# Beyond Sentential Semantic Parsing: Tackling the Math SAT with a Cascade of Tree Transducers

**Mark Hopkins, Cristian Petrescu-Prahova, Roie Levin,**
**Ronan Le Bras, Alvaro Herrasti,** and **Vidur Joshi**
Allen Institute for Artificial Intelligence
Seattle, WA

## Abstract

We present an approach for answering questions that span multiple sentences and exhibit sophisticated cross-sentence anaphoric phenomena, evaluating on a rich source of such questions – the math portion of the Scholastic Aptitude Test (SAT). By using a tree transducer cascade as its basic architecture, our system (called EUCLID) propagates uncertainty from multiple sources (e.g. coreference resolution or verb interpretation) until it can be confidently resolved. Experiments show the first-ever results (43% recall and 91% precision) on SAT algebra word problems. We also apply EUCLID to the public Dolphin algebra question set, and improve the state-of-the-art $F_1$-score from 73.9% to 77.0%.

## 1 Introduction

Math word problems pose questions that are challenging for current question answering (QA) systems to solve. Consider the following question originating from a study guide for the Math SAT[1]:

> **Example 1:** Suppose $3x + y = 15$, where $x$ is a positive integer. What is the difference between the largest possible value of $y$ and the smallest possible value of $x$, assuming that $y$ is also a positive integer?

The correct response is 11; however its relationship with the other numbers in the question (3 and 15) is oblique and not easily mapped to an operator tree or equation template. This encourages us to build a semantic parser that produces an explicit representation of what the question is asking, if we want to make quantitative progress on the question set. However, while it is not hard to formalize the semantics:

$$X \times Y = \{(x, y) \mid 3x + y = 15, x, y \in \mathbb{Z}^+\}$$
$$X = \{x \mid (x, y) \in X \times Y\}$$
$$Y = \{y \mid (x, y) \in X \times Y\}$$
$$\textbf{solve: } \max Y - \min X$$

it is not clear how to devise a compositional transformation from the original question to the formal semantics, since the meaning is dispersed throughout the discourse, such that neither the maximization nor the minimization can be locally derived from some subtree of the syntactic structure.

Moreover, SAT questions quickly reach the limits of preprocessing tools like anaphora resolution:

> **Example 2:** $\langle r, s, t \rangle$ In the sequence above, if each term after the first is x more than the previous term, what is the average of r, s, and t in terms of r and x?

Understanding this question requires a nuanced resolution of *each term after the first* to the subsequence $\langle s, t \rangle$, a coreference resolution beyond the grasp of the current state of the art.

Generally speaking, question discourse (with its complex cross-sentence semantics and anaphora) has not been a major focus of QA research. In this paper, we use math SAT questions to develop an approach to handling question discourse. Our parser uses an intermediate semantic language that allows complex semantics (like those of Example 1) to be compositionally constructed from a multi-sentence question passage (Section 5.1). By architecting our semantic parser as a cascade of nondeterministic tree transducers (Gécseg and Steinby, 1997), we can propagate uncertainty until it can be

[1]The Math SAT is a standardized exam administered to college-bound high school students in the United States.

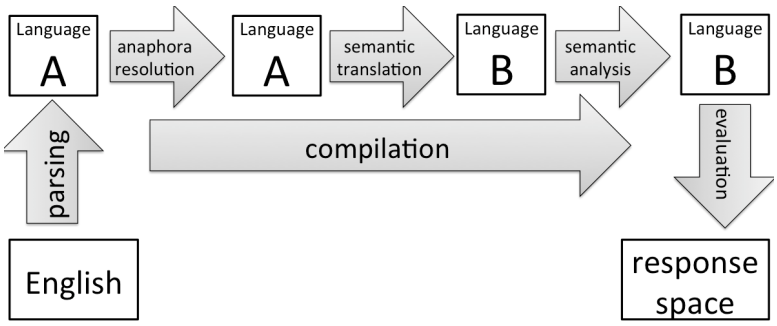


Figure 1: High-level view of EUCLID's architecture.

confidently resolved – sometimes as late as during program interpretation (Section 6). The integrated approach also allows us to handle novel classes of anaphoric phenomena by framing anaphora resolution as an operation on a parse forest decorated with implicits (Section 5.2).

Ultimately we produce an end-to-end system (called EUCLID) that achieves 43% recall and 91% precision on SAT closed-vocabulary algebra questions, a subset (described in more detail in the next section) that constitutes approximately 45% of a typical math SAT. We also achieve state-of-the-art results on the publicly released Dolphin question set (Shi et al., 2015), a set of more than 1500 algebra questions released by Microsoft Research.

## 2 Anatomy of a Math SAT

To assess our semantic parser, we compiled three question sets. Two question sets were created from sample SAT exams found in study guides (published by Kaplan and McGraw-Hill). We used the Kaplan set (12 exams, 648 total questions) for training/development and the McGraw-Hill set (13 exams[2], 686 total questions) for devtest. We reserved official practice exams (8 exams, 396 total questions) released by the College Board for final testing. We did not subselect questions from the exams, rather we used them in their entirety.[3] We encoded mathematical formatting using LATEX.

During the compilation of these questions, they were split into 4 broad categories:

1. **Algebra (closed vocabulary)** (e.g. Examples 1 and 2) **:** Algebra questions drawn from a limited mathematical vocabulary.

2. **Algebra (open vocabulary)** (e.g. "At a basketball tournament involving 8 teams, each team played 4 games with each of the other teams. How many games were played at this tournament?") **:** Algebra questions drawn from an open-ended vocabulary.

3. **Geometry:** Geometry questions, typically involving a diagram.

4. **Other** A catch-all for questions that do not fall neatly into the above categories.

In this paper, we focus our attention on closed-vocabulary algebra, which constitutes approximately 45% of the questions.

## 3 Related Work

Most of the recent work on math questions has focused on open-vocabulary algebra problems, also known as math story problems. Benchmark datasets include Alg514 (Kushman et al., 2014), AI2 (Hosseini et al., 2014), Illinois and Commoncore (Roy and Roth, 2015), and DRAW (Upadhyay and Chang, 2016). A common property of these datasets is that they have been curated such that any given question can be solved by a limited-depth operator tree (AI2, Illinois, Commoncore) or a limited set of equation templates (Alg514 and DRAW). Because of this, it is feasible to use discriminative approaches (Kushman et al., 2014; Hosseini et al., 2014; Roy and Roth, 2015; Zhou et al., 2015; Koncel-Kedziorski et al., 2015; Mitra and Baral, 2016) that extract the quantities, featurize the question, and then perform a weighted search over the space of instantiated operator trees or equation templates. However it is not clear how one can extend these discriminative techniques to handle the complex semantics found in Examples 1 and 2.

[2] 12 full exams + 1 PSAT

[3] One exception: we exclude the "comparison"-style questions (discontinued in 2005) from pre-2005 exams.

Very recently, (Matsuzaki et al., 2017) published a paper about their semantic parsing approach to pre-university math problems (harvested from Japanese exams rather than the Math SAT). It is challenging to do a direct comparison, since they report results only on the Japanese-language exams. They report end-to-end system results of 11% recall and 50% precision.

(Shi et al., 2015) harvested a fairly diverse set of closed-vocabulary algebra problems (called Dolphin) from the web and provided the first results on that dataset. Here, we demonstrate how to handle the more complex discourse semantics and anaphoric phenomena found in Math SAT questions, and establish a new state-of-the-art result on the Dolphin benchmark.

## 4 System Overview

Figure 1 shows a high-level view of our QA system. We will give a general overview in this section, and then explore more advanced concepts and examples in the subsequent section.

### 4.1 Intermediate Languages

Our QA system has two basic languages that mediate the transformation from the question passage to the answer: a syntactic language $\mathcal{A}$ and a semantic language $\mathcal{B}$.

Syntactic language $\mathcal{A}$ has a constituent-style syntax convenient[4] for the tree transducers in our cascade. In Figure 2, we show an example. We have three basic node types: *clauses*, *entities* (these correspond to noun phrases), and *details* (these correspond to adjectival and adverbial modifiers). Each node has a table of *fields* (key-value pairs) that store child relationships and auxiliary information like tense and number. For brevity, this additional structure is omitted from Figure 2, but a more explicit visualization can be found in Figure 4 (top).

A program in semantic language $\mathcal{B}$ is a set of constraint declarations. For instance, the question from Figure 2 ("Let $m+3 < 15$. If $m$ is a positive integer, what is the sum of all values of $m$?") compiles to the semantic program in Figure 3. When the form of the tree is unimportant, it will be convenient to use a more legible LISP-style format,

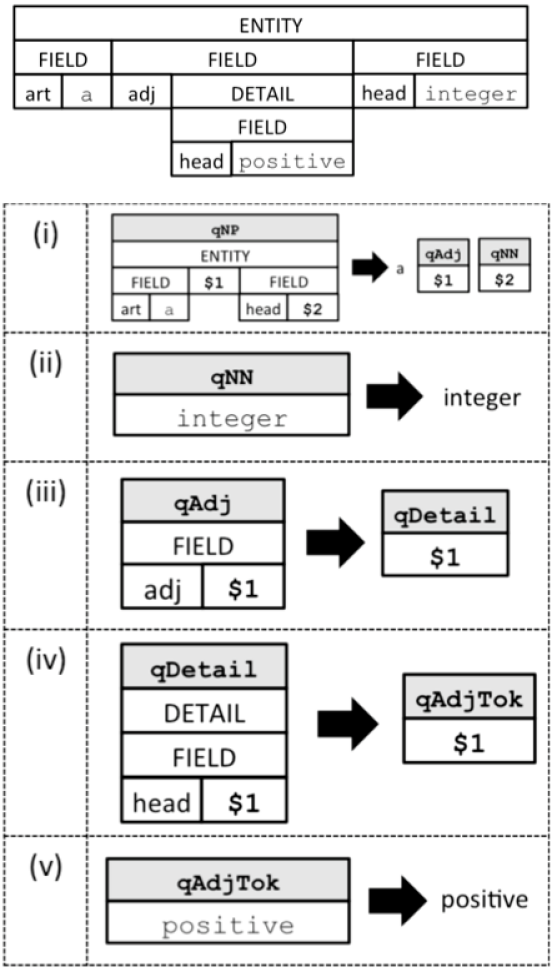


Figure 4: Example XTOPs transducer rules (bottom) used to derive a syntactic parse from the noun phrase "a positive integer" (via backward application of the transducer).

e.g.

$(< (+\ m\ 3)\ 15)$
$(> m\ 0)$
$(\mathsf{int}\ m)$
$(\mathsf{proto}\ m\ M)$
$(= ?q\ (\mathsf{sum}\ M))$

Every constraint in this program should be easily understandable, except for $(\mathsf{proto}\ m\ M)$, which loosely means that $M$ is the set of all possible values of $m$. In Section 5.1, we discuss the proto directive in more detail.

### 4.2 Syntactic Parsing

The first stage of our QA system parses the question passage into language $\mathcal{A}$[5]. We implemented

[4] We experimented with adopting an existing syntax, like the Penn Treebank Syntax or the Stanford Dependency Syntax, but it turned out to be easier to develop the syntax in parallel with the needs of our system. Having said that, it is not intended to be wildly different from those formalisms.

[5] Recall: language $\mathcal{A}$ is the syntactic language described in the previous section. An example is shown in Figure 2.

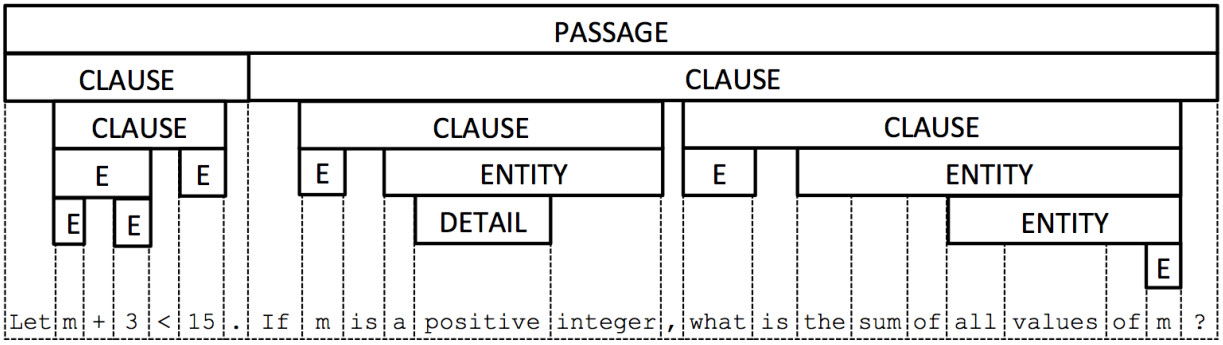


Figure 2: Example syntactic parse. For convenience, we show the correspondence of the nodes of our syntactic parse (top) to the original question passage (bottom). In the parse tree, "E" stands for "ENTITY".

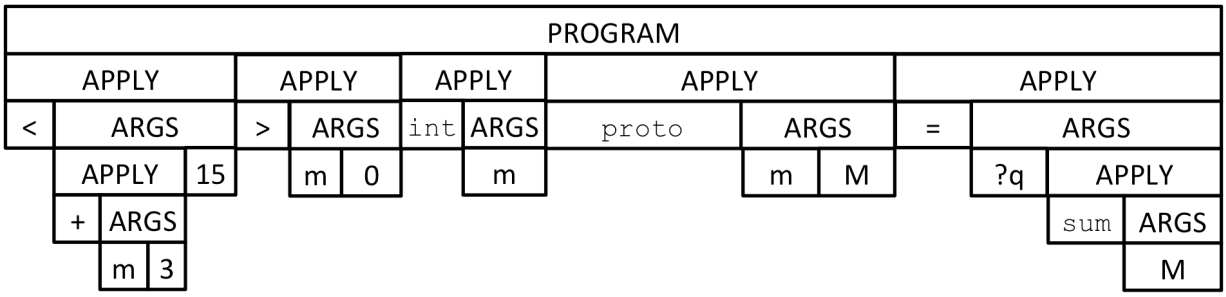


Figure 3: Example semantic program for the question "Let $m + 3 < 15$. If $m$ is a positive integer, what is the sum of all values of $m$?"

the parser as the backward application of an extended top-down tree-to-string (XTOPs) transducer[6].

We refer the reader to (Maletti et al., 2009) for a theoretical presentation of XTOPs, and instead give a brief intuitive presentation of the device. An XTOPs transducer defines a top-down transformation from a tree language to a string language, via a set of stateful rewrite rules. For instance, rules (i) through (v) of Figure 4 can generate the string "a positive integer" from the $\mathcal{A}$-tree pictured at the top of the figure, given start state **qNP**.

Given an XTOPs transducer $M$, we can parse string $s$ through *backward application* of the transducer, i.e. compute the set of trees $M^{-1}(s)$ that could have generated string $s$ from the start state. Efficient backward application of XTOPs transducers is supported by packages like Tiburon (May and Knight, 2006).

Our XTOPs transducer has approximately 140 states and 550 engineered rules (approximately 200 of these rules are used for parsing formal mathematics and a subset of LaTeX). Most lexical rules are automatically generated on-the-fly from WordNet (Miller, 1995).

### 4.3 Compilation

We then compile the parses of the question passage, by running them forward through a cascade of bottom-up tree transducers (Engelfriet, 1975). Again we refer the reader to the literature (Maletti, 2011, 2014) for a theoretical presentation of bottom-up tree transducers, and use Figure 5 to provide intuition about the device. A bottom-up tree transducer defines a transformation from a tree language to a (possibly different) tree language, via a set of stateful bottom-up rewrite rules.

In Figure 5, we show how this transformation works in the context of the semantic translation step, which uses a multi bottom-up transducer (MBOT) to map our syntactic language $\mathcal{A}$ into our semantic language $\mathcal{B}$. There is a single state (indicated by a gray shaded rectangle) that has two children: (i) a *return value*, and (ii) a set of *side-effect* statements.

[6] We chose to implement the parsing step by engineering a transducer rather than using an off-the-shelf statistical parser. While we tried to retrofit a parser – e.g. as done by (Seo et al., 2015) – to serve our needs, it turned out to be somewhat more robust (and relatively simple) to engineer our own.

The first rule application transforms "all values of $m$" into a *return value* of $M$ (a new variable introduced to indicate the set of all values of variable $m$) and a *side-effect* (proto $m$ $M$), indicating that $M$ equals the set of all possible values of $m$. The second rule application transforms "the sum of $M$" into a *return value* of (sum $M$), and propagates upward the accumulated side-effects.

We implemented all three compilation steps from Figure 1 (anaphora resolution, semantic translation, and semantic analysis) as the forward application of a bottom-up tree transducer. Anaphora resolution resolves any nodes that refer to other nodes in the tree. Semantic translation translates syntactic language $\mathcal{A}$ into semantic language $\mathcal{B}$. Semantic analysis type-checks the trees for internal consistency.

### 4.4 Interpretation

Finally, each derived $\mathcal{B}$-tree is sent to an evaluator to obtain an answer. Our main evaluator is a wrapped version of Z3 (de Moura and Bjorner, 2008), a widely used Satisfiability Modulo Theories (SMT) solver. If it does not find an answer, we fall back to a numeric optimization solver similar to one used by (Seo et al., 2015).

## 5 Spotlights

Having provided a bird's eye view in the last section, we now spotlight some key details of our QA system.

### 5.1 Spotlight: Complex Aggregations

A core challenge of semantic parsing is how best to read complex semantic phenomena from a syntactic representation. Two such phenomena are superlatives and counting. GeoQuery (Zelle and Mooney, 1996) has examples[7] of these, as does[8] WebQuestions (Berant et al., 2013). Unfortunately, it is not clear how existing strategies for dealing with aggregative constructs (e.g. (Liang et al., 2011)) can be extended to the more complex multi-sentence questions found on the SATs. For instance, the basic semantics of Example 1 (enumerated in Section 1) is dispersed throughout the question passage, such that neither the maximization nor the minimization can be locally derived from some subtree of the dependency structure.

[7] e.g. "What is the capital of the state that borders the most states?"

[8] e.g. "How many pets did John F. Kennedy own?"

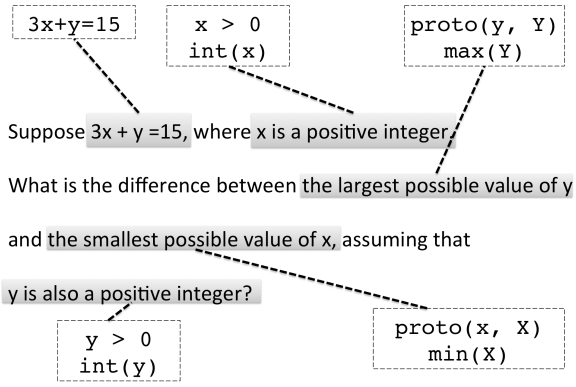


Figure 6: Understanding complex aggregations by decomposing them into order-independent atoms.

To deal with this challenge, we designed our semantic language $\mathcal{B}$ to decompose the semantics of aggregative constructs into order-independent atoms. Consider the following restatement of the semantics of Example 1:

$$\begin{gathered}\mathsf{proto}(\dot{x}, X)\\ \mathsf{proto}(\dot{y}, Y)\\ 3\dot{x} + \dot{y} = 15\\ \dot{x} > 0\\ \dot{y} > 0\\ \dot{x} \in \mathbb{Z}\\ \dot{y} \in \mathbb{Z}\\ \textbf{solve: } \max Y - \min X\end{gathered}$$

where $\mathsf{proto}(\dot{z}, Z)$ designates that a variable $\dot{z}$ should be treated as the prototype variable of a statement in set-builder notation, i.e. $Z = \{\dot{z} \mid ...\}$. We treat any other statement featuring prototype variable $\dot{z}$ as a constraint appearing on the right side of the set-builder statement. If there are multiple prototype statements, they are grouped into a single set-builder statement (as occurs with $\dot{x}$ and $\dot{y}$ in our example).

The power of this decomposition is that it can be reconstructed piecemeal from an arbitrarily complex passage. The atomic statements can be interpreted locally in an arbitrary order, as in Figure 6, then synthesized into set-builder notation during evaluation.

### 5.2 Spotlight: Complex Anaphoric Phenomena

The anaphora resolution task (Ge et al., 1998) is typically defined at the lexical level. For instance,

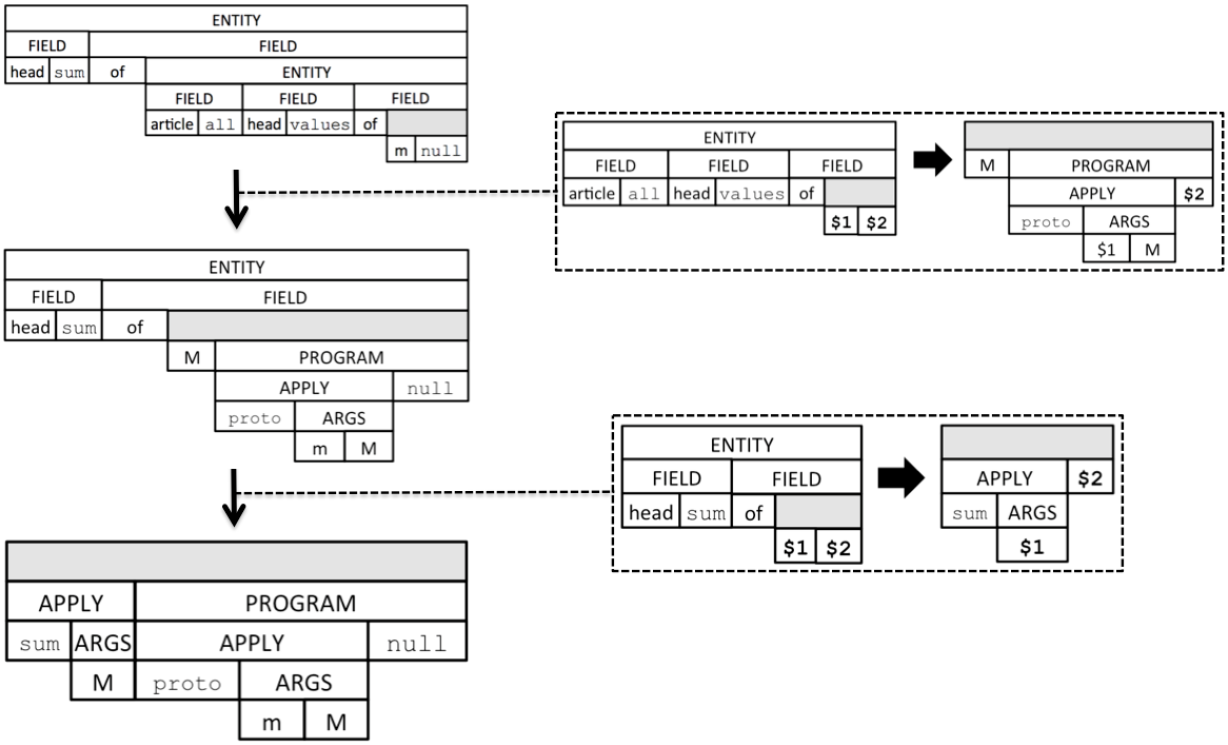


Figure 5: Example semantic translation using an MBOT.

in the sentence "c is equal to its square," a traditional evaluation like the CoNLL-2011 Shared Task (Pradhan et al., 2011) would ask whether the string "its" is aligned to the string "c". These evaluations also assume that both the reference and the referent (a.k.a. antecedent) are contiguous strings in the text.

Math SAT problems exhibit a host of new challenges that fall outside traditionally studied definitions of anaphora resolution:

- **One-to-many coreference**[9] *(One integer is 5 more than another. What is the sum of the numbers?)*: "The numbers" refers to two discontiguous strings: "one integer" and "another".

- **Implicit set reference** *(Two numbers sum to 5. If the first is 2, what is the second?)*: "The second" implies a latent set that needs to be resolved (to "two numbers") in order to understand the sentence. This phenomenon is shown in Figure 7.

- **Implicit clausal reference** *(If 7 is divided by 3, what is the remainder?)*: "The remainder" implies a latent clause that needs to be resolved (to "7 is divided by 3") in order to understand the sentence.

We address this broader class of anaphora by a two-pass process:

1. First, we introduce implicit sets and clauses when appropriate. For instance, implicit sets are introduced for superlative and ordinal constructions, while implicit clauses are introduced for functional nouns like "remainder." In Figure 7, these implicits are depicted as bracketed phrases (i.e. [of a set]).

2. Anaphora resolution then proceeds as a bottom-up tree-labeling process, shown in Figure 7. For each subtree, a resolution function $\rho$ partially maps subtree entities to subtree nodes. Note that ancestors can overwrite the resolutions of their descendants. This occurs in the second example of Figure 7, where the implicit set E7 is initially resolved to implicit set E4, but is later resolved to the entity E1 ("two numbers") once it comes into scope.

In our initial system, the resolution function $\rho$ was engineered heuristically. We later replaced this

[9] A recent ACL paper (Vala et al., 2016) has provided a preliminary treatment of this phenomenon.

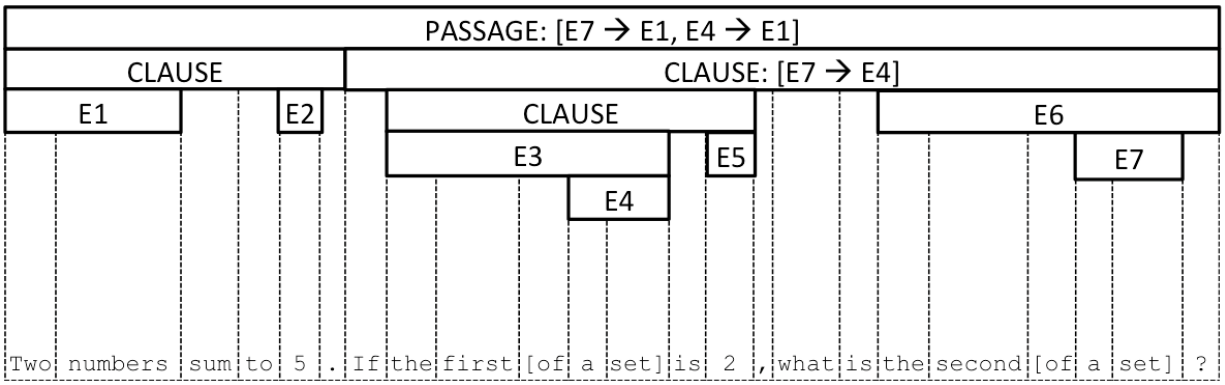


Figure 7: Bottom-up anaphora resolution in our QA system. For convenience, we show the correspondence of the nodes of our syntactic parse (top) to the original question passage (bottom). In the parse tree, "E" is an abbreviation for ENTITY.

with a learned version (by using our system to generate training data). Due to space considerations, details are omitted.

## 6 Results with the Unweighted Nondeterministic Cascade

In the basic cascade from Section 4, the number of trees passed from module to module can expand, but it can also contract (for instance, in the semantic translation step, there can be multiple ways of translating a parse, or none at all). This allows the QA system to disambiguate question passages by eliminating parses for which there is no consistent semantics. On the subset of the Kaplan questions for which at least one parse exists, the average number of trees after the parsing step is 7.5. The average number of trees after the semantic analysis step goes down to only 2.4. At that point, obviously we need to choose some priority in which to feed these finalized programs to the evaluation module. Using a simple heuristic (process smaller programs first), we obtain 70.2% recall and 95.8% precision on the Kaplan closed-vocabulary algebra questions[10].

This high precision can be partially attributed to the fact that most SAT questions are multiple-choice (thus we can sequentially evaluate the finalized programs until we find a viable answer). We do not have that luxury on the Dolphin dataset, a set of direct-answer algebra questions curated by Microsoft Research (split into a development set of 374 questions and a test set of 1504 questions). On the subset of the development questions for which at least one parse exists (90.3% of the questions), the average number of trees after the parsing step is 4.3. The average number of trees after the semantic analysis step goes down to 1.5. Our basic system obtains 66.3% recall on the development questions. Naturally the precision is not as high as on the multiple choice questions, but surprisingly we still obtain 85.5% precision, even with an unweighted cascade.

## 7 Introducing a Parse Ranker

Most of this precision loss is due to legitimate parse ambiguity that cannot be resolved through semantic interpretability alone. Rather, the disambiguation requires some additional pragmatic convention. Consider the example: "When the reciprocal of three times a number is subtracted from the reciprocal of the number, the result is one sixth. Find the number." By interpreting "the reciprocal of three" as $\frac{1}{3}$, the meaning of this question becomes "When $\frac{1}{3}$ times a number is subtracted from the reciprocal of the number, the result is one sixth. Find the number." This is not however the most human-intuitive interpretation of the question. Somehow the system must identify the pragmatic cues that cause humans to disprefer this interpretation.

To identify these cues, we insert a *parse ranking module* between the parsing module and the anaphora resolution module (see Figure 1 for a reminder of the system components). The goal of the parse ranker is to associate a lower cost to "more intuitive" interpretations when there are multiple plausible syntactic interpretations. The

[10] Recall and precision numbers are computed over the entire set of questions, regardless of whether they have a valid parse.

| | recall | prec. | F1 |
|---|---|---|---|
| **Kaplan (non-blind)** | 70.2 | 95.8 | 81.0 |
| **McGraw-Hill (blind)** | 41.0 | 91.8 | 56.7 |
| **Official (blind)** | 43.1 | 90.8 | 58.5 |

Table 1: Results on the closed-vocabulary algebra subsets of our Math SAT question sets.

| | EUCLID | | | (Shi et al., 2015) | | |
|---|---|---|---|---|---|---|
| | **rec.** | **prec.** | **F1** | **rec.** | **prec.** | **F1** |
| **dev** | 78.1 | 97.0 | 86.5 | - | - | - |
| **test** | 65.1 | 94.1 | 77.0 | 60.3 | 95.4 | 73.9 |

Table 2: Results on the Dolphin question sets. The increase in recall is statistically significant with a $P$-value $< 0.01$.

rest of the cascade propagates these costs. Similar to existing work, e.g. (Charniak and Johnson, 2005), we implement the cost function as an $L_1$-regularized logistic regression model.

Adding the trained parse ranker module improves performance on the Dolphin development set to 75.7% recall and 97.3% precision (from 66.3% recall and 85.5% precision).

## 8 Final Results

Results from our final system are shown in Table 1 (for the closed-vocabulary algebra subsets of our math SAT question sets) and Table 2 (for the Dolphin question sets). EUCLID generalizes reasonably well to the blind SAT questions, achieving approximately 60% of the system's recall on the training questions, at a precision of approximately 91%. To give a sense of the extent of the generalization from training to test, Table 3 offers a couple of correctly answered questions from the blind[11] McGraw-Hill set, plus their closest analog in the training questions (by edit distance). The performance on the blind test sets (including all questions, not just closed-vocabulary algebra) corresponds to an SAT score of approximately 350 (out of 800). A random-guessing baseline has an expected score of 200.

Table 4 provides a failure analysis on the McGraw-Hill data, categorizing a sample of 50 questions. Half of the questions failed to have a

[11]Apart from harvesting a sample of correctly answered questions for this analysis, the McGraw-Hill set was kept completely blind. The official set was left completely untouched.

| **development (blind)** | **training** |
|---|---|
| Set $M$ consists of the consecutive integers from -15 to $y$, inclusive. If the sum of all of the integers in set $M$ is 70, how many numbers are in the set? | If the sum of the consecutive integers from -15 to $x$, inclusive, is 51, what is the value of $x$? |
| If $a$, $b$, and $c$ are positive even integers such that $a < b < c$ and $a + b + c = 60$, then the greatest possible value of $c$ is | If $x$ and $y$ are different positive integers and $3x + y = 17$, the difference between the largest possible value of $y$ and the smallest possible value of $x$ is |

Table 3: Some correctly answered questions on the blind McGraw-Hill set, and their closest parallel (by edit distance) in the training set (Kaplan).

| **development (blind)** | **training** |
|---|---|
| failed to parse | 50% |
| failed to map parse into a semantic program | 24% |
| failed to produce an answer from any semantic program | 18% |
| produced an incorrect answer | 8% |

Table 4: Error analysis on the blind McGraw-Hill set, surveying the first point of failure for a sample of 50 incorrectly answered questions.

valid parse. Roughly a quarter of the questions had at least one valid parse, but none of these resulted in a semantic program. 18% of the questions compiled into at least one semantic program, but none of these produced an answer when fed to the interpreter. 8% of the questions compiled into a semantic form that produced an incorrect answer.

Besides the Math SAT datasets, EUCLID also has state-of-the-art performance on the public Dolphin question set, achieving an absolute recall improvement[12] of nearly 5% with a small loss in precision. This raises the state-of-the-art $F_1$-score on this data set from 73.9% to 77.0%.

[12]This improvement is statistically significant with a P-value $< 0.01$.

| genre | dataset | blind? | recall | precision | F1-score |
|---|---|---|---|---|---|
| closed algebra | Kaplan | no | 70.2 | 95.8 | 81.0 |
| | McGraw-Hill | yes | 41.0 | 91.8 | 56.7 |
| | official | yes | 43.1 | 90.8 | 58.5 |
| geometry | Kaplan | no | 11.7 | 95.0 | 20.8 |
| | McGraw-Hill | yes | 5.6 | 76.9 | 10.4 |
| open algebra | hosseini-ma2 | no | 34.7 | 57.5 | 43.3 |
| | hosseini-ma1 | yes | 29.9 | 45.4 | 36.1 |

Table 5: Snapshot of early progress across several subgenres of the Math SAT. In our early stages, we are hillclimbing on the hosseini datasets from (Hosseini et al., 2014), which are simpler than the open-vocabulary algebra questions from the Math SAT.

## 9 Towards a Broad-Coverage SAT solver

This paper reports on the first steps of a longer-term initiative to build a unified system that can pass the math SAT. We have made some preliminary forays into extending the system to handle the more complex subdomains described in Section 2, namely open-vocabulary algebra and geometry. Key research challenges presented by these new domains are:

- **Mapping into richer semantic languages:** The math story problems of open-vocabulary algebra require languages that reason about state change and can introduce assumptions not explicitly represented in the text.
- **Robustly synthesizing diagram and text information:** For geometry questions, we are building on key early work in this area performed by (Seo et al., 2014, 2015).
- **Extending the system in a scalable way:** Writing new transducer rules for each new domain is not a sustainable way to extend our system. We are exploring how to use natural language to "program" our system, e.g. by automatically inducing transducer rules for paraphrase text.

A snapshot of our current progress is shown in Table 5.

## 10 Discussion

In the process of creating a QA system for math SAT questions, this project has yielded several general strategies for beyond-sentential semantic parsing. For instance:

- One can modularize the parser as a cascade of tree transducers, allowing uncertainty about anaphora resolution and lexical interpretation to be propagated until it can be confidently resolved, sometimes as late as program interpretation (see Section 6).
- One can atomize complex semantic phenomena (e.g. aggregative constructs) into small order-independent pieces. This allows a simpler transformation from a syntactic form, because these atoms can be locally recognized, incrementally composed, and globally reconstituted into a structured semantics (see Section 5.1).
- One can reframe bread-and-butter NLP tasks (e.g. anaphora resolution) to fit better within (and take advantage of) the context of the transducer cascade (see Section 5.2)

An important focus of this paper has been issues of representation, namely how to develop intermediate structured languages that facilitate the automatic transformation of question discourse into a response. Because we can use the resulting QA system to generate gold intermediate trees for any correctly answered question in our dataset, one way to view this work is as a data annotation project. One distinguishing advantage is that our intermediate languages come with a "proof of usefulness." They are not designed based on speculative utility – rather, they have already proven useful in the context of a functioning QA system.

## Acknowledgments

The authors would like to thank Luke Zettlemoyer, Jayant Krishnamurthy, Oren Etzioni, and the anonymous reviewers for valuable feedback on earlier drafts of the paper.

## References


Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic parsing on freebase from question-answer pairs. In *EMNLP*.

Eugene Charniak and Mark Johnson. 2005. Coarse-to-fine n-best parsing and maxent discriminative reranking. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 173–180. Association for Computational Linguistics.

Joost Engelfriet. 1975. Bottom-up and top-down tree transformationsa comparison. *Mathematical systems theory*, 9(2):198–231.

Niyu Ge, John Hale, and Eugene Charniak. 1998. A statistical approach to anaphora resolution.

Ferenc Gécseg and Magnus Steinby. 1997. Tree languages. In *Handbook of formal languages*, pages 1–68. Springer.

Mohammad Javad Hosseini, Hannaneh Hajishirzi, Oren Etzioni, and Nate Kushman. 2014. Learning to solve arithmetic word problems with verb categorization. In *EMNLP*.

Rik Koncel-Kedziorski, Hannaneh Hajishirzi, Ashish Sabharwal, Oren Etzioni, and Siena Dumas Ang. 2015. Parsing algebraic word problems into equations. *TACL*, 3:585–597.

Nate Kushman, Luke S. Zettlemoyer, Regina Barzilay, and Yoav Artzi. 2014. Learning to automatically solve algebra word problems. In *ACL*.

Percy Liang, Michael I. Jordan, and Dan Klein. 2011. Learning dependency-based compositional semantics. In *ACL*.

Andreas Maletti. 2011. How to train your multi bottom-up tree transducer. In *ACL*.

Andreas Maletti. 2014. The power of regularity-preserving multi bottom-up tree transducers. In *CIAA*.

Andreas Maletti, Jonathan Graehl, Mark Hopkins, and Kevin Knight. 2009. The power of extended top-down tree transducers. *SIAM J. Comput.*, 39:410–430.

Takuya Matsuzaki, Takumi Ito, Hidenao Iwane, Hirokazu Anai, and Noriko H. Arai. 2017. Semantic parsing of pre-university math problems. In *ACL*.

Jonathan May and Kevin Knight. 2006. Tiburon: A weighted tree automata toolkit. In *CIAA*.

George A Miller. 1995. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41.

Arindam Mitra and Chitta Baral. 2016. Addressing a question answering challenge by combining statistical methods with inductive rule learning and reasoning. In *AAAI*.

Leonardo Mendona de Moura and Nikolaj Bjorner. 2008. Z3: An efficient smt solver. In *TACAS*.

Sameer Pradhan, Lance A. Ramshaw, Mitchell P. Marcus, Martha Palmer, Ralph M. Weischedel, and Nianwen Xue. 2011. Conll-2011 shared task: Modeling unrestricted coreference in ontonotes. In *CoNLL Shared Task*.

Subhro Roy and Dan Roth. 2015. Solving general arithmetic word problems. In *EMNLP*.

Min Joon Seo, Hannaneh Hajishirzi, Ali Farhadi, and Oren Etzioni. 2014. Diagram understanding in geometry questions. In *AAAI*.

Min Joon Seo, Hannaneh Hajishirzi, Ali Farhadi, Oren Etzioni, and Clint Malcolm. 2015. Solving geometry problems: Combining text and diagram interpretation. In *EMNLP*.

Shuming Shi, Yuehui Wang, Chin-Yew Lin, Xiaojiang Liu, and Yong Rui. 2015. Automatically solving number word problems by semantic parsing and reasoning. In *EMNLP*.

Shyam Upadhyay and Ming-Wei Chang. 2016. Annotating derivations: A new evaluation strategy and dataset for algebra word problems. *CoRR*, abs/1609.07197.

Hardik Vala, Andrew Piper, and Derek Ruths. 2016. The more antecedents, the merrier: Resolving multi-antecedent anaphors. In *ACL*.

John M. Zelle and Raymond J. Mooney. 1996. Learning to parse database queries using inductive logic programming. In *AAAI/IAAI, Vol. 2*.

Lipu Zhou, Shuaixiang Dai, and Liwei Chen. 2015. Learn to solve algebra word problems using quadratic programming. In *EMNLP*.