# Semantic Parsing of Pre-university Math Problems

**Takuya Matsuzaki**[1], **Takumi Ito**[1], **Hidenao Iwane**[2], **Hirokazu Anai**[2], **Noriko H. Arai**[3]
[1] Nagoya University, Japan
{matuzaki,takumi_i}@nuee.nagoya-u.ac.jp
[2] Fujitsu Laboratories Ltd., Japan
{iwane,anai}@jp.fujitsu.com
[3] National Institute of Informatics, Japan
arai@nii.ac.jp

## Abstract

We have been developing an end-to-end math problem solving system that accepts natural language input. The current paper focuses on how we analyze the problem sentences to produce logical forms. We chose a hybrid approach combining a shallow syntactic analyzer and a manually-developed lexicalized grammar. A feature of the grammar is that it is extensively typed on the basis of a formal ontology for pre-university math. These types are helpful in semantic disambiguation inside and across sentences. Experimental results show that the hybrid system produces a well-formed logical form with 88% precision and 56% recall.

## 1 Introduction

Frege and Russell, the initiators of the mathematical logic, delved also into the exploration of a theory of natural language semantics (Frege, 1892; Russell, 1905). Since then, symbolic logic has been a fundamental tool and a source of inspiration in the study of language meaning. It suggests that the formalization of the two realms, mathematical reasoning and language meaning, is actually the two sides of the same coin – probably, we could not even conceive the idea of formalizing language meaning without *grounding it* onto mathematical reasoning. This point was first clarified by Tarski (1936; 1944) mainly on formal languages and then extended to natural languages by Davidson (1967). Montague (1970a; 1970b; 1973) further embodied it by putting forward a terrifyingly arrogant and attractive idea of seeing a natural language *as* a formal language.

The automation of end-to-end math problem solving thus has an outstanding status in the research themes in natural language processing. The conceptual basis has been laid down, which connects text to the truth (= answer) through reasoning. However, we have not seen a fully automated system that instantiates it end-to-end. We wish to add a piece to the big picture by materializing it.

Define the two straight lines $L_1$ and $L_2$ on the $xy$-plane as $L_1$: $y = 0$ ($x$-axis) and $L_2$: $y = \sqrt{3}x$. Let $P$ be a point on the $xy$-plane. Let $Q$ be the point symmetric to $P$ about the straight line $L_1$, and let $R$ be the point symmetric to $P$ about the straight line $L_2$. Answer the following questions:

(1) Let $(a, b)$ be the coordinates of $P$, then represent the coordinates of $R$ using $a$ and $b$.

(2) Assuming that the distance between the two points $Q$ and $R$ is 2, find the locus $C$ of $P$.

(3) When the point $P$ moves on $C$, find the maximum area of the triangle $PQR$ and the coordinates of $P$ that gives the maximum area.

(Hokkaido Univ., 1999-Sci-3)

Figure 1: Example problem

Past studies have mainly targeted at primary school level arithmetic word problems (Bobrow, 1964; Charniak, 1969; Kushman et al., 2014; Hosseini et al., 2014; Shi et al., 2015; Roy and Roth, 2015; Zhou et al., 2015; Koncel-Kedziorski et al., 2015; Mitra and Baral, 2016; Upadhyay et al., 2016). In their nature, arithmetic questions are quantifier-free. Moreover they tend to include only $\wedge$ (and) as the logical connective. The main challenge in these works was to extract simple numerical relations (most typically equations) from a real-world scenario described in a text.

Seo et al. (2015) took SAT geometry questions as their benchmark. However, the nature of SAT geometry questions restricts the resulting formula's complexity. In §3, we will show that none of them includes $\forall$ (for all), $\vee$ (or) or $\rightarrow$ (implies). It suggests that this type of questions require little need to analyze the logical structure of the problems beyond conjunctions of predicates.

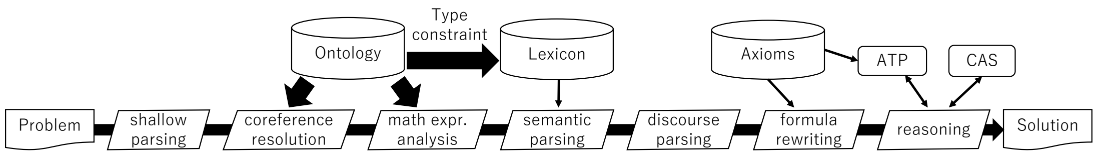

Figure 2: Overview of the end-to-end math problem solving system

We take pre-university math problems falling in the theory of real-closed fields (RCF) as our benchmark because of their variety and complexity. The subject areas include real and linear algebra, complex numbers, calculus, and geometry. Furthermore, many problems involve more than one subject: e.g., algebraic curves and calculus as in Fig. 1. Their logical forms include all the logical connectives, quantifiers, and $\lambda$-abstraction. Our goal is to recognize the complex logical structures precisely, including the scopes of the quantifiers and other logical operators.

In the rest of the paper, we first present an overview of an end-to-end problem solving system (§2) and analyze the complexity of the pre-university math benchmark in comparison with others (§3). Among the modules in the end-to-end system, we focus on the sentence-level semantic parsing component and describe an extensively-typed grammar (§4 and §5), an analyzer for the math expressions in the text (§6), and two semantic parsing techniques to fight against the scarcity of the training data (§7) and the complexity of the domain (§8). Experimental results show the effectiveness of the presented techniques as well as the complexity of the task through an in-depth analysis of the end-to-end problem solving results (§9).

## 2 End-to-end Math Problem Solving

Fig. 2 presents an overview of our end-to-end math problem solving system. A math problem text is firstly analyzed with a dependency parser. Anaphoric and coreferential expressions in the text are then identified and their antecedents are determined. We assume the math formulas in the problems are encoded in MathML presentation mark-up. A specialized parser processes each one of them to determine its syntactic category and semantic content. The semantic representation of each sentence is determined by a semantic parser based on Combinatory Categorial Grammar (CCG) (Steedman, 2001, 2012). The output from the CCG parser is a ranked list of sentence-level logical forms for each sentence.

| Dataset | Succeeded | | Failed | |
|---|---|---|---|---|
| | Success% | Avg. Time | Timeout | Other |
| DEV | 75.3% (131/174) | 10.5s | 16.7% | 8.1% |
| TEST | 78.2% (172/220) | 16.2s | 15.0% | 6.8% |

Table 1: Performance of the reasoning module on *manually* formalized pre-university problems

After the sentence-level processing steps, we determine the logical relations among the sentence-level logical forms (discourse parsing) by a simple rule-based system. It produces a tree structure whose leaves are labeled with sentences and internal nodes with logical connectives. Free variables in the logical form are then bound by some quantifiers (or kept free) and their scopes are determined according to the logical structure of the problem. A semantic representation of a problem is obtained as a formula in a higher-order logic through these language analysis steps.

The logical representation is then rewritten using a set of axioms that define the meanings of the predicate and function symbols in the formula, such as `maximum` defined as follows:

$$\mathtt{maximum}(x, S) \leftrightarrow x \in S \land \forall y(y \in S \rightarrow y \leq x),$$

as well as several logical rules such as $\beta$-reduction. We hope to obtain a representation of the initial problem expressed in a decidable math theory such as RCF through these equivalence-preserving rewriting. Once we find such a formula, we invoke a computer algebra system (CAS) or an automatic theorem prover (ATP) to derive the answer.

The reasoning module (i.e., the formula rewriting and the deduction with CAS and ATP) of the system has been extensively tested on a large collection of manually formalized pre-university math problems that includes more than 1,500 problems. It solves 70% of the them in the time limit of 10 minutes per problem. Table 1 shows the rate of successfully solved problems in the *manually* formalized version of the benchmark problems used in the current paper.

| | Prob-lems | Avg. tokens | Avg. sents. | Uniq. words | Atoms | $\exists$ | $\forall$ | $\lambda$ | $\wedge$ | $\vee$ | $\neg$ | $\rightarrow$ | Unique sketches |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Jobs | 640 | 9.83 | 1.00 | 391 | 4.63 | 1.71 | 0.00 | 0.00 | 1.06 | 0.01 | 0.13 | 0.00 | 8 |
| GeoQuery | 880 | 8.56 | 1.00 | 284 | 4.25 | 1.70 | 0.00 | 1.04 | 1.18 | 0.00 | 0.02 | 0.00 | 20 |
| Geometry | 119 | 23.64 | 1.74 | 202 | 11.00 | 7.45 | 0.00 | 0.06 | 1.00 | 0.00 | 0.04 | 0.00 | 4 |
| Univ (dev) | 174 | 70.34 | 3.45 | 363 | 10.99 | 5.10 | 1.10 | 1.11 | 1.71 | 0.02 | 0.49 | 0.35 | 76 |
| Univ (test) | 220 | 70.85 | 4.02 | 366 | 9.70 | 4.58 | 1.10 | 1.00 | 1.62 | 0.02 | 0.28 | 0.23 | 72 |

Table 2: Profile of pre-university math benchmark data and other semantic parsing benchmark data sets

| Jobs | | GeoQuery | | Geometry | | Univ (dev) | |
|---|---|---|---|---|---|---|---|
| $\exists P$ | 81% | $\exists P$ | 46% | $\exists P$ | 94% | $\exists P$ | 25% |
| $P$ | 6% | $\exists P(\lambda \exists P)$ | 24% | $\exists(P \wedge \neg P)$ | 3% | $\exists(P \wedge \neg P)$ | 7% |
| $\exists(P \wedge \neg \exists P)$ | 5% | $P(\lambda \exists P)$ | 8% | $\exists(P \wedge P(\lambda P))$ | 2% | $P(\lambda \exists P)$ | 5% |
| $\exists(P \wedge \neg P)$ | 5% | $\exists(P \wedge P(\lambda \exists P))$ | 7% | $P(\lambda \exists P)$ | 1% | $\exists(P \wedge P(\lambda f))$ | 4% |
| | 97% | | 85% | | 100% | | 41% |

Table 3: Top four most frequest sketches and their coverage over the dataset

| Sketch | Freq. |
|---|---|
| $\forall(P \rightarrow \exists(\forall(P \rightarrow P) \wedge P))$ | 2 |
| $\exists(\exists(\neg P \wedge P) \wedge P \wedge P(\lambda f)) \wedge P(\lambda(P \rightarrow P)))$ | 1 |
| $\exists(P \wedge P(\lambda(\neg P \wedge \exists(\exists P \wedge P))))$ | 1 |
| $\exists(P \wedge P(\lambda f)) \wedge P(\lambda(\neg P \wedge P)) \wedge P(\lambda P))$ | 1 |

Table 4: Less frequent sketches in Univ (dev)

## 3 Profile of the Benchmark Data

Our benchmark problems, Univ, were collected from the past entrance exams of seven top-ranked universities in Japan. In the exams held in odd numbered years from 1999 to 2013, we exhaustively selected the problems which are ultimately expressible in RCF. They occupied 40% of all the problems. We divided the problems into two sets: Dev for development (those from year 1999 to 2005) and Test for test (those from year 2007 to 2013). Dev was used for the lexicon development and the tuning of the end-to-end system. The problem texts (both in English and Japanese) with MathML mark-up and manually translated logical forms are publicly available at https://github.com/torobomath.

The manually translated logical forms were formulated in a higher-order semantic language introduced later in the paper. The translation was done as faithfully as possible to the original wordings of the problems. They thus keep the inherent logical structures expressed in natural language.

Table 2 lists several statistics of the Univ problems in the English version and their manual formalization. For comparison, the statistics of three other benchmarks are also listed. Jobs and GeoQuery are collections of natural language queries against databases. They have been widely used as benchmarks for semantic parsing (e.g., Tang and Mooney, 2001; Zettlemoyer and Collins, 2005, 2007; Kwiatkowski et al., 2010, 2011; Liang et al., 2011). The queries are annotated with logical forms in Prolog. We converted them to equivalent higher-order formulas to collect comparable statistics. Geometry is a collection of SAT geometry questions compiled by Seo et al. (2015). We formalized the Geometry questions[1] in our semantic language in the same way as Univ.

In Table 2, the first column lists the number of problems. The next three provide statistics of the problem texts: average number of words and sentences in a problem ('Avg. tokens' and 'Avg. sents'), and the number of unique words in the whole dataset.[2] They reveal that the sentences in Univ are significantly longer than the others and more than three sentences have to be correctly processed for a problem.

The remaining columns provide the statistics about the logical complexities of the problems. 'Atoms' stands for the average number of the occurrences of predicates per problem. The next three columns list the number of variables bound by $\exists$, $\forall$, and $\lambda$. We count sequential occurrences of the same binder as one. The columns for $\wedge$, $\vee$, $\neg$, and $\rightarrow$ list the average number of them per problem.[3] We can see Univ includes a wider variety of quantifiers and connectives than the others.

The final column lists the numbers of unique 'sketches' of the logical forms in the dataset. What

[1]Including *all* conditions expressed in the diagrams.

[2] All the math formulas in the texts were replaced with a special token "MATH" before counting words.

[3] $\wedge$ and $\vee$ was counted as operators with arbitrary arity. E.g., there is only one $\wedge$ in $A \wedge B \wedge C$.

we call 'sketch' here is a signature that encodes the overall structure of a logical form. Table 3 shows the top four most frequent sketches observed in the datasets. In a sketch, $P$ stands for a (conjunction of) predicate(s) and $f$ stands for a term. $\exists$, $\forall$, and $\lambda$ stand for (immediately nested sequence of) the binders.

To obtain the sketch of a formula $\phi$, we first replace all the predicate symbols in $\phi$ to $P$ and function symbols and constants to $f$. We then eliminate all variables in $\phi$ and 'flatten' it by applying the following rewriting rules to the sub-formulas in $\phi$ in the bottom-up order:

$$\begin{gathered} f(..., f(\alpha_1, \alpha_2, ..., \alpha_n), ...) \Rightarrow f(..., \alpha_1, \alpha_2, ..., \alpha_n, ...) \\ P(..., f(\alpha_1, \alpha_2, ..., \alpha_n), ...) \Rightarrow P(..., \alpha_1, \alpha_2, ..., \alpha_n, ...) \\ \alpha \lor \alpha \lor \beta \Rightarrow \alpha \lor \beta, \quad \alpha \land \alpha \Rightarrow \alpha \\ \exists\exists\psi \Rightarrow \exists\psi, \quad \forall\forall\psi \Rightarrow \forall\psi, \quad \lambda\lambda\psi \Rightarrow \lambda\psi \end{gathered}$$

Finally, we sort the arguments of $P$s and $f$s and remove the duplicates among them. For instance, to obtain the sketch of the following formula:

$$\forall k \forall m \left( \begin{array}{c} \mathtt{maximum}(m, \mathtt{set}(\lambda e.(e < k))) \\ \rightarrow k - 1 \leq m \land m < k \end{array} \right),$$

we replace the predicate/function symbols as in:

$$\forall k \forall m \left( \begin{array}{c} P(m, f(\lambda e.P(e, k))) \\ \rightarrow P(f(k, f), m) \land P(m, k) \end{array} \right),$$

and then eliminate the variables to have:

$$\forall\forall(P(f(\lambda P)) \rightarrow P(f(f)) \land P),$$

and finally flatten it to:

$$\forall(P(\lambda P) \rightarrow P).$$

Table 3 shows that a wide variety of structures are found in UNIV while other data sets are dominated by a small number of structures. Table 4 presents some of less frequent sketches found in UNIV (DEV). In actuality, 67% of the unique sketches found in UNIV (DEV) occur only once in the dataset. These statistics suggest that the distribution of the logical structures found in UNIV, and math text in general, is very long-tailed.

## 4 A Type System for Pre-university Math

Our semantic language is a higher-order logic (lambda calculus) with parametric polymorphism. Table 5 presents the types in the language. The atomic types are defined so that they capture the selectional restriction of verbs and other argument-taking phrases as precisely as possible. For instance, an equation in real domain, e.g., $x^2 - 1 = 0$, can be regarded as a set of reals, i.e., $\{x \mid x^2 - 1 = 0\}$. However, we never say 'a solution of a set.' We thus discriminate an equation from a set in the type system even though the concept of equation is mathematically dispensable.

| | |
|---|---|
| truth values | `Bool` |
| numbers | `Z` (integers), `Q` (rationals), `R` (reals), `C` (complex) |
| polynomials | `Poly` |
| single variable functions | `R2R` ($\mathbb{R} \rightarrow \mathbb{R}$), `C2C` ($\mathbb{C} \rightarrow \mathbb{C}$) |
| single variable equations | `EqnR` (in $\mathbb{R}$), `EqnC` (in $\mathbb{C}$) |
| points in 2D/3D space | `2d.Point`, `3d.Point` |
| geometric objects | `2d.Shape`, `3d.Shape` |
| vectors and matrices | `2d.Vector`, `3d.Vector` |
| matrices | `2d.Matrix`, `3d.Matrix` |
| angles | `2d.Angle`, `3d.Angle` |
| number sequences | `Seq` |
| cardinals and ordinals | `Card`, `Ord` |
| ratios among numbers | `Ratio` |
| limit values of functions | `LimitVal` |
| integer division | `QuoRem` |
| polymorphic containers | `SetOf`($\alpha$), `ListOf`($\alpha$) |
| polymorphic tuples | `Pair`($\alpha, \beta$), `Triple`($\alpha, \beta, \gamma$) |

Table 5: Types defined in the semantic language

Entities of equation and set are built by constructor functions that take a higher-order term as the argument as in $\mathtt{eqn}(\lambda x.x^2 - 1)$ and $\mathtt{set}(\lambda x.x^2 - 1)$. Related concepts such as 'solution' and 'element' are defined by the axioms for corresponding function and predicate symbols:

$$\begin{gathered} \forall f \forall x(\mathtt{solution}(x, \mathtt{eqn}(f)) \leftrightarrow fx) \\ \forall s \forall x(\mathtt{element}(x, \mathtt{set}(s)) \leftrightarrow sx). \end{gathered}$$

Distinction of cardinal numbers (`Card`) and ordinal numbers (`Ord`), and the introduction of 'integer division' type (`QuoRem`) are also linguistically motivated. The former is necessary to capture the difference between, e.g., '$k$-th integer in $n_1, n_2, \ldots, n_m$' and '$k$ integers in $n_1, n_2, \ldots, n_m$.' An object of type `QuoRem` is conceptually a pair of integers that represent the quotient and the remainder of integer division. It is linguistically distinct from the type of `Pair(Z,Z)` because, e.g., in

> Select a pair of integers $(n, m)$ and divide $n$ by $m$. If the remainder (of $\phi$) is zero, ...

the null (i.e., omitted) pronoun $\phi$ has 'the result of division $n/m$' as its antecedent but not $(n, m)$.

Polymorphism is a mandatory part of the language. Especially, the semantics of plural noun

$$
>\dfrac{>\dfrac{\dfrac{\text{When}}{\begin{array}{c}S/(S\backslash NP)/Sa\\ :\lambda P.\lambda Q.\pi_2(P)\to Q(\pi_1(P))\end{array}}\quad\dfrac{\text{any } k \text{ in } K \text{ is divided by } m,}{\begin{array}{c}Sa\\ :(\texttt{quorem}(k,m),(\exists k;k\in K))\end{array}}}{S/(S\backslash NP):\lambda Q.(\exists k;k\in K)\to Q(\texttt{quorem}(k,m))}\quad>\dfrac{\dfrac{\text{the quotient}}{\begin{array}{c}\mathbf{T}\backslash NP/(\mathbf{T}\backslash NP)\\ :\lambda P.\lambda x.P(\texttt{quo\_of}(x))\end{array}}\quad\dfrac{\text{is 3.}}{\begin{array}{c}S\backslash NP\\ :\lambda x.(x=3)\end{array}}}{S\backslash NP:\lambda x.\texttt{quo\_of}(x)=3}}{S:(\exists k;k\in K)\to\texttt{quo\_of}(\texttt{quorem}(k,m))=3}
$$

Figure 3: Sketch of the derivation tree for a sentence including an action verb and quantification

phrases is expressed by polymorphic lists and tuples: e.g., 'the radii of the circles $C_1$, $C_2$, and $C_3$' is of type `ListOf(R)` and 'the function $f$ and its maximum value' is of type `Pair(R2R,R)`.

## 5 Lexicon and Grammar

### 5.1 Combinatory Categorial Grammar

An instance of CCG grammar consists of a *lexicon* and a small number of *combinatory rules*. A lexicon is a set of *lexical items*, each of which associates a word surface form with a syntactic category and a semantic function: e.g.,

*sum* :: NP/PP : $\lambda x.\texttt{sum\_of}(x)$

*intersects* :: S\NP/PP : $\lambda y.\lambda x.\texttt{intersect}(x,y)$

A syntactic category is one of atomic categories, such as NP, PP, and S, or a complex category in the form of $\mathbf{X}/\mathbf{Y}$ or $\mathbf{X}\backslash\mathbf{Y}$, where $\mathbf{X}$ and $\mathbf{Y}$ are syntactic categories.

The syntactic categories and the semantic functions of constituents are combined by applying combinatory rules. The most fundamental rules are forward ($>$) and backward ($<$) application:

$$
>\dfrac{\mathbf{X}/\mathbf{Y}:f\quad\mathbf{Y}:x}{\mathbf{X}:fx}\qquad<\dfrac{\mathbf{Y}:x\quad\mathbf{X}\backslash\mathbf{Y}:f}{\mathbf{X}:fx}
$$

The atomic categories are further classified by *features* such as `num`(ber) and `case` of noun phrases. In the current paper, the features are written as in NP[`num=pl`,`case=acc`].

### 5.2 A Japanese CCG Grammar and Lexicon

We developed a Japanese CCG following the analysis of basic constructions by Bekki (2010) but significantly extending it by covering various phenomena related to copula verbs, action verbs, argument-taking nouns, appositions and so forth. The semantic functions are defined in the format of a higher-order version of dynamic predicate logic (Eijck and Stokhof, 2006). The dynamic property is necessary to analyze semantic phenomena related to quantifications, such as donkey anaphora. In the following examples, we use English instead of Japanese and the standard notation of higher-order logic for the sake of readability.

We added two atomic categories, Sn and Sa, to the commonly used S, NP, and N. Category Sn is assigned to a proposition expressed as a math formula, such as '$x > 0$'. Semantically it is of type `Bool` but syntactically it behaves both like a noun phrase and a sentence.

Category Sa is assigned to a sentence where the main verb is an action verb such as *add* and *rotate*. Such a sentence introduces the result of the action as a discourse entity (i.e., what can be an antecedent of coreferential expressions). The action verbs can also mediate quantification as in:

When any $k \in K$ is divided by $m$, the quotient is 3.
$\forall k(k \in K \to \texttt{quo\_of}(\texttt{quorem}(k,m)) = 3)$

where $\texttt{quorem}(k,m)$ represents the result of the division (i.e., the pair of the quotient and the remainder) and `quo_of` is a function that extracts the quotient from it. To handle such phenomena, we posit the semantic type of Sa as `Pair(`$\alpha$`, Bool)` where the two components respectively bring the result of an action and the condition on it (including quantification). Fig. 3 presents a derivation tree for the above example.[4]

The atomic category NP, N, and Sa in our grammar have `type` feature. Its value is one of the types defined in the semantic language or a *type variable* when the entity type is underspecified. The lexical entry for '(an integer) *divides* (an integer)' and '(a set) *includes* (an element)' would thus have the following categories (other features than `type` are not shown):

*divides* :: S\NP[`type=Z`]/NP[`type=Z`]

*includes* :: S\NP[`type=SetOf(`$\alpha$`)`]/NP[`type=`$\alpha$]

When defining a lexical item, we don't have to explicitly specify the `type` features in most cases. They can be usually inferred from the definition of

[4] In Fig. 3, the semantic part is in the dynamic logic format as in our real grammar where the dynamic binding $(\exists x;\phi) \to \psi$ is interpreted as $\forall x(\phi \to \psi)$ in the standard predicate logic. Following our analysis of an analogous construction in Japanese, the null pronoun after 'the quotient' is filled by analysing the second clause as including a gap rather than filling it by zero-pronoun resolution.

the semantic function. In the above example, *divides* will have $\lambda y.\lambda x.(x|y)$ and *includes* will have $\lambda y.\lambda x.(y \in x)$ as their semantic functions. For both cases, the `type` feature of the NP arguments can be determined from the type definitions of the operators $|$ and $\in$ in the ontology.

The lexicon currently includes 54,902 lexical items for 8,316 distinct surface forms, in which 5,094 lexical items for 1,287 surface forms are for function words and functional multi-word expressions. The number of unique categories in the lexicon is 10,635. When the `type` features are ignored, there are still 4,026 distinct categories.

## 6 Math Expression Analysis

The meaning of a math expression is composed with the semantic functions of surrounding words to produce a logical form. We dynamically generate lexical items for each math expression in a problem. Consider the following sentence including two 'equations':

If $a^2{-}4{=}0$, then $x^2{+}ax{+}1{=}0$ has a real solution.

The latter, $x^2{+}ax{+}1 = 0$, should receive a lexical item of a noun phrase, NP : $\texttt{eqn}(\lambda \texttt{x}.\texttt{x}^2 + \texttt{a} + \texttt{1})$, but the former, $a^2{-}4 = 0$, should receive category S since it denotes a proposition. Such disambiguation is not always possible without semantic analysis of the text. We thus generate more than one lexical item for ambiguous expressions and let the semantic parser make a choice.

To generate the lexical items, we first collect appositions to the math expressions, such as 'integer $n$ and $m$' and 'equation $x^2 + a = 0$,' and use them as the type constraints on the variables and the compound expressions. Compound expressions are then parsed with an operator precedence parser (Aho et al., 2006). Overloaded operators, such as $+$ for numbers and vectors, are resolved using the type constrains whenever possible. Finally, we generate all possible interpretations of the expressions and select appropriate syntactic categories.

We have seen only three categories of math expressions: NP, Sn, and **T**/(**T**\NP). The last one is used for a NP with post-modification, as in:

$$> \dfrac{\dfrac{n > 0}{\mathbf{T}/(\mathbf{T}\backslash NP) \\ : \lambda P.(n > 0 \wedge P(n))} \qquad \dfrac{\text{is an even number}}{S\backslash NP \\ : \lambda x.(\texttt{even}(x))}}{S : n > 0 \wedge \texttt{even}(n)}$$

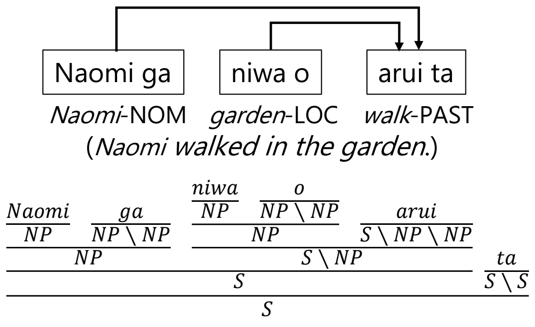

Figure 4: Bunsetsu dependency structure (top) and CCG derivation tree (bottom)

## 7 Two-step Semantic Parsing

Two central issues in parsing are the cost of the search and the accuracy of disambiguation. Supervised learning is commonly used to solve both. It is however very costly to create the training data by manually annotating a large number of sentences with CCG trees. Past studies have tried to bypass it by so-called weak supervision, where a parser is trained only with the logical form (e.g., Kwiatkowski et al. 2011) or even only with the answers to the queries (e.g., Liang et al. 2011).

Although the adaptation of such methods to the pre-university math data is an interesting future direction, we developed yet another approach based on a hybrid of shallow dependency parsing and the detailed CCG grammar. The syntactic structure of Japanese sentences has traditionally been analyzed based on the relations among word chunks called *bunsetsu*s. A bunsetsu consists of one or more content words followed by zero or more function words. The dependencies among bunsetsus mostly correspond to the predicate-argument and inter-clausal dependencies (Fig. 4). The dependency structure hence matches the overall structure of a CCG tree only leaving the details unspecified.

We derive a full CCG-tree by using a bunsetsu dependency tree as a constraint. We assume: (i) the fringe of each sub-tree in the dependency tree has a corresponding node in the CCG tree. We call such a node in the CCG tree 'a matching node.' We further assume: (ii) a matching node is combined with another CCG tree node whose span includes at least one word in the head bunsetsu of the matching node. Fig. 5 presents an example of a sentence consisting of four bunsetsus (rounded squares), each of which contains two words. In the figure, the $i$-th cell in the $k$-th row from the bottom is the CKY cell for the span from $i$-th to

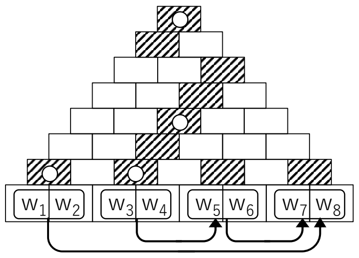

Figure 5: Restricted CKY parsing based on a shallow dependency structure

($i$+$k$-1)-th words. Under the two assumptions, we only need to fill the hatched cells given the dependency structure shown below the CKY chart. The hatched cells with a white circle indicate the positions of the matching nodes.

Even under the constraint of a dependency tree, it is impractical to do exhaustive search. We use beam search based on a simple score function on the chart items that combines several features such as the number of atomic categories in the item. We also use $N$-best dependency trees to circumvent the dependency errors. The restricted CKY parsing is repeated on the $N$-best dependency trees until a CCG tree is obtained. Our hope is to reject a dependency error as violation of the syntactic and semantic constraints encoded in the CCG lexicon. In the experiment, we used a Japanese dependency parser developed by Kudo and Matsumoto (2002). We modified it to produce $N$-best outputs and used up to 20-best trees per sentence.

## 8 Global Type Coherency

The well-typedness of the logical form is usually guaranteed by the combinatory rules. However, they do not always guarantee the type coherency among the interpretations of the math expressions.

For instance, consider the following derivation:

$$\dfrac{\dfrac{\text{if } x+y\in U,}{S/S : \lambda P.(\mathtt{add_R}(x,y)\in U \to P)} \quad \dfrac{\text{then } x+z\in V.}{S : \mathtt{add_V}(x,y)\in V}}{S : \mathtt{add_R}(x,y)\in U \to \mathtt{add_V}(x,z)\in V} >$$

The $+$ symbol is interpreted as the addition of real numbers ($\mathtt{add_R}$) in the first clause but that of vectors ($\mathtt{add_V}$) in the second one. The logical form is not typable because the two occurrences of $x$ must have different types. The forward application rule does not reject this derivation since the categories of the two clauses perfectly match the rule schema.

We can reject such inconsistency by doing type checking on the logical form at every step of the

**Algorithm 1** Global type coherence check

```
procedure PARSEPROBLEM
    Envs ← $\emptyset$; AllDerivs ← []
    for each sentence $s$ in the problem do
        Chart ← INITIALIZECKYCHART($s$, Envs)
        Derivs ← TWOSTEPPARSING($s$, Chart)
        Envs ← UPDATEENVIRONMENTS(Envs, Derivs)
        AllDerivs ← AllDerivs $\oplus$ [Derivs]
    return AllDerivs

// $s$: a sentence; Envs: a set of environments
procedure INITIALIZECKYCHART($s$, Envs)
    Chart ← empty CKY chart
    for each token $t$ in $s$ do
        for each lexical item $C : f$ for $t$ do
            // $C$: category, $f$: semantic function
            if $t$ is a math expression then
                for each environment $\Gamma \in$ Envs do
                    if $\Gamma$ is unifiable with FV($f$) then
                        add $(C, \Gamma \sqcup \mathrm{FV}(f))$ to Chart
            else   // $t$ is a normal word
                add $(C, \emptyset)$ to Chart
    return Chart

FV($f$): the environment that maps the free variables in a semantic function $f$ to their principal types determined by type inference on $f$.

// Envs: a set of environments; Derivs: derivations trees
procedure UPDATEENVIRONMENTS(Envs, Derivs)
    NewEnvs ← $\emptyset$ // environments for the next sentence
    for each derivation $d \in$ Derivs do
        $\Gamma$ ← the environment at the root of $d$
        if $\Gamma \neq \emptyset$ then   // update the environments
            NewEnvs ← NewEnvs $\cup \{\Gamma\}$
        else  // no update: there was no math expression
            NewEnvs ← NewEnvs $\cup$ Envs
    // eliminate those subsumed by other environments
    return MOSTGENERALENVIRONMENTS(NewEnvs)
```

derivation. It is however quite time consuming because we cannot use dynamic programming any more and need to do type checking on numerous chart items. Furthermore, such type inconsistency may happen *across* sentences. Instead, we consider the *type environment* while parsing. A type environment, written as $\{v_1 : T_1, v_2 : T_2, \dots\}$, is a finite function from variables to type expressions. A pair $v : T$ means that the variable $v$ must be of type $T$ or its instance (e.g., $\mathtt{SetOf(R)}$ is an instance of $\mathtt{SetOf}(\alpha)$). For example, the logical form of the first clause of the above sentence is typable under $\{x\!:\!\mathtt{R}, y\!:\!\mathtt{R}, z\!:\!\alpha, U\!:\!\mathtt{SetOf(R)}, V\!:\!\beta\}$, but that of the second clause isn't. Please refer, e.g., to (Pierce, 2002) for the formal definitions. Two environments $\Gamma_1$ and $\Gamma_2$ are *unifiable* iff there exists a substitution $\sigma$ that maps the type variables in $\Gamma_1$ and $\Gamma_2$ to some type expressions so that $\Gamma_1\sigma = \Gamma_2\sigma$ holds. We write $\Gamma_1 \sqcup \Gamma_2$ for the result of such substitution (i.e., unification) with the

$$\dfrac{\dfrac{\dfrac{n}{(NP[\alpha],\{n:\alpha\})} \;>\; \dfrac{\dfrac{\text{divides}}{(S\backslash NP[\mathtt{Z}]/NP[\mathtt{Z}],\emptyset)}\quad\dfrac{12}{(NP[\mathtt{Z}],\emptyset)}}{(S\backslash NP[\mathtt{Z}],\emptyset)}}{(S,\{n:\mathtt{Z}\})}< \;>\; \dfrac{\dfrac{\text{iff}}{(S\backslash S/Sn,\emptyset)}\quad\dfrac{n\in U}{(Sn,\{n:\beta,U:\mathtt{SetOf}(\beta)\})}}{(S\backslash S,\{n:\beta,U:\mathtt{SetOf}(\beta)\})}}{(S,\{n:\mathtt{Z},U:\mathtt{SetOf}(\mathtt{Z})\})}<$$

Figure 6: CCG parsing with type environment

| Dataset | Correct | Time-out | Wrong | No RCF | Parse failure |
|---|---|---|---|---|---|
| DEV | 27.6% | 10.9% | 12.1% | 12.1% | 37.4% |
| TEST | 11.4% | 1.8% | 11.4% | 6.8% | 68.6% |

(Correct: correct answer; Timeout: reasoning did not finish in 10 min; Wrong: wrong answer; No RCF: no RCF formula was obtained by rewriting the logical form; Parse failure: at least one sentence in the problem did not receive a CCG tree)

Table 6: Result of end-to-end problem solving

| Dataset | Dep. train | Parsed Sentences (%) | | | |
|---|---|---|---|---|---|
| | | N=1 | N=5 | N=10 | N=20 |
| DEV | News | 48.9 | 69.1 | 72.6 | 76.6 |
| | News+Math | 70.5 | 81.6 | 84.6 | 86.4 |
| TEST | News | 46.6 | 58.7 | 61.9 | 64.7 |
| | News+Math | 59.3 | 65.3 | 66.9 | 68.3 |

Table 7: Fraction of sentences on which a CCG tree was obtained in top $N$ dependency trees

most general $\sigma$ (most general unifier, mgu).

We associate a type environment with each chart item and refine it through parsing. The type constraints implied in a discourse are accumulated in the environment and block the generation of incoherent derivations (Algorithm 1). Fig. 6 presents an example of a parsing result, in which the type constraints implied in the two clauses are unified at the root and the type of $U$ is determined. When we apply a combinatory rule, we first check if the environments of the child chart items are unifiable. If so, we put the unified environment in the parent item and apply the unifier to the `type` features in the parent category. For instance, the forward application rule is revised as follows:

$$(X/Y, \Gamma_1) + (Y, \Gamma_2) \rightarrow (X\sigma, \Gamma_1 \sqcup \Gamma_2),$$

where $\sigma$ is the mgu of $\Gamma_1$ and $\Gamma_2$ and $X\sigma$ means the application of $\sigma$ to the `type` features in $X$.[5]

## 9 Experiments and Analysis

This section presents the overall performance of the current end-to-end system and demonstrates the effectiveness of the proposed parsing techniques. We also present an analysis of the failures.

Table 6 presents the result of end-to-end problem solving on the UNIV data. It shows the failure in the semantic parsing is a major bottleneck in the current system. Since a problem in UNIV includes more than three sentences on average, parsing a whole problem is quite a high bar for a semantic parser. It is however necessary to solve it by the nature of the task. Once a problem-level logical form was produced, the system yielded a correct solution for 44% of such problems in DEV and 36% in TEST.

Table 7 lists the fraction of the sentences on which the two-step parser produced a CCG tree within top-$N$ dependency trees. We compared the results obtained with the dependency parser trained only on a news corpus (News) (Kurohashi and Nagao, 2003), which is annotated with bunsetsu level dependencies, and that trained additionally with a math problem corpus consisting of 6,000 sentences[6] (News+Math). The math problem corpus was developed according to the same annotation guideline for the news corpus. The attachment accuracy of the dependency parser was 84% on math problem text when trained only on the news corpus but improved to 94% by the addition of the math problem corpus. The performance gain by increasing $N$ is more evident in the results with the News parser than that with the News+Math parser. It suggests the grammar properly rejected wrong dependency trees, which were ranked higher by the News parser. The effect of the additional training is very large at small $N$s and still significant at $N = 20$. It means that we successfully boosted both the speed and the success rate of CCG parsing only with the shallow dependency annotation on in-domain data.

[5] To be precise, we also consider the type constraints induced through the unification of the *categories*. It can be seen in the derivation step for "$n$ divides 12" in Fig. 6, where the new constraint $n$ :Z is induced by the unification of NP[$\alpha$] and NP[Z] and merged into the environment of the parent.

[6] No overlap with DEV and TEST sections of UNIV.

| Dataset | Parsing w/ type env. | Typing failure (%) | Correct answer (%) |
|---|---|---|---|
| DEV | no | 9.8% | 21.8% |
| | yes | 0.6% | 27.6% |
| TEST | no | 8.6% | 8.6% |
| | yes | 0.0% | 11.4% |

Table 8: Effect of parsing with type environment

| Freq. | Reason for the parse failures (on TEST-2007) |
|---|---|
| 17 | Unknown usage of known content words |
| 9 | Unknown content words |
| 8 | Errors in coreference resolution |
| 4 | Missing math expression interpretaions |
| 3 | Unknown usage of known function words |
| 3 | Unknown function words |
| 2 | No correct dependency tree in 20-best |

Table 9: Reasons for the parse failures

| | Dataset | Precision | Recall |
|---|---|---|---|
| sentence-level | DEV-1999 | 83% (64/77) | 72% (64/ 89) |
| | TEST-2007 | 88% (64/73) | 56% (64/114) |
| problem-level | DEV-1999 | 75% (18/24) | 45% (18/40) |
| | TEST-2007 | 50% (8/16) | 15% (8/53) |

Table 10: Accuracy of logical forms

| Error type | DEV-1999 | TEST-2007 |
|---|---|---|
| Bind a variable or leave it free | 6 | 2 |
| Wrong math expr. interpretaion | 6 | 1 |
| Quantifier choice | 0 | 3 |
| Quantifier scope | 1 | 1 |
| Logical connective choice | 1 | 1 |
| Logical connective scope | 1 | 0 |
| Others | 1 | 2 |

Table 11: Types of errors in the logical forms

Table 8 shows the effect of CCG parsing with type environments. The column headed 'Typing failure' is the fraction of the problems on which no logical form was obtained due to typing failure. Parsing with type environment eliminated almost all such failures and significantly improved the number of correct answers. The remaining type failure was due to beam thresholding where a necessary derivation fell out of the beam.

Table 9 lists the reasons for the parse failures on 1/4 of the TEST section (the problems taken from exams on 2007). In the table, "unknown usage" means a missing lexical item for a word already in the lexicon. "Unknown word" means no lexical item was defined for the word. Collecting unknown usages (especially that of a function word) is much harder than just compiling a list of words. Our experience in the lexicon development tells us that once we find a usage example, in the large majority of the cases, it is not difficult to write down its syntactic category and semantic function. Table 9 suggests that we can efficiently detect and collect unknown word usages through parsing failures on a large raw corpus of math problems.

Table 10 presents the accuracy of the sentence- and problem-level logical forms produced on the year 1999 subset of DEV and the year 2007 subset of TEST. Although the recall on the unseen test data is not as high as we hope, the high precision of the sentence-level logical forms is encouraging.

Table 11 provides the counts of the error types found in the wrong sentence-level logical forms produced on DEV-1999 and TEST-2007. It reveals the majority of the errors are related to the choice of quantifier ($\exists$, $\forall$, or free) and logical operators (e.g., $\rightarrow$ vs. $\leftrightarrow$) as well as the determination of their scopes. Meanwhile, we did not find an error related to the predicate-argument structure of a logical form. This fact and the results in Table 6 suggest that the selectional restrictions, encoded in the lexicon, properly rejected nonsensical predicate-argument relations. Our next step is to introduce a more sophisticated disambiguation model on top of the grammar, enjoying the properly confined search space.

## 10 Conclusion

We have explained why the task of end-to-end math problem solving matters for a practical theory of natural language semantics and introduced the semantic parsing of pre-university math problems as a novel benchmark. The statistics of the benchmark data revealed that it includes far more complex semantic structures than the other benchmarks. We also presented an overview of an end-to-end problem solving system and described two parsing techniques motivated by the scarcity of the annotated data and the need for the type coherency of the analysis. Experimental results demonstrated the effectiveness of the proposed techniques and showed the accuracy of the sentence-level logical form was 88% precision and 56% recall. Our future work includes the expansion of the lexicon with the aid of the semantic parser and the development of a disambiguation model for the binding and scoping structures.

## References

Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman. 2006. *Compilers: Principles, Techniques, and Tools (2nd Edition)*. Addison-Wesley Longman Publishing Co., Inc.

Daisuke Bekki. 2010. *Nihongo-bunpou no keishiki-riron (in Japanese)*. Kuroshio Shuppan.

Daniel Gureasko Bobrow. 1964. *Natural language input for a computer problem solving system*. Ph.D. thesis, Massachusetts Institute of Technology.

Eugene Charniak. 1969. Computer solution of calculus word problems. In *Proceedings of the 1st International Joint Conference on Artificial Intelligence*. San Francisco, CA, USA, pages 303–316. http://dl.acm.org/citation.cfm?id=1624562.1624593.

Donald Davidson. 1967. Truth and meaning. *Synthese* 17(1):304–323.

Jan Van Eijck and Martin Stokhof. 2006. The gamut of dynamic logic. In *Handbook of the History of Logic, Volume 6 Logic and the Modalities in the Twentieth Century*, Elsevier, pages 499–600.

Gottlob Frege. 1892. Über Sinn und Bedeutung. *Zeitschrift für Philosophie und philosophische Kritik* 100:25–50.

Mohammad Javad Hosseini, Hannaneh Hajishirzi, Oren Etzioni, and Nate Kushman. 2014. Learning to solve arithmetic word problems with verb categorization. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*. pages 523–533. http://aclweb.org/anthology/D/D14/D14-1058.pdf.

Rik Koncel-Kedziorski, Hannaneh Hajishirzi, Ashish Sabharwal, Oren Etzioni, and Siena Ang. 2015. Parsing algebraic word problems into equations. *Transactions of the Association for Computational Linguistics* 3:585–597. https://transacl.org/ojs/index.php/tacl/article/view/692.

Taku Kudo and Yuji Matsumoto. 2002. Japanese dependency analysis using cascaded chunking. In *CoNLL 2002: Proceedings of the 6th Conference on Natural Language Learning 2002 (COLING 2002 Post-Conference Workshops)*. pages 63–69. http://aclweb.org/anthology/W/W02/W02-2016.pdf.

Sadao Kurohashi and Makoto Nagao. 2003. *Building A Japanese Parsed Corpus*, Springer Netherlands, Dordrecht, pages 249–260.

Nate Kushman, Yoav Artzi, Luke Zettlemoyer, and Regina Barzilay. 2014. Learning to automatically solve algebra word problems. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*. pages 271–281. http://www.aclweb.org/anthology/P14-1026.

Tom Kwiatkowski, Luke Zettlemoyer, Sharon Goldwater, and Mark Steedman. 2010. Inducing probabilistic ccg grammars from logical form with higher-order unification. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*. pages 1223–1233. http://dl.acm.org/citation.cfm?id=1870658.1870777.

Tom Kwiatkowski, Luke Zettlemoyer, Sharon Goldwater, and Mark Steedman. 2011. Lexical generalization in ccg grammar induction for semantic parsing. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. pages 1512–1523. http://dl.acm.org/citation.cfm?id=2145432.2145593.

Percy Liang, Michael Jordan, and Dan Klein. 2011. Learning dependency-based compositional semantics. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. pages 590–599. http://www.aclweb.org/anthology/P11-1060.

Arindam Mitra and Chitta Baral. 2016. Learning to use formulas to solve simple arithmetic problems. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. pages 2144–2153. http://www.aclweb.org/anthology/P16-1202.

Richard Montague. 1970a. English as a formal language. In Bruno Visentini, editor, *Linguaggi nella Societa e nella Tecnica*, Edizioni di Communità, pages 189–224.

Richard Montague. 1970b. Universal grammar. *Theoria* 36(3):373–398. https://doi.org/10.1111/j.1755-2567.1970.tb00434.x.

Richard Montague. 1973. The proper treatment of quantification in ordinary english. In Patrick Suppes, Julius Moravcsik, and Jaakko Hintikka, editors, *Approaches to Natural Language*, Dordrecht, pages 221–242.

Benjamin C. Pierce. 2002. *Types and Programming Languages*. MIT Press.

Subhro Roy and Dan Roth. 2015. Solving general arithmetic word problems. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. pages 1743–1752. http://aclweb.org/anthology/D15-1202.

Bertrand Russell. 1905. On denoting. *Mind* 14(56):479–493. http://www.jstor.org/stable/2248381.

Minjoon Seo, Hannaneh Hajishirzi, Ali Farhadi, Oren Etzioni, and Clint Malcolm. 2015. Solving geometry problems: Combining text and diagram interpretation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. pages 1466–1476. http://aclweb.org/anthology/D15-1171.

Shuming Shi, Yuehui Wang, Chin-Yew Lin, Xiaojiang Liu, and Yong Rui. 2015. Automatically solving number word problems by semantic parsing and reasoning. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. pages 1132–1142. http://aclweb.org/anthology/D15-1135.

Mark Steedman. 2001. *The Syntactic Process*. Bradford Books. MIT Press.

Mark Steedman. 2012. *Taking Scope - The Natural Semantics of Quantifiers*. MIT Press. http://mitpress.mit.edu/books/taking-scope.

Lappoon R. Tang and Raymond J. Mooney. 2001. Using multiple clause constructors in inductive logic programming for semantic parsing. In *Proceedings of the 12th European Conference on Machine Learning*. pages 466–477. http://www.cs.utexas.edu/users/ai-lab/?tang:ecml01.

Alfred Tarski. 1936. The concept of truth in formalized languages. In A. Tarski, editor, *Logic, Semantics, Metamathematics*, Oxford University Press, pages 152–278.

Alfred Tarski. 1944. The semantic conception of truth: and the foundations of semantics. *Philosophy and Phenomenological Research* 4(3):341–376. http://www.jstor.org/stable/2102968.

Shyam Upadhyay, Ming-Wei Chang, Kai-Wei Chang, and Wen-tau Yih. 2016. Learning from explicit and implicit supervision jointly for algebra word problems. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. pages 297–306. https://aclweb.org/anthology/D16-1029.

Luke Zettlemoyer and Michael Collins. 2007. Online learning of relaxed ccg grammars for parsing to logical form. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*. pages 678–687. http://www.aclweb.org/anthology/D07-1071.

Luke S. Zettlemoyer and Michael Collins. 2005. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *Proceedings of the 21st Conference in Uncertainty in Artificial Intelligence*. pages 658–666.

Lipu Zhou, Shuaixiang Dai, and Liwei Chen. 2015. Learn to solve algebra word problems using quadratic programming. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. pages 817–822. http://aclweb.org/anthology/D15-1096.