The Complexity of Math Problems – Linguistic, or Computational?

Takuya Matsuzaki[1], **Hidenao Iwane**[2], **Hirokazu Anai**[2,3] **and Noriko Arai**[1]
[1] National Institute of Informatics, Japan
[2] Fujitsu Laboratories Ltd., Japan [3] Kyushu University, Japan
{takuya-matsuzaki,arai}@nii.ac.jp; {iwane,anai}@jp.fujitsu.com

Abstract

We present a simple, logic-based architecture for solving math problems written in natural language. A problem is firstly translated to a logical form. It is then rewritten into the input language of a solver algorithm and finally the solver finds an answer. Such a clean decomposition of the task however does not come for free. First, despite its formality, math text still exploits the flexibility of natural language to convey its complex logical content succinctly. We propose a mechanism to fill the gap between the simple form and the complex meaning while adhering to the principle of compositionality. Second, since the input to the solver is derived by strictly following the text, it may require far more computation than those derived by a human, and may go beyond the capability of the current solvers.

Empirical study on Japanese university entrance examination problems showed positive results indicating the viability of the approach, which opens up a way towards a true end-to-end problem solving system through the synthesis of the advances in linguistics, NLP, and computer math.

1 Introduction

Development of an NLP system usually starts by decomposing the task into several sub-tasks. Such a modular design is mandatory not only for the reusability of the component technologies and the extensibility of the system, but also for the sound and steady advancement of the research field. Each module, however, has to attack its sub-task in isolation from the entirety of the task, usually with a quite limited form and amount of knowledge. The separated sub-task is hence not necessarily easy even for human. This problem has been investigated in various directions, including the solutions to the error-cascading in pipeline models (Finkel et al., 2006; Roth and Yih, 2007, e.g.), the injection of knowledge into the processing modules (Koo et al., 2008; Pitler, 2012, e.g.), and the invention of a novel way of modularization (Bangalore and Joshi, 2010, e.g.).

In this paper, we present a simple pipeline architecture for natural language math problem solving, and investigate the issues regarding the separation of the semantic composition mechanism and the mathematical inference. Although the separation between these two may appear to be of different nature than the above-mentioned issues regarding the system modularization, as we will see later, the technical challenges there are also in the tension between the generality of an implemented theory as a reusable component, and its coverage over domain-specific phenomena.

In the system, a problem is analyzed with a Combinatory Categorial Grammar (Steedman, 2001) coupled with a semantic representation based on the Discourse Representation Theory (Kamp and Reyle, 1993) to derive a logical form. The logical form is then rewritten to the input language of a solver algorithm, such as specialized math algorithms and theorem provers. The solver finally finds an answer through inference.

Natural language problem solving in math and related domain is a classic AI task, which has served as a good test-bed for the integration of various AI technologies (Bobrow, 1964; Charniak, 1968; Gelb, 1971, e.g.). Besides its attraction as a pure intellectual challenge, it has direct applications to the natural language interface for the formal systems such as databases, theorem provers, and formal proof checkers. The necessity of the interaction between language understanding and backend solvers has been pointed out in some of the classic works and also in closely related works

International Joint Conference on Natural Language Processing, pages 73–81,
Nagoya, Japan, 14-18 October 2013.

terms $t ::= v \mid f(t_1, \ldots, t_k) \mid \Lambda v.t \mid \Lambda v.D$

conditions $C ::= P(t_1, \ldots, t_k) \mid \neg D \mid D_1 \rightarrow D_2$

DRSs $D ::= (\{v_1, \ldots, v_k\}, \{C_1, \ldots, C_m\})$

Figure 1: Syntax of DRS

such as Winograd's SHRDLU (1971). A clear separation of the two layers is, however, an essential property for a wide-coverage problem solving system since we can extend it in a modular fashion, by the enhancement of the solver or the addition of different types of solvers.

The research question in the current paper is thus summarized as follows:

1. Can we derive the logical form of the problems compositionally, with no intervention of mathematical inference, and how?

2. Can we solve such a direct translation of the text to a logical form with the current state-of-the-art automatic reasoning technology?

After a brief overview of the system pipeline (§3), we present a technique for capturing the dynamic properties of the syntax-semantics mapping in the math problem text, which, at first sight, seem to call for mathematical inference during the derivation of a logical form (§4). We then describe remaining issues we found so far in the semantic analysis of math problem text (§5). Finally, the viability of the approach is empirically evaluated on real math problems taken from university entrance examinations. In the evaluation, we apply a solver to the logical forms derived through manually annotated CCG derivations and DRSs on the problem text (§6). In the current paper, we thus exclusively focus on the formal aspect of the semantic analysis, setting aside the problem of its automation and disambiguation. The final section concludes the paper and gives future prospects including the automatic processing of the math text.

2 Preliminaries

2.1 Discourse Representation Structure

We use a variant of Discourse Representation Structure (DRS) (Kamp and Reyle, 1993) for the semantic representation. DRS has been developed for the formal analysis of various discourse phenomena, such as anaphora and quantifier scopes beyond a single sentence.

Fig. 1 shows the syntax of DRS used in this paper.[1] In the definitions, f and P respectively denote a function and a predicate symbol and v denotes a variable. The definition is slightly extended from that by van Eijck and Kamp (2011) for incorporating higher-order terms. A term of the form $\Lambda v.M$ denotes lambda abstraction in the object language, which is used to represent (mathematical) functions and sets[2]; we reserve λ for denoting the abstraction over DRSs (and terms) for the composition of DRSs. We define the interpretation of a DRS D indirectly through its translation D° to a (higher-order) predicate logic as in Fig. 2.

As defined in Fig. 2, a DRS $D = (\mathbf{V}, \mathbf{C})$ is basically interpreted as a conjunction of the conditions in $\mathbf{C}$ that is quantified existentially by all the variables in $\mathbf{V}$. However, as in the second clause in Fig. 2, the variables in the antecedent of an implication are universally quantified and their scopes also cover the succedent; this definition is utilized in the analysis of sentences including indefinite NPs, such as donkey sentences.

The mechanism of the DRS composition in this paper is based on the formulation by van Eijck and Kamp (2011). They use an operation called merge (denoted by $\bullet$) to combine two DRSs. Assuming no conflicts of variable names, it can be defined as: $(\mathbf{V}_1, \mathbf{C}_1) \bullet (\mathbf{V}_2, \mathbf{C}_2) := (\mathbf{V}_1 \cup \mathbf{V}_2, \mathbf{C}_1 \cup \mathbf{C}_2)$. Roughly speaking, this operation amounts to form the conjunction of the conditions in $\mathbf{C}_1$ and $\mathbf{C}_2$ allowing the conditions in $\mathbf{C}_2$ to refer to the variables in $\mathbf{V}_1$. Consider the following discourse:

s_1: A monkey^x is sleeping.
s_2: It$_x$ holds a banana.

Assuming the anaphoric relation indicated by the super/sub-scripts, we have their DRSs as follows:

$$D_1 = (\{x\}, \{\mathrm{monkey}(x), \mathrm{sleep}(x)\})$$
$$D_2 = (\{y\}, \{\mathrm{banana}(y), \mathrm{hold}(x, y)\})$$

By merging them, we have

$$D_1 \bullet D_2 = \left(\{x, y\}, \left\{ \begin{array}{l} \mathrm{monkey}(x), \mathrm{sleep}(x), \\ \mathrm{banana}(y), \mathrm{hold}(x, y) \end{array} \right\}\right),$$

which is translated to $\exists x.\exists y.(\mathrm{monkey}(x) \wedge \cdots \wedge \mathrm{hold}(x, y))$ as expected.

[1] Disjunction can be defined by using implication and negation: $D_1 \vee D_2 := (\{\}, \{\neg D_1\}) \rightarrow D_2$.

[2] We represent the application of a Λ-term to another term, such as $(\Lambda x.D)t$ and $(\Lambda x.t_1)t_2$, either by a special predicate $\mathrm{App}(f, x) \equiv fx$ or a function $\mathrm{app}(f, x) := fx$ according to the type of f. Compound terms of the form $t_1 t_2$ are hence not in the definitions.

$$\text{Assuming } D_1 = (\{v_1, \ldots, v_k\}, \{C_1, \ldots, C_m\}),$$
$$D_1^\circ := \exists v_1 \ldots \exists v_k.\, (C_1^\circ \wedge \cdots \wedge C_k^\circ)$$
$$(D_1 \to D_2)^\circ := \forall v_1 \ldots \forall v_k.\, ((C_1^\circ \wedge \cdots \wedge C_m^\circ) \to D_2^\circ)$$
$$(\neg D)^\circ := \neg D^\circ$$
$$(P(t_1, t_2, \ldots))^\circ := P(t_1^\circ, t_2^\circ, \ldots)$$
$$(f(t_1, t_2, \ldots))^\circ := f(t_1^\circ, t_2^\circ, \ldots)$$
$$(\Lambda v.D)^\circ := \Lambda v.(D^\circ)$$
$$(\Lambda v.t)^\circ := \Lambda v.(t^\circ)$$
$$v^\circ := v$$

Figure 2: Translation of DRS to HOPL

$$\dfrac{\dfrac{\text{When}}{\mathsf{S/S/S}}}{\mathsf{S/S} : \lambda P.\lambda Q.P \to Q} > \dfrac{\dfrac{\dfrac{\text{the centers of } C_1 \text{ and } C_2}{\mathsf{S/(S\backslash NP)}}}{: \lambda P.(\{x, x_1, x_2\}, \{x = [x_1, x_2], x_1 = \text{center\_of}(C_1), x_2 = \text{center\_of}(C_2)\}) \bullet Px} \quad \dfrac{\dfrac{\text{coincide}}{\mathsf{S\backslash NP}}}{: \lambda x.(\{\}, \{\text{coincide}(x)\})}}{\mathsf{S} : (\{x, x_1, x_2\}, \{x = [x_1, x_2], x_1 = \text{center\_of}(C_1), x_2 = \text{center\_of}(C_2), \text{coincide}(x)\})} >$$
$$\mathsf{S/S} : \lambda Q.(\{x, x_1, x_2\}, \{x = [x_1, x_2], x_1 = \text{center\_of}(C_1), x_2 = \text{center\_of}(C_2), \text{coincide}(x)\}) \to Q$$

Figure 3: A part of CCG derivation tree

$$> \frac{X/Y : f \quad Y : a}{X : fa} \qquad >\mathsf{B} \frac{X/Y : f \quad Y/Z : g}{X/Z : \lambda x.f(gx)}$$

Figure 4: Example of combinatory rules

2.2 Combinatory Categorial Grammar

Combinatory Categorial Grammar (CCG) (Steedman, 2001) is a lexicalized grammar formalism. In CCG, the association between a word w and its syntactic/semantic property is specified by a lexical entry of the form $w := C : S$, where C is the category of w and S is the semantic interpretation of w. A category is either a basic category (e.g., $\mathsf{S}, \mathsf{N}, \mathsf{NP}$) or a complex category of the form X/Y or $X\backslash Y$. For instance, we can assign the following categories and semantic interpretations to the region notation "$[0, +\infty)$" and a bare noun phrase "positive number":

$$[0, +\infty) := \mathsf{NP} : \Lambda x.(\{\}, \{x \geq 0\})$$
$$\text{positive number} := \mathsf{N} : \lambda x.(\{\}, \{x > 0\})$$

since the region notation behaves as a proper noun and it can be represented by its characteristic function, while "positive number" functions like a common noun (recall that Λ is for the abstraction in the object language and λ stands for the abstraction for the DRS composition). A handful of combinatory rules define how the categories and the semantic interpretations of constituents are combined to derive a larger phrase. Fig. 4 shows two of the rules. A part of a derivation tree for "When the centers of C_1 and C_2 coincide" is shown in Fig. 3. As shown in the figure, the semantic representation in DRS is composed by the beta reduction and the DRS merge operation. As we will see in §4, there are certain types of discourse for which the basic DRS composition machinery described so far does not suffice. We will return to this after a brief description of the whole system.

3 A Simple Pipeline for Natural Language Math Problem Solving

The main result in the current paper is a mechanism of semantic composition and an empirical support for our overall design choice. Although the NLP modules for the automatic processing and disambiguation are still under development, we show a brief overview of the whole system to give a clear image on the different representations of a problem at different stages of the pipeline.

From text to logical form The system receives a problem text with LaTeX-style markup on the symbolic mathematical expressions: e.g.,

> Let `$a>0$`, `$b≤0$`, and `$0<p<1$`. `$P(p, p^2)$` is on the graph of the function `$y=ax-bx^2$`. Write `$b$` in terms of `$a$` and `$p$`.

We process the mathematical expressions with a symbolic expression analyzer and produce their possible interpretations as lexical entries. For instance, `$y=ax-bx^2$` in the above example will receive at least two interpretations:

$$\texttt{\$y=ax-bx\^{}2\$} := \mathsf{S} : (\{\}, \{y = ax - bx^2\})$$
$$\texttt{\$y=ax-bx\^{}2\$} := \mathsf{NP} : \Lambda x.ax - bx^2.$$

The first lexical entry is for the usages such as "Hence $y = ax - bx^2$," where the expression denotes a proposition and behaves as a sentence. The second entry is for the usage as a noun phrase as in the example, which stands for a function.

We add such dynamically generated lexical entries to the lexicon and then analyze the sentences with a CCG parser. From the resulting CCG derivation trees, we will obtain a DRS for each sentence. For the above example, we will have the following DRSs (the third one is in the extended

language we'll introduce in the next section):

$$D_1 = (\{\}, \{a > 0, b \leq 0, 0 < p, p < 1\})$$
$$D_2 = (\{\}, \{P = (p, p^2), \mathrm{on}(P, \Lambda x.ax - bx^2)\})$$
$$D_3 = \mathrm{Find}(b')\left[cc; \exists^{-1}a; \exists^{-1}p; b = b'\right]$$

A discourse structure analyzer receives the DRSs and determines the logical relations among them while selecting an antecedent for each anaphoric expression. The net result of this stage is a large DRS that represents the whole problem. For the above example, we have their sequencing as the result: $D_1; D_2; D_3$. The sequencing operator (;) basically means conjunction (merge) of the DRSs, but it is also used to connect the meanings of a declarative sentence and an imperative sentence. The large DRS is then translated by a process defined in the next section, giving a HOPL formula enclosed by a directive to the solver:

$$\mathrm{Find}(b')\left[\begin{array}{l} a > 0 \wedge b \leq 0 \wedge 0 < p \wedge p < 1 \wedge \\ \exists P.\left(\begin{array}{l} P = (p, p^2) \wedge \\ \mathrm{on}(P, \Lambda x.ax - bx^2) \wedge\ b = b' \end{array}\right) \end{array}\right],$$

where $\mathrm{Find}(v)[\phi]$ is a directive to find the value of variable v that satisfies the condition ϕ.

From logical form to solver input Many of the current automatic reasoners operate on first-order formulas. To utilize them, we hence have to transform the HOPL formula in a directive to an equivalent first-order formula. Such transformation is of course not possible in general. However, we found that a greedy rewriting procedure suffices for that purpose on all of the high-school level math problems used in the experiment.

In the rewriting procedure, we iteratively apply several equivalence-preserving transformations including the beta-reduction of Λ-terms and rewriting of the predicates and functions using their definitions. For the above example, by using some trivial simplifications and the definition of $\mathrm{on}(\cdot, \cdot)$:

$$\forall x.\forall y.\forall f.\,(\mathrm{on}((x, y), f) \leftrightarrow (y = fx))\,,$$

we have the following directive holding a first-order formula:

$$\mathrm{Find}(b')\left[\begin{array}{l} a > 0 \wedge b \leq 0 \wedge 0 < p \wedge p < 1 \wedge \\ p^2 = ap - bp^2 \wedge b = b' \end{array}\right].$$

Solver Algorithms In addition to the generic first-order theorem provers, we can use specific algorithms as the solver when the formula is expressible in certain theories. Among them, many mathematical and engineering problems can be naturally translated to formulas consisting of polynomial equations, inequalities, quantifiers ($\forall, \exists$) and boolean operators ($\wedge, \vee, \neg, \rightarrow$, etc). Such formulas construct sentences in the first-order theory of real closed fields (RCF).

In his celebrated work, Tarski (1951) showed that RCF allows quantifier-elimination (QE): for any RCF formula $\phi(x_1, \ldots, x_n)$, there exists an equivalent quantifier-free formula $\psi(x_1, \ldots, x_n)$ in the same vocabulary. For example, the formula $\exists x.(x^2 + ax + b \leq c)$ can be reduced to a quantifier-free formula $a^2 - 4b + 4c \geq 0$ by QE.

Automated theorem proving is usually very costly. For example, QE for RCF is doubly exponential on the number of quantifier alternations in the input formula. The problems containing only six variables may be hard for today's computer with the best algorithm known. However, several positive results have been attained as the result of extensive search for practical algorithms during the last decades (see (Caviness and Johnson, 1998)). Efficient software systems of QE have been developed on several computer algebra systems, such as SyNRAC (Iwane et al., 2013).

4 Formal Analysis of Math Problem Text

In this section, we first summarize the most prominent issues we found so far in the linguistic analysis of high-school/college level math problems and then present a solution.

4.1 Problems

Context-dependent meanings of superlatives and their alike The meaning of a superlatives and semantically similar expressions such as "maximum" generally depends highly on the context. For example, the interpretation of "John was the tallest" depends on the group (of people) that is prominent in the discourse:

> There were ten boys. John was the tallest.

This context-dependency can be made more explicit by paraphrasing it to a comparative (Heim, 2000): "John was taller than *anyone else*," where "anyone else" refers, depending on the context, to the group against which John was compared.

In math text, however, we can usually determine the range of the "anyone else" without ambiguity:

> Assume $a + b = 3$. Find the maximum value of ab.

Here, the set of values that should be compared against the maximum value is, with no ambiguity, all the possible values of ab that is determined by the preceding context. Once we have a representation of such a set, it is easy to write the semantic interpretation of the phrase "maximum value of α." But, how can we obtain a representation of such a set without inference?

Discrimination between free/bound variable We can explicitly specify that a variable should be interpreted as being free, as in:

> Let R be a square with perimeter l.
> Write the area of R in terms of l.

This discourse may be translated to

$$\mathrm{Find}(a)\left[\exists R.\left(\begin{array}{l}\mathrm{is\_square(R)}\wedge\\ \mathrm{perimeter\_of}(R)=l\wedge\\ \mathrm{area\_of}(R)=a\end{array}\right)\right]$$

but not to

$$\mathrm{Find}(a)\left[\exists R.\exists l.\left(\begin{array}{l}\mathrm{is\_square(R)}\wedge\\ \mathrm{perimeter\_of}(R)=l\wedge\\ \mathrm{area\_of}(R)=a\end{array}\right)\right]$$

since, assuming the proper definitions of the functions and predicates, the first one is equivalent to $\mathrm{Find}(a)[a = l^2/16]$ but the second one is equivalent to $\mathrm{Find}(a)[a > 0]$. How can we specify a variable be *not* bound?

Imperatives Math problems usually include imperatives such as "Find/Write...," and "Prove/Show...". How can we derive correct interpretations of those imperatives, which depend on the semantic content of preceding declarative sentences, but are not a part of the declarative meaning of a discourse?

4.2 Solution by iDRS

Although the above-mentioned phenomena are quite common in math problem text, we found it is difficult to derive the meanings of such expressions within the basic compositional DRS framework introduced in §2. All of the examples above involve the manipulation and modification of the context in a discourse.

We present an extension of the DRS composition mechanism that covers expressions like the above examples. The basic idea is to introduce another layer of semantic representation called iDRS hereafter, which provides a device to manipulate the representation of the preceding context during the semantic composition.[3]

terms $t ::= v \mid f(t_1, \ldots, t_k) \mid \Lambda v.t \mid \Lambda v.I$

iDRS $I ::= P(t_1, \ldots, t_k) \mid \neg I \mid I_1 \rightarrow I_2 \mid$
$\exists v \mid I_1; I_2 \mid \exists^{-1} v \mid \mathrm{Find}(v)[I] \mid \mathrm{Show}[I] \mid cc$

Figure 5: Syntax of iDRS

First we define the syntax of iDRS as in Fig. 5. In the definition, the variables P, f, t, and v follows the same convention as in the DRS definition. In words, an iDRS represents either a DRS condition (the first row of the definition of I), a quantification $\exists v$, which corresponds to a DRS having only one variable, $(\{v\}, \{\})$, a sequencing $I_1; I_2$ of two iDRSs, or the new ingredients in the rest of the definition that will be explained shortly.

The "anti-quantifier" $\exists^{-1} v$ means an operation that cancels the quantification on v that precedes $\exists^{-1} v$. $\mathrm{Find}(v)[I]$ is a directive that requires to find the set of the values of variable v which satisfy the condition represented by I. Similarly, $\mathrm{Show}[I]$ is a directive that requires to prove the statement represented by I. Note that these two directives are not specific to any solvers; The choice of the solver depends on the theory (e.g., RCF) under which the formula in a directive is understood. The last element, cc, can be considered as a special 'variable', through which we can always retrieve an iDRS representation of the context that precedes the position marked by the cc.

Using these new ingredients, we can now write, for instance, the semantic representation of the phrase "maximum value" as follows:

$$\mathsf{N/NP_{of}} : \lambda x.\lambda m.\max(\Lambda y.(cc; y = x), m),$$

assuming that the two-place predicate $\max(s, m)$ is defined to be true iff m is the maximum element in the set s (represented by a Λ-term). A sentence "the maximum value of x is m" will thus have $\max(\Lambda y.(cc; y = x), m)$ as its semantic representation, which means that m is the maximum value of x that satisfies the condition specified by the preceding context.

[3]This approach shares much with a kind of dynamic semantics such as those by Bekki (2000) and Brasoveanu (2012), in which a representation of the context can also be accessed in the semantic language. An important difference is that in their approaches the context is represented as a set of assignment functions, while we represent them directly as an iDRS. This difference is crucial for our purpose since we eventually need to obtain a (first-order) formula on which an automatic reasoner operates.

$$
\begin{array}{rcl}
\{\!\{I_1; I_2\}\!\}_c & := & \{\!\{I_1\}\!\}_c; \{\!\{I_2\}\!\}_{c;[\![I_1]\!]_c} \\
\{\!\{\mathrm{Find}(v)[I]\}\!\}_c & := & \mathrm{Find}(v)\,[[\![I]\!]_c] \\
\{\!\{\mathrm{Show}[I]\}\!\}_c & := & \mathrm{Show}\,[[\![I]\!]_c] \\
\{\!\{I_1 \to I_2\}\!\}_c & := & \{\!\{I_2\}\!\}_{c;[\![I_1]\!]_c} \\
\{\!\{I\}\!\}_c & := & \epsilon
\end{array}
\qquad
\begin{array}{rcl}
[\![cc; I]\!]_c & := & c; [\![I]\!]_c \\
[\![cc]\!]_c & := & c \\
[\![I_1; I_2]\!]_c & := & [\![I_1]\!]_c; [\![I_2]\!]_{c;[\![I_1]\!]_c} \\
[\![I_1 \to I_2]\!]_c & := & [\![I_1]\!]_c \to [\![I_2]\!]_{c;[\![I_1]\!]_c} \\
[\![\neg I]\!]_c & := & \neg [\![I]\!]_c \\
[\![\mathrm{Find}(v)[I]]\!]_c & := & \exists v; [\![I]\!]_c \\
[\![\mathrm{Show}[I]]\!]_c & := & [\![I]\!]_c
\end{array}
\qquad
\begin{array}{rcl}
[\![P(t_1, \ldots)]\!]_c & := & P([\![t_1]\!]_c, \ldots) \\
[\![\exists v]\!]_c & := & \exists v \\
[\![\exists^{-1} v]\!]_c & := & \exists^{-1} v \\
[\![v]\!]_c & := & v \\
[\![f(t_1, \ldots)]\!]_c & := & f([\![t_1]\!]_c, \ldots) \\
[\![\Lambda v.t]\!]_c & := & \Lambda v.[\![t]\!]_c \\
[\![\Lambda v.I]\!]_c & := & \Lambda v.[\![I]\!]_c
\end{array}
$$

Figure 6: Transformation from iDRS to directive sequence

Let's take the following problem as an example:

> Let $p > 0$. R is a rectangle whose perimeter is p. Find the maximum value of the area of R as a function of p.

We have its iDRS representation shown below, by parsing the sentences and composing the resulting iDRSs into one (in this case, just by sequencing the three sentences' iDRSs):

$$
\left[\begin{array}{l}
0 < p; \\
\mathrm{is\_rectangle}(R); \mathrm{perimeter\_of}(R) = p; \\
\exists m; \max(\Lambda x.\,[cc; x = \mathrm{area\_of}(R)]\,, m); \\
\mathrm{Find}(a)[cc; \exists^{-1} p; a = m]
\end{array}\right]
$$

We then bind all free variables in the iDRS at their narrowest scopes:

$$
\left[\begin{array}{l}
\exists p; 0 < p; \\
\exists R; \mathrm{is\_rectangle}(R); \mathrm{perimeter\_of}(R) = p; \\
\exists m; \max(\Lambda x.\,[cc; x = \mathrm{area\_of}(R)]\,, m); \\
\mathrm{Find}(a)[cc; \exists^{-1} p; a = m]
\end{array}\right]
$$

This amounts to assume each variable appearing in a problem text is, unless it is explicitly quantified, interpreted to be existentially quantified as default, and to be universally quantified if it appears in the antecedent of an implication.

The iDRS is then processed by the functions $\{\!\{\cdot\}\!\}_c$ and $[\![\cdot]\!]_c$ defined in Fig. 6. In the definition, ϵ stands for an empty sequence. The function $\{\!\{\cdot\}\!\}_c$ extracts the imperative meaning from an iDRS, using $[\![\cdot]\!]_c$ as a 'sub-routine' that extracts the declarative meaning from an iDRS. The suffix (c) of the two functions stands for the preceding context represented as an iDRS. When $[\![\cdot]\!]_c$ processes a sequence $I_1; I_2$ or an implication $I_1 \to I_2$, the declarative content of I_1 (i.e., $[\![I_1]\!]_c$) is appended to the preceding context c, and $c; [\![I_1]\!]_c$ is passed as the preceding context when processing I_2. When $[\![\cdot]\!]_c$ finds a cc variable, it substitutes the cc with the current context stored in the suffix.

By applying $\{\!\{\cdot\}\!\}_\epsilon$ to the iDRS of a problem, we can extract the logical form of the problem as a sequence of directives. For the example problem, we have a single directive as follows:

$$
\mathrm{Find}(a)\left[\begin{array}{l}
\exists p; 0 < p; \\
\exists R; \mathrm{is\_rectangle}(R); \mathrm{perimeter\_of}(R) = p; \\
\exists m; \max(\Lambda x.\left[\begin{array}{l}\exists p; 0 < p; \\ \exists R; \mathrm{is\_rectangle}(R); \\ \mathrm{perimeter\_of}(R) = p; \\ \exists m; x = \mathrm{area\_of}(R)\end{array}\right], m); \\
\exists^{-1} p; a = m
\end{array}\right]
$$

Now, by the definition of $\{\!\{\cdot\}\!\}_c$ and $[\![\cdot]\!]_c$, the iDRS I inside a directive $\mathrm{Find}(v)[I]$ or $\mathrm{Show}[I]$ includes only those elements that have a counterpart in the basic DRS except for the "anti-quantifiers." We can hence convert it to a HOPL formula, by first canceling the quantifications $\exists v$ that precede $\exists^{-1} v$ (i.e., deleting all occurrences of $\exists v$ that appear before an occurrence of $\exists^{-1} v$ in the iDRS, and deleting $\exists^{-1} v$ itself), then converting it to a DRS by replacing the sequencing operator ';' to the merge operator, and finally translating it to a HOPL formula according to Fig. 2.

5 Remaining Issues in the Semantic Analysis of Math Problem Text

The mechanism presented in §4 significantly enhanced the coverage of the analysis over real problems. We however found several phenomena that can not be handled now.

Free/bound variable distinction without a cue phrase We have presented a mechanism to 'unbind' the variables specified by a cue phrase, such as "(find x) in terms of (y)." Some types of variables however have to be left free even without any explicit indication, e.g.:

> Let $p > 0$. Find the area of a circle with radius p, centered at the origin.

Assuming $\mathrm{circle}(x, y, r)$ denotes a circle with radius r and centered at (x, y), we want to derive

$$
\mathrm{Find}(a)\,[p > 0; a = \mathrm{area\_of}(\mathrm{circle}(0, 0, p))]\,,
$$

but our default variable binding rule gives

$$\mathrm{Find}(a)\,[\exists p; p > 0; a = \mathrm{area\_of}(\mathrm{circle}(0, 0, p))]\,.$$

This directive means to find the range of the areas of the circles with arbitrary radii, which is apparently not a possible reading of the problem. We found such cases in 3 out of the 32 test problems used in the experiment shown later.

Scope inversion by a cue phrase The hierarchy of the quantifier scopes in math text mostly follows the linear order of the appearance of the variables (either overtly quantified or not). This general rule can however be superseded by the effect of a cue phrase, as shown in the example problem and its possible translation in Fig. 7. In the figure, the formula inside the Show-directive mostly follows the discourse structure, in that the predicates from the first and the second sentence respectively form the antecedent and the succedent of the implication. The quantification on F is however dislocated from its default scope, i.e., the succedent, and moved to the outset of the formula by the effect of the underlined cue phrases. To handle such cases correctly, we would need a more involved mechanism for the manipulation of the context representation through the cc variable.

Idiomatic expressions As in other text genres, idiomatic multiword expressions are also problematic as can be seen in the following example:

> By choosing x sufficiently large, $y = 1/x$ can be made as close to 0 as desired.

As the example shows, a set phrase involving complex syntactic relations, e.g., "can do X as Y as desired by choosing Z sufficiently W" and "X approaches Y as Z approaches W," can convey idiomatic meanings in math.

6 Empirical Results

We tested the feasibility of our approach on a set of problems selected from Japanese university entrance exams. Specifically, we wanted 1) to test the coverage of the semantic composition mechanism presented in §4 on real problems, and 2) to verify that there is no significant loss in the capability of the system due to the additional computational cost incurred by the separation of the semantic analysis from the mathematical reasoning.

The second point was confirmed by providing the ideal (100% correct) output from the (forthcoming) NLP components to a state-of-the-art automatic reasoner and comparing the result against the performance of the reasoner on the input formulated by a human expert. Specifically, we manually gave the semantic representations of the problems as iDRSs or CCG derivation trees, and then automatically rewrote them into the language of RCF. The resulting formulas were fed to a solver to see whether the answers be returned in a realistic amount of time (30 seconds). The solver was implemented on SyNRAC (Iwane et al., 2013), which is an RCF-QE solver implemented as an add-on to Maple, and the (in)equation solving commands of Maple.

The problems were taken from the entrance exams of five first-tier universities in Japan (Tokyo U., Kyoto U., Osaka U., Kyushu U., and Hokkaido U.) for fiscal year 2001, 2003, 2005, 2007, 2009 and 2011. There were 249 problems in total. From them, we first eliminated those that included almost no natural language text, such like calculation problems. We then chose, from the remaining non-straightforward word problems, all the problems which could be solved with SyNRAC and Maple when the input was formulated by an expert of computer algebra. The formulation by an expert was done, of course, with no manual calculation, but otherwise it was freely done including the division of the solving process into several steps of QE and (in)equation solving.

As the result of that, we got 32 test problems, each of which contained 3.9 sentences on average. They include problems on algebra (of real and complex numbers), 3D and 2D geometry, calculus, and their combinations. For analyzing the result in more detail, we divided the problems into 78 sub-problems for which the correctness of the answers can be judged independently.

6.1 From discourse analysis to the solution

For the first experiment, we manually encoded the problems in the form of iDRSs. Each sentence in a problem was first encoded as a single iDRS, and the sentence-level iDRSs were combined (again manually) into a problem-level iDRS using the connectives defined in the iDRS syntax. In the manual encoding, the granularity of the representation, i.e., the smallest units of the semantic representation, was kept at the level of the actual words in the text whenever possible, intending that the resulting iDRSs closely match the representation

Problem: Point P is on the circle $x^2 + y^2 = 4$ and l_P is the normal line to the circle at P. Show that l_P passes through a fixed point F irrespective of P.

$$\text{Show } \left[\exists F.\ \left(\forall P.\forall l_P.\ \left(\left(\begin{array}{l} P \text{ is on } x^2+y^2=4 \text{ and} \\ l_P \text{ is the normal line to the circle at } P \end{array}\right) \rightarrow l_P \text{ passes through } F\right)\right)\right]$$

Figure 7: Scope inversion by cue phrases

Let $O(0,0)$, $A(2,6)$, $B(3,4)$ be 3 points on the coordinate plane. Draw the perpendicular to line AB through O, which meets AB at C. Let s, t be real numbers, and let P be such that $OP = s\overrightarrow{OA} + t\overrightarrow{OB}$. Answer the following questions.

(1) Calculate the coordinates of point C, and write $|\overrightarrow{CP}|^2$ in terms of s and t.

(2) Let s be constant, and let t vary in the range $t \geq 0$. Calculate the minimum of $|\overrightarrow{CP}|^2$.

Figure 8: Kyushu University 2009 (Science Course) Problem 1

composed from word-level semantic representations. In the iDRS encoding of the 32 problems, the context-fetching mechanism through 'cc' variable was needed in 15 problems and the canceling of quantification was needed in 6 problems. These mechanisms thus significantly enhanced the coverage of the semantic composition machinery.

After rewriting the iDRSs to RCF formulas[4], we fed them to the solver and got perfect answers for 19 out of the 32 problems. Out of the 78 sub-problems, 56 sub-problems (72%) were successfully solved. 12% of the sub-problems (9 sub-problems) failed due to the timeout in the QE solver. Besides the timeout, a major cause of the failures (7 sub-problems) was the fractional power (mainly square root) in the formula. Although we can mechanically erase the fractional powers to get an RCF formula, it was not implemented in the solver.[5] The remaining 6 sub-problems needed the free/bound variable distinction without any cue phrase (§5). Although half of them could be solved by manually specifying the free variables, we did not count them as solved here.

6.2 From syntactic analysis to the answer

We chose 14 problems from the 19 problems which were fully solved with the iDRS encodings. We manually analyzed the text following the CCG-based analyses of basic Japanese constructions given by Bekki (2010). We annotated the 44 sentences in the 14 problems with full CCG derivation trees and anaphoric links. We selected the 14 problems so that they cover different types of grammatical phenomena as much as possible. The final CCG lexicon contained 240 lexical entries (109 for function words and the rest for content words). The iDRS representations were then derived by (automatically) composing the semantic representations of the words according to the derivation trees and combining the sentence-level iDRSs to a problem-level iDRS as in the first experiment. Out of the 14 problems, we got fully correct answers for 13 problems. In the 14 problems, there were 33 sub-problems and we got correct answers for 32 of them; On only one sub-problem, the solver could not return an answer within the time limit. Fig. 8 shows an English translation of one of the 13 problems successfully solved with the CCG derivation trees as the input.

Overall, the results on the real exam problems were very promising: 72% of the sub-problems were successfully solved with the formula derived from a sentence-by-sentence, direct encoding of the problem. The experiment with manually annotated CCG derivation trees further showed that there was almost no additional cost introduced by the mechanical derivation of the logical forms from the word-level semantic representations.

7 Conclusion and Prospects

We have presented a logic-based architecture for automatic problem solving. The experiments on the university entrance exams showed positive results indicating the viability of the modular design.

Future work includes the development of the processing modules, i.e., the symbolic expression analyzer, the parser, and the discourse structure analyzer. Another future work is to incorporate different types of solvers to the system for covering a wider range of problems, with the ability to choose a solver based on the content of a problem.

[4]The knowledge-base used to rewrite the HOPL formulas to first-order RCF formulas included 230 axioms for 86 predicates and 98 functions.

[5]In the formulation by the human expert, the use of square roots were avoided by encoding the conditions differently (e.g., $x \geq 0 \wedge x^2 = 2$ instead of $\sqrt{x} = 2$).

References

Srinivas Bangalore and Aravind K. Joshi. 2010. *Supertagging: Using Complex Lexical Descriptions in Natural Language Processing*. Bradford Books. MIT Press.

Daisuke Bekki. 2000. *Typed Dynamic Logic for Compositional Grammar*. Ph.D. thesis, University of Tokyo.

Daisuke Bekki. 2010. *Formal Theory of Japanese Syntax*. Kuroshio Shuppan. (In Japanese).

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

Adrian Brasoveanu. 2012. The grammar of quantification and the fine structure of interpretation contexts. *Synthese*, pages 1–51.

Bob F. Caviness and Jeremy R. Johnson, editors. 1998. *Quantifier Elimination and Cylindrical Algebraic Decomposition*. Springer-Verlag, New York.

Eugene Charniak. 1968. Carps: a program which solves calculus word problems. Technical report, Massachusetts Institute of Technology.

Jenny Rose Finkel, Christopher D. Manning, and Andrew Y. Ng. 2006. Solving the problem of cascading errors: approximate bayesian inference for linguistic annotation pipelines. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, EMNLP '06, pages 618–626, Stroudsburg, PA, USA. Association for Computational Linguistics.

Jack P. Gelb. 1971. Experiments with a natural language problem-solving system. In *Proceedings of the 2nd international joint conference on Artificial intelligence*, IJCAI'71, pages 455–462, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Irene Heim. 2000. Degree operators and scope. In *Proceedings of Semantics and Linguistic Theory 10*, pages 40–64. CLC Publications.

Hidenao Iwane, Hitoshi Yanami, Hirokazu Anai, and Kazuhiro Yokoyama. 2013. An effective implementation of symbolic-numeric cylindrical algebraic decomposition for quantifier elimination. *Theoretical Computer Science*. (in press).

Hans Kamp and Uwe Reyle. 1993. *From Discourse to Logic: Introduction to Modeltheoretic Semantics of Natural Language, Formal Logic and Discourse Representation Theory*. Studies in Linguistics and Philosophy. Kluwer Academic.

Terry Koo, Xavier Carreras, and Michael Collins. 2008. Simple semi-supervised dependency parsing. In *Proceedings of ACL-08: HLT*, pages 595–603, Columbus, Ohio, June. Association for Computational Linguistics.

Emily Pitler. 2012. Attacking parsing bottlenecks with unlabeled data and relevant factorizations. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 768–776, Jeju Island, Korea, July. Association for Computational Linguistics.

Dan Roth and Wen-tau Yih. 2007. Global inference for entity and relation identification via a linear programming formulation. In Lise Getoor and Ben Taskar, editors, *Introduction to Statistical Relational Learning*. MIT Press.

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

Alfred Tarski. 1951. *A Decision Method for Elementary Algebra and Geometry*. University of California Press, Berkeley.

Jan van Eijck and Hans Kamp. 2011. Discourse representation in context. In Johan van Benthem and Alice ter Meulen, editors, *Handbook of Logic and Language, Second Edition*, pages 181–252. Elsevier.

Terry Winograd. 1971. Procedures as a representation for data in a computer program for understanding natural language. Technical report, Massachusetts Institute of Technology, Feb. MIT AI Technical Report 235.