Symbolic computation and machine learning on logical formulas

Munehiro Kobayashi

February 2017

Symbolic computation and machine learning on logical formulas

Munehiro Kobayashi

Doctoral Program in Mathematics

Submitted to the Graduate School of
Pure and Applied Sciences
in Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy in
Science

at the
University of Tsukuba

Preface

In this thesis, the contents of the papers [19, 24] as well as some basic notions relating these studies are presented.

These studies are intended to contribute to computer algebra. Computer algebra is a scientific field in computational mathematics which deals with tree structures that represent the mathematical objects. These tree structures are constructed by applying function symbols (*e.g.* $+$, $-$, sin, *etc.*) and logical symbols (*e.g.* $=$, $\wedge$, $\vee$, $\neg$, $\forall$, $\exists$, *etc.*) inductively to constant symbols and rational numbers. The computations to evaluate the equalities, to find more simple forms, and to rewrite into the normal forms of the expressed mathematical objects are the main problems in computer algebra. The studies in computer algebra result in, for example, algorithms to make the factorization or to calculate the greatest common divisor (GCD) of polynomials with rational coefficients.

Beyond the polynomials, we handle logical formulas in computer algebra. One of the most important examples of such computing on formulas is algorithms to perform quantifier elimination (QE) for real closed fields (RCF), the first one of which was stated by Tarski [30]. The manipulation of logical formulas is a strong tool to cope with complex problems. Our focus is on symbolic computation on logical formulas throughout the series of studies presented in the thesis.

Part I is devoted to describe the elementary definitions and ideas used in the studies. In chapter 1, compactness theorem, which is the most basic theorem in model theory, and a model theoretic proof of QE for RCF is explained. In chapter 2, a short tutorial on support vector machines is given.

The research contributions are stated in part II. In chapter 3 and 4, we present the results from the activities in "Todai robot project – Can a Robot Get Into the University of Tokyo?" [1, 22]. It is a project to develop an artificial intelligence system to pass the entrance exam for the University of Tokyo by 2021, which was launched in 2011 by the National Institute of Informatics, Japan. The math solver of the project uses the techniques in computer algebra such as RCF-QE and rewriting of the higher-order logic (HOL) formulas to solve the math problems. In chapter 3, the attempt to achieve higher efficiency for QE problems for non-prenex formulas by modifying the order of processing subformulas is described [19]. We also investigate the possibility of automatizing manual design for effective heuristics that requires trials and errors using the machine learning techniques. In chapter 4, the relation between the computational complexity to process the HOL formulas and the difficulty levels that the corresponding math problems expected to have is studied [24].

Acknowledgements

I would like to thank my advisor Dr. Akira Terui for the acceptance of my unexpected transfer and the continuous support since then. I am also grateful to Prof. Akito Tsuboi, who is the advisor of my master's degree, for providing a precious opportunity to gain knowledge in mathematical logic. Besides my advisors, I would like to thank Prof. Noriko H. Arai for the permission for me to act as a member of Tofai robot project as well as the financial support for my activity, and to Prof. Hirokazu Anai for sparing his time to make the research plan and put the valuable comments on my paper.

Dr. Kota Takeuchi, Dr. Hidenao Iwane, and Dr. Takuya Matsuzaki have helped me in a great extent to develop my skill and knowledge in mathematical logic, computer algebra, and sciences beyond mathematics. My sincere appreciation goes to them, as I am confident that these experiences remain still helpful in the future,

For taking care of all paperworks related to the university, I am indebted to the adept assistance of the administrative office of the Math Department of the University of Tsukuba. I likewise thank the generous office of Prof. Arai for providing assistance in accomplishing many administrative papers related to the Todai robot project.

Finally, I express my great appreciation to my fellow graduate students for making the duration of my doctoral studies fun and being there during the stressful times. I extend my utmost gratitude to my family and friends for their constant love and support. Without the help of everyone listed here, the completion of my doctoral studies would not be possible. For this and many more, I again humbly express my gratitude.

Contents

Part I

Preliminaries

Chapter 1

First order logic

In this chapter, we see a model theoretic proof of quantifier elimination (QE) for real closed fields (RCF). The first proof to this fact was given by Tarski [30], although here we employ another method presented in [31, 21].

We introduce the following notations: ω represents the set of all natural numbers including zero; a symbol with an overbar means a tuple of finite length, *e.g.* $\bar{a} = (a_1, \dots, a_n)$ for some $n \in \omega$; $f(\bar{a})$ stands for the tuple $(f(a_1), \dots, f(a_n))$ if f is a function of arity one; $a_{<\omega}$ denotes a countably infinite sequence $(a_0, a_1, \dots)$.

1.1 Compactness theorem

We start with a proof of compactness theorem, which is one of the most elementary theorem in model theory. First, we define the sets of terms and formulas which are subsets of the set of all finite sequences of the symbols in the language, the variables, and the logical symbols: $=$, $\wedge$, $\neg$, and $\exists$. A language L is a set of constant symbols, function symbols and predicate symbols. Every function symbol and predicate symbol in L is assigned an arity. Let us fix a set of variables here. We inductively define the set of terms as a subset of all finite sequences of the symbols in L and the variables.

Definition 1.1.1 (Terms)**.** For a finite sequence t of symbols in L and variables, we say t is a term if t satisfies either of the following three conditions:

- t is a variable.
- t is a constant symbol in L.
- There are L-terms $t_1, \dots, t_m$ and an m-ary function symbol F in L such that t has the form of $F(t_1, \dots, t_m)$.

We write $t(x_1, ..., x_n)$ for a term t if the variables that appear in t are contained in the variables $x_1, ..., x_n$. We say that a term t is closed if t does not have variables at all.

Similarly, we can construct the L-formulas from L, the variables and the logical symbols $\wedge$, $\vee$, $\neg$, $\rightarrow$, $\forall$, and $\exists$.

Definition 1.1.2 (L-formulas)**.** For a finite sequence φ of the symbols from L, the variables and $=$, we say φ is an L-atomic formula if φ satisfies either of the following conditions:

- There are L-terms t_1, t_2 such that φ is $t_1 = t_2$.
- There are L-terms $t_1, \dots, t_m$ and an m-ary predicate symbol P in L such that φ has the form of $P(t_1, \dots, t_m)$.

For a finite sequence φ that consists of the symbols from L, the variables, $=$, $\wedge$, $\neg$, and $\exists$, we say φ is an L-formula if φ satisfies either of the following conditions:

· φ is an L-atomic formula.

· There is an L-formula ψ such that φ is $\neg\psi$.

· There are L-formulas ψ_1, ψ_2 such that φ is $\psi_1 \wedge \psi_2$.

· There are an L-formula ψ and a variable x such that φ is $\exists x\, \psi$.

Remark 1.1.3. For simplicity, we note that the finite sequences of the symbols $\varphi \vee \psi$, $\varphi \to \psi$, and $\forall x\, \varphi$ are shorthands for $\neg(\neg\varphi \wedge \neg\psi)$, $\neg\varphi \vee \psi$, and $\neg\exists x\, \neg\varphi$, respectively. Thus, if φ, ψ are L-formulas and x is a variable, then $\varphi \vee \psi$, $\varphi \to \psi$, and $\forall x\, \varphi$ are L-formulas.

A variable x that appears in φ is called a bounded variable if it is within the scope of a quantifier $\forall x$ or $\exists x$. For example, in the formula $\forall x\, (\neg P(x, y))$, x is a bounded variable, while y is not. Such y is called a free variable. If the free variables in φ are contained in $x_1, ..., x_n$, we write $\varphi(x_1, \ldots, x_n)$ to denote φ. A formula without free variables is called a sentence.

L-terms formally represent 'objects' and L-formulas formally represent 'statements' expressed in a language L, while our language L is just a set of symbols and each symbol in L does not have a particular meaning. With a structure we can think of the meanings of L.

Definition 1.1.4. An L-structure is a tuple $(M, (X^M)_{X \in L})$ satisfying the following condition:

(1) M is a non-empty set.

(2) For each symbol $X \in L$,

(a) if $X = c$ (a constant symbol), then c^M is an element in M,

(b) if $X = F$ (an m-ary function symbol), then F^M is an m-ary function from M^m to M, and

(c) if $X = P$ (an n-ary predicate symbol), then P^M is a subset of M^n.

We can augment a language using a subset of a structure and expand the structure naturally.

Definition 1.1.5. Let M be an L-structure and A be a subset of M. By preparing a new constant symbol c_a for each element $a \in A$, we define the language $L_A = L \cup \{\, c_a \mid a \in A \,\}$. We let $(c_a)^M = a$ and to expand the L-structure M to the L_A-structure.

The following definition gives a semantics of a term and formula.

Definition 1.1.6. For an L-structure M and a closed L-term t, we define the interpretation t^M of t in M by induction on the construction of t as follows:

· If t is c for some constant symbol c, then $t^M = c^M$.

· If $t_1, \ldots, t_m$ are closed L-terms, F is an m-ary function symbol and t is $F(t_1, \ldots, t_m)$, then $t^M = F^M(t_1^M, \ldots, t_m^M)$.

Definition 1.1.7. Let M be an L-structure, and φ be an L-sentence. We define the notion that φ holds in M, or $M \models \varphi$ by induction on the construction of formulas:

· If φ is of the form $s = t$ where s and t are closed L-terms, then

$$M \models \varphi \text{ if } s^M = t^M.$$

· If φ is of the form $P(t_1, \ldots, t_n)$ where $t_1, \ldots, t_n$ are closed L-terms and P is an n-ary predicate symbol, then

$$M \models \varphi \text{ if } (t_1^M, \ldots, t_n^M) \in P^M.$$

· If φ is of the form $\varphi_1 \wedge \varphi_2$ where φ_1, φ_2 are L-sentences, then

$$M \models \varphi \text{ if } M \models \varphi_1 \text{ and } M \models \varphi_2.$$

· If φ is of the form $\neg\psi$ where ψ is an L-sentences, then

$$M \models \varphi \text{ if it is not the case } M \models \psi.$$

· If φ is of the form $\exists x\, \psi(x)$ where ψ is an L-sentences, then

$$M \models \varphi \text{ if there exists } a \in M \text{ such that } M \models \psi(a).$$

Remark 1.1.8. By the above definition, the interpretation of an L-formula $\varphi(x_1, \dots, x_n)$ with free variables can be introduced if the variables are substituted with some elements in M. Let $a_1, \dots, a_n$ be elements in M. By expanding the L-structure M to the L_M-structure, we have the L_M-sentence $\varphi(c_{a_1}, \dots, c_{a_n})$. The definition gives the interpretation of the L_M-sentence $\varphi(c_{a_1}, \dots, c_{a_n})$. Thus we write $M \models \varphi(a_1, \dots, a_n)$ for $\varphi(c_{a_1}, \dots, c_{a_n})$. If $M \models \varphi(\bar{a})$, we also say that $\varphi(\bar{x})$ is satisfied by $\bar{a}$ in M.

Definition 1.1.9. (1) A theory in a language L, or an L-theory in short, is a set of L-sentences.

(2) We say that an L-structure M is a model of an L-theory T if we have $M \models \varphi$ for any $\varphi \in T$.

(3) Let T be a L-theory and φ is L-sentence. We write $T \models \varphi$ if $M \models \varphi$ for any model M of T.

(4) A theory is consistent if it has a model.

(5) A consistent theory T in a language L is complete if for any L-sentence φ, $T \cup \{\varphi\}$ or $T \cup \{\neg\varphi\}$ is inconsistent.

(6) Let M be an L-structure. $\mathrm{Th}_L(M)$ is the set of L-sentences that are satisfied in M. $\mathrm{Th}_L(M)$ is called the theory of M. In particular, we call the set of L_M-sentences $\mathrm{Th}_{L_M}(M)$ the elementary diagram of M where L_M is the language extending L by adding new constants $\{\, c_m \mid m \in M \,\}$ and expanding the structure M with $c_m^M = m$.

Theorem 1.1.10 (Compactness Theorem)**.** *Let T be an L-theory. Then the following are equivalent:*

(1) *T has a model.*

(2) *Any finite subset of T has a model.*

Proof. We show (2) implies (1). The other direction is trivial. For simplicity, we assume that L is countable. Let $c_{<\omega}$ be a set of new constant symbols, *i.e.* $c_{<\omega} \cap L = \emptyset$, and let $L^* = L \cup c_{<\omega}$. Let $(\varphi_i(x))_{i\in\omega}$ be an enumeration of all L^*-formulas of which free variable is only x. We can assume the new constant symbols that appear in $(\varphi_i(x))_{i<n}$ are contained in $\{\, c_0, \dots, c_{n-1} \,\}$ for all $n \in \omega$.

Claim A. *Let the L^*-theory $T' = T \cup \{\, \exists x\, \varphi_i(x) \to \varphi_i(c_i) \mid i \in \omega \,\}$. Any finite subset of T' has a model.*

Proof of Claim A. Take any subset S of T' and suppose that

$$S = \{\, \psi_1, \dots, \psi_n \,\} \cup \{\, \exists x\, \varphi_i(x) \to \varphi_i(c_i) \mid i < m \,\},$$

where $\psi_i \in T$. We show S has a model. By the assumption on T, $\{\, \psi_1, \dots, \psi_n \,\}$ has a model N. Let us assign the interpretation of L^* in N. For $i = 0, \dots, m-1$, if $\varphi_i(x)$ has a solution in N, let the interpretation of c_i be the solution. This is possible because c_i does not appear in $\varphi_j(x)$ for $j = 0, \dots, i-1$. The case $\varphi_i(x)$ does not have a solution in N or $i \geq m$, let the interpretation of c_i be some element in N. With this interpretation, N also becomes a model of S. $\boxed{\text{end of the proof of Claim A}}$

Next, we use Zorn's lemma to get T^* such that T^* is a maximal L^*-theory among the theories which contain T' and any finite subset of which has a model. T^* becomes a complete L^*-theory. In fact, if L^*-sentence θ is not in T^*, then for any finite $\Delta \subset T^*$ and its model N, θ does not hold in N, so $\neg\theta$ holds in N and $\Delta \cup \{ \neg\theta \}$ has a model. This means any finite subset of $T' \cup \{ \neg\theta \}$ has a model, and $\neg\theta \in T^*$ by the maximality of T^*.

We now define an L^*-structure M. Let $\mathcal{A}$ be the set of all closed L^*-terms and $\sim$ be the binary relation such that $t \sim s$ if and only if $t = s \in T^*$ for $t, s \in \mathcal{A}$. This $\sim$ becomes equivalence relation on $\mathcal{A}$ by the choice of T^*. We put the universe of M be $\mathcal{A}/\sim$. We define the interpretation of L^* in M as follows:

- For constant symbol c, let $c^M = [c]$.
- For m-ary function symbol F, let $F^M([t_1], \ldots, [t_m]) = [F(t_1, \ldots, t_m)]$.
- For m-ary predicate symbol P, let $([t_1], \ldots, [t_m]) \in P^M \Leftrightarrow P(t_1, \ldots, t_m) \in T^*$.

By the maximality of T^*, we can see that this interpretation is well-defined , and that $t^M = [t]$ holds for all $t \in \mathcal{A}$.

Claim B. *M is a model of T^**

Proof of Claim B. For an arbitrary $\varphi \in T^*$, we show φ holds in M. We use induction on the number l of the logical symbols in φ.

For the case $l = 0$, φ is an atomic formula. If φ has the form of $t = s$ for some $t, s \in \mathcal{A}$, then $t \sim s$ and $t^M = [t] = [s] = s^M$ holds. Hence φ holds in M. If φ has the form of $P(t_1, \ldots, t_m)$, then

$$\begin{aligned} P(t_1, \ldots, t_m) \in T^* &\Leftrightarrow ([t_1], \ldots, [t_m]) \in P^M \\ &\Leftrightarrow (t_1^M, \ldots, t_m^M) \in P^M \\ &\Leftrightarrow M \models P(t_1, \ldots, t_m), \end{aligned}$$

and so φ holds in M.

Suppose the case l holds and φ has $l+1$ logical symbols. Recall that $\psi_1 \vee \psi_2$, $\psi_1 \to \psi_2$, and $\forall x\, \psi$ are shorthands for $\neg(\neg\psi_1 \wedge \neg\psi_2)$, $\neg\psi_1 \vee \psi_2$, and $\neg\exists x\, \neg\psi$, respectively. If φ has the form of $\psi_1 \wedge \psi_2$ or $\neg\psi$, then the case is trivial. The remaining case is when φ has the form of $\exists x\, \psi(x)$. By the definition of $T^* \supset T'$, $\exists x\, \psi(x) \to \psi(c_i)$ is an element of T^* for some $i \in \omega$. Thus since any finite subset of T^* has a model and T^* is maximal, $\psi(c_i)$ is in T^*. Here, by induction hypothesis, $\psi(c_i)$ holds in M and so $[c_i]$ is the solution for $\psi(x)$ in M. Therefore $\exists x\, \psi(x)$ holds in M. $\boxed{\text{end of the proof of Claim B}}$

Hence M is a model of T when considered as an L-structure. □

1.2 Model theoretic proof of quantifier elimination for real closed fields

Here we proceed to show the fact that the theory RCF admits QE. First, we give the definition of the axiom of RCF as well as QE.

Definition 1.2.11. Let $\mathcal{L}_{\mathrm{OR}}$ be the language $\{ <, 0, 1, +, -, \cdot \}$. The axiom RCF for real closed ordered fields is the set of the $\mathcal{L}_{\mathrm{OR}}$-sentences consists of the following:

- axioms for ordered fields,
- the sentence $\forall x\, \exists y\, x > 0 \to x = y^2$, and

· the sentences of the form $\forall x_1 \dots \forall x_{2n+1} \exists y\, y^{2n+1} + x_1 y^{2n} + \cdots + x_{2n+1} = 0$.

In the following of this section, we write $\top$ and $\bot$ for a tautology and a contradiction, respectively. When we have a constant symbol c in the language, $\top$ and $\bot$ can be represented by quantifier-free sentences such as $c = c$ and $\neg c = c$. If the language does not have a constant symbol, there are no quantifier-free sentences. In such a case, $\top$ and $\bot$ can be taken as $\exists x\ x = x$ and $\exists x\ \neg x = x$, respectively , which are two trivial formulas but with a quantifier $\exists$.

Definition 1.2.12. An L-theory T admits quantifier elimination if for any L-formula $\varphi(\bar{x})$, there exists an L-formula $\psi(\bar{x})$ such that $\psi(\bar{x})$ is either quantifier-free, $\top$, or $\bot$, and satisfies the condition $T \models\ \forall \bar{x}\, \varphi(\bar{x}) \leftrightarrow \psi(\bar{x})$.

1.2.1 QE tests

We prepare some equivalent conditions for a theory to admit QE before we work on the theory RCF. The first equivalent condition is rather simple.

Definition 1.2.13. An L-formula is called to be primitive if it has a form of $\exists y\, \varphi(\bar{x}, y)$, where φ is a conjunction of atomic formulas and negations of atomic formulas.

Proposition 1.2.14. *An L-theory T admits quantifier elimination if and only if any primitive formula is equivalent to a quantifier-free formula under T.*

The proof of proposition 1.2.14 is easy. Just use the fact that the existential quantifier distributes over logical disjunctions, and notice that an existentially quantified formula in disjunctive normal form is equivalent to a disjunction of primitive formulas.

A more subtle condition for the QE test is stated in terms of embeddings.

Definition 1.2.15. Let M and N be two L-structures, A be a subset of M, and f be a partial map $f : A \to N$.

(1) We say that f is an L-embedding if it holds that

$$M \models \varphi(\bar{a}) \text{ if and only if } N \models \varphi(f(\bar{a})),$$

for every quantifier-free L-formula $\varphi(\bar{x})$ and any $\bar{a} \in A$

(2) We say that f is an A-elementary map if we have

$$M \models \varphi(\bar{a}) \text{ if and only if } N \models \varphi(f(\bar{a})),$$

for every L-formula $\varphi(\bar{x})$ and any $\bar{a} \in A$. We simply say f is an elementary map when A is clear.

(3) Suppose M is a substructure of N. We say that N is an elementary extension of M or M is an elementary submodel of N, and write $M \prec N$, if the inclusion map is an elementary map.

We need the next lemma to strengthen the second condition for QE test.

Lemma 1.2.16. *Let M and N be L-structures and A be a non-empty subset of M. The following hold:*

(1) $\mathrm{cl}(A) = \left\{ t^M(\bar{a}) \,\middle|\, \bar{a} \in A \text{ and } t(\bar{x}) \text{ is an } L\text{-term} \right\}$ *is a smallest substructure of M containing A.*

(2) *An arbitrary L-embedding $f : A \to N$ can be extended to an L-embedding $\bar{f} : \mathrm{cl}(A) \to N$.*

Proof. (1) It is trivial that any substructure containing A also contains $\mathrm{cl}(A)$. Hence it suffices to prove that $\mathrm{cl}(A)$ is a substructure of M. In general, there always exists some interpretation for any constant or predicate symbol on any non-empty subset, so particularly on $\mathrm{cl}(A)$. We show that there is an interpretation for a function symbol $f(x_1, \dots, x_n)$ in L on $\mathrm{cl}(A)$. For any $\alpha_1, \dots, \alpha_n$

in $\mathrm{cl}(A)$, each α_i has a form of $t_i^M(\bar{a})$. Notice $t(\bar{x}) = f(t_1(\bar{x}), \dots, t_n(\bar{x}))$ is an L-term. Thus $t^M(\bar{a})$ is an element in $\mathrm{cl}(A)$. We can define the interpretation of f at $(\alpha_1, \dots, \alpha_n)$ by

$$f^{\mathrm{cl}(A)}(\alpha_1, \dots, \alpha_n) = t^M(\bar{a}).$$

The value is independent of the choice of $t_i(\bar{x})$ and $f^{\mathrm{cl}(A)}$ is well-defined.

(2) Let $f : A \to N$ be an L-embedding and $\varphi(x_1, \dots, x_m)$ be any quantifier free formula. We define $\bar{f} : \mathrm{cl}(A) \to N$ by

$$\bar{f}(\alpha) = t^N(f(a_1), \dots, f(a_k)),$$

where $t(\bar{x})$ is the L-term such that $\alpha = t^M(a_1, \dots, a_k)$. This definition is well-defined since f is an L-embedding. We need to show that for all $\alpha_1, \dots, \alpha_n$ in $\mathrm{cl}(A)$, $M \models \varphi(\alpha_1, \dots, \alpha_m)$ if and only if $N \models \varphi(\bar{f}(\alpha_1), \dots, \bar{f}(\alpha_m))$. We can find the L-terms $t_i(\bar{x})$ such that $\alpha_i = t_i^M(a_1, \dots, a_n)$. Thus we have

$$\begin{aligned}
&N \models \varphi(\bar{f}(\alpha_1), \dots, \bar{f}(\alpha_m)) \\
\Leftrightarrow\ &N \models \varphi(t_1(f(a_1), \dots, f(a_n)), \dots, t_m(f(a_1), \dots, f(a_n))) \\
\Leftrightarrow\ &M \models \varphi(t_1(a_1, \dots, a_n), \dots, t_m(a_1, \dots, a_n)) \\
\Leftrightarrow\ &M \models \varphi(\alpha_1, \dots, \alpha_m).
\end{aligned}$$

□

Here we see the second condition that gives a useful criteria to show RCF admits QE. We use the following notation in the proof below.

Definition 1.2.17. Let M be an L-structure as well as A and B be subsets of M. We prepare a set of new variables X and fix an enumeration for X as $(x_b)_{b \in B}$.

(1) We define $\mathrm{Diag}_A(B)$, the diagram of B over A, and $\mathrm{tp}_A(B)$, the type of B over A by

$$\begin{aligned}
&\mathrm{Diag}_A(B) = \{\, \varphi(x_{b_1}, \dots, x_{b_n})\text{: quantifier-free } L_A\text{-formula} \mid M \models \varphi(b_1, \dots b_n),\ (b_1, \dots, b_n) \in B \,\} \\
&\mathrm{tp}_A(B) = \{\, \varphi(x_{b_1}, \dots, x_{b_n})\text{: } L_A\text{-formula} \mid M \models \varphi(b_1, \dots, b_n),\ (b_1, \dots, b_n) \in B \,\}
\end{aligned}$$

(2) Let $\Phi(\bar{x})$ be a set of L-formulas of which free variables are contained in $\bar{x}$, and T be a theory. We say that $\Phi(\bar{x})$ is satisfiable in T, if there is a model M of T and a tuple $\bar{a} \in M$ such that $M \models \varphi(\bar{a})$ for all $\varphi \in \Phi$.

Proposition 1.2.18. *Let T be a complete L-theory. The following are equivalent:*

(i) T admits quantifier elimination.

(ii) For any models M and N of T, subsets A and A' with $A \subsetneq A' \subset M$, and L-embedding $f : A \to N$, f can be extended to an L-embedding $f' : A' \to N'$, where N' is an elementary extension of N.

Moreover, we can replace subsets A and A' in the statement (ii) by substructures A and A' without loss of equivalence.

Proof. Suppose T admits quantifier elimination, and M and N are models of T. Since f is an L-embedding, we have $\mathrm{Diag}(A) = \mathrm{Diag}(f(A))$. By quantifier elimination, it holds that $\mathrm{tp}(A) = \mathrm{tp}(f(A))$. Hence, because $p_A(X) = \mathrm{tp}_A(A')$ is satisfiable in M, $p_{f(A)}(X)$ is finitely satisfiable in N. By compactness theorem, we have a witness B of $p_{f(A)}(X)$ in some elementary extension N' of N.

For the converse, we use a contrapositive argument. Suppose T does not admit quantifier elimination. By proposition 1.2.14, there exists a primitive formula $\varphi(\bar{x})$ that is not equivalent to a quantifier-free formula. Notice $\varphi(\bar{x})$ is not equivalent to $\top$ nor $\bot$, so $T \not\models \forall \bar{v}\, \varphi(\bar{v})$ and $T \not\models \forall \bar{v}\, \neg\varphi(\bar{v})$ holds. Hence both $T \cup \{\, \varphi(\bar{v}) \,\}$ and $T \cup \{\, \neg\varphi(\bar{v}) \,\}$ are satisfiable. Let $\Gamma(\bar{v})$ be the set

$$\Gamma(\bar{v}) = \{\, \psi(\bar{v}) \mid \psi \text{ is a quantifier-free formula such that } T \models \forall \bar{v}\, \varphi(\bar{v}) \to \psi(\bar{v}) \,\}.$$

Since φ is not equivalent to a quantifier-free formula, we have $T \not\models \forall \bar{v}\, \psi(\bar{v}) \to \varphi(\bar{v})$, for any $\psi(\bar{v})$ in Γ. Hence $T \cup \Gamma(\bar{v}) \cup \{ \neg\varphi(\bar{v}) \}$ is finitely satisfiable, and so satisfiable. We introduce new constant symbols $\bar{d}$ and take a model N of $T \cup \Gamma(\bar{d}) \cup \{ \neg\varphi(\bar{d}) \}$. Let $\bar{a}$ be the interpretation of $\bar{d}$ in N, and $\Delta(\bar{v})$ be $\mathrm{Diag}(\bar{a})$

Claim. $T \cup \Delta(\bar{d}) \cup \{ \varphi(\bar{d}) \}$ *is consistent.*

Proof of Claim. For any $\psi(\bar{v})$ in $\Delta(\bar{v})$, it holds that $\neg\psi(\bar{v}) \notin \mathrm{Diag}(\bar{a})$ and so that $\neg\psi(\bar{v}) \notin \Gamma(\bar{v})$. By the definition of Γ, we have $T \models \exists \bar{v}\, \varphi(\bar{v}) \wedge \psi(\bar{v})$. This shows that $T \cup \{ \psi(\bar{d}),\ \varphi(\bar{d}) \}$ is consistent, and so $T \cup \Delta(\bar{d}) \cup \{ \varphi(\bar{d}) \}$ is finitely satisfiable. By compactness, we have shown the claim. $\boxed{\text{end of the proof of Claim}}$

Let M be a model of $T \cup \Delta(\bar{d}) \cup \{ \varphi(\bar{d}) \}$, and $\bar{a'}$ be the interpretation of $\bar{d}$ in M. Because $\varphi(\bar{v})$ is primitive, φ has the form $\exists y\, \varphi'(\bar{x}, y)$ for some quantifier-free $\varphi'(\bar{x}, y)$. We put b in M to be a witness of $\exists y\, \varphi'(\bar{a'}, y)$. We have found, as we desired, the L-embedding $f : \bar{a'} \mapsto \bar{a}$ that cannot be extended to an L-embedding $\bar{a'}b \to N'$ for any elementary extension N' of N, since $\neg\exists y\, \varphi'(\bar{a}, y)$ holds in N and N'.

For the moreover part, use lemma 1.2.16 to extend the L-embeddings to the suitable substructures.

□

1.2.2 QE for RCF

In this subsection, we follow the argument in [31] to prove QE for RCF. We use some facts about ordered domains.

Definition 1.2.19. Let A be an ordered domain.

(i) A real closure of A is a real closed ordered field extending A and algebraic over the fraction field of A.

(ii) For subsets C and D of A, we say $C < D$ holds if $c < d$ for any $c \in C$ and $d \in D$. We may write $a < D$ and $C < a$ if C and D respectively is a singleton set $\{ a \}$.

(iii) A pair (C, D) of subsets of A is said to be a cut in A if it holds that $C \cup D = A$ and $C < D$.

Fact 1.2.20 (Artin and Schreier 1926)**.** *The following hold:*

(i) *Any ordered domain A has a real closure $\overline{A}$.*

(ii) *Let A be an ordered domain and F be a real closed field. Any embedding of ordered domains $A \to F$ extends to an embedding $\overline{A} \to F$.*

(iii) *Let A be a real closed field and $A(b)$ and $A(c)$ be two ordered field extensions with $b, c \notin A$. If b and c determine the same cut in A, i.e. $\{ a \in A \mid a < b \} = \{ a \in A \mid a < c \}$ holds, then there is an A-isomorphism σ of ordered fields from $A(b)$ onto $A(c)$ with $\sigma(b) = \sigma(c)$.*

Finally, we obtain the following theorem.

Theorem 1.2.21. RCF *admits quantifier elimination.*

Proof. We apply proposition 1.2.18 to prove this theorem. Let K and L be real closed ordered fields, A and A' be ordered subrings with $A \subsetneq A' \subset K$, and $i : A \to L$ be an embedding of ordered rings. We are going to show i can be extended to an embedding $A' \to L'$ for some elementary extension L' of L. By fact 1.2.20 (ii), we can reduce the above to the case that A itself is a real closed ordered field. Take any $b \in A' \setminus A$. Then b determines a cut (U, V) in A, where $U = \{ x \in A \mid x < b \}$ and $V = \{ x \in A \mid b < x \}$. Thus $i(U) < i(V)$ in L. By a compactness argument, we can find a suitable elementary extension L' of L so that there is an element $c \in L'$ such that $i(U) < c < i(V)$: If we put $\Delta(x) = \{ u < x \mid u \in U \} \cup \{ x > v \mid v \in V \}$ as well as $\Delta_0(x)$ to be a finite subset of Δ, then there exist elements s and t in L with $s < t$ such that a formula $s < x < t$ implies Δ_0, and thus Δ_0 is satisfied with an element in L such as $(s + t)/2$. Then by fact 1.2.20 (iii) above we can extend i to an ordered field embedding from $A(b)$ into L'. □

Chapter 2

Support vector machines

A support vector machine is an algorithm to obtain a classification hypothesis from a given set of training samples. After Vapnik first introduced SVMs in 1979 [32], their descriptions are specifically written in the books [33, 34]. In this chapter, we see an elementary explanation on SVMs. For a more thorough tutorial, refer to [3].

2.1 Learning for binary linearly separable samples

In this section, we see the most elementary SVMs which learn binary classifications from linearly separable samples. We put a set of sample data T as $T = \{ (x_i, y_i) \in \mathbb{R}^n \times \{-1, 1\} \mid 1 \le i \le k \}$. Our goal is to find an n-dimension hyperplane $\langle x_i, w\rangle + b = 0$ which separates positive samples $P = \{ x_i \mid y_i = 1 \}$ and negative samples $N = \{ x_i \mid y_i = -1 \}$. Besides, we determine the one that maximize the sum of distances from itself to the set of points P and N.

This problem can be mathematically expressed by the following minimization problem:

$$\text{(OP)}\begin{cases} \text{Minimize: } \|w\|^2, \\ \text{subject to: } y_i(\langle x_i, w\rangle + b) - 1 \ge 0 \ (1 \le i \le k). \end{cases}$$

We put the feasible region to be X. Let us translate this minimization problem (OP) into another problem using the results in convex optimization. First, we let $\bar{\lambda} = (\lambda_i)_{1\le i\le k}$ and the Lagrange function

$$\mathcal{L}(w, b, \bar{\lambda}) = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{k} \lambda_i(y_i(\langle x_i, w\rangle + b) - 1).$$

Observe that it holds

$$\max_{\bar{\lambda}\ge 0} \mathcal{L}(w, b, \bar{\lambda}) = \begin{cases} \frac{1}{2}\|w\|^2 & (w, b) \text{ in } X, \\ \infty & \text{otherwise,} \end{cases}$$

where $\bar{\lambda} \ge 0$ denotes the condition $\lambda_i \ge 0$ for each $1 \le i \le k$. Then we have

$$\frac{1}{2} \min_{(w,b)\in X} \|w\|^2 = \min_{w,b} \max_{\bar{\lambda}\ge 0} \mathcal{L}(w, b, \bar{\lambda}).$$

Here, we can use a theorem in convex quadratic optimization. Namely, we can apply strong duality to the optimization problem (OP), because (OP) is a convex quadratic programming and X satisfies Slater's condition. Therefore, it holds that

$$\min_{w,b} \max_{\bar{\lambda}\ge 0} \mathcal{L}(w, b, \bar{\lambda}) = \max_{\bar{\lambda}\ge 0} \min_{w,b} \mathcal{L}(w, b, \bar{\lambda}).$$

Further, we can calculate the part $\min_{w,b} \mathcal{L}(w, b, \bar{\lambda})$ using the equivalent condition that the partial derivatives of each component of w and b are all equal to zero at the solution point (w^*, b^*):

$$\left.\frac{L(w,b,\bar{\lambda})}{\partial w_j}\right|_{(w,b)=(w^*,b^*)} = w_j^* - \sum_{i=1}^{k} \lambda_i y_i x_{ij} = 0,$$
$$\left.\frac{L(w,b,\bar{\lambda})}{\partial b}\right|_{(w,b)=(w^*,b^*)} = \sum_{i=1}^{k} \lambda_i y_i = 0.$$

Finally, we get the following maximization problem in variables $\bar{\lambda}$:

$$\text{(DP)}\begin{cases} \text{Maximize: } \sum_{i=1}^{k} \lambda_i - \frac{1}{2} \sum_{i,j=1}^{k} \lambda_i \lambda_j y_i y_j \langle x_i, x_j \rangle, \\ \text{subject to: } \sum_{i=1}^{k} \lambda_i y_i = 0,\ \bar{\lambda} \geq 0. \end{cases}$$

The solution for the original problem (OP) is given by $w = \sum_{i=1}^{k} \lambda_i y_i x_i$, while we cannot solve for the implicitly determined b from the solution for (DP). We can determine b by the additional conditions $\lambda_i(y_i(\langle x_i, w\rangle + b) - 1) = 0$ $(1 \leq i \leq k)$, which are derived from the Karush-Kuhn-Tucker conditions for (OP).

The problem (DP) has the important property that the training data x_i only appear in the form of the inner product. In the original problem (OP), the number of variables is n, which is the number of the dimension of the training samples, and the number of constraint conditions is k, the number of samples. On the other hand, (DP) has k variables and k constraint conditions, which is independent from n after we calculate the inner product between the training samples. This property help us to efficiently perform the algorithm and generalize it to the nonlinear case.

2.2 Non-separable case

With the above setting for separable data, no feasible solution can be found if non-separable data are given. This happens because the constraint conditions in (OP) are too strict. We deal with this problem by relaxing the constraint condition, whenever needed. We introduce a positive slack variable ξ_i for the i-th sample to obtain a loosened constraint

$$y_i(\langle x_i, w\rangle + b) - 1 + \xi_i \geq 0.$$

Notice that ξ_i with $\xi_i > 1$ causes the i-th sample classified into the class conflicting with the label. Thus $\sum_i \xi_i$ is an upper bound on the number of the training errors. We charge an extra cost to the training errors by changing the objective function of (OP):

$$(\text{OP}_C)\begin{cases} \text{Minimize: } \|w\|^2 + C \sum_{i=1}^{k} \xi_i, \\ \text{subject to: } y_i(\langle x_i, w\rangle + b) - 1 + \xi_i \geq 0, \text{ and } \xi_i \geq 0\ (1 \leq i \leq k), \end{cases}$$

where C is a hyper parameter, *i.e.* a parameter determined by the user. A larger C results in a separating hyperplane more loyal to the information of the sample points. With a smaller (positive) C, we obtain a classifier that more aggressively ignores the sample points likely to be exceptional.

The optimization problem (OP_C) can be solved in a similar way to (OP). Let X denote the feasible region of (OP_C). We put $\bar{x}_i = (\xi_i)_{1 \leq i \leq k}$, $\bar{\lambda} = (\lambda_i)_{1 \leq i \leq k}$, and $\bar{\mu} = (\mu_i)_{1 \leq i \leq k}$ and the Lagrange function

$$\mathcal{L}(w, b, \bar{\xi}, \bar{\lambda}, \bar{\mu}) = \frac{1}{2}\|w\|^2 + C \sum_{i=1}^{k} \xi_i - \sum_{i=1}^{k} \lambda_i(y_i(\langle x_i, w\rangle + b) - 1 + \xi_i) - \sum_{i=1}^{k} \mu_i \xi_i.$$

We use strong duality again to have

$$\min_{(w,b,\bar{\xi})\in X}\left(\frac{1}{2}\|w\|^2 + C\sum_{i=1}^{k}\xi_i\right) = \max_{\substack{\bar{\lambda}\geq 0\\ \bar{\mu}\geq 0}}\min_{w,b,\bar{x}i}\mathcal{L}(w,b,\bar{\xi},\bar{\lambda},\bar{\mu}).$$

Concerning $\min_{w,b,\bar{\xi}}\mathcal{L}(w,b,\bar{\xi},\bar{\lambda},\bar{\mu})$, at the solution point $(w^*,b^*,\bar{\mu}^*)$ it holds that

$$\begin{aligned}
\left.\frac{L(w,b,\bar{\xi},\bar{\lambda},\bar{\mu})}{\partial w_j}\right|_{(w,b)=(w^*,b^*,\bar{\mu}^*)} &= w_j^* - \sum_{i=1}^{k}\lambda_i y_i x_{ij} = 0,\\
\left.\frac{L(w,b,\bar{\xi},\bar{\lambda},\bar{\mu})}{\partial b}\right|_{(w,b)=(w^*,b^*,\bar{\mu}^*)} &= \sum_{i=1}^{k}\lambda_i y_i = 0,\\
\left.\frac{L(w,b,\bar{\xi},\bar{\lambda},\bar{\mu})}{\partial \xi_j}\right|_{(w,b)=(w^*,b^*,\bar{\mu}^*)} &= C - \lambda_j - \mu_j = 0.
\end{aligned}$$

Finally, we obtain the dual problem of (OP_C):

$$(\mathrm{DP}_C)\begin{cases}\text{Maximize: } \displaystyle\sum_{i=1}^{k}\lambda_i - \frac{1}{2}\sum_{i,j=1}^{k}\lambda_i\lambda_j y_i y_j\langle x_i, x_j\rangle,\\[2ex] \text{subject to: } \displaystyle\sum_{i=1}^{k}\lambda_i y_i = 0, \text{ and } 0\leq\bar{\lambda}\leq C.\end{cases}$$

The solution for (OP_C) is given by the same condition $w = \sum_{i=1}^{k}\lambda_i y_i x_i$ as in the separable case. The Karush-Kuhn-Tucker conditions for (OP_C) imply that b is given by $\lambda_i(y_i(\langle x_i, w\rangle + b) - 1) = 0$ with such i that $0 < \lambda_i < C$ holds.

2.3 Multiclass classification problems

One way to handle the multiclass case using SVMs is learning the one-versus-rest classifiers. For this method, in order to deal with an m-class classification, we train the m binary classifiers each of which labels are positive for just one class and negative for the other. The predicted class for an unknown sample is the class that corresponding classifier yields the largest positive distance.

More specifically, for the samples $T = \{\,(x_i, y_i)\in\mathbb{R}^n\times\{\,1,\dots,m\,\}\mid 1\leq i\leq k\,\}$, we define the training data $T_j\subset\mathbb{R}^n\times\{\,-1,1\,\}$ that corresponds to the label j by the following:

$$(x_i,y_i)\in T_j \Leftrightarrow \begin{cases} y_i = 1 & \text{if } (x_i,j)\in T, \text{ and}\\ y_i = -1 & \text{if } (x_i,j)\notin T.\end{cases}$$

Then m binary classifiers are trained by SVMs to obtain the hyperplane $\langle w_j, x\rangle + b_j = 0$ $(1\leq j\leq m)$. An unknown sample $\tilde{x}\in\mathbb{R}^n$ is predicted to belong to the class such that maximize the distance from the hyperplanes, which are expressed by

$$\frac{|\langle w_j,\tilde{x}\rangle + b_j|}{\|w_j\|}.$$

2.4 Kernel methods

The generalization of SVMs to nonlinear classification is achieved by kernel methods, where we nonlinearly embed training data into a higher dimensional vector space to find a more effective hyperplane to separate them. Here, we only need to calculate the inner product in the higher dimensional vector space between the embedded data to solve the maximization problem (DP)/(DP$_C$) and so the minimization problem (OP)/(OP$_C$). Thus, we can train a classifier without the explicit expression of nonlinear embedding.

A kernel is the function of which value is an inner product of the embedded samples.

Definition 2.4.22. Let X be a set and K be a map $X \times X \to \mathbb{R}$.

(i) K is positive definite if $\sum_{i,j} c_i c_j K(x_i, x_j) \geq 0$ holds for any natural number N, real numbers $c_1, \ldots, c_N$, and $x_1, \ldots, x_N$ in X.

(ii) K is said to be a kernel if K is symmetric and positive definite.

The following theorem states that a kernel is in fact an inner product in some Hilbert space. It is known that the generalized form of the theorem holds for a separable metric space X [7].

Theorem 2.4.23 (Mercer 1909)**.** *Let X be a countable set. The following hold* :

(i) *For any Hilbert space $(H, \langle\ ,\ \rangle)$ and embedding $\varphi : X \to H$, the map $\langle \varphi(*), \varphi(*) \rangle$ is a kernel.*

(ii) *For any kernel K, there exist a Hilbert space $(H, \langle\ ,\ \rangle)$ and an embedding $\varphi : X \to H$ such that $K(x, y) = \langle \varphi(x), \varphi(y) \rangle$ holds.*

By using kernel methods, we can solve not only nonlinear classification problems on vector spaces, but also classification problems on sets with some structures. Another way to learn a classifier on structured sets is specifying the feature vector of an element and reduce the problem to classification on the vector space, which amounts to express explicitly the embedding from the set to the vector space. On the other hand, by using kernel methods we can perform SVMs keeping embeddings to vector spaces implicit.

Part II

Research contribution

Chapter 3

Efficient subformula orders for real quantifier elimination of non-prenex formulas

This chapter is a reprint of [19] and is a joint work with H. Iwane, T. Matsuzaki and H. Anai.

3.1 Introduction

In this study we aim at speeding up quantifier elimination (QE) methods for *non-prenex formulas* over the reals by automatizing a heuristic procedure in QE computation using a machine learning method.

When we discuss real QE algorithms, we usually assume that input first-order formulas are in prenex form. We say a formula is in prenex form if it has the form of a string of quantifiers followed by a quantifier-free part. However, given formulas are not always prenex ones in practice. Transforming non-prenex formulas to prenex form causes to miss the algebraic independency between variables and then the QE computation tends to be hard. Hence, in view of practical computation, the following strategy is considered to be effective: dividing a given non-prenex formula into subformulas, applying QE to each subformulas, and then logically integrating the results. Furthermore, we can obtain further efficiency by utilizing intermediate QE results to simplify remaining formulas before applying QE to them.

Here we face the problem to determine the order of performing QE for the subformulas. Total time required for achieving QE for the input non-prenex formula depends heavily on the order. Ordinarily, some heuristic methods to choose an appropriate order for subformulas are implemented (e.g., "easy to solve" subformula is first). For example, this strategy is successfully employed in [12].

In spite of its practical efficacy, studies on QE for non-prenex formulas are few [29]. Actually, the existing computer algebra systems that accept non-prenex formulas as inputs for QE execution are only Mathematica and SyNRAC [13] on Maple. We focus on QE for non-prenex formulas to improve efficiency of QE algorithm in practical use.

Our target problem is to determine an appropriate order for improving the efficiency of QE computation for non-prenex formulas. In this study we design some heuristics methods and others based on machine learning, and examine their effectiveness through detailed analysis of the experimental results over 2,306 nontrivial QE problems.

The 2,306 nontrivial QE problems are provided from the activity in "Todai robot project – Can a Robot Get Into the University of Tokyo?" [1, 22], which is a project to develop an artificial intelligence system to automatically solve natural language math problems of university entrance examinations. As 2,306 samples are sufficient to apply machine learning, the binary classification problem of if a formula is easy to perform QE for was learned with support vector machines (SVMs).

So far there are some attempts to propose heuristic methods for determining variable orders aiming at speeding up elimination methods in computer algebra. Dolzmann et al. gave heuristics that estimates the optimal projection order for cylindrical algebraic decomposition (CAD) [6]. Their study is based on a statistical analysis. Later, Huang et al. applied machine learning to choose heuristics for selecting projection order for CAD [9]. They showed SVMs, which are based on statistical learning theory, achieved a good switching of heuristics and the obtained mixture of heuristics performed better than any single heuristics.

While Huang et al. mainly measured the numbers of cells produced by CAD, we made our study more practical by dealing with time performance. We arranged this study in hope with a possible contribution to Todai robot project. Thus, among time performances, one of which is simple execution time, we most value the property that a QE computation finishes in a fixed time.

Our experimental results on more than 2,000 QE problems also indicate that machine learning based methods are promising for heuristic processes if we have sufficient amount of data (input problems and computational results). This fact is beneficial in efficiently designing an effective method for such heuristics parts since usually developing effective heuristic methods takes a lot of trials and errors.

The rest of this section is organized as follows. Section 3.2 shows the problem we discuss. Section 3.3 provides some methods to determine an appropriate order of QE computation for subformulas from an input non-prenex formula. Section 3.4 is devoted to explain the computational results of the proposed methods and detailed analysis. The concluding remarks are made in section 3.5.

3.2 Problem

In this study, we study efficient QE for non-prenex formulas over the reals. The following formula φ gives an example of a non-prenex formula:

$$\varphi \equiv \varphi_1 \vee \varphi_2 \vee \varphi_3, \tag{3.1}$$

where

$$\begin{aligned}
&\varphi_1 \equiv \exists x_0(-x_0 \leq -1),\\
&\varphi_2 \equiv \exists x_1(x_1 \leq 0), \text{ and}\\
&\varphi_3 \equiv \exists x_2 \exists x_3 \exists x_4 \Big(\exists x_5\big((x_2^3x_4x_5 - x_2^2x_4 - x_2^2x_5 + x_2 + 1 = 0 \vee\\
&x_2^3x_4x_5x_6 - x_2^2x_4x_5 - x_2^2x_4x_6 - x_2^2x_5x_6 + x_2x_4 + x_2x_5 + x_2x_6 - x_6 = 0) \;\wedge\\
&\qquad\qquad (x_2^3x_4x_6 - x_2^2x_4 - x_2^2x_6 + x_2 + 1 = 0 \vee\\
&\quad x_2^3x_4x_5x_6 - x_2^2x_4x_5 - x_2^2x_4x_6 - x_2^2x_5x_6 + x_2x_4 + x_2x_5 + x_2x_6 - x_5 = 0)\big) \;\wedge\\
&\qquad\qquad \big(x_2^3x_3 - x_2^2x_3 - x_2^2 + x_2 + 1 = 0 \vee\\
&\qquad x_2^3x_3x_4 - x_2^2x_3x_4 - x_2^2x_3 - x_2^2x_4 + x_2x_3 + x_2x_4 + x_2 - x_4 = 0\big) \;\wedge\\
&\qquad\qquad \big(x_2^3x_4 - x_2^2x_4 - x_2^2 + x_2 + 1 = 0 \vee\\
&\qquad x_2^3x_3x_4 - x_2^2x_3x_4 - x_2^2x_3 - x_2^2x_4 + x_2x_3 + x_2x_4 + x_2 - x_3 = 0\big)\Big).
\end{aligned}$$

Our approach is to modify the order of processing subformulas by varying ordering functions on formulas.

The paper [12] of Iwane et al. states that sort orderings on formulas affect the efficiency of QE computation. The paper treats the problem of constructing formulas equivalent to university entrance examination problems using the natural language processing, and the problem of solving them by performing QE for the constructed formulas. According to [12], construction of formulas by the current natural language processing results in large and complicated QE problems, and solving such problems simply by a general QE algorithm is not realistic. The following approaches are proposed by them to be effective in QE computation if the input QE problem is much redundant: use of special QE algorithms, simplification of intermediate formulas, and decomposition of the formula into separately solvable parts. Algorithm 1 (adapted from [12]) shows their procedure to perform QE for a non-prenex formula. $\mathrm{QE}_{\mathrm{prenex}}$ in Algorithm 1 executes QE using traditional algorithms such as CAD, virtual term substitution, and QE by real root counting [11] in the form $\forall x(f(x) > 0)$ and $\forall x(x \geq 0 \to f(x) > 0)$. Their proposed QE algorithm rearranges an input formula into non-prenex form, makes sorting over the subformulas, and in that order performs QE recursively for each subformula. The conditions obtained from solved (i.e., quantifier-free) subformulas are immediately utilized to simplify the remaining subformulas. Hence subformulas should be sorted in the order from easiest to hardest. Our study investigated the effect of changing 'Sort' in Algorithm 1.

Algorithm 1 $\mathrm{QE}_{\mathrm{main}}(\varphi, \psi_{\mathrm{nec}}, \psi_{\mathrm{suf}})$

```
Input: a first-order formula $\varphi$, quantifier-free formulas $\psi_{\mathrm{nec}}$ and $\psi_{\mathrm{suf}}$
Output: a quantifier-free formula $\varphi'$ s.t. $\varphi \wedge \psi_{\mathrm{nec}} \vee \psi_{\mathrm{suf}}$ is equivalent to $\varphi' \wedge \psi_{\mathrm{nec}} \vee \psi_{\mathrm{suf}}$
 1: if $\varphi$ is quantifier-free then
 2:     return SIMPLIFY($\varphi, \psi_{\mathrm{nec}}, \psi_{\mathrm{suf}}$)
 3: else if $\varphi$ is a prenex formula then
 4:     return $\mathrm{QE}_{\mathrm{prenex}}(\varphi, \psi_{\mathrm{nec}}, \psi_{\mathrm{suf}})$
 5: else if $\varphi \equiv \mathsf{Q}_1 x_1 \cdots \mathsf{Q}_m x_m\ \xi$ where $\mathsf{Q}_j \in \{\exists, \forall\}$ then
 6:     /* $\xi$ is not quantifier-free */
 7:     return $\mathrm{QE}_{\mathrm{prenex}}(\mathsf{Q}_1 x_1 \cdots \mathsf{Q}_m x_m \mathrm{QE}_{\mathrm{main}}(\xi, \psi_{\mathrm{nec}}, \psi_{\mathrm{suf}}), \psi_{\mathrm{nec}}, \psi_{\mathrm{suf}})$
 8: else if $\varphi \equiv \vee \xi_i$ then
 9:     return $\neg \mathrm{QE}_{\mathrm{main}}(\neg\varphi, \neg\psi_{\mathrm{suf}}, \neg\psi_{\mathrm{nec}})$
10: /* $\varphi \equiv \wedge \xi_i$ */
11: if $\xi_i \equiv (f_i = 0)$ and $f_i$ is reducible ($f_i = \prod g_{i,j}$) then
12:     return $\mathrm{QE}_{\mathrm{main}}(\vee_j (g_{i,j} = 0 \wedge (\wedge_{k \neq i} \xi_k)), \psi_{\mathrm{nec}}, \psi_{\mathrm{suf}})$
13: $\{\varphi_i\}_{i=1}^n \leftarrow$ SORT($\{\xi_i\}_{i=1}^n$)
14: for all $i$ such that $1 \leq i \leq n$ do
15:     $\varphi_i \leftarrow \mathrm{QE}_{\mathrm{main}}(\varphi_i, \psi_{\mathrm{nec}}, \psi_{\mathrm{suf}})$
16:     $\psi_{\mathrm{nec}} \leftarrow \psi_{\mathrm{nec}} \wedge \varphi_i$
17: return $\wedge_i \varphi_i$
```

3.3 Methodology

In this section, we introduce the ordering functions. Our aim is to find an *ordering function* that makes QE computation efficient, and to develop a method to obtain such an ordering function systematically by using machine learning. Our ordering functions can be divided into three types – heuristics, random, and machine learning. The way SVMs are utilized to define ordering functions is also described.

3.3.1 Ordering Functions

We experimented with the sorting determined by the following ordering functions on formulas:

Heuristics. The measure described below is calculated from the formula, and the order of QE computation for subformulas was determined by comparing the measure. We identified 10 measures for heuristics:

ADG [12]. The measure is a weighted sum of 1, 2, 3, 19, 27, 45 in Table 3.1.

nvar, npoly, sotd, msotd, mtdeg, mdeg, and mterm. The measures are 1, 2, 3, 7, 8, 9, and 10 in Table 3.1 respectively.

nvar_rev and npoly_rev. The measures are 1 and 2 in Table 3.1 respectively. These ordering functions give the reversed orders of 'nvar' and 'npoly' respectively, and were designed to investigate the worst cases.

Random. Each subformula is assigned random priority.

Machine Learning. We use an SVM to calculate the *decision value*, which is described in subsection 3.3.2, of a formula using the model obtained by training in advance. The formula which has the greater decision value is the smaller formula.

As the above configuration suggests, the ordering functions by heuristics and machine learning are similar in the way that both search for the best combination of features shown in Table 3.1 to obtain efficient sorting in QE computation.

3.3.2 Features and Labeling for Machine Learning

Features. In order to perform machine learning over formulas with SVM, we need to characterize a formula by a fixed length *feature vector* of numerical values. Each value of the vector is supposed to capture some feature of the formula. We chose 58 features shown in Table 3.1 to construct such a vector for a given formula. The features are taken from other studies [6, 2, 9] and derived from the conditions of special QE algorithms. While there are apparently overlapped features that are expressed with other features such as feature 4 expressed with feature 3 divided by feature 2, these dependent features are selected in the aim of emphasizing the nonlinear relation among the defining features. Although machine learning technique basically evaluates all features and their relations rather flatly, one way to make SVMs sensitive to a particular relation among features is adding a new dependent feature that perceive the target relation well. Another way to modify SVMs in focused features and their relations is designing an appropriate kernel function, but we decided to stick to the traditional radial basis function (RBF) kernel for the simplicity of implementation required for experiment. We also prepared diminished features of size 43 by omitting the rows 4, 14, 15, 16, and 18 in Table 3.1.

Labeling. We also need training samples labeled with either $+1$ or -1 to train an SVM. In our experiment, a feature vector in training samples was labeled $+1$ if the corresponding QE problem was solved in N seconds, and -1 otherwise. We experimented with N varying 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 30 and 55. Among these thresholds, 55 seconds is the time that separates the least 90 percent from the top 10 percent of the execution times.

Use of SVMs for Ordering Function. We used SVMs to learn binary classification problems, which is the most elementary function of SVMs. There are two steps to make binary classification using SVMs. First, we train a model with a set of training samples which were vectors labeled either $+1$ (positive) or -1 (negative). Second, we request an SVM to predict if a subformula is likely to belong to the positive class or the negative class.

In the first step, an SVM calculates a hyperplane that divides positive and negative training samples in some vector space. In the second step, we can obtain a decision value, which is a kind of distance from a new sample to the dividing hyperplane. Decision values can be considered as a measure of the confidence in a correct prediction, and so the formula with the largest decision value was solved first. Fig. 3.1 illustrates the procedure to make sorting with an ordering function determined using an SVM.

In order to configure ordering functions, we trained 17 models in total by changing labels on training data with varying threshold time, parameter selection of SVMs, and feature selection.

3.4 Computational Experiments

In this section, computational experiments performed are described. QE computations were made using the Maple-package SyNRAC [13]. For the machine learning experiment, we used LIBSVM [4], which is an implementation of an SVM written in C++. We ran all the computational experiments on a computer with an Intel(R) Xeon(R) CPU E7-4870 2.40GHz and 1007 GB of memory.

Table 3.1: Identified Features of a Formula for SVM Learning. The features other than 55-58 are calculated for three types of variables: all, quantified, and free in order. We use the following notations to define features of a formula φ: $\mathsf{Var}(\varphi)$ is the set of all variables, quantified variables, or free variables appearing in φ, $\mathsf{Poly}(\varphi)$ is the set of all polynomials p appearing in φ such that $\deg_v(p) > 0$ for some $v \in \mathsf{Var}(\varphi)$, and $\mathsf{Term}(p)$ is the set of all terms appearing in a polynomial p.

Feature number	Label	Description
1, 19, 37	nvar	Number of variables
2, 20, 38	npoly	Number of polynomials
3, 21, 39	sotd	Sum of total degrees ($\sum_{p\in\mathsf{Poly}(\varphi)}\sum_{t\in\mathsf{Term}(p)}\sum_{v\in\mathsf{Var}(\varphi)}\deg_v(t)$)
4, 22, 40	asotd	Average sum of total degrees w.r.t. npoly (sotd/npoly)
5, 23, 41	atdeg	Average total degree w.r.t. npoly ($\sum_{p\in\mathsf{Poly}(\varphi)}\max_{t\in\mathsf{Term}(p)}(\sum_{v\in\mathsf{Var}(\varphi)}\deg_v(t))/\text{npoly}$)
6, 24, 42	aterm	Average number of terms w.r.t. npoly ($\sum_{p\in\mathsf{Poly}(\varphi)}\sum_{t\in\mathsf{Term}(p)} 1/\text{npoly}$)
7, 25, 43	msotd	Maximum sum of total degrees ($\max_{p\in\mathsf{Poly}(\varphi)}\sum_{t\in\mathsf{Term}(p)}\sum_{v\in\mathsf{Var}(\varphi)}\deg_v(t)$)
8, 26, 44	mtdeg	Maximum total degree ($\max_{p\in\mathsf{Poly}(\varphi)}\max_{t\in\mathsf{Term}(p)}\sum_{v\in\mathsf{Var}(\varphi)}\deg_v(t)$)
9, 27, 45	mdeg	Maximum degree ($\max_{p\in\mathsf{Poly}(\varphi)}\max_{v\in\mathsf{Var}(\varphi)}\deg_v(p)$)
10, 28, 46	mterm	Maximum number of terms ($\max_{p\in\mathsf{Poly}(\varphi)}\sum_{t\in\mathsf{Term}(p)} 1$)
11, 29, 47	ndeg1	Number of polynomials with degree 1 ($\sum_{p\in\mathsf{Poly}(\varphi),\max_{t\in\mathsf{Term}(p)}\sum_{v\in\mathsf{Var}(\varphi)}\deg_v(t)=1} 1$)
12, 30, 48	ndeg2	Number of polynomials with degree 2
13, 31, 49	ndeg3	Number of polynomials with degree 3
14, 32, 50	rdeg1	Ratio of ndeg1 and npoly (ndeg1/npoly)
15, 33, 51	rdeg2	Ratio of ndeg2 and npoly (ndeg2/npoly)
16, 34, 52	rdeg3	Ratio of ndeg3 and npoly (ndeg3/npoly)
17, 35, 53	mcoef	Maximum absolute value of coefficients ($\max_{p\in\mathsf{Poly}(\varphi)}\max_{c\in\mathsf{Coeff}(p)}\lvert c\rvert$)
18, 36, 54	acoef	Average absolute value of coefficients w.r.t. npoly (mcoef/npoly)
55		Ratio of the number of the symbol '$=$' and npoly
56		Ratio of the number of the symbol '$\neq$' and npoly
57		Ratio of the number of the symbol '$<$' and npoly
58		Ratio of the number of the symbol '$\leq$' and npoly

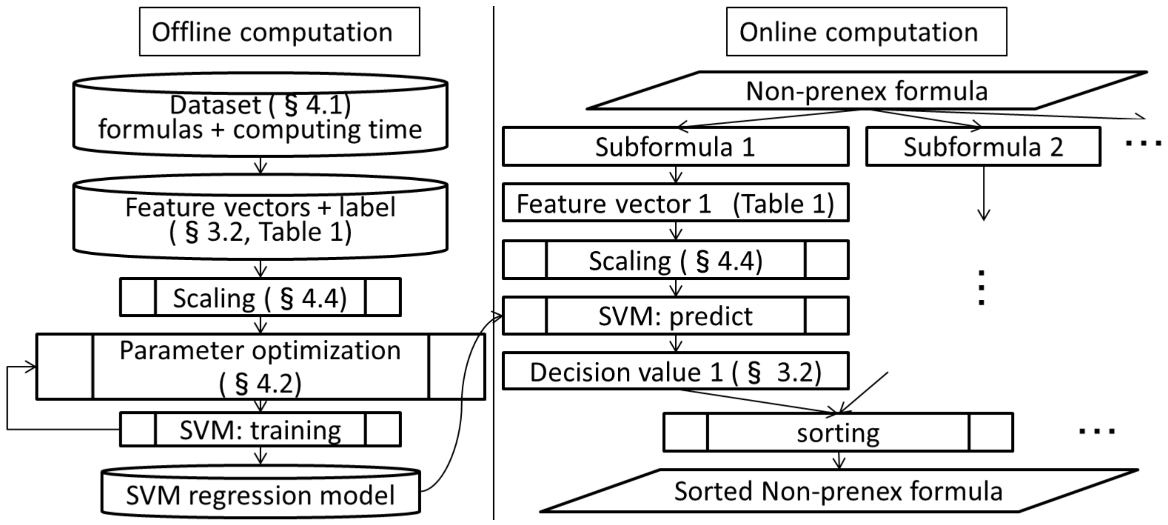

Offline computation
Dataset (§ 4.1) formulas + computing time
Feature vectors + label (§ 3.2, Table 1)
Scaling (§ 4.4)
Parameter optimization (§ 4.2)
SVM: training
SVM regression model
Online computation
Non-prenex formula
Subformula 1
Subformula 2
Feature vector 1 (Table 1)
Scaling (§ 4.4)
SVM: predict
Decision value 1 (§ 3.2)
sorting
Sorted Non-prenex formula

Figure 3.1: Flow of Sorting by Using SVMs

Table 3.2: Number of Formulas for which QE Computation Took Less than N Seconds. These execution times were measured using the 'ADG' ordering function.

N	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
#	433	846	1055	1261	1379	1476	1627	1694	1725
N	1.0	2.0	3.0	4.0	5.0	6.0	7.0	8.0	9.0
#	1752	1884	1937	1958	1977	1990	1999	2005	2013
N	10	20	30	100	150	200	300	500	600
#	2015	2040	2057	2096	2100	2102	2109	2113	2116

3.4.1 Datasets

We have extracted 2,306 first-order formulas that were generated when the automated natural language math problem solving system (developed by the Todai robot project) solved the problems, which were also collected by the project. The problems were originally taken from three sources: "Chart-shiki", Japanese university entrance exams, and International Mathematical Olympiads. "Chart-shiki" is a popular problem book series for preparing university entrance examinations. The math problem solving system accepts problems in algebra, linear algebra, geometry, pre-calculus, calculus, combinatorics, Peano arithmetic, and higher order formula as its inputs. Among these problems, the problems which can be expressed in the language of real closed field were utilized to generate our dataset, where such kind of problems are mainly in algebra, linear algebra, and geometry.

For each formula, we executed QE computation with SyNRAC and recorded the execution time, where we set the limit of timeout to 600 seconds. Table 3.2 shows the distribution of execution times for the 2,116 formulas for which QE computation stopped in less than 600 seconds. There are 187 formulas which took more than 600 seconds to finish QE computation. The other 4 problems caused error while computing QE with SyNRAC, which were omitted from the training set for machine learning. The size 2,302 of the training set is large enough for the application of machine learning with SVMs. For example, three cases of application of SVMs are introduced in [8], and the sizes of the training data are 3,089, 391, 1,243 while the numbers of features are 4, 20, 21 respectively.

Parameter Optimization of SVMs We selected the RBF kernel for our machine learning, which is a popular choice when the number of features is smaller than the number of training set. In our case, the number of features was 58 and the number of training set was 2,302.

We used a python script "grid.py," which is distributed with LIBSVM, for grid search over parameters (C, γ) where C is the cost parameter and γ is the parameter of the RBF kernel. The script "grid.py" performs cross validation to estimate the *accuracy* of each parameter combination in the

Table 3.3: An Example of QE Execution Times and Feature Vectors of Formulas. Features with an asterisk are omitted in machine learning for '<9_dim'.

Feature no.	time(sec)	1	2	3	4*	5	6	7	8	9	10	11	12	13
φ_1	0.078	1	1	1	1	1	2	1	1	1	2	1	0	0
φ_2	0.037	1	1	1	1	1	1	1	1	1	1	1	0	0
φ_3	1551.989	5	8	136	17	5	6.5	25	6	3	8	0	0	0
Feature no.	14*	15*	16*	17	18*	19	20	21	22*	23	24	25	26	27	28
φ_1	1	0	0	1	1	1	1	1	1	1	2	1	1	1	2
φ_2	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1
φ_3	0	0	0	1	1	4	8	125	15.6	4.6	6.5	21	5	3	8
Feature no.	29	30	31	32*	33*	34*	35	36*	37	38	39	40*	41	42	43
φ_1	1	0	0	1	0	0	1	1	0	0	0	0	0	0	0
φ_2	1	0	0	1	0	0	1	1	0	0	0	0	0	0	0
φ_3	0	0	0	0	0	0	1	1	1	3	11	3.7	1	7	5
Feature no.	44	45	46	47	48	49	50*	51*	52*	53	54*	55	56	57	58
φ_1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
φ_2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
φ_3	1	1	8	3	0	0	1	0	0	1	1	1	0	0	0

specified range, and outputs the parameter combination that recorded the highest accuracy. Here, accuracy is defined as the number of correct predictions divided by the total number of predictions. The script "grid.py" was executed twice to obtain the optimal parameters. We specified the options of "grid.py" `-log2c`, `-log2g`, `-w1` and `-w-1`. The options `-log2c` and `-log2g` specify the range and the incremental width of C and γ respectively. The values of the options `-log2c` and `-log2g` were set to -5, 15, 2 and 3, -15, -2 respectively in the first execution, which are default, and in the second time the values were set to $C_0 - 1$, $C_0 + 1$, 0.2 and $\gamma_0 - 1$, $\gamma_0 + 1$, 0.2 respectively, where (C_0, γ_0) is the output of the first execution. The option `-w1` and `-w-1` are for handling unbalance in training data, and were set to the ratio of the number of training vector labeled $+1$ and -1.

An Illustrative Example We again consider the non-prenex formula (3.1) in section 3.2. What we do here is to perform QE for φ using an ordering function configured with an SVM. QE for φ can be divided into 3 QE subproblems for φ_1, for φ_2, and for φ_3. In this case, φ_1 and φ_2 are trivially true, and so φ is true. The order of processing subproblems are determined by decision values calculated with LIBSVM. Table 3.3 shows the QE computation time and the features of φ_1, φ_2, and φ_3. Table 3.4 shows the decision value and the order of QE subproblems. The QE computation for φ with ordering functions '<2', '<3', and '<7' solved the subproblem φ_3 first, and did not finish in 600 seconds. On the other hand, the QE computation with the other ordering functions started to solve the subproblems for φ_1 or φ_2, and finished computation in less than 1 second.

3.4.2 Experimental Results

We made two types of experiments. Experiment 1 investigated the efficiency of QE computation when the prepared ordering functions were used. Experiment 2 justifies Experiment 1 in the point that the apparently common training and test set did not cause to overestimate the result of ordering functions based on machine learning. The amount of computation is large as it took more than 40 hours to make a series of QE computation with a single ordering function.

Experiment 1. We prepared 28 ordering functions in total. Among those, 10 functions were heuristics and 1 function was random, which are described in section 3.3.1. The rest 17 functions which were configured using machine learning consist of 13 functions named '<1', . . . , '<55' that were designed by simply following the setting given in subsections 3.3.2 and 3.4.1, three functions named '<4_opt', '<8_opt', and '<30_opt' of which parameters were chosen by assuming the output of the first step (C_0, γ_0) in subsection 3.4.1 are $(10, -1)$, and 1 function named '<9_dim' where machine learning was performed on the diminished features described in subsection 3.3.2. After extracting feature vectors from 2,302 training samples, each feature was scaled to be between 0 and 1.

Table 3.4: Example of Decision Values

	φ_1	φ_2	φ_3	Order
<1	−4.143236	−4.126463	−7.026860	$\varphi_2 \to \varphi_1 \to \varphi_3$
<2	4.924164	4.659644	6.950127	$\varphi_3 \to \varphi_1 \to \varphi_2$
<3	2.092471	2.145601	4.441247	$\varphi_3 \to \varphi_2 \to \varphi_1$
<4	1.428065	1.430775	−0.009359	$\varphi_2 \to \varphi_1 \to \varphi_3$
<4_opt	1.891265	1.887521	−2.485134	$\varphi_1 \to \varphi_2 \to \varphi_3$
<5	1.981705	2.103514	0.978644	$\varphi_2 \to \varphi_1 \to \varphi_3$
<6	2.063262	2.127833	−2.001929	$\varphi_2 \to \varphi_1 \to \varphi_3$
<7	1.462456	1.469122	3.834284	$\varphi_3 \to \varphi_2 \to \varphi_1$
<8	4.010731	4.015434	0.297530	$\varphi_2 \to \varphi_1 \to \varphi_3$
<8_opt	23.705870	23.860115	−0.525661	$\varphi_2 \to \varphi_1 \to \varphi_3$
<9	25.655055	25.840633	−1.913830	$\varphi_2 \to \varphi_1 \to \varphi_3$
<9_dim	1.761298	1.933184	−0.091157	$\varphi_2 \to \varphi_1 \to \varphi_3$
<10	36.241697	36.340194	−3.770675	$\varphi_2 \to \varphi_1 \to \varphi_3$
<20	38.722068	38.077265	6.001291	$\varphi_1 \to \varphi_2 \to \varphi_3$
<30	6.523585	6.529640	0.220255	$\varphi_2 \to \varphi_1 \to \varphi_3$
<30_opt	21.553545	21.639367	1.056053	$\varphi_2 \to \varphi_1 \to \varphi_3$
<55	21.408608	21.551922	−1.358177	$\varphi_2 \to \varphi_1 \to \varphi_3$

As baseline, we ran the experiment with randomized order of the subformulas. We repeated it with fourteen different seeds of the random number generator. We will show the average computation time over the fourteen runs as well as the best and the worst cases in the fourteen runs chosen for each problem.

We performed QE for the 2,306 formulas with the 28 prepared ordering functions. Table 3.5 shows the statistics on the results. The columns "solved", "error", and "timeout" give the number of formulas for which QE computation finished in less than 600 seconds, for which QE caused error, and for which computation took more than 600 seconds respectively. The errors occurred in the experiments were due to bugs in Maple and SyNRAC, or computer memory shortage. The columns "average", "average(log)", and "median" state respectively the average execution time of solved QE problems, the average logarithm of execution times of solved QE problems, and the median of execution times where timed out problems assumed to take infinite seconds to stop. The values of "average" and "median" are given in units of seconds. The row 'rand_aver' shows the results averaged over the fourteen runs with random subformula order. 'rand_best' (resp. 'rand_worst') shows the sum of the best (resp. worst) result in the fourteen runs for each problem.

An overall argument on Table 3.5. Here, we especially value the number of solved problems in less than 600 seconds, because the property if QE computation for a formula finishes in a fixed time or not is critical in practical use rather than whether a computation takes 0.1 or 1 seconds to finish. From this point of view, the results of 'nvar_rev', 'npoly_rev', 'random', '<1', and '<2' are considerably bad compared to the results of the other orderings. The reason '<1' and '<2' behaved poorly is presumed that the learning algorithm failed to set large weight on the hardest problems in the negative class since there were relatively easy problems mixed in the negative class.

An effect of parameter selection. We also observe that the result '<4' and '<8' are relatively inferior to the orderings of close condition. Table 3.6 shows the parameters used to train models. The rows "$\log_2 C$" and "$\log_2 \gamma$" list the logarithm to the base 2 of cost and gamma parameters used respectively, and the row "nSV" lists the number of support vectors. The parameters of '<4' and '<8' lie apart from those of '<3', '<5', '<6', '<7', '<9', '<10', and '<55'. Besides, the numbers of support vectors of '<4' and '<8' are larger than expected from other values. This note suggests that parameter selection affects the performance, and actually '<4_opt' and '<8_opt', of which parameters were similar to '<9' etc., solves more problems in time. We used accuracy as the target of the hyperparameter optimization in this experiment, but it would be effective to replace accuracy with other known measures such as the Matthews correlation coefficient, which is reported to have performed well in the application of SVMs to the problem of the optimal projection order for CAD [9].

Minor contribution of omitted features. Moreover, we see '<9' is just slightly better when compared with '<9_dim'. This means the omitted 15 features did not contribute much. The analysis what feature contributes much is left to further study.

Significance of the achieved result. Table 3.7 illustrates the contrast in QE computation time among 4 heuristics 'ADG', 'nvar', 'npoly', and 'sotd', and 5 ordering functions using machine learning '<1', '<3', '<5', '<7', and '<9'. The column "id" shows the identification numbers of formulas, and each row shows the time consumed to make QE for the corresponding formula. The row "total" lists the number of the formulas for which QE computation finished in 600 seconds shown in Table 3.7. The example in subsection 3.4.1 corresponds to the problem with "id" 2,051 in Table 3.7. The problems which are not shown in Table 3.7 were either solved in 600 seconds or timed out regardless of the choice of ordering functions.

Notice it was only less than 40 problems out of 2,306 that the results of "solved" and "timeout" changed by ordering functions. If we consider the results of only a small number of problems changed, it is valuable to have achieved the ordering functions that solved more problems by machine learning such as '<9' compared with the ordering functions by heuristics such as 'ADG' despite the modest difference between them. From another view point of the computation time, '<9' can be evaluated to perform better than 'ADG' again. We can observe in Table 3.7 that it took relatively long time to perform QE for the problems of which results altered by ordering functions. Hence, solving more problems tends to cause larger average computation time since the averages are taken over solved problems. '<9' compared to 'ADG' recorded the average computation time of the same level, less average of logarithm of the computation time, and smaller median of the computation time. Thus '<9' can be stated to have achieved better performance in overall speed than 'ADG'. The fact that effective ordering functions were obtained by machine learning implies machine learning can save a lot of time because searching for the best application of features by trials and errors consumes long time, where it takes about 40 hours to make QE computation for 2,306 problems in a single experiment. On the other hand, we also found that no single ordering function can be regarded as the best. This shows that there is still room for study in sort ordering for efficient QE for non-prenex formulas.

Experiment 2. In order to justify the result of Experiment 1, it is necessary to estimate the effect of the overlap of the training set and the test set. We achieved this by taking another experiment of cross validation. The 2,302 training samples were shuffled and divided into 6 groups, and 6 models were trained using 5 of the 6 groups as training data. For the machine learning, we set the threshold for labeling to be 9 seconds and parameters $(\log_2 C, \log_2 \gamma)$ to be $(10, -1)$ which were the same parameters for the case '<9'. QE computation with the ordering function using the model obtained was executed for each formula in the remaining group.

Table 3.8 shows the statistics on the results of the cross validation experiment. The row '<9' is already shown in Table 3.5, and is printed again for reference.

First, all the 4 problems which were excluded from cross validation because of the error also caused error or timed out with the ordering function '<9'. Therefore, the number of solved problems with 'cross_validation' was smaller by 3 than that of '<9' with the erroneous input excluded. From these results, the effect of ill generalization caused by the common training and test set is estimated to be quite minor in this study.

The purpose of machine learning is to obtain a model that estimates the general characteristics by analyzing given training data. In order to attain this goal, the characteristics that only appear in the training set need to be abstracted. We usually verify that the abstraction is properly made by

Table 3.5: Statistics on Computation Time for QE with Varying Ordering Functions

	solved	error	timeout	average(sec)	average(log)	median(sec)
ADG	2116	3	187	4.770	-1.038	0.337
nvar	2115	4	187	4.741	-0.993	0.358
npoly	2114	4	188	4.594	-1.134	0.267
sotd	2114	4	188	4.269	-1.217	0.234
msotd	2114	4	188	4.240	-1.212	0.233
mtdeg	2114	4	188	4.172	-1.189	0.262
mdeg	2115	3	188	4.005	-1.230	0.222
mterm	2114	4	188	4.982	-1.150	0.256
nvar_rev	2097	2	207	4.857	-1.129	0.260
npoly_rev	2098	2	206	4.560	-1.119	0.268
rand_aver	2104.93	3.64	197.43	4.764	-1.182	0.241
rand_best	2121	2	183	4.569	-1.389	0.190
rand_worst	2087	2	217	4.531	-0.890	0.356
<1	2099	4	203	5.141	-0.934	0.300
<2	2102	3	201	5.320	-0.995	0.332
<3	2115	3	188	4.673	-0.928	0.360
<4	2112	4	190	4.732	-0.959	0.384
<4_opt	2118	2	186	5.020	-1.085	0.279
<5	2117	3	186	5.153	-0.928	0.392
<6	2115	2	189	4.540	-1.155	0.261
<7	2115	2	189	4.508	-1.178	0.244
<8	2113	3	190	4.806	-1.105	0.267
<8_opt	2117	2	187	4.714	-1.142	0.245
<9	2118	2	186	4.750	-1.165	0.239
<9_dim	2116	4	186	4.718	-0.984	0.363
<10	2117	2	187	4.578	-0.956	0.362
<20	2116	3	187	4.451	-1.200	0.235
<30	2114	2	190	4.886	-1.173	0.245
<30_opt	2115	3	188	4.628	-1.199	0.234
<55	2114	3	189	4.808	-0.993	0.351

Table 3.6: Employed Parameters of Machine Learning and the Number of Support Vectors

	<1	<2	<3	<4	<5	<6	<7	<8	<9	<10	<20	<30	<55
$\log_2 C$	12	15.4	9.0	5.4	9.6	11.6	10.8	3.4	10.0	10.8	15.2	7.2	11.8
$\log_2 \gamma$	-3.6	-4.8	-3.0	0.8	-1.8	-2.2	-3.0	1.0	-1.0	-0.8	-2.2	0.8	-1.4
nSV	538	427	428	462	374	354	390	492	324	297	276	350	274

	<4_opt	<8_opt	<30_opt	<9_dim
$\log_2 C$	10.4	9.2	10.6	13.4
$\log_2 \gamma$	-1.2	-0.8	-0.6	0
nSV	359	325	285	292

Table 3.7: Timing Data (in seconds) for Order Functions. TO expresses the computation time was greater than 600 seconds, and ERR expresses the computation caused error.

id	ADG	nvar	npoly	sotd	<1	<3	<5	<7	<9
2051	0.12	0.13	0.12	0.12	0.19	TO	0.16	TO	0.15
1142	0.57	0.61	0.64	0.60	TO	0.63	0.64	0.65	0.63
1944	TO	TO	10.69	16.75	TO	6.92	8.24	17.40	4.61
889	5.01	5.74	5.52	5.56	TO	5.64	5.62	6.00	5.98
272	2.98	3.55	3.26	3.26	ERR	5.32	5.74	6.21	6.76
1024	8.15	11.87	10.20	10.22	TO	10.31	10.28	10.54	10.88
993	TO	9.74	TO	TO	TO	10.44	10.05	10.92	13.45
999	TO	TO	TO	TO	12.54	13.33	13.35	13.30	13.57
1528	22.28	25.46	24.57	23.90	TO	25.06	23.27	23.60	23.24
2167	28.49	114.20	21.46	21.31	TO	48.71	27.14	24.27	27.32
2294	37.91	TO	31.60	31.80	TO	48.95	28.88	27.41	31.93
1088	58.49	TO	TO	TO	TO	74.81	32.61	TO	32.43
1039	31.24	32.11	32.38	31.65	32.27	32.46	32.44	33.46	32.69
1112	40.31	39.73	39.03	40.14	120.55	40.05	39.67	TO	39.40
1234	45.87	49.48	41.81	42.72	14.92	39.83	39.78	15.17	44.14
32	42.21	43.80	292.48	44.66	TO	41.92	44.04	43.76	44.25
1239	39.87	48.73	43.35	44.19	13.44	44.93	40.2	13.79	44.94
2045	67.18	60.06	61.08	60.17	TO	71.15	TO	60.30	TO
1101	61.42	61.16	63.67	61.85	TO	62.39	61.29	60.74	62.39
2168	62.42	62.72	64.16	61.87	TO	60.44	153.39	62.74	74.04
33	69.83	75.44	76.99	76.54	TO	69.87	68.68	76.28	75.79
1040	80.14	87.35	107.34	87.68	TO	TO	89.18	89.93	87.61
28	199.02	208.30	TO	TO	TO	TO	202.83	210.17	209.45
151	239.64	218.08	244.91	246.53	TO	243.18	236.42	243.94	248.93
1125	TO	TO	TO	TO	TO	252.36	252.79	296.94	250.25
1092	266.09	153.79	TO	149.94	323.97	TO	TO	TO	275.69
5	279.42	285.78	295.58	289.15	TO	283.14	277.4	291.48	292.78
17	341.99	293.81	230.59	345.67	272.41	352.92	307.32	242.58	339.28
112	TO	TO	TO	TO	409.60	TO	TO	TO	TO
610	437.12	455.59	456.30	TO	393.91	TO	TO	TO	TO
1091	523.86	528.97	476.57	466.62	TO	475.28	521.76	363.77	472.13
2216	525.31	482.94	485.39	481.33	TO	573.73	566.82	482.50	487.02
150	513.11	592.43	589.52	596.24	586.34	571.58	548.01	586.74	586.56
total	28	27	26	26	11	27	29	27	30

examining if the model obtained from the training set is also valid on a validation set which contains different data from the training set.

The problem in the overlap of the training set and the test set in this study lies in the possibility that the results in Table 3.5 was obtained by making use of the characteristics special only to the 2,302 training data in solving the 2,306 QE problems, and the same method is not valid on new QE problems any more.

If this case happened, the result of 'cross_validation' would be considerably bad, because characteristics special only to 5 groups of the training set should not be valid on the remaining group.

On the other hand, the difference between the number of the solved problems with the order '<9' and 'cross_validation' were 3, and is possible to arise from the parameter selection. This is because we can conclude that the performance of the ordering function using machine learning is not expected to be local to the 2,306 QE problems used. The above argument justifies that in Experiment 1, the result of the ordering functions with machine learning are not overestimated and valid.

3.5 Conclusion

We showed that ordering of subformulas affects the performance of QE computation for non-prenex formulas. The ordering determined by heuristics performed well compared to random baseline, but the ordering by machine learning performed still better when the training vectors were appropriately labeled. These results suggest that machine learning can save a lot of effort to design heuristics by trials and errors. Meanwhile, we also observed that improper labeling and parameter selection can cause poor performance.

Since our study focused on only one particular dataset of math problems, additional validation

Table 3.8: Results of Experiment 2

	solved	error	timeout	average(sec)	average(log)	median(sec)
<9	2118	2	186	4.750	-1.165	0.239
cross_validation	2115	0	187	4.378	-1.109	0.273
group 1	356	0	28	5.509	-1.189	0.217
group 2	356	0	28	5.454	-1.067	0.281
group 3	346	0	38	5.487	-0.937	0.321
group 4	351	0	33	2.637	-1.364	0.214
group 5	350	0	33	4.868	-0.729	0.406
group 6	356	0	27	2.329	-1.318	0.188

on other datasets is necessary for our method to be more convincing. Also, further study should be made to make full use of machine learning extracting valuable patterns in formulas. Our experiment is suggestive to proceed application of machine learning to heuristic portions in computer algebra algorithms.

Chapter 4

Investigation on difficulty levels of math problems using an automated math solving system

This study is based on [24] and is a joint work with T. Matsuzaki and N. H. Arai.

4.1 Introduction

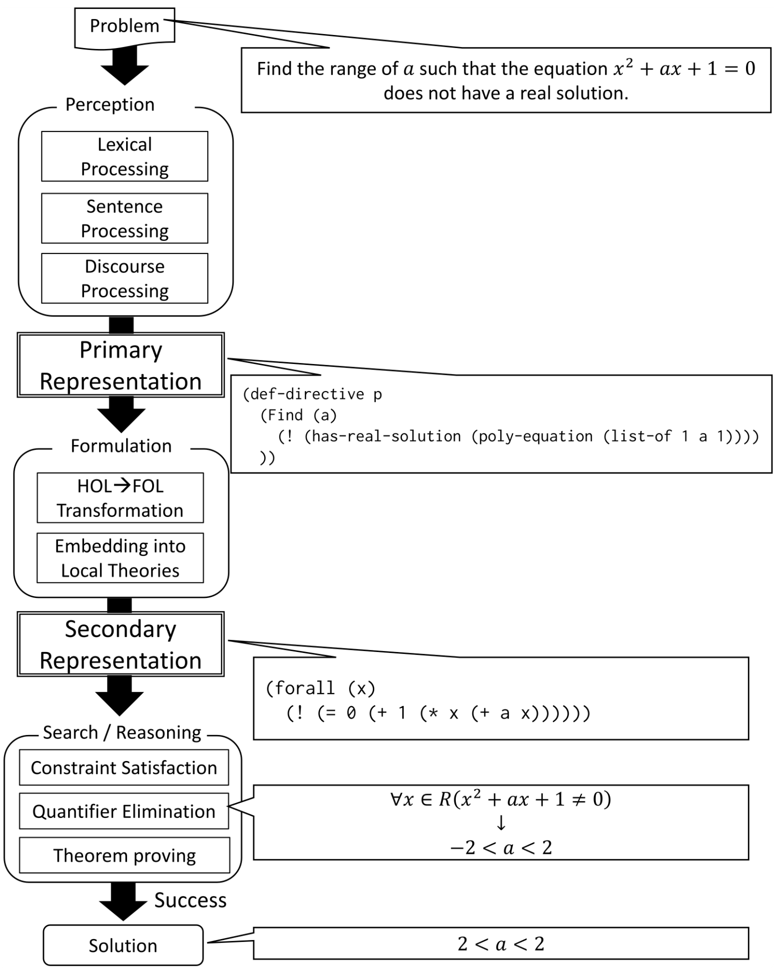
Some math problems are much more difficult than others to solve although they do not require higher level of mathematical knowledge or techniques. Nine dot problem and mutilated draughtboard problem are examples of such problems. The question what determines the difficulty arises.

In classical information-processing models, the difficulty of a given problem is explained by its computational complexity; the cost of search [17, 20]. In contrast, Gestalts explain the phenomena by the term *insights* [10, 26]. A problem is called an *insight problem* when it requires the recognition or restructuring of a key feature of the problem (representation change) to be solved.

One of the major criticisms to classical information-processing account is that it has no mechanism to implement representation change since problem solving is understood as search within a well-defined problem space [27]. In fact, search space explosion is almost always inevitable if one tries to enlarge the framework (theory) of the problem to implement representation change. For example, it is a well-known fact that almost all the mathematical activities can be formalized in *Zermelo-Fraenkel's set theory* (ZF), thus representation change can be formalized as proof search in ZF. However, we cannot hope the search is terminated in realistic time since search space of ZF is too vast. On the other hand, representation change account has also some downsides. It does not provide any process model, and the analysis remains qualitative but not quantitative [20].

In this study, we present a new information-processing model which enables us to include representation change account. Based on the flow chart of insight problem solving [27], we first specify the perceptual process as translation of a given problem to a formula in higher-order logic (HOL). We defined a language that contains the symbols representing the notions likely to be used in the math problems of high-school level so that the translation is kept as straightforward as possible. In other words, we design perceptual process to require no insight, but it rather corresponds to natural language and image processing. This is worth mentioning since the inputs of existing information-processing account are usually not obtainable without insight regardless in which theories the problems are represented [25, 5, 18]. The obtained HOL formula is considered to be the *primary* problem representation. Usually there are many possible interpretations of the primary problem representation in different mathematical local theories. For example, mutilated draughtboard problem can be embedded not only to propositional logic but also to Peano Arithmetic and Presburger Arithmetic. The possible interpretations of the primary representation are called *secondary* representation.

We take theory of real closed field (RCF) as an example of local theories, and implement interpretation process from the primary to secondary representation. We adopt quantifier elimination (QE)



```
(def-directive p
  (Find (a)
    (! (has-real-solution (poly-equation (list-of 1 a 1))))
  ))
```

```
(forall (x)
  (! (= 0 (+ 1 (* x (+ a x))))))
```

Figure 4.1: End-to-end problem solving model

algorithm as an approximate process of searching for solutions [14], and develop a prototype system to solve geometry and introductory calculus problems.

We manually formalize more than four hundred math problems from three different sources in our representation language as a benchmark. The problems are translated so that they can be obtainable automatically from the problem text using state-of-the-art natural language processing theories and techniques [16, 28, 35]. One source of the problems is exercise books, the second is university entrance examinations, and the last is International Mathematics Olympiad (IMO). Though all the problems require mathematical knowledge and techniques no more than high-school level, they have different level of difficulties. We assume naturally that more insight problems can be found in IMO than in other two because IMO problems are known to be solvable by only few mathematically talented students.

4.2 Model for solving math problem

Fig. 4.1 presents an overview of our problem solving model. It consists of three modules. The perception module translates a problem to a primary representation expressed in HOL by language processing. The formulation module transforms the primary representation to the secondary representation, which itself is another HOL formula, that is interpretable in a local theory such as RCF. Finally, the search/reasoning module works on the secondary representation. The rest of this section provides more details on the three modules, while we especially look closely into the second formulation module here. For a more specific description, refer to [24, 23]

Perception module The perception module is organized along a hierarchy in natural language: words, sentences, and discourses (*i.e.* sequences of sentences). The lexical processing unit identifies the parts-of-speech and other syntactic properties of the words and math formulas in a problem. The sentence processing unit translates each sentence in the problem to a formal representation. The discourse processing unit combines the sentence-level semantic representations into a single formula.

Solution search/reasoning In the current implementation, we adopt a QE algorithm for RCF [13] as an example for solution search. The architecture of the model however allows to use several reasoning algorithms in the solution search, such as automated theorem proving and combinatorial search algorithms. In such a setting, a proper reasoning algorithm shall be selected depending on the secondary representation produced in the formulation module module.

4.2.1 Formulation module

The formulation module deals with symbolic manipulations of HOL formulas. This module receives a primary problem representation and transforms it to a secondary representation. While both of the problem representations are HOL formulas, a primary representation typically includes higher order terms and redundant expressions, yet a secondary representation has only terms that can be handled in a solver implemented in the solution search/reasoning module. The process consists of two steps. One is the transformation of the higher-order terms and predicates to first-order ones. The other is the transformation of formulas to those interpretable in a local theory, such as RCF and propositional logic. The former is usually a routine procedure for a person with necessary math knowledge. The latter requires a target theory to be chosen beforehand. In our current prototype implementation, a mechanism to choose an appropriate local theory has not yet developed. In the experiments, we chose RCF as the target local theory and confirmed that many pre-university math problems can be mechanically reformulated in RCF. The rest of this subsection summarizes the implementation of the two reformulation steps.

Higher-order to first-order transformation The primary representation often includes higher-order elements (λ-abstractions), which denote functions (*e.g.* $\lambda x.x^2$) and conditions (*e.g.* $\lambda y.(|y| < 1)$). They are necessary to translate the natural language expressions such as "the function which maps $x \in \mathbf{R}$ to x^2" and "The absolute value of y is less than 1. The same condition also applies to x." We eliminate such higher-order elements to obtain a first-order formula. In the current implementation, this is done by iteratively applying a handful of transformation rules such as β-reduction and variable elimination by substitution ($\exists x(x = \alpha \wedge \phi(x)) \leftrightarrow \phi(\alpha)$).

Reformulation into RCF In the prototype implementation, a primary representation is rewritten into the language of RCF. The first-order language of RCF consists of polynomial equations and inequalities, logical connectives and quantifiers. We developed a set of axioms in HOL that define various math concepts such as:

$$\forall S \forall m(\text{minimum}(S, m) \leftrightarrow (m \in S \wedge (\forall m' \in S\, m \leq m'))).$$

The primary representation is iteratively rewritten with these axioms until the resulting equivalent formula is in the first-order language of RCF. There is no theoretical guarantee that such a formula is eventually found even when it exists. We empirically examined how often it succeeds in the experiments.

4.2.2 Representation language

A problem is represented by a directive in an s-expression, which we call an HOL formula. Fig. 4.2 shows an example of an HOL formula. three types of directives are prepared to be used in the HOL formulas:

Problem

Find the maximum value of $ax + by$, where x, y, a and b are reals that satisfy $x^2 + y^2 = 1$ and $a^2 + 4b^2 = 4$.

```
(def-directive p
  (Find (max)
    (exists (C1 C2) (&&
      (= C1 (Lam x (PLam y (= 1 (+ (^ x 2) (^ y 2))))))
      (= C2 (Lam a (PLam b (= 4 (+ (^ a 2) (* 4 (^ b 2)))))))
      (maximum
        (set-by-def (PLam v (exists (x y a b f) (&&
          (= f (Lam x (Lam y (Lam a (Lam b
            (+ (* a x) (* b y))
          )))))
          (PLamApp (LamApp C1 x) y)
          (PLamApp (LamApp C2 a) b)
          (= v (LamApp (LamApp (LamApp (LamApp f x) y) a) b))
        ))))
        max
      )))))
```

Figure 4.2: An example of HOL formulas

· `Show`$[\varphi]$ is a proof problem to prove φ.

· `Find`$(v)[\varphi(v)]$ is a problem to find all v that satisfies $\varphi(v)$.

· `Draw`$(v)[\varphi(v)]$ is a problem to draw a geometric object v defined by $\varphi(v)$.

An HOL formula is a higher-order predicate logic (typed lambda calculus) with parametric polymorphism. An HOL formula consists of lambda expressions, logical connectives, quantifiers, and the various function and predicate symbols that are defined by the axioms we have implemented. Parametric polymorphism is employed to utilize polymorphic lists and sets in HOL formulas and enable us to define various operations on them with a transparent description of the axioms.

We identified more than fifty types such as `Bool` (truth values), `Z` (integers), `Q` (rational numbers), `R` (real numbers), `C` (complex numbers), `ListOf`(α) (polymorphic lists), `SetOf`(α) (polymorphic sets), `Point` (points in $\mathbb{R}^2$ and $\mathbb{R}^3$), `Shape` (sets of `Points`), `Equation` (equations in real domain), and so on. The types we use are redundant in the sense that we can eliminate a term of some type by using some others. For example, a term of `Equation` type can also be expressed by the expression $f(x) = 0$ with some function f of type `R` $\rightarrow$ `R`. We use an abundant set of types to realize more efficient environment in organizing the axioms and confirming the validity of the HOL formulas.

4.3 Methodology

4.3.1 Problems

We collected 461 problems taken from the three sources: exercise books (**Ex**), Japanese university entrance exams (**Univ**), and International Mathematical Olympiads (**IMO**). The **Ex** problems were sampled from a popular exercise book series. The problems in the books are marked with one to five stars in accordance with their difficulty: one to three stars signify the textbook exercise level and four and five stars signify the university entrance exam level. We sampled approximately the same number

Table 4.1: Subject areas of the benchmark problems

	Ex	Univ	IMO
Algebra	0	10	21
Linear Algebra	14	62	0
Geometry	81	65	94
Pre-calculus	0	75	0
Calculus	6	33	0
total	101	245	115

of problems from those marked with one, two, and three stars. The **Univ** problems were taken from the past entrance exams of seven top Japanese national universities. The **IMO** problems were taken from the past IMOs held in 1959 through 2014.

We examined the problems and exhaustively selected those that are possibly formulated (by humans) in the theory of RCF. The distinction between RCF and non-RCF problems was made solely on the basis of the essential mathematical content of the problems. The selected problems thus contain problems in several subject areas as shown in Table 4.1.

The problems were manually formalized in HOL. We used those manually formalized problems rather than the outputs of our implementation of the perception module because an automatic translation of a text to a logical representation inevitably includes errors due to the ambiguity in the natural language. Operators, who all majored in computer science and/or mathematics, were trained to translate the problems as faithfully as possible to the original natural language statements following the design of the perception module. In formalizing the problems, some problems are the combinations of the smaller questions. In such cases, we divide the problem into the questions and make an HOL formula for each question. We distinguish the problems and the questions in the experiment below.

4.3.2 Settings for machine learning

We performed machine learning over formulas in HOL to predict solvability of the problems by the system. The SVM library LIBSVM [4] was used for the machine learning experiments. The following paragraphs explain the feature vectors, the labeling and the training options for LIBSVM.

Features. We put 400 dimensional feature vectors to characterize a formula in HOL representing questions. The feature vectors are basically term frequency vectors, where the variables are classified and counted by their inferred types and boundedness independent of their literal names.

The representation by a frequency vector, or the bag-of-words representation, is one of the most popular representation methods for text data. Since our HOL formulas have more varieties of symbols compared with the formulas studied in traditional mathematical logic such as RCF, we can employ a representation method likely to be used in text categorization.

The dimension size of 400 is rather small as a bag-of-words representation, since in the usual text categorization the feature spaces contain more than 10,000 dimensions [15].

Labeling. We prepared two types of labeling. One is the three class labeling representing the source of the questions *i.e.* **Ex**, **Univ**, and **IMO**.

The other is a binary labeling based on the system answers obtained by the computation on each formula. First, the prototype system was run on the formulas with the time limit of 3,600 seconds for each questions. Then, those questions that the system output the right answers were labeled with $+1$, and the other questions -1.

Kernel Selection and Parameter Optimization We chose the linear kernel to train SVM models, since the number of the samples ($= 879$) is not so large compared with the size of features ($= 400$). The cost parameter C in SVMs was optimized over $\log_2 C = -5, -3, -1, 1, 3, 5, 7, 9, 11, 13,$ and 15.

Table 4.2: Overall benchmark results

Problem Source	Solved		Failed		
	Solved (%)	Time (sec) min/med/avg/max	FM (%)	TO (%)	WR (%)
Ex	75.2	1 / 4 / 19.6 / 1069	7.9	13.9	3.0
Univ	65.3	1 / 7 / 37.8 / 1061	7.3	22.9	4.5
IMO	26.1	2 / 10 / 56.4 / 513	10.4	60.0	3.5

Table 4.3: Results for **Ex** problems by number of stars

#★	Succeeded		Failed	
	Success %	Time (sec) min/med/avg/max	FM (%)	TO (%)
1	82.4 (28/34)	1 / 4 / 5.3 / 39	11.8	5.9
2	73.5 (25/34)	2 / 4 / 6.4 / 39	5.9	11.8
3	69.7 (23/33)	2 / 4 / 51.2 / 1069	6.1	24.2
4	63.2 (24/38)	2 / 6 / 35.8 / 589	10.5	23.7
5	54.3 (19/35)	3 / 10 / 197.8 / 3245	2.9	42.9

4.4 Experiments

4.4.1 Basic statistics of the problems

We executed the computation for solving the benchmark questions, setting the limit of timeout to 3,600 seconds. We ran all the computational experiments on a computer with an AMD Opteron(tm) Processor 6344 and 132 GB of memory. Table 4.2 shows the number of successfully solved problems, minimum, median, average, and maximum (wallclock) time spent on solved problems, number of failures in the reformulation of the primary HOL representation into RCF (FM), number of failures due to timeout (TO), and wrong answers (WR). Wrong answers were due to bugs in the current implementation.

Overall, the performances on the **Ex**, **Univ**, and **IMO** problems seem to well reflect the inherent differences in their difficulty levels. We conducted χ^2-test on the difference in the rates of success on them. The difference between **IMO** and other sources were statistically significant ($p < 0.01$) though that between **Ex** and **Univ** was not ($p = 0.09$).

We further examined how well the system performance correlates with the fine-grained difficulty level assessed by human experts. Table 4.3 lists the performance figures for the **Ex** problems (one to three stars) and additional problems sampled from those marked with four and five stars in the same practice books.

Table 4.4 presents several syntactic features of the benchmark problems. The figures are averaged over the problems taken from each source. It reveals that, the syntactic features of the **IMO** problems are not very different from the exercise problems in **Ex** except for the distribution of variable binders ($\forall$, $\exists$, λ).

4.4.2 Prediction on the sources

In addition to the basic syntactic features listed in Table 4.4, we maybe able to estimate the difficulty of a problem by the vocabulary (*i.e.* distribution of function/relation symbols). To see this, we trained two types of classifiers in the manner of subsection 4.3.2. In this experiment, we trained a three-class classifier that predicts the source where a question is derived. Table 4.5 lists the confusion matrix of the classification obtained by 5-fold cross-validation.

The experiment result shows that the predictions on the question sources are quite accurate. The number of the questions wrongly predicted as their sources are **Ex** in spite of their true sources are **Univ** is relatively large. We can interpret the result that the former questions in the problems in

Univ contain easy questions which can be used as exercise problems in **Ex**.

4.4.3 Prediction on the solvability by the system

In this experiment, we trained a binary classifier that predicts whether or not a problem is solvable by the prototype system in one hour. Table 4.6 lists the precision, the recall, and the accuracy of the classification obtained by 5-fold cross-validation. Also, we obtained the problem-wise predictions by combining the question-wise predictions by considering a correct problem-wise prediction is made if all the predictions of the questions in the problem are correct. Table 4.6 lists the performance of the problem-wise predictions.

The definitions of the precision, recall, and accuracy are:

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}},\ \text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}},\ \text{and accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}},$$

where TP (resp. FP) is the numbers of samples which are correctly (resp. wrongly) predicted 'solvable' and TN (resp. FN) is the number of samples which are correctly (resp. wrongly) predicted 'unsolvable.'

From this experiment result, we can observe that the predictions are not so correct compared to the predictions on the question sources. Among the predictions, the ratios relating to TP in **Ex** and TN in **IMO** are relatively high, while the ratios concerning TN in **Univ** and TP in **IMO** are low. This shows that the predictions on the questions/problems that are obviously easy or difficult yield good performance, but the predictions on the ones that appear to be easy yet actually difficult (and the converse cases) tend to fail.

4.4.4 Discussions

The tendency that the performances of the system becomes worse on the expectedly difficult problems was observed. It can be said that the performance of the system reflects the difficulties of the problems, *i.e.* a difficult problem is harder to solve also for a computer.

Further analysis on the performances of the system using SVMs suggests that there are two types of the difficulties: the difficulty that can be explained by the superficial properties such as the vocabulary and the number of symbols in HOL formulas and the difficulty that cannot. The frequency of the appearance of the symbols in the HOL formulas representing the problems has the clear character depending on the sources of the problems. Since we chose the sources to have the different level of difficulties, the good prediction on the problem sources proposes that the superficial properties can represent some kind of difficulty.

However, the character of the frequency is not so clear between those that can be solved by the system within the time limit and those cannot. Clearly, there are the cases that replacing the symbol $+$ by $\times$ or changing a rational coefficient can critically affect the property of the problems and the computational complexity. On the other hand, it is possible that such replacements yield the similar problems to the original ones, and both problems are solvable by the system. In our setting using frequency vectors, we do not use the information of the structure of the HOL formulas. Thus even if a critical replacement has taken to a problem, two formulas are evaluated very similar or even the same. We think that this kind of difficulty that depends on the specific structure of the formulas decreased the quality of the predictions.

4.5 Conclusion

We introduced a model of math problem solving and the representation language to represent math problems. Our experimental results on more than four hundred problems show that the prototype implementation reflects the difficulties of the problems quite precisely. Also, we identified the two types of difficulties for the system: one derived from the the superficial properties of formulas, and the other arises from the specific structure of formulas. This observation encourages to make a further study on a machine learning technique on structured data to achieve a better prediction. The experiment

Table 4.4: Syntactic profiles of the formalized problems

	Ex	**Univ**	**IMO**
# of ∀	2.2	2.0	5.8
# of ∃	5.3	9.3	3.1
# of λ	1.3	2.1	0.1
# of relations	12.5	19.8	13.8
# of functions	19.9	36.3	21.9
# of bound variables	8.8	13.4	9.1
# of free variables	3.0	3.1	1.8

Table 4.5: Confusion matrix of the sources prediction

		Prediction by SVM		
		Ex	**Univ**	**IMO**
True	**Ex**	181	18	14
Source	**Univ**	63	445	19
	IMO	22	14	103

Accuracy: 82.9% (729/879)

Table 4.6: Accuracy of the solvability prediction with respect to the questions

Source	Precision	Recall	TN/(TN + FN)	TN/(TN + FP)	Accuracy
Ex	95.8% (160/167)	91.4% (160/175)	67.4% (31/46)	81.6% (31/38)	89.7% (191/213)
Univ	83.9% (319/380)	83.5% (319/382)	57.1% (84/147)	57.9% (84/145)	76.5% (403/527)
IMO	51.1% (24/47)	54.5% (24/44)	78.3% (72/92)	75.8% (72/95)	69.1% (96/139)
All	84.7% (503/594)	83.7% (503/601)	65.6% (187/285)	67.3% (187/278)	78.5% (690/879)

Table 4.7: Accuracy of the solvability prediction with respect to the problems

Source	Precision	Recall	TN/(TN + FN)	TN/(TN + FP)	Accuracy
Ex	93.0% (66/71)	86.8% (66/76)	66.7% (20/30)	80.0% (20/25)	85.1% (86/101)
Univ	78.9% (116/147)	72.5% (116/160)	55.1% (54/98)	63.5% (54/85)	69.4% (170/245)
IMO	46.2% (18/39)	60.0% (18/30)	84.2% (64/76)	75.3% (64/85)	71.3% (82/115)
All	77.8% (200/257)	75.2% (200/266)	67.6% (138/204)	70.8% (138/195)	73.3% (338/461)

results indicate the model correctly captures the difficulty of the problems and hence it can serve as a basis of a quantitative study on representation change. Future work includes further analysis of the difficulty of math problems in light of our information-processing account and development of computational models of theory choice.

Bibliography

[1] Noriko H Arai, Takuya Matsuzaki, Hidenao Iwane, and Hirokazu Anai. Mathematics by machine. In *Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation*, pages 1–8. ACM, 2014.

[2] Christopher W. Brown and James H. Davenport. The complexity of quantifier elimination and cylindrical algebraic decomposition. In *Proceedings of the 2007 International Symposium on Symbolic and Algebraic Computation*, ISSAC '07, pages 54–60, New York, NY, USA, 2007. ACM.

[3] Christopher JC Burges. A tutorial on support vector machines for pattern recognition. *Data mining and knowledge discovery*, 2(2):121–167, 1998.

[4] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(3):27, 2011.

[5] Shang-Ching Chou. *Mechanical geometry theorem proving*, volume 41. Springer Science & Business Media, 1988.

[6] Andreas Dolzmann, Andreas Seidl, and Thomas Sturm. Efficient projection orders for CAD. In *Proceedings of the 2004 international symposium on Symbolic and algebraic computation*, pages 111–118. ACM, 2004.

[7] David Haussler. Convolution kernels on discrete structures. Technical Report UCS-CRL-99-10, University of California at Santa Cruz, Santa Cruz, CA, USA, 1999.

[8] Chih-Wei Hsu, Chih-Chung Chang, Chih-Jen Lin, et al. A practical guide to support vector classification, 2003.

[9] Zongyan Huang, Matthew England, David Wilson, James H Davenport, Lawrence C Paulson, and James Bridge. Applying machine learning to the problem of choosing a heuristic to select the variable ordering for cylindrical algebraic decomposition. In *Intelligent Computer Mathematics*, pages 92–107. Springer, 2014.

[10] Matthew I Isaak and Marcel Adam Just. Constraints on thinking in insight and invention. In Robert J. Sternberg and Janet E. Davidson, editors, *The nature of insight*, pages 281–325. MIT Press, 1995.

[11] Hidenao Iwane, Hiroyuki Higuchi, and Hirokazu Anai. An effective implementation of a special quantifier elimination for a sign definite condition by logical formula simplification. In *Computer Algebra in Scientific Computing*, pages 194–208. Springer, 2013.

[12] Hidenao Iwane, Takuya Matsuzaki, N Arai, and Hirokazu Anai. Automated natural language geometry math problem solving by real quantifier elimination. In *Proceedings of the 10th International Workshop on Automated Deduction in Geometry*, pages 75–84, 2014.

[13] Hidenao Iwane, Hitoshi Yanami, and Hirokazu Anai. SyNRAC: A toolbox for solving real algebraic constraints. In *Mathematical Software–ICMS 2014*, pages 518–522. Springer, 2014.

[14] Hidenao Iwane, Hitoshi Yanami, and Hirokazu Anai. Synrac: A toolbox for solving real algebraic constraints. In *Mathematical Software–ICMS 2014*, pages 518–522. Springer, 2014.

[15] Thorsten Joachims. Text categorization with support vector machines: Learning with many relevant features. In *European conference on machine learning*, pages 137–142. Springer, 1998.

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

[17] Craig A. Kaplan and Herbert A. Simon. In search of insight. *Cognitive psychology*, 22(3):374–419, 1990.

[18] Manfred Kerber and Martin Pollet. A tough nut for mathematical knowledge management. In *Mathematical Knowledge Management*, pages 81–95. Springer, 2006.

[19] Munehiro Kobayashi, Hidenao Iwane, Takuya Matsuzaki, and Hirokazu Anai. Efficient subformula orders for real quantifier elimination of non-prenex formulas. In *International Conference on Mathematical Aspects of Computer and Information Sciences*, pages 236–251. Springer, 2015.

[20] James N. MacGregor, Thomas C. Ormerod, and Edward P. Chronicle. Information processing and insight: A process model of performance on the nine-dot and related problems. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 27(1):176, 2001.

[21] David Marker. *Model theory*, volume 217 of *Graduate Texts in Mathematics.* Springer-Verlag, New York, 2002. An introduction.

[22] Takuya Matsuzaki, Hidenao Iwane, Hirokazu Anai, and Noriko H Arai. The most uncreative examinee: a first step toward wide coverage natural language math problem solving. In *Twenty-Eighth AAAI Conference on Artificial Intelligence*, pages 1098–1104, 2014.

[23] Takuya Matsuzaki, Hidenao Iwane, Munehiro Kobayashi, Yiyang Zhan, Ryoya Fukasaku, Jumma Kudo, Hirokazu Anai, and Noriko H Arai. Race against the teens–benchmarking mechanized math on pre-university problems. In *International Joint Conference on Automated Reasoning*, pages 213–227. Springer, 2016.

[24] Takuya Matsuzaki, Munehiro Kobayashi, and Noriko H. Arai. An information-processing account of representation change: International mathematical olympiad problems are hard not only for humans. In A. Papafragou, D. Grodner, D. Mirman, and J.C. Trueswell, editors, *Proceedings of the 38th Annual Conference of the Cognitive Science Society, CogSci 2016*, pages 2297–2302, 2016.

[25] Allen Newell and Herbert Alexander Simon. *Human problem solving*, volume 104. Prentice-Hall, Englewood Cliffs, NJ, 1972.

[26] Stellan Ohlsson. Information-processing explanations of insight and related phenomena. In *Advances in the psychology of thinking*, pages 1–44. Harvester Wheatsheaf, 1992.

[27] Michael Öllinger, Gary Jones, and Günther Knoblich. The dynamics of search, impasse, and representational change provide a coherent explanation of difficulty in the nine-dot problem. *Psychological research*, 78(2):266–275, 2014.

[28] Mark Steedman. *The Syntactic Process.* Bradford Books. MIT Press, 2001.

[29] Adam W. Strzeboński. Real quantifier elimination for non-prenex formulas. unpublished manuscript, 2011.

[30] Alfred Tarski. *A decision method for elementary algebra and geometry.* University of California Press, 1951.

[31] Lou van den Dries. Classical model theory of fields. In *Model theory, algebra, and geometry*, volume 39 of *Math. Sci. Res. Inst. Publ.*, pages 37–52. Cambridge Univ. Press, Cambridge, 2000.

[32] Vladimir Vapnik. *Estimation of dependences based on empirical data.* Springer Series in Statistics. Springer-Verlag, New York-Berlin, 1982. Translated from the Russian by Samuel Kotz.

[33] Vladimir N. Vapnik. *The nature of statistical learning theory.* Springer-Verlag, New York, 1995.

[34] Vladimir N. Vapnik. *Statistical learning theory.* Adaptive and Learning Systems for Signal Processing, Communications, and Control. John Wiley & Sons, Inc., New York, 1998. A Wiley-Interscience Publication.

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