International Journal of Innovative
Computing, Information and Control
Volume **19**, Number **5**, October **2023**

ICIC International ©2023 ISSN 1349-4198
pp. 1517–1530

COMPUTATION CONTROL BY PRIORITIZED EQUIVALENT TRANSFORMATION RULES

Kiyoshi Akama[1] and Ekawit Nantajeewarawat[2,*]

[1]Information Initiative Center
Hokkaido University
Kita 11, Nishi 5, Kita-ku, Sapporo, Hokkaido 060-0811, Japan
akama@iic.hokudai.ac.jp

[2]School of Information, Computer and Communication Technology
Sirindhorn International Institute of Technology
Thammasat University
99 Moo 18, Km. 41 on Paholyothin Highway, Khlong Luang, Pathum Thani 12120, Thailand
*Corresponding author: ekawit@siit.tu.ac.th

Received July 2022; revised December 2022

Abstract. *The conventional concept of computation in logical problem solving is based on procedural reading of logical formulas. In contrast, computation in the equivalent transformation (ET) framework is successive application of an unlimited number of ET rules, which are procedures and not logical formulas. Computation by ET rules truly extends the solvability of logical problems. For solving logical problems on clauses, we propose in this paper prioritized ET rules. A set R of ET rules with priority ordering is employed, and at each computation step an applicable ET rule with best priority in R is selected and applied. This method can be used to reduce a search space by introducing new rules and adjusting rule priority, and is useful for solving a large class of logical problems with guarantee of strict correctness of computation results.*
Keywords: Model-intersection problem, Equivalent transformation, Rule priority, Computation control, Proof problem, Query-answering problem, Correctness

1. **Introduction.** By first-order formulas, we mean first-order formulas without built-in constraint atoms. By full first-order formulas, we mean first-order formulas possibly with built-in constraint atoms. Proof problems on first-order formulas historically constitute the most important problem class in logical problem solving. The conventional resolution-based theory, however, fails to establish a correct proof method for full first-order formulas [1]. A query-answering (QA) problem is an "all-answers finding" problem to satisfy a given logical consequence relation on first-order formulas. SLD resolution has been proposed for solving QA problems on definite clauses [2]. The conventional resolution-based theory, however, has failed to establish a correct QA solution method for full first-order formulas [3]. It has been revealed that the conventional logical computation theory has both representational limitation and computational limitation.

To break the representational and computational limitations of first-order formulas, LPSF (Logical Problem Solving Framework) was invented [4]. LPSF is an axiomatic logic and can be applied to conventional logics. LPSF can also generate new logics. KR-Logic (knowledge-representation-logic) was created on LPSF as an extension of first-order logic [5]. KR-Logic overcomes the representational and computational limitations of first-order logic [1, 3, 5, 6].

DOI: 10.24507/ijicic.19.05.1517

The conventional concept of computation in logical problem solving is based on procedural reading of logical formulas. Only one rule of resolution is used in the conventional computation by SLD resolution, and control means to select an occurrence of an atom in a clause at each step of computation. Control in LPSF, in comparison, is to select a rule in a given rule set at each computation step. One main characteristic of LPSF computation is to use various equivalent transformation (ET) rules. Computation in the ET framework is successive application of an unlimited number of ET rules, which are procedures and not logical formulas. Computation by ET rules truly extends the solvability of logical problems. In contrast, the conventional resolution-based computation can be regarded as a simpler computational framework with a single transformation rule such as resolution or unfolding.

For solving logical problems on clauses in the ET framework, we propose in this paper prioritized ET rules. A set R of ET rules with priority ordering is employed, and at each computation step an applicable ET rule with best priority in R is selected and applied. Prioritizing a set of ET rules is a simple control mechanism, where the first applicable rule in a given sequence of prioritized ET rules is selected and applied to a problem. Prioritized ET rules can be used generally by any LPSF solver including one for KR-Logic. Since we have various ET rules, control with prioritized ET rules is very useful for better computation performance.

This method can be used to reduce a search space by introducing new rules on demand and adjusting rule priority, and is useful for solving a large class of logical problems with guarantee of strict correctness of computation results. One computation control may give the shortest and finite path to the end of computation, while another computation control may result in non-termination, i.e., never-ending computation with no answer being obtained. By assigning a higher priority to an ET rule whose application yields a smaller number of clauses, resource minimization control is possible. In practical problem solving, resource for computation is limited. We can minimize execution time to reach a conclusion under space constraints by giving higher priorities to lower-splitting rules.

The rest of the paper is organized as follows. Section 2 explains the LPSF theory, which presents a solution method based on ET. Section 3 proposes prioritized ET rules for computation control. Correctness of computation with prioritized ET rules directly follows from the LPSF theory. Section 4 introduces a puzzle that is used as the main example in this paper. The puzzle is represented by first-order formulas, and is converted by Skolemization to a set of clauses in a clause space. Section 5 explains several ET rules used in this paper mainly by examples. Section 6 illustrates computation with resource minimization control by introducing new specialized rules and adjusting rule priority. Section 7 provides conclusions.

The notation that follows holds thereafter. Given a set A, $pow(A)$ denotes the power set of A. Given two sets A and B, $Map(A, B)$ denotes the set of all mappings from A to B, and for any partial mapping f from A to B, $dom(f)$ denotes the domain of f, i.e., $dom(f) = \{a \mid (a \in A)\ \&\ (f(a) \text{ is defined})\}$.

2. LPSF Theory.

2.1. LPSF theory.

The conventional concept of computation in logical problem solving is based on procedural reading of logical formulas, while computation in our theory is successive application of an unlimited number of equivalent transformation (ET) rules. To overcome the limited computation power of conventional logical problem solving methods [1, 3], a general theory, called a logical problem solving framework (LPSF), was invented [4], which extends the most basic concepts of logic and computation as follows.

- LPSF utilizes a general concept of logical structure.
- LPSF regards model-intersection (MI) problems [7] as the main class of logical problems.
- LPSF takes ET rules as units of procedures.

MI problems form a large class of logical problems. Proof problems and query-answering problems on first-order logic are considered as subclasses of MI problems. By solving MI problems, we can solve proof problems and query-answering problems on first-order logic. A general principle for solving MI problems on clauses is ET, where problems are solved by repeated problem simplification using ET rules. Efficiency of computation is basically determined by

- a set R of ET rules used for the computation, and
- selection of an ET rule in R at each computation step.

ET rules and computation control may give an ET sequence that reaches a final problem description in finite steps or may produce an infinite sequence without giving any answer to the original problem.

2.2. **Control and ET sequences.** Let R be a set of ET rules. Let *Prob* be the set of all MI problems. Given a sequence $[P_0, P_1, \ldots, P_i]$ of problems in *Prob*, a control gives an ET rule in R that is applicable to P_i. More precisely, a control *ctrl* is a partial mapping from the set of all sequences of problems in *Prob* to the ET-rule set R that satisfies the following two conditions. First, $ctrl([P_0, P_1, \ldots, P_i])$ is an ET rule in R that is applicable to P_i. Secondly, $ctrl([P_0, P_1, \ldots, P_i])$ is undefined if no rule in R is applicable to P_i.

Given an initial problem P_0 and a rule set R, a control *ctrl* determines a finite or infinite problem sequence $[P_0, P_1, \ldots]$ such that i) if *ctrl* gives a rule r_i to a problem P_i in the sequence, then the next problem P_{i+1} is obtained by $(P_i, P_{i+1}) \in r_i$, and ii) if *ctrl* gives no rule to a problem P_i in the sequence, then P_i is the last problem and the problem sequence terminates.

2.3. **An LPSF-based solver.** We can invent a logical problem solving method using LPSF by designing four parameters: 1) a canonical logical structure $\mathcal{L}$, 2) a set R of ET rules, 3) a control *ctrl*, and 4) a set A of answer mappings. An answer mapping is a mapping to obtain a computed answer of a given MI problem from the final state of computation. For each initial problem, an LPSF-based solver determines a problem sequence, a final problem, and a computed answer. A solver is correct if the following condition is satisfied: for any initial problem, if the problem sequence determined by the solver reaches the domain of an answer mapping, then the computed answer is equal to the answer to the initial problem.

LPSF with these input parameters works as a problem solver for MI problems as follows.

- A problem q formalized on $\mathcal{L}$ is received as input.
- The input problem q is taken as the initial state P_0.
- P_0 is successively transformed by using ET rules in R, with rule selection being determined by *ctrl*.
- Computation is a sequence of problems $P_0, P_1, P_2, \ldots$ on $\mathcal{L}$.
- If the computation $P_0, P_1, P_2, \ldots$ reaches the domain of an answer mapping *ans* in A at P_n, then LPSF outputs $ans(P_n)$ as an answer.

3. **Computation Control by Prioritized ET Rules.** Based on the LPSF theory and the definition of control described in Section 2, we introduce computation control by prioritized ET rules for LPSF solvers. The correctness of computation with prioritized ET rules follows from the LPSF theory.

3.1. **Models and MI problems.** Let $\mathcal{G}_{\mathrm{u}}$ be a set, the elements of which are regarded as ground user-defined atoms. A canonical logical structure $\mathcal{L}$ is a triple $\langle \mathcal{K}, \mathcal{I}, \nu \rangle$ that satisfies the following conditions [4].

1) $\mathcal{K}$ is a set.
2) $\mathcal{I} = \mathit{pow}(\mathcal{G}_{\mathrm{u}})$.
3) ν is a mapping from $\mathcal{K}$ to $\mathit{Map}(\mathcal{I}, \{\mathit{true}, \mathit{false}\})$.

An element of $\mathcal{K}$ is called a *description* and that of $\mathcal{I}$ is called an *interpretation*. A typical description is a logical formula in the first-order syntax. A set of clauses on first-order logic is also a typical example of a description.

An interpretation $I \in \mathcal{I}$ is a *model* of a description $k \in \mathcal{K}$ iff $\nu(k)(I) = \mathit{true}$. The set of all models of a description $k \in \mathcal{K}$ is denoted by $\mathit{Models}(k)$. The intersection of all models of k is denoted by $\bigcap \mathit{Models}(k)$. Note that when $\mathit{Models}(k)$ is the empty set, $\bigcap \mathit{Models}(k) = \mathcal{G}_{\mathrm{u}}$.

Hereafter, let Cs be a description on $\mathcal{L}$. A *model-intersection problem* (for short, *MI problem*) on $\mathcal{L}$ is a pair $\langle \mathit{Cs}, \varphi \rangle$, where φ is a mapping from $\mathit{pow}(\mathcal{G}_{\mathrm{u}})$ to some set W. The mapping φ is called an *extraction mapping*. The answer to this problem, denoted by $\mathit{ans}_{\mathrm{MI}}(\mathit{Cs}, \varphi)$, is defined by

$$\mathit{ans}_{\mathrm{MI}}(\mathit{Cs}, \varphi) = \varphi\left(\bigcap \mathit{Models}(\mathit{Cs})\right).$$

Given an MI problem $P = \langle \mathit{Cs}, \varphi \rangle$, $\mathit{ans}_{\mathrm{MI}}(\mathit{Cs}, \varphi)$ is simply written as $\mathit{ans}_{\mathrm{MI}}(P)$.

3.2. **Control and prioritized ET rules.** A control selects an ET rule that is applied to a problem at each step of computation based on a given problem sequence starting from the initial problem to the current problem. Prioritized ET rules give a simple control that selects an ET rule based only on the current problem P_i.

We use priority of ET rules for computation control, where the first applicable rule in a given sequence of rules is selected and applied to a given logical problem. Let *Prob* be the set of all MI problems. Let R be a set of ET rules on *Prob*. Define R^+ as the set of all sequences of mutually distinct rules in R. A sequence $[r_1, r_2, \ldots, r_m]$ in R^+ is also denoted by $r_1 \geq r_2 \geq \cdots \geq r_m$. A rule r_i in $\{r_1, r_2, \ldots, r_m\}$ is given priority i by $[r_1, r_2, \ldots, r_m]$. Hence $[r_1, r_2, \ldots, r_m]$ is also called a prioritized ET rule set.

Let $R_p = [r_1, r_2, \ldots, r_m] \in R^+$. Let P be a problem. R_p is applicable to P with r_j if

1) none of $r_1, r_2, \ldots, r_{j-1}$ is applicable to P, and
2) r_j is applicable to P.

Note that j is determined uniquely by P and R_p. For each element R_p in R^+, define $\mathit{rule}(R_p)$ as the set of all pairs (P, P') such that R_p is applicable to P with r_j, and $(P, P') \in r_j$. Since the relation $\mathit{rule}(R_p)$ is regarded as a rule, R_p can be identified as a rule by the relation $\mathit{rule}(R_p)$.

3.3. **ET sequences and control by prioritized ET rules.** Let R be a set of ET rules on *Prob*. Let P_0 be an initial problem and R_p a sequence of ET rules in R. The sequence R_p of ET rules determines

- a rule set $\{r_1, r_2, \ldots, r_m\}$ if $R_p = [r_1, r_2, \ldots, r_m]$, and
- a control, i.e., R_p and P determine r_j uniquely such that R_p is applicable to P with r_j (Section 3.2).

Definition 3.1. *A sequence $[P_0, P_1, \ldots]$ of problems in Prob is an ET sequence iff $\mathit{ans}_{\mathrm{MI}}(P_i) = \mathit{ans}_{\mathrm{MI}}(P_{i+1})$ for any P_i and P_{i+1} in the sequence.*

Since all rules in R_p are ET rules, R_p produces a finite or infinite ET sequence.

Definition 3.2. *A mapping comp gives a finite sequence of problems, i.e.,*

$$comp(P_0, R_p) = [P_0, P_1, P_2, \ldots, P_n],$$

if i) $(P_i, P_{i+1}) \in R_p$ *for any* $i \in \{0, 1, \ldots, n-1\}$ *and ii)* R_p *is not applicable to* P_n. *Otherwise* $comp(P_0, R_p)$ *gives an infinite sequence of problems* $[P_0, P_1, P_2, \ldots]$ *such that* $(P_i, P_{i+1}) \in R_p$ *for any* $i \geq 0$.

3.4. **Correctness of computation with prioritized ET rules.** An MI problem $\langle Cs, \varphi \rangle$, where Cs is a description and φ is an extraction mapping, can be solved as follows.

1) Let ans be an answer mapping.
2) Prepare a sequence R_p of ET rules.
3) Take P_0 such that $P_0 = \langle Cs, \varphi \rangle$ to start computation from P_0.
4) Obtain $comp(P_0, R_p) = [P_0, P_1, P_2, \ldots, P_n]$, and $P_n = \langle Cs_n, \varphi_n \rangle$.
5) If the computation reaches the domain of ans, i.e., $\langle Cs_n, \varphi_n \rangle \in dom(ans)$, then compute the answer by using the answer mapping ans, i.e., output $ans(Cs_n, \varphi_n)$.

There are many possible computation paths depending on the selection of a sequence R_p in R^+. Every output computed by taking any arbitrary computation path is correct.

Theorem 3.1. *When an ET sequence starting from* $P_0 = \langle Cs, \varphi \rangle$ *reaches* P_n *in* $dom(ans)$, *the above procedure gives the correct answer to* $\langle Cs, \varphi \rangle$.

Proof: Since $[P_0, P_1, \ldots, P_n]$ is an ET sequence, $ans_{\mathrm{MI}}(Cs, \varphi) = ans_{\mathrm{MI}}(Cs_n, \varphi_n)$. Since ans is an answer mapping,

$$ans(Cs_n, \varphi_n) = ans_{\mathrm{MI}}(Cs_n, \varphi_n) = ans_{\mathrm{MI}}(Cs, \varphi) = \varphi\left(\bigcap Models(Cs)\right).$$

Hence, $ans(Cs_n, \varphi_n) = \varphi(\bigcap Models(Cs))$. □

3.5. **Finite solvability.** From an initial problem P_0 and a prioritized rule set $R_p \in R^+$,

$$finalProb(P_0, R_p) = P_n$$

means that an initial problem P_0 is sucessively transformed into a final problem P_n by R_p. Given a logical structure and a set of answer mappings, one typical computational improvement is obtained by creation and accumulation of ET rules. Assume that a logical structure is given, and A is a set of answer mappings. Let $FSA(R, A)$ denote the set of all problems that can be solved successfully using a set R of ET rules and a set A of answer mappings by some control, i.e.,

$$\begin{aligned} FSA(R, A) = \{P_0 \mid\ & (finalProb(P_0, R_p) = P_n)\ \&\\ & (P_n \in dom(ans))\ \&\\ & (ans \in A)\ \&\\ & (R_p \in R^+)\}. \end{aligned}$$

The set $FSA(R, A)$ is called the *finitely solvable area* with respect to R and A assuming rule priority.

3.6. **The plan of the paper.** The LPSF theory and the theory of control by prioritized ET rules described so far can be applied to any logical structures such as first-order logic and KR-Logic. From the next section onwards, we apply the theory of control by prioritized ET rules to clauses on first-order logic and discuss computation control by prioritized ET rules. We will explain several ET rules that may be applicable to clauses on first-order logic. If we have more ET rules, priority-based control is more effective. By specializing the applicability of ET rules, we can obtain ET rules that are useful for smaller clause-splitting. We will take a proof problem in Section 4, explain several ET

rules in a clause space in Section 5, and show that priority control by specialized ET rules is useful for resource minimization in Section 6.

4. **A Proof Problem and Its Formalization.** We introduce a proof problem, called the Steamroller puzzle. It is represented by a set of first-order formulas and is transformed into an MI problem described by a set of clauses.

4.1. **Schubert's Steamroller puzzle.** The Steamroller puzzle was presented by Lenhart Schubert in 1978 as a challenge to automated-deduction systems. It was considered to be too hard for existing theorem provers at that time due to its big search space. Much work has been done related to the Steamroller puzzle to improve efficiency of computation [8, 9, 10, 11, 12]. This puzzle will be used in Section 6 to explain computation control with prioritized ET rules and to illustrate the effect of computation control.

This puzzle is a proof problem. Formalization of the Steamroller puzzle as a proof problem based on the conventional theory is given in Section 4.6. Since proof problems can be transformed into MI problems according to our theory, we also give a formulation of the Steamroller puzzle as an MI problem in Section 4.6.

4.2. **Formalization.** The problem description of the Steamroller puzzle is as follows: Wolves, foxes, birds, caterpillars, and snails are animals, and there are some of each of them. Also there are some grains, and grains are plants. Every animal either likes to eat all plants or all animals much smaller than itself that like to eat some plants. Caterpillars and snails are much smaller than birds, which are much smaller than foxes, which in turn are much smaller than wolves. Wolves do not like to eat foxes or grains, while birds like to eat caterpillars but not snails. Caterpillars and snails like to eat some plants. Therefore, there is an animal that likes to eat a grain-eating animal.

The following predicates will be used for formalization of the Steamroller puzzle.

$A(x)$	– x is an animal	$W(x)$	– x is a wolf
$F(x)$	– x is a fox	$B(x)$	– x is a bird
$C(x)$	– x is a caterpillar	$S(x)$	– x is a snail
$G(x)$	– x is a grain	$P(x)$	– x is a plant
$M(x,y)$	– x is much smaller than y	$E(x,y)$	– x likes to eat y

Figure 1 shows the first-order formulas representing this problem [10]. All sentences except the last one form the background knowledge and are represented by F_1-F_{22} in Figure 1. The third sentence of the problem description ("Every animal either likes to eat all plants or all animals much smaller than itself that like to eat some plants") is represented by the first-order formula F_{13} in Figure 1. The last sentence of the problem description ("Therefore there is an animal that likes to eat a grain-eating animal") is regarded as a conclusion to be proved and is represented by the first-order formula F_{23} in Figure 1.

4.3. **Clause space $\mathbf{CLS_B}$.** Let $\mathcal{A}_{\rm u}$ be the set of all user-defined atoms. Let $\mathcal{G}_{\rm u}$ be the set of all ground user-defined atoms. Let $\mathcal{A}_{\rm c}$ be the set of all built-in constraint atoms. Let $\mathcal{G}_{\rm c}$ be the set of all ground built-in constraint atoms.

A *clause* C on $\mathcal{A}_{\rm u} \cup \mathcal{A}_{\rm c}$ is a formula of the form

$$a_1, \ldots, a_m \leftarrow b_1, \ldots, b_n,$$

where each of $a_1, \ldots, a_m, b_1, \ldots, b_n$ is a user-defined atom in $\mathcal{A}_{\rm u}$ or a constraint atom in $\mathcal{A}_{\rm c}$. All variables occurring in C are implicitly universally quantified and their scope is restricted to the clause C itself. Let $lhs(C)$ and $rhs(C)$ denote the set of all atoms in the left-hand side of C and the set of all those in the right-hand side of C, respectively. When there is only one user-defined atom in $lhs(C)$, C is called a *definite clause*. A clause is

F_1: $\exists x : W(x)$
F_2: $\exists x : F(x)$
F_3: $\exists x : B(x)$
F_4: $\exists x : C(x)$
F_5: $\exists x : S(x)$
F_6: $\forall x : (W(x) \to A(x))$
F_7: $\forall x : (F(x) \to A(x))$
F_8: $\forall x : (B(x) \to A(x))$
F_9: $\forall x : (C(x) \to A(x))$
F_{10}: $\forall x : (S(x) \to A(x))$
F_{11}: $\exists x : G(x)$
F_{12}: $\forall x : (G(x) \to P(x))$
F_{13}: $\forall x : [A(x) \to [[\forall y : (P(y) \to E(x,y))] \vee [\forall y : (A(y) \wedge M(y,x) \wedge (\exists z : (E(y,z) \wedge P(z))) \to E(x,y))]]]$
F_{14}: $\forall x : (C(x) \to (\forall y : (B(y) \to M(x,y))))$
F_{15}: $\forall x : (S(x) \to (\forall y : (B(y) \to M(x,y))))$
F_{16}: $\forall x : (B(x) \to (\forall y : (F(y) \to M(x,y))))$
F_{17}: $\forall x : (F(x) \to (\forall y : (W(y) \to M(x,y))))$
F_{18}: $\forall x : (W(x) \to (\forall y : ((F(y) \vee G(y)) \to \neg E(x,y))))$
F_{19}: $\forall x : (B(x) \to (\forall y : (C(y) \to E(x,y))))$
F_{20}: $\forall x : (B(x) \to (\forall y : (S(y) \to \neg E(x,y))))$
F_{21}: $\forall x : (C(x) \to (\exists y : (P(y) \wedge E(x,y))))$
F_{22}: $\forall x : (S(x) \to (\exists y : (P(y) \wedge E(x,y))))$
F_{23}: $\exists x : [A(x) \wedge [\exists y : (A(y) \wedge [\exists z : (G(z) \wedge E(y,z))] \wedge E(x,y))]]$

FIGURE 1. First-order formulas representing the Steamroller puzzle

ground if all atoms in the clause are ground. The set of all clauses is denoted by $\mathrm{CLS_B}$. The *problem clause space* is the powerset of $\mathrm{CLS_B}$. Next, we will give general concepts related to a set of clauses in $\mathrm{CLS_B}$, such as interpretations, models, canonical logical structures, and MI problems.

4.4. **Interpretations and models.** An *interpretation* is defined as a subset of $\mathcal{G}_\mathrm{u}$. A ground user-defined atom g is true under an interpretation I iff g belongs to I. Unlike ground user-defined atoms, the truth values of ground constraint atoms are predetermined independently of interpretations. Let TCON denote the set of all true ground constraint atoms, i.e., a ground constraint atom g is true iff $g \in \mathrm{TCON}$.

A ground clause $C = (a_1, \ldots, a_m \leftarrow b_1, \ldots, b_n) \in \mathrm{CLS_B}$ is true under an interpretation I (in other words, I *satisfies* C) iff at least one of the following conditions is satisfied:

1) There exists $i \in \{1, \ldots, m\}$ such that $a_i \in I \cup \mathrm{TCON}$;
2) There exists $i \in \{1, \ldots, n\}$ such that $b_i \notin I \cup \mathrm{TCON}$.

An interpretation I is a *model* of a clause set $Cs \subseteq \mathrm{CLS_B}$ iff for any clause $C \in Cs$ and any substitution θ for usual variables, if $C\theta$ is a ground clause, then $C\theta$ is true under I.

4.5. **A canonical logical structure and MI problems on it.** To apply the LPSF theory to solving the Steamroller puzzle, we introduce a canonical logical structure $\mathcal{L} = \langle \mathcal{K}, \mathcal{I}, \nu \rangle$ as follows:

- Let $\mathcal{K} = pow(\mathrm{CLS_B})$.
- Let $\mathcal{I} = pow(\mathcal{G}_\mathrm{u})$.

- Let ν: $\mathcal{K} \to \mathit{Map}(\mathcal{I}, \{\mathit{true}, \mathit{false}\})$ be defined by: for any $Cs \subseteq \mathrm{CLS_B}$, $\nu(Cs)$ is a mapping from $\mathcal{I}$ to $\{\mathit{true}, \mathit{false}\}$ such that for each $G \in \mathcal{I}$, $\nu(Cs)(G) = \mathit{true}$ if G is a model of Cs, and $\nu(Cs)(G) = \mathit{false}$ if G is not a model of Cs.

An MI problem on $\mathrm{CLS_B}$ is a pair $\langle Cs, \varphi\rangle$, where $Cs \subseteq \mathrm{CLS_B}$ and φ is an extraction mapping from $\mathit{pow}(\mathcal{G}_\mathrm{u})$ to some set W. The answer to this problem, denoted by $\mathit{ans}_\mathrm{MI}(Cs, \varphi)$, is defined by $\mathit{ans}_\mathrm{MI}(Cs, \varphi) = \varphi(\bigcap \mathit{Models}(Cs))$.

4.6. **Formalization of the puzzle as an MI problem.** Referring to F_1-F_{23} in Figure 1, let $F = F_1 \wedge F_2 \wedge \cdots \wedge F_{22}$. The Steamroller puzzle is first formalized as $\mathit{Models}(F \wedge \neg F_{23}) = \emptyset$, which corresponds to the formalization as a proof problem $F \models F_{23}$ in the conventional theory. By using the conventional Skolemization, $F \wedge \neg F_{23}$ is converted into a clause set Cs_1 consisting of the clauses in Figure 2, where w, f, b, c, s, and g are Skolem constants, and f_1 and f_2 are Skolem functions. Since $\mathit{Models}(F \wedge \neg F_{23}) = \emptyset$ iff $\mathit{Models}(Cs_1) = \emptyset$, we have a new formalization $\mathit{Models}(Cs_1) = \emptyset$.

C_1: $W(w) \leftarrow$
C_2: $F(f) \leftarrow$
C_3: $B(b) \leftarrow$
C_4: $C(c) \leftarrow$
C_5: $S(s) \leftarrow$
C_6: $G(g) \leftarrow$
C_7: $A(x) \leftarrow W(x)$
C_8: $A(x) \leftarrow F(x)$
C_9: $A(x) \leftarrow B(x)$
C_{10}: $A(x) \leftarrow C(x)$
C_{11}: $A(x) \leftarrow S(x)$
C_{12}: $P(x) \leftarrow G(x)$
C_{13}: $E(x, y), E(x, z) \leftarrow A(x), P(y), A(z), M(z, x), P(u), E(z, u)$
C_{14}: $M(x, y) \leftarrow C(x), B(y)$
C_{15}: $M(x, y) \leftarrow S(x), B(y)$
C_{16}: $M(x, y) \leftarrow B(x), F(y)$
C_{17}: $M(x, y) \leftarrow F(x), W(y)$
C_{18}: $\leftarrow W(x), F(y), E(x, y)$
C_{19}: $\leftarrow W(x), G(y), E(x, y)$
C_{20}: $E(x, y) \leftarrow B(x), C(y)$
C_{21}: $\leftarrow B(x), S(y), E(x, y)$
C_{22}: $P(f_1(x)) \leftarrow C(x)$
C_{23}: $E(x, f_1(x)) \leftarrow C(x)$
C_{24}: $P(f_2(x)) \leftarrow S(x)$
C_{25}: $E(x, f_2(x)) \leftarrow S(x)$
C_{26}: $\leftarrow A(x), A(y), G(z), E(y, z), E(x, y)$

FIGURE 2. Clausal form

For any $Cs \subseteq \mathrm{CLS_B}$, $\mathit{Models}(Cs) = \emptyset$ iff $\bigcap \mathit{Models}(Cs) = \mathcal{G}_u$. As a result, the formalization $\mathit{Models}(Cs_1) = \emptyset$ is equivalent to $\bigcap \mathit{Models}(Cs_1) = \mathcal{G}_u$, which is also equivalent to $\varphi_1(\bigcap \mathit{Models}(Cs_1)) = \mathit{yes}$, where φ_1 is a mapping from $\mathit{pow}(\mathcal{G}_\mathrm{u})$ to $\{\mathit{yes}, \mathit{no}\}$ such that for any $G \subseteq \mathcal{G}_\mathrm{u}$, $\varphi_1(G) =$ "yes" if $G = \mathcal{G}_\mathrm{u}$, otherwise $\varphi_1(G) =$ "no". Hence we have an MI problem $\langle Cs_1, \varphi_1\rangle$ such that

$$\mathit{ans}_\mathrm{MI}(Cs_1, \varphi_1) = \varphi_1\left(\bigcap \mathit{Models}(Cs_1)\right).$$

To solve this MI problem, Cs_1 is transformed equivalently. While only the resolution and factoring inference rules are used in the conventional proof theory, an LPSF-based solver uses all possible ET rules, including unfolding and the ET rules described in Section 5.

5. **ET Rules by Examples.** We explain the ET rules used in this paper by examples. Their strict definitions and correctness proofs can be found elsewhere [13, 14].

5.1. **Resolution.** Resolution is a transformation rule for producing a new clause, called a resolvent, from two clauses. For instance, suppose that Cs contains the two clauses:

C_1: $p(x) \leftarrow q(x), r(x, 4), s(x)$
C_2: $r(1, y), t(y, z) \leftarrow u(y), v(z)$

By applying the resolution rule to C_1 and C_2, a new clause

C_3: $p(1), t(4, z) \leftarrow q(1), s(1), u(4), v(z)$

is added to Cs as the resolvent.

5.2. **Factoring.** Two atoms in the same side of a clause are unified to give a new clause. For instance, suppose that Cs contains the clause:

C_1: $p(x) \leftarrow q(x), r(x, 4), r(3, y)$

Then a new clause

C_2: $p(3) \leftarrow q(3), r(3, 4)$

is added to Cs. Suppose that Cs contains the clause:

C_3: $p(x), r(x, 4), r(3, y) \leftarrow$

A new clause

C_4: $p(3), r(3, 4) \leftarrow$

is added.

5.3. **Erasing independent satisfiable atoms.** Let C be a clause and B a set of atoms. Let $C \ominus B$ be defined as the clause obtained from C by removing all atoms in B from its right-hand side. That is, $C \ominus B$ is defined by $lhs(C \ominus B) = lhs(C)$ and $rhs(C \ominus B) = rhs(C) - B$. This rule, referred to as (erase) in Section 6, changes C into $C \ominus B$ if i) B and $(lhs(C) \cup rhs(C)) - B$ have no common variable and ii) B can be instantiated to be true under the condition of $Cs - \{C\}$. For instance, suppose that Cs contains the two clauses:

C_1: $p(f(2, 6)) \leftarrow$
C_2: $r(y) \leftarrow p(f(x, 6)), q(y)$

Then $p(f(x, 6))$ can be removed from C_2.

5.4. **Elimination of subsumed clauses.** This rule, referred to as (subsumed) in Section 6, removes a clause C from a clause set Cs if C is subsumed by some clause in Cs. For instance, suppose that Cs contains the two clauses:

C_1: $h_1, h_2 \leftarrow b_1, b_2$
C_2: $h_1 \leftarrow b_2$

Then C_1 can be removed from Cs.

5.5. **Elimination of an atom in a clause's left-hand side.** This rule, called the forwarding transformation rule, erases an atom a in the left-hand side of a clause by using a negative clause $(\leftarrow b)$, where b is an atom, such that a is an instance of b. For instance, suppose that Cs contains the two clauses:

C_1: $p(X, 3), h \leftarrow q(X)$
C_2: $\leftarrow p(Y, Z)$

Then C_1 can be changed into a new clause $(h \leftarrow q(X))$.

6. **Priority-Based Control with Specialized ET Rules.** For more flexible and more efficient computation, we use priority-based control with specialized ET Rules.

6.1. **Resource minimization by prioritized ET rules.** An MI solver in this theory receives a problem description and a sequence of ET rules as input, produces an ET sequence, and obtains the final answer when the computation reaches the domain of a given answer mapping [4]. In practical problem solving, resource for computation is limited. For each state $S = \langle Cs, \varphi \rangle$, the number of clauses as well as that of all atoms in Cs should not exceed a certain limit. Finding good computation control is essential for the success of solving problems in practical time. Prioritization of resource-restricted specialized ET rules is useful for resource minimization control. We try to minimize memory and time by adjusting prioritized ET rules. Based on the fact that a smaller amount of space consumption tends to decrease total execution time, we will design a set of ET rules and their priority basically so as to minimize space consumption at each state.

6.2. **Restricted resolution and restricted factoring.** Resolution adds a resolvent of two clauses and always increases the number of clauses by one. Let (res i) be defined as an ET rule for making a resolution step with a resolvent containing not more than i atoms. Since a smaller number of atoms are better for saving space and for finding simpler clauses, (res i) should have higher priority than (res j) when $i < j$.

Factoring also adds a clause; it always increases the number of clauses by one. Let (fac i) be defined as a factoring rule with a new clause containing not more than i atoms. Again, for finding simpler clauses and saving space, (fac i) should have higher priority than (fac j) when $i < j$.

6.3. **Restricted unfolding.** Unfolding decreases the number of clauses by one when the number of resolvents is zero, does not change the number of clauses when it produces only one resolvent, and increases the number of clauses otherwise.

Assume that Cs is a clause set. Unfolding of Cs with not more than i resolvents is a transformation rule that satisfies the following conditions.

1) Letting $D \subseteq Cs$ be a set of definite clauses used for unfolding, this rule is applicable to a body atom b in Cs when

$$|\{C \mid (C \in D) \ \& \ (b \text{ and } \mathit{head}(C) \text{ are unifiable})\}| \leq i.$$

2) The result of the transformation is the same as that of usual unfolding.

Let (udi i) be defined as a transformation rule that applies unfolding to an atom in the right-hand side of a clause if it results in not more than i resolvents. According to the principle of giving higher priorities to lower-splitting rules, (udi i) should have higher priority than (udi j) when $i < j$.

6.4. **A solution to the steamroller problem.** Let (erase) be an ET rule for erasing independent satisfiable atoms (see Section 5.3), (subsumed) an ET rule for elimination of subsumed clauses (see Section 5.4), and (fwd) the forwarding transformation rule, which erases an atom in the left-hand side of a clause by using a negative clause (see Section 5.5).

When we take the rule priority

(udi 1) $\geq$ (erase) $\geq$ (subsumed) $\geq$ (fwd) $\geq$ (udi 2) $\geq$ (udi 3) $\geq$ (udi 5),

the Steamroller puzzle is solved by 211 rule applications, where (udi 1) is applied 116 times, (erase) 15 times, (subsumed) 38 times, (fwd) 17 times, (udi 2) 8 times, (udi 3) 4 times, and (udi 5) 13 times. Clause splitting by (udi i) gives less than or equal to i clauses. If $i > 1$, the number of clauses may increase. Each of (erase) and (fwd) does not change the number of clauses. The rule (subsumed) decreases the number of clauses by one. As a result, we have the changes of the number of clauses as shown in Figure 3. As depicted in Figure 4, the number of all atoms in the clauses in each computation state also changes similarly.

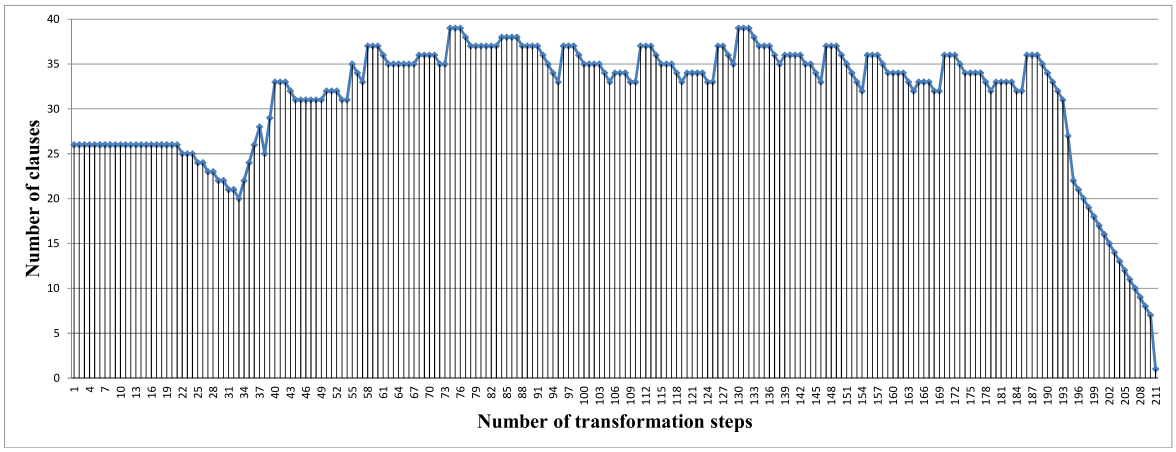

FIGURE 3. Changes of the number of clauses

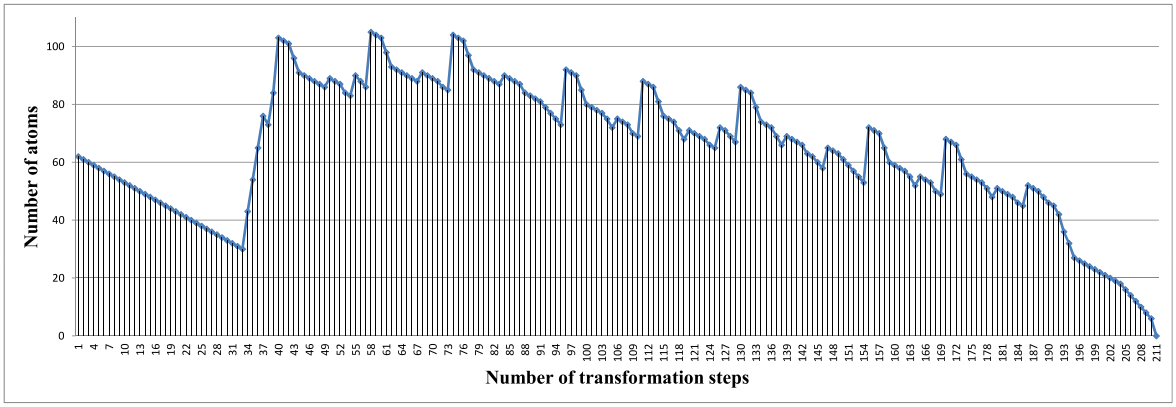

FIGURE 4. Changes of the number of atoms

6.5. **Comparison.** By deleting (udi 2) and (udi 3) from the above priority, we have

$$\text{(udi 1)} \geq \text{(erase)} \geq \text{(subsumed)} \geq \text{(fwd)} \geq \text{(udi 5)},$$

which gives only 90 steps to obtain the same solution. However, when we remove the rule (udi 1), i.e., when we take

$$\text{(erase)} \geq \text{(subsumed)} \geq \text{(fwd)} \geq \text{(udi 5)},$$

we need 269 steps to reach the final singleton of the empty clause, showing that prioritized application of (udi 1) is important for efficient computation.

So far we have introduced three priority controls, which are referred to as "Cont01", "Cont02", and "Cont03". Changes of the number of clauses resulting from these priority controls when solving the Steamroller puzzle are shown in Figure 5.

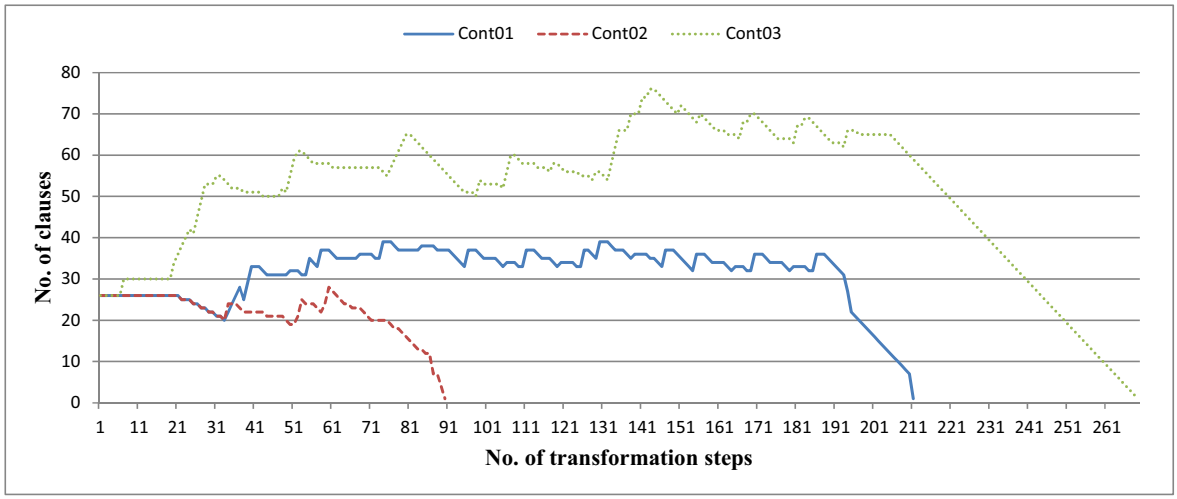

FIGURE 5. Computation control comparison

Since resolution and factoring are ET rules, the conventional resolution proof method is covered by our framework. For example, we can take prioritized ET rules (res 99) $\geq$ (fac 99) to solve proof problems by the resolution and factoring ET rules.

Figure 6 compares ET computation using the priority control "Cont01" with computation by resolution and factoring. Since each resolution step adds one resolvent of two clauses, each step increases the number of clauses by one. In the proof with resolution and factoring, computation goes on the upward straight line in Figure 6, which may exceed the space limitation before the computation terminates.

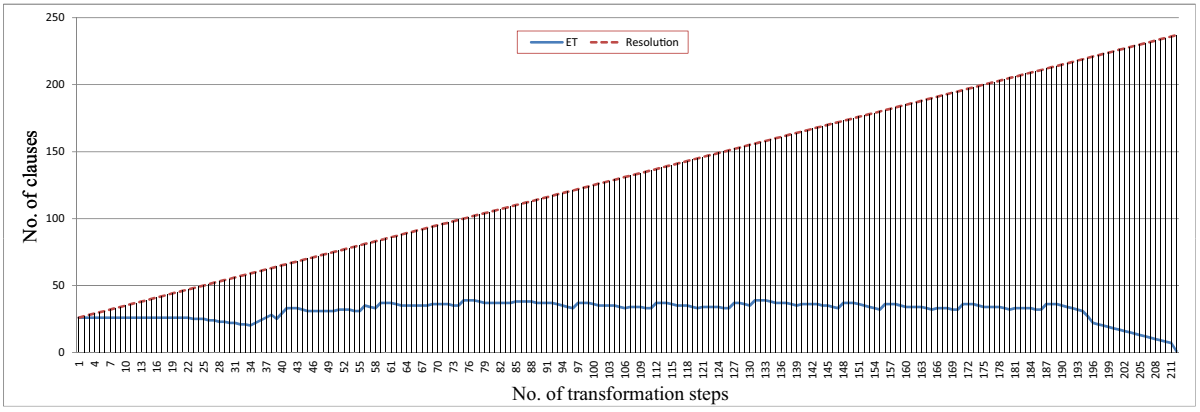

FIGURE 6. Comparing computation by ET with that by resolution and factoring

7. **Conclusions.** The conventional first-order logic has both representational limitation and computational limitation. LPSF is an axiomatic logic that overcomes these limitations by enabling the employment of various forms of extended logical formulas and various ET rules. Computation in the ET framework is successive application of an unlimited number of ET rules. Computation by ET rules extends the solvability of logical problems.

We have proposed a prioritized set of ET rules, which can be regarded as a sequence of ET rules. It gives a simple control mechanism, where the first applicable rule in the rule sequence is selected and applied to a problem. We can search for successful ET sequences with smaller cost by adjusting priorities of ET rules.

Since we have various ET rules, control with prioritized ET rules is very useful for better computation performance. With the principle of generating specialized ET rules

and assigning higher priorities to lower-splitting rules, resource minimization control of clauses can be realized. Resource minimization control is useful for finding an ET sequence that reaches a final problem description in finite steps, avoiding an infinite sequence without giving any answer to an originally given problem.

Control with prioritized ET rules is mainly intended to be used for unfolding-based top-down computation. Priority control with specialized ET rules extends the successful range of unfolding-based top-down search. One future research is to develop a new control by a mixture of top-down and bottom-up search in LPSF-based computation.

Control with prioritized ET rules can also be applied to the conventional proof theory. Each resolution/factoring step adds one clause in the conventional proof method. The number of clauses grows monotonically, which may exceed the space limitation before the computation reaches a given goal. By using non-splitting ET rules other than the resolution and factoring rules, we can suppress explosion of problem size and may avoid infinite computation without producing an answer. Resource minimization control using various ET rules is indispensable for overcoming the computational limitation of the conventional proof theory.

Acknowledgment. This work was partially supported by i) JSPS KAKENHI Grant Numbers 25280078 and 26540110, ii) the Center of Excellence in Intelligent Informatics, Speech and Language Technology and Service Innovation (CILS), Thammasat University, and iii) the Intelligent Informatics and Service Innovation (IISI) Center, Sirindhorn International Institute of Technology (SIIT), Thammasat University. The authors also gratefully acknowledge the helpful comments and suggestions from the reviewers.

REFERENCES

[1] K. Akama and E. Nantajeewarawat, Solving proof problems with equivalent transformation rules, *International Journal of Innovative Computing, Information and Control*, vol.18, no.1, pp.331-344, 2022.

[2] J. W. Lloyd, *Foundations of Logic Programming*, 2nd Edition, Springer-Verlag, 1987.

[3] K. Akama and E. Nantajeewarawat, Solving query-answering problems based-on equivalent transformation, *International Journal of Innovative Computing, Information and Control*, vol.18, no.5, pp.1547-1558, 2022.

[4] K. Akama and E. Nantajeewarawat, A foundation of logical problem solving, *International Journal of Innovative Computing, Information and Control*, vol.18, no.5, pp.1559-1570, 2022.

[5] K. Akama and E. Nantajeewarawat, Knowledge-representation-logic: An extension of first-order logic, *International Journal of Innovative Computing, Information and Control*, vol.18, no.4, pp.1055-1069, 2022.

[6] K. Akama and E. Nantajeewarawat, Skolemization that preserves logical meanings, *International Journal of Innovative Computing, Information and Control*, vol.17, no.1, pp.1-13, 2021.

[7] K. Akama and E. Nantajeewarawat, Formalization of logical problems as model-intersection problems on an extended clause space, *International Journal of Innovative Computing, Information and Control*, vol.17, no.4, pp.1103-1117, 2021.

[8] R. Manthey and F. Bry, SATCHMO: A theorem prover implemented in Prolog, *Proc. of the 9th International Conference on Automated Deduction*, Argonne, Illinois, pp.415-434, 1988.

[9] F. J. Pelletier, Seventy-five problems for testing automatic theorem provers, *Journal of Automated Reasoning*, vol.2, no.2, pp.191-216, 1986.

[10] M. Stickel, Schubert's steamroller problem: Formulations and solution, *Journal of Automated Reasoning*, vol.2, no.2, pp.89-104, 1986.

[11] C. Walther, A mechanical solution of Schubert's steamroller by many-sorted resolution, *Artificial Intelligence*, vol.26, no.2, pp.217-224, 1985.

[12] T. C. Wang and W. W. Bledsoe, Hierarchical deduction, *Journal of Automated Deduction*, vol.3, no.1, pp.35-77, 1987.

[13] K. Akama and E. Nantajeewarawat, Unfolding existentially quantified sets of extended clauses, *Proc. of the 8th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management*, vol.2, KEOD, Porto, Portugal, pp.96-103, 2016.

[14] K. Akama, E. Nantajeewarawat and T. Akama, Computation control by prioritized ET rules, *Proc. of the 10th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management*, vol.2, KEOD, Seville, Spain, pp.84-95, 2018.

Author Biography

Kiyoshi Akama received the B.Eng. and M.Eng. degrees in Control Engineering from Tokyo Institute Technology, Japan, in 1973 and 1975, respectively; and the D.Eng. degree in Control Engineering from Tokyo Institute Technology, Japan, in 1989. He was an assistant professor at Faculty of Engineering, Tokyo Institute Technology, Japan, 1979-1981; a lecturer at Faculty of Letters, Hokkaido University, Japan, 1981-1989; an associate professor at Faculty of Engineering, Hokkaido University, Japan, 1989-1999; a professor at Center for Multimedia Studies, Hokkaido University, Japan, 1999-2003; a professor at Information Initiative Center, Hokkaido University, Japan, 2003-2013; a specially-appointed professor at Information Initiative Center, Hokkaido University, 2013-2015; a professor at Graduate School of Information Science and Technology, Hokkaido University, Japan, 1999-2015.

Dr. Akama is currently an emeritus professor of Hokkaido University, Japan. His research interests include artificial intelligence, computer science, logic and computation, program generation and computation based on the equivalent transformation model, programming paradigms, and knowledge representation.

Ekawit Nantajeewarawat received the B.Eng. degree in Computer Engineering from Chulalongkorn University, Thailand, in 1987; and the M.Eng. and D.Eng. degrees in Computer Science from the Asian Institute of Technology, Thailand, in 1991 and 1997, respectively.

Dr. Nantajeewarawat is currently an associate professor of computer science at Sirindhorn International Institute of Technology, Thammasat University, Thailand. His research interests include knowledge representation, automated reasoning, rule-based equivalent transformation, program synthesis, formal ontologies, and object-oriented modelling.