International Journal of Innovative Computing, Information and Control
Volume **19**, Number **2**, April **2023**

ICIC International ©2023 ISSN 1349-4198
pp. 563–578

SOLVING PROOF PROBLEMS WITH BUILT-IN EQUALITY IN KR-LOGIC

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 November 2022

Abstract. *In human proofs of mathematical problems, such as proofs in group theory, term rewriting is widely used. However, they are not mathmatically correct proofs on first-order logic. The failure comes from the limitations of first-order logic itself. When we take first-order logic as the underlying logic, correct representation of unknown functions cannot be obtained due to the lack of representation power of the logic. We demonstrate that a mathmatically correct proof for a problem in group theory can be obtained on KR-Logic (knowledge-representation-logic), which is an extension of first-order logic and can represent built-in constraints and unknown functions correctly. We propose a class of term rewriting rules. A term rewriting rule preserves the sets of all models of formulas in KR-Logic. Representation by formulas and clauses in KR-Logic and computation by rewriting rules in KR-Logic are developed. This theory will provide a foundation for integrating logical inference and functional rewriting under a broader concept of equivalent transformation.*

Keywords: Proof problem, Query-answering problem, Function variable, Built-in equality, KR-Logic, Equivalent transformation, Term rewriting rule, Model preservation

1. **Introduction.** Term rewriting has been used in computation as a basic tool for transforming a term into a new term [1, 2, 3]. Terms are expressions with nested sub-expressions. A rewrite rule is applied by replacing an instance of its left-hand side with the same instance of its right-hand side. A term rewriting system (TRS) is a (finite or infinite) set of rewriting rules. Computation by term rewriting is repeated rewriting of terms by application of rewriting rules. Traditional TRSs play a role of a natural model of sequential computation. Given a word problem with a set of rewriting rules, the Knuth-Bendix completion algorithm (named after Knuth and Bendix [4]) tries to transform the set of equations that corresponds to the set of rewriting rules into a confluent term rewriting system to solve the word problem efficiently.

To consider relations between term rewriting and logical inference, we take a proof problem in group theory, which is to prove commutativity, i.e., $\forall x \in G$, $\forall y \in G$: $x \cdot y = y \cdot x$, assuming that a group G satisfies one additional property: $\forall x \in G$: $x \cdot x = e$. We introduce a name f for the mapping $(\cdot)$. Usually this problem is solved by term rewriting:

$$f(a,b) = f(e, f(a,b)) = f(f(f(b,a), f(b,a)), f(a,b)) = \cdots = f(b,a),$$

DOI: 10.24507/ijicic.19.02.563

i.e., we start with $f(a, b)$, and repeat application of rewriting rules, and we finally reach $f(b, a)$, which proves that $f(a, b) = f(b, a)$.

This proof is repeated term rewriting, while a logical proof is repeated inference (such as resolution). Term rewriting is not considered as inference. It is believed that term rewriting is different from logical reasoning [1]. In the conventional theory of logic, where resolution is a major method of proving theorems, proof by term rewriting cannot be formalized well. First-order logic fails to formally prove theorems in group theory by term rewriting.

We argue that the conventional theory of first-order logic should be extended into a new theory of logic so as to include proof by term rewriting as one of the major methods of proving theorems. LPSF (logical problem solving framework) [5] supports various logics by defining logics axiomatically. It can generate various logics. KR-Logic (knowledge-representation-logic) was created on LPSF as an extension of first-order logic [6]. KR-Logic overcomes the representational and computational limitations of first-order logic [6, 7, 8, 9].

We investigate the group problem based on the LPSF theory. We point out that proof by term rewriting cannot be formalized on first-order logic. We construct a theory of proof by term rewriting on KR-Logic. We propose a class of term rewriting rules on KR-Logic. Term rewriting rules preserve the sets of all models of formulas.

This theory will provide a foundation for understanding term rewriting from a logical side. Term rewriting rules, unfolding rules, and the resolution rule are all equivalent transformation (ET) rules that can be used to transform clauses equivalently. In ET-based computation, logical inference and functional rewriting co-exist, both of them being instances of a broader concept of equivalent transformation.

The rest of the paper is organized as follows. Section 2 takes a proof problem of group theory as a target problem to be formalized and solved. Its solution outline in first-order logic is explained. Section 3 reviews LPSF and explains how to apply LPSF for solving the proof problem. Section 4 formalizes the given problem using first-order logic with constraints and Herbrand semantics (LP-Logic). Section 5 formalizes the given problem using KR-Logic. Section 6 constructs a canonical logical structure of clauses in KR-Logic. Section 7 proposes a class of term rewriting rules. Section 8 gives correctness theorems for term rewriting. Section 9 shows how to solve the given proof problem correctly by repeated application of the proposed term rewriting rules. Section 10 concludes the paper.

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.

2. A Proof Problem and Its Informal Solution.

We take a proof problem in group theory and show an intuitive solution by term rewriting. This is an informal proof, not a mathematically correct proof. We will try to formalize this solution on two logics in subsequent sections.

2.1. A proof problem of a group.

Consider the following definition of a group: $\langle G, (\cdot), e, (^{-1})\rangle$ is a group iff

1) G is a set,
2) $(\cdot)$ is a mapping from $G \times G$ to G,
3) $e \in G$,
4) $(^{-1})$ is a mapping from G to G, and
5) they satisfy the following conditions:
   A1 (associativity): $\forall x, y, z \in G$: $x \cdot (y \cdot z) = (x \cdot y) \cdot z$
   A2 (identity): $\forall x \in G$: $x \cdot e = e \cdot x = x$

A3 (inverse): $\forall x \in G$, $\exists x^{-1} \in G$: $x \cdot x^{-1} = x^{-1} \cdot x = e$

Suppose that a group $\langle G, (\cdot), e, (^{-1}) \rangle$ with the three axioms above has one additional property:

$$\forall x \in G\colon\ x \cdot x = e.$$

Then we want to prove that this group has the property:

$$\text{(Commutativity) } \forall x \in G,\ \forall y \in G\colon\ x \cdot y = y \cdot x.$$

2.2. **An informal proof with equations.** Assume that

1) $(\cdot)$ is a mapping f from $G \times G$ to G, and
2) $(^{-1})$ is a mapping i from G to G.

Using these mappings, we have the following equations:

$$\begin{aligned}
&f(x, f(y, z)) = f(f(x, y), z)\\
&f(x, e) = x\\
&f(e, x) = x\\
&f(x, i(x)) = e\\
&f(i(x), x) = e\\
&f(x, x) = e
\end{aligned}$$

By replacement of terms using these equations, computation proceeds as follows:

$$\begin{aligned}
f(a, b) &= f(e, f(a, b))\\
&= f(f(f(b, a), f(b, a)), f(a, b))\\
&= f(f(b, a), f(f(b, a), f(a, b)))\\
&= f(f(b, a), f(b, f(a, f(a, b))))\\
&= f(f(b, a), f(b, f(f(a, a), b)))\\
&= f(f(b, a), f(b, f(e, b)))\\
&= f(f(b, a), f(b, b))\\
&= f(f(b, a), e)\\
&= f(b, a).
\end{aligned}$$

Starting with $f(a, b)$, we finally reach $f(b, a)$, which means that $f(a, b) = f(b, a)$.

2.3. **The plan of the paper.** The proof shown in this section is an informal proof, not a mathematically correct proof. The difficulty comes from its underlying logic, i.e., first-order logic. We will investigate the group problem based on the LPSF theory [5], since the LPSF theory can give many candidate logics. Among others, we take LP-Logic and KR-Logic. LP-Logic uses first-order formulas, while KR-Logic uses a superclass of first-order formulas.

- In Section 3, we will explain an LPSF-based solution method for logical problems.
- In Section 4, we will try to formalize the group problem on LP-Logic using function symbols and built-in equalities. We will conclude that LP-Logic fails to formalize the group problem correctly.
- In Section 5, we will try to formalize the group problem on KR-Logic using function variables and built-in equalities. We will conclude that KR-Logic succeeds in formalizing the group problem correctly.

In Sections 6, 7, 8 and 9, we will develop a method of solving the group problem on KR-Logic.

3. **How to Give Correctness to the Informal Method.** We explain how to apply LPSF for solving the proof problem in Section 2 in order to give correctness to the informal method.

3.1. **Logical problem solving based on LPSF.** We review the LPSF theory [5], which provides a general foundation for logical problem solving methods. LPSF 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 [10] as the main class of logical problems.
- LPSF takes equivalent transformation rules (ET rules) as units of procedures.

LPSF is a generator of logical problem solving methods. Input parameters of LPSF consist of (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. By changing the four parameters, LPSF can generate various logics, which may be new logics or existing ones [5]. By the generality covered by the four parameters, LPSF is useful for inventing new logics or comparing different logics.

Given a logical problem, we try to solve it using LPSF as follows.

1) Set a logical structure $\mathcal{L} = \langle \mathcal{K}, \mathcal{I}, \nu \rangle$.
2) Prepare a set R of ET rules.
3) Determine a control *ctrl*.
4) Determine a set A of answer mappings.
5) Obtain a solution method M determined by the four parameters, which is denoted by $M = \mathrm{LPSF}(\mathcal{L}, R, \mathit{ctrl}, A)$.
6) Formalize the given problem as an MI problem $\langle \mathit{Cs}, \varphi \rangle$, where $\mathit{Cs} \in \mathcal{K}$.
7) Solve the MI problem $\langle \mathit{Cs}, \varphi \rangle$ using M.

Given the above three parameters, i.e., a canonical logical structure ($\mathcal{L}$), an ET rule set (R), and a set of answer mappings (A), we obtain a solution method $M(\mathit{ctrl})$ for MI problems with a parameter *ctrl*:

$$M(\mathit{ctrl}) = \mathrm{LPSF}(\mathcal{L}, R, \mathit{ctrl}, A).$$

Each time we take a parameter *ctrl*, we can try to run $M(\mathit{ctrl})$ for solving problems.

3.2. **Applying LPSF to this proof problem.** To apply LPSF for solving the proof problem in Section 2.1, we consider the following four parameters of LPSF.

1) Canonical logical structure: We will explain that LP-Logic is not sufficient for the representation of the problem (Section 4). We explain a new logical structure on KR-Logic, consisting of extended clauses with existential quantification of function variables (Sections 5 and 6).
2) ET rules: We introduce a class of term rewriting rules from equational clauses in a given clause set (Section 7).
3) A set of answer mappings: We consider the singleton set $\{\mathit{ans}\}$ as a set of answer mappings, where the answer mapping *ans* is defined as follows: For any set *Cs* of clauses, $\mathit{ans}(\mathit{Cs}) = \mathit{yes}$ if *Cs* contains an empty clause ($\leftarrow$), and $\mathit{ans}(\mathit{Cs}) = \mathit{no}$ otherwise. The answer mapping *ans* means that when computation reaches a set of clauses with an empty clause ($\leftarrow$), the proof problem is solved affirmatively, i.e., it is proved.
4) Control: We take some control *ctrl*, the details of which are not discussed in this paper.

According to the LPSF theory, we will obtain a correctness-based computation for the proof problem (Sections 8 and 9).

4. **Representation in LP-Logic.** We use built-in constraints for representing equality. $\mathrm{FOL_c}$ is the set of all first-order formulas with built-in constraints. The logical structure of first-order logic with built-in constraints ($\mathrm{FOL_c}$) is called LP-Logic. We try to formalize the problem described in Section 2.1 in LP-Logic.

4.1. **Example: Formalization with formulas in LP-Logic.** The alphabet of LP-Logic is a quadruple $\langle \mathbb{F}, \mathbb{V}, \mathit{Pred}, \mathit{Pred}_{\mathrm{C}} \rangle$, where $\mathbb{F}$ is a set of usual function symbols, $\mathbb{V}$ is a set of usual variables, *Pred* is a set of user-defined predicate symbols, and $\mathit{Pred}_{\mathrm{C}}$ is a set of built-in constraint predicate symbols. Each element in $\mathbb{F}$ has an arity, which is a non-negative integer. A term in $\mathrm{T}(\mathbb{F})$ is called a *ground term*.

Assume that $g \in \mathit{Pred}$ and $g(x)$ represents the information that x is in the underlying set G of the group being considered. Let $eq \in \mathit{Pred}_{\mathrm{C}}$ be a built-in constraint predicate symbol defined by $eq(t_1, t_2)$ is true iff $t_1 = t_2$ for any ground terms t_1 and t_2. Assume that f, i, and e are binary, unary, and 0-ary function symbols, respectively, in $\mathbb{F}$, and x, y, and z are usual variables in $\mathbb{V}$. Let K denote the conjunction of the following first-order formulas:

F_1: $\forall x \forall y$: $(g(x) \wedge g(y) \rightarrow g(f(x, y)))$
F_2: $\forall x$: $(g(x) \rightarrow g(i(x)))$
F_3: $g(e)$
F_4: $\forall x \forall y \forall z$: $((g(x) \wedge g(y) \wedge g(z)) \rightarrow eq(f(x, f(y, z)), f(f(x, y), z)))$
F_5: $\forall x$: $(g(x) \rightarrow eq(f(x, e), x))$
F_6: $\forall x$: $(g(x) \rightarrow eq(f(e, x), x))$
F_7: $\forall x$: $(g(x) \rightarrow eq(f(x, i(x)), e))$
F_8: $\forall x$: $(g(x) \rightarrow eq(f(i(x), x), e))$
F_9: $\forall x$: $(g(x) \rightarrow eq(f(x, x), e))$

We want to prove that $K \rightarrow (\forall x \forall y$: $(g(x) \wedge g(y)) \rightarrow eq(f(x, y), f(y, x)))$ is valid, i.e., it is true with respect to every interpretation. Its negation is $K \wedge \neg(\forall x \forall y$: $(g(x) \wedge g(y)) \rightarrow eq(f(x, y), f(y, x)))$, which is equivalent to $K \wedge (\exists x \exists y$: $(g(x) \wedge g(y) \wedge \neg eq(f(x, y), f(y, x))))$.

4.2. **Inappropriateness of formalization in LP-Logic.**

4.2.1. *Inappropriateness of LP-Logic.* We discuss in this section the limited expressive power of LP-Logic, where the Herbrand semantics is taken for first-order logic with built-in constraints ($\mathrm{FOL_c}$). Although it is considered to be natural for representation of and computation for logical problems, we show that LP-Logic lacks correct representation of evaluable terms.

The predicate *eq* is defined based on the relation '=' on ground terms. Since '=' means syntactical coincidence, e.g., if *cons* and *nil* are ground terms, we have $eq(nil, nil)$ and $eq(cons(nil, nil), cons(nil, nil))$.

The fact that f, i, and e are function symbols in $\mathbb{F}$, however, results in inconvenience as follows: Consider the atom $eq(f(x, e), x)$ as an example. What is the meaning of this equality? It says the term $f(x, e)$ is equal to the another term x. Whatever value the variable x may take, however, $f(x, e)$ cannot be equal to x due to the inclusion of x inside $f(x, e)$.

4.2.2. *Towards KR-Logic.* The intuitive meaning of the equality $eq(f(x, e), x)$ should be as follows: f computes a value, say a term t, from x and e, and the resulting term t is equal to x. f should not be a function symbol in $\mathbb{F}$, but an evaluable function. Hence, intuitively, we need two types of "functions", i.e.,

1) constructors for constructing compound terms, and

2) evaluable functions for mapping ground terms.

$\mathrm{FOL_c}$ (together with Herbrand semantics) supports only the first type of functions, while we need the second type of functions for representing $(\cdot)$, e and $(^{-1})$. This is the reason why we cannot represent this proof problem correctly in $\mathrm{FOL_c}$.

5. **Representation in KR-Logic.** We use function variables for representing evaluable functions. $\mathrm{KRL_c}$ is the set of all first-order formulas with built-in constraints and function variables. The logical structure consisting of formulas in $\mathrm{KRL_c}$ is called KR-Logic. We try to formalize the problem described in Section 2.1 in KR-Logic.

5.1. **Alphabet and terms in KR-Logic.** An alphabet $\langle \mathbb{F}, \mathbb{V}, \mathbb{FC}, \mathbb{FV}, \mathit{Pred}, \mathit{Pred}_{\mathrm{C}} \rangle$ is assumed, where $\mathbb{F}$ is a set of constructors, $\mathbb{V}$ is a set of usual variables, $\mathbb{FC}$ is a set of function constants, $\mathbb{FV}$ is a set of function variables, *Pred* is a set of user-defined predicate symbols, and $\mathit{Pred}_{\mathrm{C}}$ is a set of built-in constraint predicate symbols. Each element in $\mathbb{F} \cup \mathbb{FC} \cup \mathbb{FV}$ is associated with a non-negative integer, called its arity.

Definition 5.1. *A* term *on* $\langle \mathbb{F}, \mathbb{V}, \mathbb{FC}, \mathbb{FV} \rangle$*, which is also simply called a term, is inductively defined as follows.*

1) A 0-ary element in $\mathbb{F} \cup \mathbb{FC} \cup \mathbb{FV}$ *is a term.*
2) If $v \in \mathbb{V}$*, then* v *is a term.*
3) If $f \in \mathbb{F} \cup \mathbb{FC} \cup \mathbb{FV}$*, the arity of* f *is* $n > 0$*, and* $t_1, \ldots, t_n$ *are terms, then* $f(t_1, \ldots, t_n)$ *is a term.*

The set of all terms on $\langle \mathbb{F}, \mathbb{V}, \mathbb{FC}, \mathbb{FV} \rangle$ is denoted by $\mathrm{T}(\mathbb{F}, \mathbb{V}, \mathbb{FC}, \mathbb{FV})$. Let $\mathrm{T}(\mathbb{F}) = \mathrm{T}(\mathbb{F}, \emptyset, \emptyset, \emptyset)$. Let $\mathrm{T}(\mathbb{F}, \mathbb{FC}) = \mathrm{T}(\mathbb{F}, \emptyset, \mathbb{FC}, \emptyset)$. A term in $\mathrm{T}(\mathbb{F})$ is called a *ground term*.

5.2. **Atoms and WFFs.** A *user-defined atom* takes the form $p(t_1, \ldots, t_n)$, where p is an n-ary user-defined predicate in *Pred* and the t_i are terms. A *built-in constraint atom*, also simply called a *constraint atom* or a *built-in atom*, takes the form $c(t_1, \ldots, t_n)$, where c is an n-ary built-in constraint predicate in $\mathit{Pred}_{\mathrm{C}}$ and the t_i are terms. A *func-atom* [11] is an expression of the form $\mathit{func}(f, t_1, \ldots, t_n, t_{n+1})$, where f is either an n-ary function constant in $\mathbb{FC}$ or an n-ary function variable in $\mathbb{FV}$, and the t_i are terms. An atom is a *ground* atom if all of its arguments are ground terms. Let $\mathcal{A}_{\mathrm{u}}$ be the set of all user-defined atoms, $\mathcal{A}_{\mathrm{c}}$ the set of all built-in constraint atoms, and $\mathcal{F}$ the set of all *func*-atoms.

A *well-formed formula* (for short, *wff*) is constructed inductively by finitely many applications of the following rules.

1) *Atoms:* If α is an atom in $\mathcal{A}_{\mathrm{u}} \cup \mathcal{A}_{\mathrm{c}} \cup \mathcal{F}$, then α is a wff.
2) *Negation:* If α is a wff, then $\neg\alpha$ is a wff.
3) *Binary connectives:* If α and β are wffs, then $\alpha \wedge \beta$, $\alpha \vee \beta$, $\alpha \rightarrow \beta$, and $\alpha \leftrightarrow \beta$ are wffs.
4) *Quantification for usual variables:* If α is a wff and v is a variable in $\mathbb{V}$, then $\forall v\colon \alpha$ and $\exists v\colon \alpha$ are wffs.
5) *Quantification for function variables:* If α is a wff and f_v is a variable in $\mathbb{FV}$, then $\forall f_v\colon \alpha$ and $\exists f_v\colon \alpha$ are wffs.

5.3. **Example: Formalization with formulas in KR-Logic.** Let $g \in \mathit{Pred}$, with $g(x)$ being intended to mean that x is in the underlying set G of the group under consideration. Let $\mathit{eq} \in \mathit{Pred}_{\mathrm{C}}$ be a built-in predicate defined by $\mathit{eq}(t_1, t_2)$ is true iff $t_1 = t_2$ for any $t_1, t_2 \in \mathrm{T}(\mathbb{F})$. Assume that x, y, and z are variables in $\mathbb{V}$. Assume also that $\$f$, $\$i$, and $\$e$ are binary, unary, and 0-ary function variables, respectively, in $\mathbb{FV}$. Let K' denote the conjunction of the following formulas in KR-Logic:

F_1: $\forall x \forall y$: $(g(x) \wedge g(y) \rightarrow g(\$f(x, y)))$
F_2: $\forall x$: $(g(x) \rightarrow g(\$i(x)))$
F_3: $g(\$e)$
F_4: $\forall x \forall y \forall z$: $((g(x) \wedge g(y) \wedge g(z)) \rightarrow eq(\$f(x, \$f(y, z)), \$f(\$f(x, y), z)))$
F_5: $\forall x$: $(g(x) \rightarrow eq(\$f(x, \$e), x))$
F_6: $\forall x$: $(g(x) \rightarrow eq(\$f(\$e, x), x))$
F_7: $\forall x$: $(g(x) \rightarrow eq(\$f(x, \$i(x)), \$e))$
F_8: $\forall x$: $(g(x) \rightarrow eq(\$f(\$i(x), x), \$e))$
F_9: $\forall x$: $(g(x) \rightarrow eq(\$f(x, x), \$e))$

The only difference between the representation in Section 4.1 and that in this section is the change of function symbols such as f, i, and e in $\mathbb{F}$ into function variables in $\mathbb{FV}$. Function symbols such as *cons* and *nil* in $\mathbb{F}$ remain in $\mathbb{F}$. Syntactically, the prefix '$' is added to f, i, and e to make $\$f$, $\$i$, and $\$e$, while the others are unchanged.

The problem is then formalized as follows:

$$\forall \$f \forall \$i \forall \$e\text{: } K' \rightarrow (\forall x \forall y\text{: } (g(x) \wedge g(y)) \rightarrow eq(\$f(x, y), \$f(y, x))).$$

Let E denote the above formula. To solve the proof problem, we want to prove that $Models(\neg E) = \emptyset$, where $Models(\neg E)$ denotes the set of all models of $\neg E$. Note that this formalization in KR-Logic is similar to the formalization in first-order logic except (i) the use of function variables in place of function symbols, and (ii) existential quantification of function variables at the top of the formula $\neg E$.

6. A Canonical Logical Structure in KR-Logic.

We use clauses for computation. We describe a new logical structure consisting of extended clauses with existential quantification of function variables in KR-Logic.

6.1. Extended clauses in KR-Logic.

There are two types of variables: usual variables and function variables. An *extended clause* C on $\mathcal{A}_{\rm u} \cup \mathcal{A}_{\rm c} \cup \mathcal{F}$ is a formula of the form

$$a_1, \ldots, a_m \leftarrow b_1, \ldots, b_n, \mathbf{f}_1, \ldots, \mathbf{f}_p,$$

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 built-in constraint atom in $\mathcal{A}_{\rm c}$, and $\mathbf{f}_1, \ldots, \mathbf{f}_p$ are *func*-atoms.

All usual variables occurring in C are implicitly universally quantified and their scope is restricted to the extended clause C itself. Given a set Cs of extended clauses, all function variables occurring in Cs are implicitly existentially quantified at the top level of Cs, and their scope covers all the clauses in Cs. The set of all extended clauses is denoted by $\mathrm{ECLS_N}$. An extended clause will also be called a *clause* when no confusion is caused.

A function variable can be instantiated into function constants. (fev) is a specialization for function evaluation. A function constant f_c of arity 0 can be reduced by (fev) into a ground term g in $\mathrm{T}(\mathbb{F})$ according to the function associated with f_c, which is denoted by $f_c \circ (fev) = g$. A term $f_c(t_1, t_2, \ldots, t_n)$ such that f_c is a function constant of arity $n \geq 1$ and $t_1, t_2, \ldots, t_n$ are ground terms in $\mathrm{T}(\mathbb{F})$ can be reduced by (fev) to a ground term g in $\mathrm{T}(\mathbb{F})$ according to the function associated with f_c, which is denoted by $f_c(t_1, t_2, \ldots, t_n) \circ (fev) = g$. Given a term t, all subterms contained in t are specialized recursively by (fev). For instance, $add1(one) \circ (fev) = 2$ if $one \circ (fev) = 1$ and $add1(1) \circ (fev) = 2$. Let $bind(\mathbb{V})$ be the set of all bindings on $\mathbb{V}$. A substitution on $\mathbb{V}$ is a sequence of elements in $bind(\mathbb{V})$. Let $\mathcal{S}_\theta$ be the set of all substitutions on $\mathbb{V}$. Let $bind(\mathbb{FV})$ be the set of all bindings on $\mathbb{FV}$. A substitution on $\mathbb{FV}$ is a sequence of elements in $bind(\mathbb{FV})$. Let $\mathcal{S}_\sigma$ be the set of all substitutions on $\mathbb{FV}$. Let GCL be the set of all ground clauses consisting of only ground user-defined atoms with no occurrence of any function constant in $\mathbb{FC}$. A clause in GCL is obtained from an extended clause in $\mathrm{ECLS_N}$ by (1) instantiation of variables in $\mathbb{FV}$, (2)

instantiation of variables in $\mathbb{V}$, (3) evaluation of all evaluable subterms in $\mathrm{T}(\mathbb{F}, \mathbb{FC})$ by (*fev*) into ground terms in $\mathrm{T}(\mathbb{F})$, and (4) removal of true ground built-in atoms and true ground *func*-atoms in the resulting clause body.

6.2. **Models.** There are two kinds of variables: usual variables in $\mathbb{V}$ and function variables in $\mathbb{FV}$. In general, terms that appear in a clause are in $\mathrm{T}(\mathbb{F}, \mathbb{V}, \mathbb{FC}, \mathbb{FV})$. Variables in $\mathbb{V}$ are specialized by a substitution in $\mathcal{S}_\theta$. Function variables in $\mathbb{FV}$ are specialized by a substitution in $\mathcal{S}_\sigma$. There is one more specialization (*fev*), which is for function evaluation. Terms in $\mathrm{T}(\mathbb{F}, \emptyset, \mathbb{FC}, \emptyset)$ are specialized by (*fev*) into terms in $\mathrm{T}(\mathbb{F}, \emptyset, \emptyset, \emptyset)$. A clause may be instantiated into a ground clause with terms in $\mathrm{T}(\mathbb{F})$ by application of $\theta \in \mathcal{S}_\theta$, $\sigma \in \mathcal{S}_\sigma$ and (*fev*).

An interpretation is a subset of $\mathcal{G}_\mathrm{u}$. Let *Cs* be a set of clauses. An interpretation I is a *model* of *Cs* iff *Cs* is true with respect to I. Let *Models*(*Cs*) denote the set of all models of *Cs*. A clause is ground iff all terms occurring in it are in $\mathrm{T}(\mathbb{F})$. If *Cs*′ is the set of all ground instances of clauses in *Cs*, then *Models*(*Cs*) = *Models* (*Cs*′).

6.3. **Example: Formalization in clausal form in KR-Logic.** Referring to E and K' in Section 5.3, the negation of E is $\exists \$f \exists \$i \exists \$e$: $K' \wedge \neg(\forall x \forall y$: $(g(x) \wedge g(y)) \rightarrow eq(\$f(x,y), \$f(y,x)))$, which is equivalent to

$$\exists \$f \exists \$i \exists \$e\text{: } K' \wedge (\exists x \exists y\text{: } (g(x) \wedge g(y) \wedge \neg eq(\$f(x,y), \$f(y,x)))).$$

By lifting up the existential quantifiers ($\exists x$ and $\exists y$) using new function variables ($\$a$ and $\$b$), we have

$$\exists \$f \exists \$i \exists \$e \exists \$a \exists \$b\text{: } K' \wedge g(\$a) \wedge g(\$b) \wedge \neg eq(\$f(\$a, \$b), \$f(\$b, \$a)).$$

Hence, we obtain the following extended clauses:

C_1: $g(\$f(x,y)) \leftarrow g(x), g(y)$
C_2: $g(\$i(x)) \leftarrow g(x)$
C_3: $g(\$e) \leftarrow$
C_4: $eq(\$f(x, \$f(y,z)), \$f(\$f(x,y), z)) \leftarrow g(x), g(y), g(z)$
C_5: $eq(\$f(x, \$e), x) \leftarrow g(x)$
C_6: $eq(\$f(\$e, x), x) \leftarrow g(x)$
C_7: $eq(\$f(x, \$i(x)), \$e) \leftarrow g(x)$
C_8: $eq(\$f(\$i(x), x), \$e) \leftarrow g(x)$
C_9: $eq(\$f(x,x), \$e) \leftarrow g(x)$
C_{10}: $g(\$a) \leftarrow$
C_{11}: $g(\$b) \leftarrow$
C_{12}: $\leftarrow eq(\$f(\$a, \$b), \$f(\$b, \$a))$

K' is the conjunction of $F_1, F_2, \ldots, F_9$ in Section 5.3 and for each $i \in \{1, 2, \ldots, 9\}$, C_i is an extended clause that is obtained from F_i. The conjunction of C_{10}, C_{11} and C_{12} corresponds to $g(\$a) \wedge g(\$b) \wedge \neg eq(\$f(\$a, \$b), \$f(\$b, \$a))$. Let $\tilde{Cs} = \{C_1, C_2, \ldots, C_{12}\}$. Then we can solve the proof problem formalized in Section 5.3 by proving that $\mathit{Models}\left(\tilde{Cs}\right) = \emptyset$.

6.4. **A canonical logical structure of extended clauses.** Let $\mathcal{G}_\mathrm{t}$ be defined as $\mathrm{T}(\mathbb{F})$. Let $\mathcal{G}_\mathrm{u}$ be defined as the set of all ground atoms of the form $p(t_1, \ldots, t_m)$, where p is an m-ary user-defined predicate in *Pred* and $t_1, \ldots, t_m$ are terms in $\mathcal{G}_\mathrm{t}$.

The main role of a logical structure is to determine the truth or falsity of each formula with respect to an interpretation $G \in \mathit{pow}(\mathcal{G}_\mathrm{u})$. We define a mapping ν: $\mathit{pow}(\mathrm{ECLS_N}) \rightarrow \mathit{Map}(\mathit{pow}(\mathcal{G}_\mathrm{u}), \{\mathit{true}, \mathit{false}\})$ as follows: For any $Cs \subseteq \mathrm{ECLS_N}$ and any $G \subseteq \mathcal{G}_\mathrm{u}$, $\nu(Cs)(G) = \mathit{true}$ iff there exists $\sigma \in \mathcal{S}_\sigma$ such that for any $C \in Cs$ and any $\theta \in \mathcal{S}_\theta$, if $C\sigma\theta \circ (\mathit{fev}) \in GCL$, then $C\sigma\theta \circ (\mathit{fev})$ is true with respect to G. For completion of defining ν, the

truth value of a ground clause in GCL with respect to a subset G of $\mathcal{G}_{\mathrm{u}}$ is defined as follows: Letting $a_1, a_2, \ldots, a_n \in \mathcal{G}_{\mathrm{u}}$ and $b_1, b_2, \ldots, b_m \in \mathcal{G}_{\mathrm{u}}$, the clause $(a_1, a_2, \ldots, a_n \leftarrow b_1, b_2, \ldots, b_m)$ is true with respect to G iff $a_1 \in G$ or $a_2 \in G$ or $\ldots$ or $a_n \in G$ or $b_1 \notin G$ or $b_2 \notin G$ or $\ldots$ or $b_m \notin G$.

$\mathcal{L} = \langle pow(\mathrm{ECLS_N}), pow(\mathcal{G}_{\mathrm{u}}), \nu \rangle$ is a canonical logical structure, i.e., (1) $\mathcal{L}$ is a logical structure, and (2) $\mathcal{L}$ has $pow(\mathcal{G}_{\mathrm{u}})$ as the interpretation domain.

7. Term Rewriting Rules.

We introduce a class of *term rewriting rules* based on equational clauses in a given clause set.

7.1. Term contexts, atom contexts, and clause contexts.

Assume that $\Box$ is a special symbol with arity 0 that does not belong to $\mathbb{F} \cup \mathbb{V} \cup \mathbb{FC} \cup \mathbb{FV}$. A *term context* is a term that contains only one occurrence of $\Box$ at a subterm position. A subterm is a part of a term, and a subterm is extended by a term context into a term. Let t be a term and tc a term context. $t \rhd tc$ is the term $tc\{\Box/t\}$. An *atom context* is an atom that contains only one occurrence of $\Box$ at a term position (an argument position). A term is a part of an atom, and a term is extended by an atom context into an atom. Let t be a term and ac an atom context. $t \rhd ac$ is the atom $ac\{\Box/t\}$. A *clause context* is a clause that contains only one occurrence of $\Box$ at an atom position. An atom is a part of a clause, and an atom is extended by a clause context into a clause. Let a be an atom and con a clause context. $a \rhd con$ is the clause $con\{\Box/a\}$. For example, $((x \rhd f(\Box)) \rhd q(\Box)) \rhd (p(5) \leftarrow \Box, r(A)) = (f(x) \rhd q(\Box)) \rhd (p(5) \leftarrow \Box, r(A)) = q(f(x)) \rhd (p(5) \leftarrow \Box, r(A)) = (p(5) \leftarrow q(f(x)), r(A))$. The sets of all term contexts, all atom contexts, and all clause contexts are denoted, respectively, by Con_T, Con_A, and Con_C.

7.2. Term rewriting rules.

We define equational clauses and term rewriting rules. An equational clause produces term rewriting rules.

7.2.1. *Equational clauses.* An *equational clause* is a clause in $\mathrm{ECLS_N}$ of the form

$$eq(t_1, t_2) \leftarrow conds,$$

where t_1 and t_2 are terms in $\mathrm{T}(\mathbb{F}, \mathbb{V}, \mathbb{FC}, \mathbb{FV})$, and $conds$ is a conjunction of atoms in $\mathcal{A}_{\mathrm{u}}$.

7.2.2. *Term rewriting rules.* A *term rewriting rule*, *TRR* for short, is a formula of the form

$$(t_1 \rightarrow t_2\colon\ conds),$$

where t_1 and t_2 are terms in $\mathrm{T}(\mathbb{F}, \mathbb{V}, \mathbb{FC}, \mathbb{FV})$, and $conds$ is a conjunction of atoms in $\mathcal{A}_{\mathrm{u}}$ ($conds$ is denoted by set notation). A term rewriting rule r of the form $(t_1 \rightarrow t_2\colon\ conds)$ is often represented by $r\colon\ (t_1 \rightarrow t_2\colon\ conds)$.

7.2.3. *From an equational clause to TRRs.* An equational clause $C = (eq(t_1, t_2) \leftarrow conds)$ determines a set of two term rewriting rules, which is denoted by $trs(C)$, given by

$$trs(C) = \{(t_1 \rightarrow t_2\colon\ conds), (t_2 \rightarrow t_1\colon\ conds)\}.$$

7.3. Example: Term rewriting rules for the sample problem.

From the equational clause

$$C = (eq(\$f(x, \$e), x) \leftarrow g(x)),$$

we have two rewriting rules, i.e., $r\colon\ (\$f(x, \$e) \rightarrow x\colon\ \{g(x)\})$ and $r'\colon\ (x \rightarrow \$f(x, \$e)\colon\ \{g(x)\})$. Consider the clauses given in Section 6.3. Since each equational clause produces two term rewriting rules, we obtain the following twelve term rewriting rules from the clause set $\{C_4, C_5, C_6, C_7, C_8, C_9\}$:

r_1: $(x \to \$f(x, \$e)\colon \{g(x)\})$ r_2: $(\$f(x, \$e) \to x\colon \{g(x)\})$
r_3: $(x \to \$f(\$e, x)\colon \{g(x)\})$ r_4: $(\$f(\$e, x) \to x\colon \{g(x)\})$
r_5: $(\$e \to \$f(x, \$i(x))\colon \{g(x)\})$ r_6: $(\$f(x, \$i(x)) \to \$e\colon \{g(x)\})$
r_7: $(\$e \to \$f(\$i(x), x)\colon \{g(x)\})$ r_8: $(\$f(\$i(x), x) \to \$e\colon \{g(x)\})$
r_9: $(\$e \to \$f(x, x)\colon \{g(x)\})$ r_{10}: $(\$f(x, x) \to \$e\colon \{g(x)\})$
r_{11}: $(\$f(\$f(x, y), z) \to \$f(x, \$f(y, z))\colon \{g(x), g(y), g(z)\})$
r_{12}: $(\$f(x, \$f(y, z)) \to \$f(\$f(x, y), z)\colon \{g(x), g(y), g(z)\})$

7.4. **Term rewriting procedure.**

7.4.1. *Assumptions and basic definitions.* Consider $\tilde{C}s = \{C_1, C_2, \ldots, C_{12}\}$ in Section 6.3. $\tilde{C}s$ consists of the equational clauses $C_4, C_5, C_6, C_7, C_8, C_9$, the definite clauses C_1, C_2, C_3, C_{10}, C_{11}, and the query clause C_{12}. Let P_1 and P_2 be sets of predicates. Let $clause(P_1, P_2)$ be the set of all clauses such that the predicates on their left-hand sides are in P_1 and the predicates on their right-hand sides are in P_2. Let $negativeClause(P_2)$ be the set of all negative clauses such that the predicates on their right-hand sides are in P_2.

Let P be a set of user-defined predicates in *Pred*. We assume in this paper that we are given a target clause set Cs such that $Cs = Cs_{eq} \cup D \cup Q$, where Cs_{eq} is a set of equational clauses in $clause(\{eq\}, P)$, D is a set of definite clauses in $clause(P, P)$, and Q is a set of query clauses in $negativeClause(\{eq\})$. We also assume that all terms occurring in Cs are in $\mathrm{T}(\mathbb{F}, \mathbb{V}, \emptyset, \mathbb{FV})$.

Assume that a term $\$f(\$f(\$b, \$a), \$e)$ in an atom $eq(\$f(\$f(\$b, \$a), \$e), \$f(\$b, \$a))$ is given. We take the rewriting rule r_2: $(\$f(x, \$e) \to x\colon \{g(x)\})$. By matching $\$f(x, \$e)$ with $\$f(\$f(\$b, \$a), \$e)$, we have a substitution $\{x/\$f(\$b, \$a)\}$. We check whether $g(\$f(\$b, \$a))$ is true. The term $\$f(\$f(\$b, \$a), \$e)$ in the atom $eq(\$f(\$f(\$b, \$a), \$e), \$f(\$b, \$a))$ is then replaced with $\$f(\$b, \$a)$ to have a new atom $eq(\$f(\$b, \$a), \$f(\$b, \$a))$ since $g(\$f(\$b, \$a))$ is true.

When we check whether $g(\$f(\$b, \$a))$ is true, all function variables in this atom are regarded as constants. We know that $g(\$f(\$b, \$a))$ follows from $\tilde{D} = \{C_1, C_2, C_3, C_{10}, C_{11}\}$, which is denoted by $\tilde{D} \models_V g(\$f(\$b, \$a))$. Since $\models_V$ does not consider instantiation of function variables, $\models_V$ is different from the usual logical consequence relation $\models$. For checking the relation $\models_V$, SLD resolution can be used by instantiation of only variables in $\mathbb{V}$ as follows:

$\leftarrow g(\$f(\$b, \$a))$ (by C_1: $g(\$f(x, y)) \leftarrow g(x), g(y)$)
$\leftarrow g(\$b), g(\$a)$ (by C_{11}: $g(\$b) \leftarrow$)
$\leftarrow g(\$a)$ (by C_{10}: $g(\$a) \leftarrow$)
$\leftarrow$

Since function variables are dealt with as if they were constants here, this computation shows that the atom $g(\$f(\$b, \$a))$ follows from the definite clauses in $\tilde{D}$ for any instantiation of function variables $\$f$, $\$b$, and $\$a$. Hence, we can conclude that $Models\left(\tilde{C}s\right) = Models\left(\tilde{C}s \cup \{(eq(\$f(\$f(\$b, \$a), \$e), \$f(\$b, \$a)) \leftarrow)\}\right)$, which gives a basis for replacing $\$f(\$f(\$b, \$a), \$e)$ with $\$f(\$b, \$a)$. More generally, an equational clause $C = (eq(t_1, t_2) \leftarrow conds)$ is specialized into $C\theta = (eq(t_1\theta, t_2\theta) \leftarrow conds\theta)$ by a matcher θ. If checking $(D \models_V conds\theta)$ is successful, we know that $Models(Cs) = Models(Cs \cup \{(eq(t_1\theta, t_2\theta) \leftarrow)\})$.

7.4.2. *Procedure for transformation by term rewriting rules.* A term rewriting rule can produce correct transformation if it has a corresponding equational clause in a given clause set Cs.

Definition 7.1. *Let Cs be a set of clauses. Let* $r = (t_1 \to t_2$*: conds*$)$ *be a term rewriting rule. The rule r is admissible to Cs iff Cs contains the equational clause* $C = (eq(t_1, t_2) \leftarrow$ *conds*$)$.

We propose a procedure for transformation by term rewriting rules, i.e., given a clause set Cs, a clause C_1 and a term rewriting rule $r = (t_1 \to t_2$: *conds*$)$, r is applied to Cs and C_1 to producing a new clause C_2 as follows.

Procedure $TRR(r, Cs, C_1)$

1) Check whether the term rewriting rule $r = (t_1 \to t_2$: *conds*$)$ is admissible to Cs.
2) Find a subterm T_1 in C_1 to determine a term context tc, an atom context ac, and a clause context *con* such that $C_1 = (((T_1 \rhd tc) \rhd ac) \rhd con)$.
3) Match t_1 with T_1 to determine a substitution θ such that $t_1\theta = T_1$.
4) Check whether $D \models_V conds\theta$.
5) Determine $T_2 = t_2\theta$.
6) Make $C_2 = (((T_2 \rhd tc) \rhd ac) \rhd con)$ by replacement of T_1 with T_2 in C_1.

Definition 7.2. *We write* $C_1 \xrightarrow{r,Cs} C_2$ *to denote that a clause* C_1 *is transformed into a clause* C_2 *by a term rewriting rule r in the existence of a set Cs of clauses according to the procedure TRR.*

8. **Correctness of Term Rewriting Rules.** Assuming that $Cs = Cs_{eq} \cup D \cup Q$ is a given set of clauses according to Section 7.4, we give correctness theorems for term rewriting.

8.1. **Term rewriting based on equality of terms.** Refer to $\tilde{Cs} = \{C_1, C_2, \ldots, C_{12}\}$ in Section 6.3. Consider the clause C_5:

$$C_5\text{: } eq(\$f(x, \$e), x) \leftarrow g(x).$$

Assume that a target clause contains a term $\$f(\$b, \$e)$. By matching $\$f(x, \$e)$ with $\$f(\$b, \$e)$ without specializing function variables, we have a substitution $\theta = \{x/\$b\}$. Then $C_5\theta = (eq(\$f(\$b, \$e), \$b) \leftarrow g(\$b))$. Since $g(\$b)$ is true by C_{11}, we have an equational unit clause:

$$EQ_1\text{: } eq(\$f(\$b, \$e), \$b) \leftarrow .$$

EQ_1 means that the results of application of some function-variable specialization and (fev) on both sides are the same, i.e., $\$f(\$b, \$e)\sigma \circ (fev) = \$b\sigma \circ (fev)$ for some function-variable specialization σ. Based on the equality, we can replace the term $\$f(\$b, \$e)$ with the term $\$b$. This replacement is the result of an application of the rewriting rule r_2: $(\$f(x, \$e) \to x$: $\{g(x)\})$ to the target clause.

8.2. **Definitions and propositions.** Before proving in Section 8.3 that term rewriting rules proposed in this paper preserve models, we introduce some concepts and propositions to be used for the proof.

A clause set Cs determines an equivalently-rewriting relation on $\mathrm{ECLS_N}$, denoted by $rel(Cs)$, as follows.

Definition 8.1. *Let* $\mathcal{S}_\theta$ *be the set of all substitutions on* $\mathbb{V}$*. Let Cs be a set of clauses. Let* t_1 *and* t_2 *be terms in* $\mathrm{T}(\mathbb{F}, \mathbb{V}, \mathbb{FC}, \mathbb{FV})$*. Let* θ *be a substitution in* $\mathcal{S}_\theta$*. Define* $rel(Cs, t_1, t_2, \theta)$ *and* $rel(Cs)$ *as follows:*

- $rel(Cs, t_1, t_2, \theta) = \{(C_1, C_2) \mid (tc \in Con_T)\& \ (ac \in Con_A)\& \ (con \in Con_C)$
  $\& \ (Models(Cs) = Models(Cs \cup \{(eq(t_1\theta, t_2\theta) \leftarrow)\}))$
  $\& \ (C_1 = ((((t_1\theta) \rhd tc) \rhd ac) \rhd con))$

$$\& \ (C_2 = ((((t_2\theta) \rhd tc) \rhd ac) \rhd con))\}.$$

- $rel(Cs) = \bigcup\{rel\,(Cs, t'_1, t'_2, \theta') \mid (t'_1 \in \mathrm{T}(\mathbb{F}, \mathbb{V}, \mathbb{FC}, \mathbb{FV}))$
$\& \ (t'_2 \in \mathrm{T}(\mathbb{F}, \mathbb{V}, \mathbb{FC}, \mathbb{FV}))$
$\& \ (\theta' \in \mathcal{S}_\theta)\}.$

A clause in $\mathrm{ECLS_N}$ may contain variables in $\mathbb{V}$ and ones in $\mathbb{FV}$. Let C be a clause and V_S a subset of $\mathbb{V} \cup \mathbb{FV}$. Let $vars(C, V_S)$ be defined as the set of all variables that belong to V_S and appear in C. A clause C is *ground with respect to* $\mathbb{V}$ if it contains no variable in $\mathbb{V}$, i.e., $vars(C, \mathbb{V}) = \emptyset$. A clause that is ground with respect to $\mathbb{V}$ may contain variables in $\mathbb{FV}$.

Since variables in $\mathbb{V}$ are quantified universally, a clause can be changed into the set of all of its ground clauses preserving models. Let Cs be a clause set, and C_1 and C_2 be clauses. Let S_1 be the set of all substitutions $\rho \in \mathcal{S}_\theta$ such that $C_1\rho$ is ground with respect to $\mathbb{V}$, i.e., $vars(C_1\rho, \mathbb{V}) = \emptyset$. Then $Models(Cs \cup \{C_1\}) = Models(Cs \cup \{C_1\rho \mid \rho \in S_1\})$. Let S_2 be the set of all substitutions $\rho \in \mathcal{S}_\theta$ such that $C_2\rho$ is ground with respect to $\mathbb{V}$, i.e., $vars(C_2\rho, \mathbb{V}) = \emptyset$. Then $Models(Cs \cup \{C_2\}) = Models(Cs \cup \{C_2\rho \mid \rho \in S_2\})$. Let S_{12} be the set of all substitutions $\rho \in \mathcal{S}_\theta$ such that $C_1\rho$ and $C_2\rho$ are ground with respect to $\mathbb{V}$, i.e., $vars(C_1\rho, \mathbb{V}) = \emptyset$ and $vars(C_2\rho, \mathbb{V}) = \emptyset$. S_1 can be changed into S_{12} keeping models. Similarly, S_2 can be changed into S_{12} keeping models. Then the next proposition holds using S_{12}.

Proposition 8.1. *Let Cs be a set of clauses. Let C_1 and C_2 be clauses. Let S_{12} be the set of all substitutions $\rho \in \mathcal{S}_\theta$ such that $vars(C_1\rho, \mathbb{V}) = \emptyset$ and $vars(C_2\rho, \mathbb{V}) = \emptyset$. Then $Models(Cs \cup \{C_1\}) = Models(Cs \cup \{C_1\rho \mid \rho \in S_{12}\})$ and $Models(Cs \cup \{C_2\}) = Models(Cs \cup \{C_2\rho \mid \rho \in S_{12}\})$.*

Proof: $Models(Cs \cup \{C_1\}) = Models(Cs \cup \{C_1\rho \mid \rho \in S_1\}) = Models(Cs \cup \{C_1\rho \mid \rho \in S_{12}\})$. $Models(Cs \cup \{C_2\}) = Models(Cs \cup \{C_2\rho \mid \rho \in S_2\}) = Models(Cs \cup \{C_2\rho \mid \rho \in S_{12}\})$. □

When we have a unit equational clause $EQ = (eq(t_1\theta, t_2\theta) \leftarrow)$ such that $t_1\theta$ and $t_2\theta$ are syntactically different, we may replace $t_1\theta$ with $t_2\theta$ in C_1 to produce C_2. The correctness of this rewriting is justified by the accumulation of ground-level equality $t_1\theta\rho\sigma = t_2\theta\rho\sigma$, where $\rho \in S_{12}$ and σ is a specialization for function variables.

Proposition 8.2. *Let Cs be a set of clauses. Let t_1 and t_2 be terms in $\mathrm{T}(\mathbb{F}, \mathbb{V}, \mathbb{FC}, \mathbb{FV})$ and $\theta \in \mathcal{S}_\theta$. Let C_1 and C_2 be clauses such that $C_1 = ((((t_1\theta) \rhd tc) \rhd ac) \rhd con)$ and $C_2 = ((((t_2\theta) \rhd tc) \rhd ac) \rhd con)$ for some $tc \in Con_T$, $ac \in Con_A$ and $con \in Con_C$. Let $EQ = (eq(t_1\theta, t_2\theta) \leftarrow)$. Let S_{12} be the set of all substitutions $\rho \in \mathcal{S}_\theta$ such that $vars(C_1\rho, \mathbb{V}) = \emptyset$ and $vars(C_2\rho, \mathbb{V}) = \emptyset$. Then $Models(Cs \cup \{EQ\} \cup \{C_1\rho \mid \rho \in S_{12}\}) = Models(Cs \cup \{EQ\} \cup \{C_2\rho \mid \rho \in S_{12}\})$.*

Proof: Assume that ρ is a substitution in S_{12}. Then $C_1\rho$ and $C_2\rho$ are ground with respect to $\mathbb{V}$, and there are contexts tc', ac' and con' such that $C_1\rho = ((((t_1\theta\rho) \rhd tc') \rhd ac') \rhd con')$ and $C_2\rho = ((((t_2\theta\rho) \rhd tc') \rhd ac') \rhd con')$. It follows that $t_1\theta\rho$ and $t_2\theta\rho$ contain no variable in $\mathbb{V}$. Let σ be a specialization for function variables corresponding to the top-level existential quantification. Then $C_1\rho\sigma\circ(fev)$ is obtained from $C_2\rho\sigma\circ(fev)$ by replacing $t_1\theta\rho\sigma \circ (fev)$ with $t_2\theta\rho\sigma \circ (fev)$. From EQ, it follows that $t_1\theta\rho\sigma \circ (fev) = t_2\theta\rho\sigma \circ (fev)$. Hence, $Models(Cs \cup \{EQ\} \cup \{C_1\rho \mid \rho \in S_{12}\}) = Models(Cs \cup \{EQ\} \cup \{C_2\rho \mid \rho \in S_{12}\})$. □

8.3. **Main theorems.** We prove that $rel(Cs)$ is a set of equivalent rewriting in the next theorem.

Theorem 8.1. *Let Cs be a set of clauses. If* $(C_1, C_2) \in rel(Cs)$*, then* $Models(Cs \cup \{C_1\}) = Models(Cs \cup \{C_2\})$.

Proof: Assume that $(C_1, C_2) \in rel(Cs)$. Then there are terms t_1 and t_2, and a substitution θ in $\mathcal{S}_\theta$ such that $(C_1, C_2) \in rel(Cs, t_1, t_2, \theta)$. Let $EQ = (eq(t_1\theta, t_2\theta) \leftarrow)$. Let S_{12} be the set of all substitutions $\rho \in \mathcal{S}_\theta$ such that $vars(C_1\rho, \mathbb{V}) = \emptyset$ and $vars(C_2\rho, \mathbb{V}) = \emptyset$.

$$\begin{aligned}
Models(Cs \cup \{C_1\}) &= (\text{since } Models(Cs) = Models(Cs \cup \{EQ\})) \\
&= Models(Cs \cup \{EQ, C_1\}) \\
&= (\text{by Proposition 8.1}) \\
&= Models(Cs \cup \{EQ\} \cup \{C_1\rho \mid \rho \in S_{12}\}) \\
&= (\text{by Proposition 8.2}) \\
&= Models(Cs \cup \{EQ\} \cup \{C_2\rho \mid \rho \in S_{12}\}) \\
&= (\text{by Proposition 8.1}) \\
&= Models(Cs \cup \{EQ, C_2\}) \\
&= (\text{since } Models(Cs) = Models(Cs \cup \{EQ\})) \\
&= Models(Cs \cup \{C_2\}).
\end{aligned}$$

□

We prove that a term rewriting rule that is admissible to a set of clauses is an equivalent transformation rule as follows.

Theorem 8.2. *Let Cs be a set of clauses. Let r be a term rewriting rule that is admissible to Cs. Let* C_1 *and* C_2 *be clauses. If* $C_1 \xrightarrow{r,Cs} C_2$*, then* $Models(Cs \cup \{C_1\}) = Models(Cs \cup \{C_2\})$.

Proof: Assume that $Cs = Cs_{eq} \cup D \cup Q$ according to Section 7.4. Since $C_1 \xrightarrow{r,Cs} C_2$, there is an equational clause $C = (eq(t_1, t_2) \leftarrow conds)$ in Cs and there are terms t_1 and t_2, a substitution θ, a term context tc, an atom context ac, and a clause context con such that $D \models_V conds\theta$, $C_1 = (((t_1\theta \rhd tc) \rhd ac) \rhd con)$ and $C_2 = (((t_2\theta \rhd tc) \rhd ac) \rhd con)$. Hence, $Models(Cs) = Models(Cs \cup \{(eq(t_1\theta, t_2\theta) \leftarrow)\})$. Then $(C_1, C_2) \in rel(Cs)$. By Theorem 8.1, $Models(Cs \cup \{C_1\}) = Models(Cs \cup \{C_2\})$. □

9. **Model-Preserving Term Rewriting on LPSF.** Term rewriting rules make transformation sequences that preserve models. They solve logical problems correctly on KR-Logic according to the LPSF theory.

9.1. **Model-preserving transformation of clauses.** Referring to $\tilde{C}s = \{C_1, C_2, \ldots, C_{12}\}$ in Section 6.3, let $\bar{C}s = \{C_1, C_2, \ldots, C_{11}\}$ and $C_q = C_{12} = (\leftarrow eq(\$f(\$a, \$b), \$f(\$b, \$a)))$. We can formalize the given problem by $Models\left(\{C_q\} \cup \bar{C}s\right) = \emptyset$. The computation starts with an initial state $St_0 = \{C_q\} \cup \bar{C}s$. After 3 steps, we reach a state $St_3 = \{C'\} \cup \bar{C}s$, where

$$C' = (\leftarrow eq(\$f(\$f(\$b, \$a), \$f(\$f(\$b, \$a), \$f(\$a, \$b))), \$f(\$b, \$a))).$$

Consider the rule r_{11}. Since $g(\$b)$, $g(\$a)$, and $g(\$f(\$a, \$b))$ are true, r_{11} can be applied to St_3 with $\theta = \{x/\$b, y/\$a, z/\$f(\$a, \$b)\}$. The next state is $St_4 = \{C''\} \cup \bar{C}s$, where

$$C'' = (\leftarrow eq(\$f(\$f(\$b, \$a), \$f(\$b, \$f(\$a, \$f(\$a, \$b)))), \$f(\$b, \$a))).$$

All steps of the computation are shown below.

$St_0 = \{(\leftarrow eq(\$f(\$a, \$b), \$f(\$b, \$a)))\} \cup \bar{C}s$

(r_3 is applied with $x = \$f(\$a, \$b)$ (since $g(\$f(\$a, \$b))$ is true))

$St_1 = \{(\leftarrow eq(\$f(\$e, \$f(\$a, \$b)), \$f(\$b, \$a)))\} \cup \bar{C}s$
(r_9 is applied with $x = \$f(\$a, \$b)$ (since $g(\$f(\$a, \$b))$ is true))
$St_2 = \{(\leftarrow eq(\$f(\$f(\$f(\$b, \$a), \$f(\$b, \$a)), \$f(\$a, \$b)), \$f(\$b, \$a)))\} \cup \bar{C}s$
(r_{11} is applied with $x = y = z = \$f(\$a, \$b)$ (since $g(\$f(\$a, \$b))$ is true))
$St_3 = \{(\leftarrow eq(\$f(\$f(\$b, \$a), \$f(\$f(\$b, \$a), \$f(\$a, \$b))), \$f(\$b, \$a)))\} \cup \bar{C}s$
(r_{11} is applied with $x = \$b$, $y = \$a$, and $z = \$f(\$a, \$b)$) (see the above explanation)
$St_4 = \{(\leftarrow eq(\$f(\$f(\$b, \$a), \$f(\$b, \$f(\$a, \$f(\$a, \$b)))), \$f(\$b, \$a)))\} \cup \bar{C}s$
(r_{12} is applied with $x = y = \$a$ and $z = \$b$ (since $g(\$a)$ and $g(\$b)$ are true))
$St_5 = \{(\leftarrow eq(\$f(\$f(\$b, \$a), \$f(\$b, \$f(\$f(\$a, \$a), \$b))), \$f(\$b, \$a)))\} \cup \bar{C}s$
(r_{10} is applied with $x = \$a$ (since $g(\$a)$ is true))
$St_6 = \{(\leftarrow eq(\$f(\$f(\$b, \$a), \$f(\$b, \$f(\$e, \$b))), \$f(\$b, \$a)))\} \cup \bar{C}s$
(r_4 is applied with $x = \$b$ (since $g(\$b)$ is true))
$St_7 = \{(\leftarrow eq(\$f(\$f(\$b, \$a), \$f(\$b, \$b)), \$f(\$b, \$a)))\} \cup \bar{C}s$
(r_{10} is applied with $x = \$b$ (since $g(\$b)$ is true))
$St_8 = \{(\leftarrow eq(\$f(\$f(\$b, \$a), \$e), \$f(\$b, \$a)))\} \cup \bar{C}s$
(r_2 is applied with $x = \$f(\$b, \$a)$ (since $g(\$f(\$b, \$a))$ is true))
$St_9 = \{(\leftarrow eq(\$f(\$b, \$a), \$f(\$b, \$a)))\} \cup \bar{C}s$
(the $eq(x, x)$ atom is removed since it is true.)
$St_{10} = \{(\leftarrow)\} \cup \bar{C}s$

According to Theorem 8.2 in Section 8.3, the first nine steps are equivalent transformation. In the last step of transformation, a constraint solving rule for equality is used. This rule is not a term rewriting rule. However, it obviously preserves models, i.e., it is an ET rule. Since the computation reaches $\{(\leftarrow)\} \cup \bar{C}s$, we know that $Models(St_{10}) = \emptyset$.

9.2. **Formalization as an MI problem and its solution.** According to the LPSF theory, the proof problem under consideration is formalized as an MI problem $\langle \{C_q\} \cup \bar{C}s, \varphi_0 \rangle$, where φ_0 is an extraction mapping defined by for any $G \subseteq \mathcal{G}_{\mathrm{u}}$, $\varphi_0(G) = yes$ if $G = \mathcal{G}_{\mathrm{u}}$, and $\varphi_0(G) = no$ otherwise. For any clause set Cs, the answer to an MI problem $\langle Cs, \varphi_0 \rangle$ is defined to be $\varphi_0(\bigcap Models(Cs))$, and

$$\varphi_0\left(\bigcap Models(Cs)\right) = yes \iff \bigcap Models(Cs) = \mathcal{G}_{\mathrm{u}} \iff Models(Cs) = \emptyset.$$

Hence, the MI problem $\langle \{C_q\} \cup \bar{C}s, \varphi_0 \rangle$ is equivalent to the proof problem to prove $Models(St_0) = \emptyset$. The sequence $St_0, St_1, \ldots, St_{10}$ is model-preserving, i.e.,

$$Models(St_0) = Models(St_1) = \cdots = Models(St_{10}),$$

and, at the same time, is an equivalent transformation sequence for solving the MI problem, i.e.,

$$\varphi_0\left(\bigcap Models(St_0)\right) = \varphi_0\left(\bigcap Models(St_1)\right) = \cdots = \varphi_0\left(\bigcap Models(St_{10})\right).$$

Since the computation reaches $St_{10} = \{(\leftarrow)\} \cup \bar{C}s$, the answer mapping $ans \in A$ (see Section 3) gives the final result yes, which means that the answer to the MI problem $\langle \{C_q\} \cup \bar{C}s, \varphi_0 \rangle$ is yes. Hence, the proof problem under consideration is successfully solved. The answer yes is correct since $\varphi_0\left(\bigcap Models(St_{10})\right) = \varphi_0(\mathcal{G}_{\mathrm{u}}) = yes$.

In this paper, term rewriting rules were proposed and proved to be model-preserving in KR-Logic. The representation space $\mathrm{KRL_c}$ and the computation space $\mathrm{ECLS_N}$ were used to solve the MI problem $\langle \{C_q\} \cup \bar{Cs}, \varphi_0 \rangle$ on KR-Logic. In LPSF-based computation, logical inference and term rewriting are used together under the principle of equivalent transformation.

10. **Conclusions.** First-order logic fails to formally prove theorems in group theory by term rewriting. KR-Logic overcomes the limitations of first-order logic. To formalize a proof by term rewriting for group theory, we developed the representation space $\mathrm{KRL_c}$ and the computation space $\mathrm{ECLS_N}$ on KR-Logic. Let Cs and Cs' be sets of clauses in $\mathrm{ECLS_N}$. Assume that a proof problem is formalized as $\bigcap Models(Cs \cup Cs') = \mathcal{G}_\mathrm{u}$ and there is an equational clause $C = (eq(t_1, t_2) \leftarrow conds)$ in Cs. Then we have a term rewriting rule r: $(t_1 \rightarrow t_2 \colon conds)$ that is applicable to a clause in Cs' and preserves models of $Cs \cup Cs'$. This is similar to other ET rules such as resolution, unfolding, and constraint solving. Specialized unfolding rules can be used since they are model-intersection preserving. Inference rules such as the resolution and factoring rules preserve models. Constraint solving rules for equality and other built-in constraint atoms preserve models. All of these are ET rules working in the general logical framework LPSF. For proving problems on $\mathrm{ECLS_N}$, we can use many kinds of transformation, such as logical inference, constraint solving, and functional rewriting. They can be integrated under a broader concept of equivalent transformation.

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] N. Dershowitz and J.-P. Jouannaud, Rewrite systems, in *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics*, J. Van Leeuwen (ed.), MIT Press, 1990.

[2] G. Huet, *Logical Foundations of Functional Programming*, Addison Wesley Publishing Company, 1990.

[3] G. Huet, Confluent reductions: Abstract properties and applications to term rewriting systems, *Journal of the Association for Computing Machinery*, vol.27, no.4, pp.797-821, 1980.

[4] D. E. Knuth and P. B. Bendix, Simple word problems in universal algebras, in *Computational Problems in Abstract Algebra*, J. Leech (ed.), Pergamon Press, 1970.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.
[11] K. Akama and E. Nantajeewarawat, Meaning-preserving skolemization, *Proc. of the 3rd International Conference on Knowledge Engineering and Ontology Development*, Paris, France, pp.322-327, 2011.
[12] K. Akama, E. Nantajeewarawat and T. Akama, Term rewriting that preserves models in KR-logic, in *Intelligent Information and Database Systems. ACIIDS 2019. Lecture Notes in Computer Science*, N. Nguyen, F. Gaol, T. P. Hong and B. Trawiński (eds.), Cham, Springer, 2019.

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.

Prof. 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.