International Journal of Innovative
Computing, Information and Control
Volume **9**, Number **9**, September **2013**

ICIC International ©2013 ISSN 1349-4198
pp. 3515–3526

UNFOLDING-BASED SIMPLIFICATION OF QUERY-ANSWERING PROBLEMS IN AN EXTENDED CLAUSE SPACE

Kiyoshi Akama[1] and Ekawit Nantajeewarawat[2]

[1]Information Initiative Center
Hokkaido University
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
P.O. Box 22, Thammasat-Rangsit Post Office, Pathum Thani 12121, Thailand
ekawit@siit.tu.ac.th

Received July 2012; revised January 2013

Abstract. *To convert a first-order formula containing an existential quantification into an equivalent clausal normal form, it is necessary to introduce function variables and accordingly extend the space of first-order formulas. This paper proposes unfolding transformation and transformation by definite-clause removal on such an extended formula space and demonstrates how they are employed to simplify query-answering problems. The presented work provides a foundation for constructing a correct method for solving query-answering problems that include unrestricted use of universal and existential quantifications.*

Keywords: Query-answering problems, Unfolding transformation, Equivalent transformation

1. **Introduction.** Recently, query-answering (QA) problems have attracted wider interest, owing partly to emerging applications involving integration between schema-level ontologies and object-level rules, e.g., interaction between Description Logics and Horn rules [1, 2], based on the Semantic Web's layered architecture. A problem in this class is concerned with finding the set of all ground instances of a given query atom that are logical consequences of a given logical formula. Equivalent transformation (ET) of formulas is essential and very useful for solving many classes of logical problems [3]. In ET-based problem solving, a logical formula representing a given problem is successively transformed into a simpler but equivalent formula that preserves the answer to the given problem. Correctness of computation is readily guaranteed by any combination of equivalent transformations. Many kinds of correct and efficient algorithms for solving logical problems have been devised successfully based on ET [4], motivating us to employ the ET principle to deal with QA problems.

Solving QA problems on first-order logic involves the conversion of first-order formulas into clausal normal forms (also called conjunctive normal forms) using Skolemization. Since the classical Skolemization does not preserve the logical meaning of a first-order formula [5], it cannot be used in an ET-based problem-solving process for QA problems on full first-order logic. Meaning-preserving Skolemization has been developed recently in [6] based on an extension of a logical formula space by incorporation of function variables. A clausal normal form resulting from meaning-preserving Skolemization determines a set of clauses in an extended clause space, called the $\mathrm{ECLS_F}$ space. Compared with usual definite clauses, a clause in this extended space may contain function variables in its

body and may have multiple heads. This paper proposes unfolding transformation on the $\mathrm{ECLS_F}$ space and illustrates how it can be applied to simplify QA problems.

To begin with, Section 2 formalizes QA problems and describes a general scheme for solving them using ET. Section 3 explains the basic idea of meaning-preserving Skolemization and introduces QA problems on the $\mathrm{ECLS_F}$ space. Section 4 presents equivalent transformation by unfolding and that by definite-clause removal in the $\mathrm{ECLS_F}$ space and explains their usefulness in solving and simplifying QA problems. Section 5 illustrates their application. Section 6 describes fundamental differences between this work and existing theories. Section 7 provides concluding remarks.

2. **Query-Answering Problems and Equivalent Transformation.** A *query-answering problem* (*QA problem*) is a pair $\langle K, a\rangle$, where K is a logical formula, representing background knowledge, and a is an atomic formula (atom), representing a query. The answer to a QA problem $\langle K, a\rangle$, denoted by $ans(K, a)$, is the set of all ground instances of a that are logical consequences of K. According to the types of background knowledge, QA problems can be classified into several subclasses, e.g., QA problems on definite clauses, where background knowledge is a set of definite clauses, QA problems on description logics (DLs), where background knowledge is a conjunction of axioms and assertions in DLs [7]. QA problems on definite clauses have been extensively discussed in logic programming [8]. QA problems on DLs have been discussed in [9]. Answering queries in Datalog and deductive databases [10] can be regarded as solving QA problems on a restricted form of definite clauses.

Given a set K of definite clauses, since K has a unique minimal model, the answer set of a QA problem $\langle K, a\rangle$ becomes the intersection of the minimal model of K and $rep(a)$, where $rep(a)$ is the set of all ground instances of a. When K is an arbitrary first-order formula, determining the answer set of a QA problem $\langle K, a\rangle$ is more complicated since K possibly has multiple minimal models, none of which is included by the others.

We aim at dealing with QA problems on full first-order logic, where background knowledge can be any arbitrary first-order formula, without any restriction on its form. Let $\langle K, a\rangle$ be a QA problem in this class. Using the set of all models of K, denoted by $Models(K)$, the set $ans(K, a)$ can be equivalently formulated as $(\bigcap Models(K)) \cap rep(a)$, where $\bigcap Models(K)$ is the intersection of all models of K. Calculating $(\bigcap Models(K)) \cap rep(a)$ directly often requires high computational cost. To reduce the cost, K is transformed into a simplified formula K' such that $(\bigcap Models(K)) \cap rep(a)$ is preserved and $(\bigcap Models(K')) \cap rep(a)$ can be determined at a low cost. It is obvious that $ans(K, a) = ans(K', a)$ if $(\bigcap Models(K)) \cap rep(a) = (\bigcap Models(K')) \cap rep(a)$. If the logical meaning of K is preserved, then the answer set $(\bigcap Models(K)) \cap rep(a)$ is always preserved. However, this answer set may not be preserved if only the satisfiability of K is preserved.

3. **Meaning-Preserving Skolemization and QA Problems on $\mathbf{ECLS_F}$.** To solve a QA problem $\langle K, a\rangle$ on first-order logic, the first-order formula K is usually converted into a conjunctive normal form. The conversion involves removal of existential quantifications by Skolemization, i.e., by replacement of an existentially quantified variable with a Skolem term determined by a relevant part of a formula prenex. The classical Skolemization, however, does not preserve the logical meaning of a formula – the formula resulting from Skolemization is not necessarily equivalent to the original one [5]. In [6], we developed a theory for extending the space of first-order logical formulas and showed how meaning-preserving Skolemization can be achieved in the obtained extended space, called the $\mathrm{ECLS_F}$ space. The basic idea of meaning-preserving Skolemization is to use existentially quantified function variables instead of usual Skolem functions. Function variables,

extended clauses, extended conjunctive normal forms and QA problems on $\mathrm{ECLS_F}$ are introduced below.

3.1. **Function constants, function variables and *func*-atoms.** A usual function symbol, say f, in first-order logic denotes an unevaluated function; it is used for constructing from existing terms, say $t_1, \ldots, t_n$, a syntactically new term, e.g., $f(t_1, \ldots, t_n)$, possibly recursively, without evaluating the new term $f(t_1, \ldots, t_n)$. A different class of functions is used in the extended space. A function in this class is an actual mathematical function, say h, on ground terms; when it takes ground terms, say $t_1, \ldots, t_n$, as input, $h(t_1, \ldots, t_n)$ is evaluated for determining an output ground term. We called a function in this class a *function constant*. Variables of a new type, called *function variables*, are introduced; each of them can be instantiated into a function constant or a function variable, but not into a usual term.

In order to clearly separate function constants and function variables from usual function symbols and usual terms, a new built-in predicate *func* is introduced. Given any n-ary function constant or n-ary function variable $\bar{f}$, an expression

$$\mathit{func}(\bar{f}, t_1, \ldots, t_n, t_{n+1}),$$

where the t_i are usual terms, is considered as an atom of a new type, called a *func-atom*. When $\bar{f}$ is a function constant and the t_i are all ground, the truth value of this atom is evaluated as follows: it is true iff $\bar{f}(t_1, \ldots, t_n) = t_{n+1}$.

3.2. **Extended clauses and existentially quantified conjunctive normal forms.** An *extended clause* C is a formula of the form

$$\forall v_1, \ldots, \forall v_m : (a_1 \vee \cdots \vee a_n \vee \neg b_1 \vee \cdots \vee \neg b_p \vee \neg \mathbf{f}_1 \vee \cdots \vee \neg \mathbf{f}_q),$$

where $v_1, \ldots, v_m$ are usual variables, each of $a_1, \ldots, a_n, b_1, \ldots, b_p$ is a usual atom or a constraint atom, and $\mathbf{f}_1, \ldots, \mathbf{f}_q$ are *func*-atoms. It is often written simply as

$$a_1, \ldots, a_n \leftarrow b_1, \ldots, b_p, \mathbf{f}_1, \ldots, \mathbf{f}_q.$$

The sets $\{a_1, \ldots, a_n\}$ and $\{b_1, \ldots, b_p, \mathbf{f}_1, \ldots, \mathbf{f}_q\}$ are called the *left-hand side* and the *right-hand side*, respectively, of the extended clause C, denoted by $\mathit{lhs}(C)$ and $\mathit{rhs}(C)$, respectively. When $n = 0$, C is called a *negative extended clause*. When $n = 1$, C is called an *extended definite clause*, the only atom in $\mathit{lhs}(C)$ is called the *head* of C, denoted by $\mathit{head}(C)$, and the set $\mathit{rhs}(C)$ is also called the *body* of C, denoted by $\mathit{body}(C)$. When $n > 1$, C is called a *multi-head extended clause*. All usual variables in an extended clause are universally quantified and their scope is restricted to the clause itself. When no confusion is caused, an extended clause, a negative extended clause, an extended definite clause and a multi-head extended clause will also be called a *clause*, a *negative clause*, a *definite clause* and a *multi-head clause*, respectively.

An *existentially quantified conjunctive normal form* (*ECNF*) is a formula of the form

$$\exists v_{h1}, \ldots, \exists v_{hm} : (C_1 \wedge \cdots \wedge C_n),$$

where $v_{h1}, \ldots, v_{hm}$ are function variables and $C_1, \ldots, C_n$ are extended clauses. It is often identified with the set $\{C_1, \ldots, C_n\}$, with implicit existential quantifications of function variables and implicit clause conjunction. Function variables in such a clause set are all existentially quantified and their scope covers all clauses in the set.

3.3. **QA problems and equivalent transformation on $\mathbf{ECLS_F}$.** The set of all ECNFs is referred to as the *extended clause space* ($ECLS_{\mathrm{F}}$). By the above identification of an ECNF with a clause set, we often regard an element of $\mathrm{ECLS_F}$ as a set of (extended) clauses. With occurrences of function variables, clauses contained in a clause set in the $\mathrm{ECLS_F}$ space are connected through shared function variables. By instantiating all function variables in such a clause set into function constants, clauses in the obtained set are totally separated[1].

A QA problem $\langle Cs, a\rangle$ such that Cs is a clause set in $\mathrm{ECLS_F}$ and a is a usual atom is called a *QA problem on* $ECLS_{\mathrm{F}}$. Given a QA problem $\langle K, a\rangle$ on first-order logic, the first-order formula K is converted by meaning-preserving Skolemization, using the conversion procedure given in [6], into a clause set Cs in the $\mathrm{ECLS_F}$ space. The obtained clause set Cs may be further transformed equivalently in this space into another clause set Cs' for problem simplification. Unfolding and other transformation rules may be used. The answer to the resulting problem $\langle Cs', a\rangle$ is $(\bigcap Models(Cs')) \cap rep(a)$.

4. Unfolding Transformation and Definite-Clause Removal on $\mathbf{ECLS_F}$.

4.1. **Need for unfolding on $\mathbf{ECLS_F}$.** Unfolding transformation is normally applied for transformation of usual definite clauses [11, 12]. A clause set in $\mathrm{ECLS_F}$, however, may contain multi-head clauses and occurrences of function variables. To provide a basis for solving a class of QA problems on $\mathrm{ECLS_F}$, which is far larger than the class of QA problems on usual definite clauses in terms of expressiveness, an unfolding rule for transforming multi-head clauses with function variables is necessary. Given a QA problem $\langle Cs, a\rangle$ on $\mathrm{ECLS_F}$, we propose in this paper (i) how to determine a set of extended definite clauses that can be used for unfolding Cs and a sufficient condition for equivalently transforming $\langle Cs, a\rangle$ by unfolding, and (ii) a sufficient condition for equivalently transforming $\langle Cs, a\rangle$ by removal of extended definite clauses.

4.2. **Unfolding operation on $\mathbf{ECLS_F}$.** Assume that Cs is a clause set in $\mathrm{ECLS_F}$, D is a definite-clause set in $\mathrm{ECLS_F}$, and occ is an occurrence of an atom b in the right-hand side of a clause C in Cs. By unfolding Cs using D at occ, Cs is transformed into

$$(Cs - \{C\}) \cup \left(\bigcup\{resolvent(C, C', b) \mid C' \in D\}\right),$$

where for each $C' \in D$, $resolvent(C, C', b)$ is defined as follows, assuming that ρ is a renaming substitution for usual variables such that C and $C'\rho$ have no usual variable in common:

1. If b and $head(C'\rho)$ are not unifiable, then $resolvent(C, C', b) = \emptyset$.
2. If they are unifiable, then $resolvent(C, C', b) = \{C''\}$, where C'' is the clause obtained from C and $C'\rho$ as follows, assuming that θ is the most general unifier of b and $head(C'\rho)$:
   (a) $lhs(C'') = lhs(C\theta)$
   (b) $rhs(C'') = (rhs(C\theta) - \{b\theta\}) \cup body(C'\rho\theta)$

The resulting clause set is denoted by $\textsc{Unfold}(Cs, D, occ)$.

4.3. **Equivalent transformation by unfolding and definite-clause removal.** For any predicate p, let $Atoms(p)$ denote the set of all atoms having the predicate p. Equivalent transformation of QA problems on $\mathrm{ECLS_F}$ using unfolding and using definite-clause removal are formulated below.

[1]Clauses in the usual conjunctive normal form are also totally separated, i.e., usual variables occurring in one clause are considered to be different from those occurring in another clause.

4.3.1. *Equivalent transformation by unfolding.* Let $\langle Cs, a\rangle$ be a QA problem on $\mathrm{ECLS_F}$. Assume that:

1. q is the predicate of the query atom a.
2. p is a predicate such that $p \neq q$.
3. D is a set of definite clauses in Cs that satisfies the following conditions:
   (a) For any definite clause $C \in D$, $\mathit{head}(C) \in \mathit{Atoms}(p)$.
   (b) For any clause $C' \in Cs - D$, $\mathit{lhs}(C') \cap \mathit{Atoms}(p) = \emptyset$.
4. *occ* is an occurrence of an atom in $\mathit{Atoms}(p)$ in the right-hand side of a clause in $Cs - D$.

Then $\langle Cs, a\rangle$ can be transformed into the QA problem $\langle \textsc{Unfold}(Cs, D, \mathit{occ}), a\rangle$.

Consider the assumption part above. Let C be the clause containing the occurrence *occ*. Only definite clauses in D can supply atoms in $\mathit{Atoms}(p)$ for the atom at *occ* (by Conditions 3 and 4). Consequently, before C can be used for generating a model of Cs by bottom up computation, some definite clause $C' \in D$ must be already used. The use of C and C' together yields the same result as the use of $\mathit{resolvent}(C, C', b)$, where b is the atom at the occurrence *occ*. Therefore Cs and $\textsc{Unfold}(Cs, D, \mathit{occ})$ have the same set of models and the answer to the original problem is preserved.

4.3.2. *Equivalent transformation by definite-clause removal.* Let $\langle Cs, a\rangle$ be a QA problem on $\mathrm{ECLS_F}$. Assume that:

1. q is the predicate of the query atom a.
2. p is a predicate such that $p \neq q$.
3. D is a set of definite clauses in Cs that satisfies the following conditions:
   (a) For any definite clause $C \in D$, $\mathit{head}(C) \in \mathit{Atoms}(p)$.
   (b) For any clause $C' \in Cs - D$, $\mathit{lhs}(C') \cap \mathit{Atoms}(p) = \emptyset$.
4. For any clause $C' \in Cs - D$, $\mathit{rhs}(C') \cap \mathit{Atoms}(p) = \emptyset$.

Then $\langle Cs, a\rangle$ can be transformed into the QA problem $\langle Cs - D, a\rangle$.

Consider the assumption part. Atoms in $\mathit{Atoms}(p)$ can only be supplied by definite clauses in D (by Condition 3). However, these atoms are not used for derivation of any instance of the query atom a and are not used for derivation of a contradiction by any negative clause in Cs, the reason being that no instance of a belongs to $\mathit{Atoms}(p)$ (by Conditions 1 and 2) and the predicate p does not occur in the right-hand side of any clause in $Cs - D$ (by Condition 4). Thus D can be removed without changing the answer to the original problem.

4.4. **Benefits of unfolding and definite-clause removal.** Unfolding is one of the most important equivalent transformations. When applied along with definite-clause removal, it is useful for (i) problem solving and (ii) problem simplification. Simple problems may be solved by unfolding alone. For example, many QA problems on definite clauses can be solved, at least in principle, using unfolding. For more difficult problems, unfolding is useful for transforming them into simplified forms, which can be subsequently solved by using other equivalent transformation rules. Alternative methods for solving restricted classes of QA problems are problem-specific solver generation [13] and conversion into satisfiability problems [14]. With these methods, problem simplification by unfolding and definite-clause removal is useful in procedures for (i) checking the applicability of a selected method, i.e., testing whether a given problem belongs to the class of problems that can be handled by the method, and (ii) generating smaller problem-specific solvers or smaller satisfiability problems.

5. **Examples.** Simplification of QA problems using unfolding and definite-clause removal will now be demonstrated.

Example 5.1. *Consider a QA problem* $\langle Cs, q(x)\rangle$*, where Cs consists of the following clauses, assuming that c is a constant:*

C_1: $p(c) \leftarrow$
C_2: $r(x) \leftarrow p(x)$
C_3: $s(x,x), s(x,y) \leftarrow p(x), r(y)$
C_4: $q(x) \leftarrow s(x,y)$

Note that C_3 *is a multi-head clause. By unfolding using* $\{C_1\}$*, the clauses* C_2 *and* C_3 *are replaced with:*

C_{2a}: $r(c) \leftarrow$
C_{3a}: $s(c,c), s(c,y) \leftarrow r(y)$

By unfolding using $\{C_{2a}\}$*, the clause* C_{3a} *is replaced with:*

C_{3b}: $s(c,c), s(c,c) \leftarrow$

By eliminating a duplicate atom in its left-hand side, C_{3b} *can be simplified into:*

C_{3c}: $s(c,c) \leftarrow$

By unfolding using $\{C_{3c}\}$*, the clause* C_4 *is replaced with:*

C_{4a}: $q(c) \leftarrow$

The resulting clause set is $\{C_1, C_{2a}, C_{3c}, C_{4a}\}$*. Since the predicate q appears only in the head of* C_{4a} *and none of the head predicates of* C_1*,* C_{2a} *and* C_{3c} *appears in the body of* C_{4a}*, the definite clauses* C_1*,* C_{2a} *and* C_{3c} *can be removed. The initial QA problem is thus equivalently transformed into the QA problem* $\langle\{C_{4a}\}, q(x)\rangle$*, from which the answer set* $\{q(c)\}$ *is readily obtained.*

As illustrated below, not all QA problems can be solved using only unfolding and definite-clause removal alone.

Example 5.2. *Assuming that c and d are constants, let Cs consist of the following clauses:*

C_1: $p(c) \leftarrow$
C_2: $r(x) \leftarrow p(x)$
C_3: $s(x,d), s(d,y) \leftarrow p(x), r(y)$
C_4: $q(x) \leftarrow s(x,d)$
C_5: $q(x) \leftarrow s(d,x)$

Consider the QA problem $\langle Cs, q(x)\rangle$*. In a way similar to the transformation in Example 5.1, Cs can be simplified using unfolding and definite-clause removal into* $Cs' = \{C', C_4, C_5\}$*, where* $C' = (s(c,d), s(d,c) \leftarrow)$*. No further unfolding is applicable to* Cs'*. The simplified problem* $\langle Cs', q(x)\rangle$ *can be solved by bottom-up computation* [15] *or by further transformation using other equivalent transformation rules in a conjunction-based clause space* [16]*. Bottom-up computation and transformation of conjunction-based clauses are beyond the scope of this paper.*

Example 5.3. *Consider the Oedipus problem described in* [7]*. Oedipus killed his father, married his mother Iokaste, and had children with her, among them Polyneikes. Polyneikes also had children, among them Thersandros, who is not a patricide. The problem is to find a person who has a patricide child who has a non-patricide child. The difficulty of this problem arises due to the absence of information as to whether Polyneikes is a patricide or not.*

Assuming that "oe," "io," "po" and "th" stand, respectively, for Oedipus, Iokaste, Polyneikes and Thersandros, this problem is represented as a QA problem $\langle K, Prob(x)\rangle$

on first-order logic, where K is the conjunction of the logical formulas $hasChild(io, oe)$, $hasChild(io, po)$, $hasChild(oe, po)$, $hasChild(po, th)$, $Pat(oe)$, $\neg Pat(th)$ and

$$\forall x\colon ((\exists y\colon (hasChild(x, y) \wedge Pat(y) \wedge (\exists z\colon (hasChild(y, z) \wedge \neg Pat(z))))) \rightarrow Prob(x)).$$

By converting K into a conjunctive normal form, the QA problem $\langle K, Prob(x)\rangle$ is transformed into a QA problem $\langle Cs, Prob(x)\rangle$, where Cs consists of the following seven clauses:

C_1: $hasChild(io, oe) \leftarrow$ $\quad$ C_2: $hasChild(io, po) \leftarrow$
C_3: $hasChild(oe, po) \leftarrow$ $\quad$ C_4: $hasChild(po, th) \leftarrow$
C_5: $Pat(oe) \leftarrow$ $\quad$ C_6: $\leftarrow Pat(th)$
C_7: $Prob(x), Pat(z) \leftarrow hasChild(x, y), hasChild(y, z), Pat(y)$

Let D be the set of all definite clauses with the head predicate $hasChild$ in Cs, i.e., $D = \{C_1, C_2, C_3, C_4\}$. Since $hasChild$ does not appear in the left-hand side of any clause in $Cs - D$, Cs can be transformed by unfolding using D as follows:

1. *By unfolding Cs using D at $hasChild(x, y)$ in C_7, the clause C_7 is replaced with the following four clauses:*

   C_8: $Prob(io), Pat(z) \leftarrow hasChild(oe, z), Pat(oe)$
   C_9: $Prob(io), Pat(z) \leftarrow hasChild(po, z), Pat(po)$
   C_{10}: $Prob(oe), Pat(z) \leftarrow hasChild(po, z), Pat(po)$
   C_{11}: $Prob(po), Pat(z) \leftarrow hasChild(th, z), Pat(th)$

2. *By unfolding the resulting clause set four times using D at $hasChild$-atoms in C_8, C_9, C_{10} and C_{11}, these four clauses are replaced with the following clauses:*

   C_{12}: $Prob(io), Pat(po) \leftarrow Pat(oe)$
   C_{13}: $Prob(io), Pat(th) \leftarrow Pat(po)$
   C_{14}: $Prob(oe), Pat(th) \leftarrow Pat(po)$

Since the predicate $hasChild$ does not appear in the right-hand side of any of C_5, C_6 and C_{12}-C_{14}, the definite clauses in D are then removed.

The resulting clause set is $Cs' = \{C_5, C_6, C_{12}, C_{13}, C_{14}\}$. At this point, Pat is the only predicate of a possible target body atom. Since each of C_{12}, C_{13} and C_{14} also contains a Pat-atom in its left-hand side, no further unfolding is applicable to Cs'. By bottom-up computation [15] *or by further equivalent transformation in a conjunction-based clause space* [16]*, the simplified problem $\langle Cs', Prob(x)\rangle$ can be solved, with the answer set being the singleton $\{Prob(io)\}$, i.e., Iokaste is the answer to this problem (no matter whether Polyneikes is a patricide).*

Example 5.4. *Next, consider the "Tax-cut" problem discussed in* [2]*. This problem is to find all persons who can have discounted tax, with the knowledge that:*

1. *Any person who has two children or more can get discounted tax.*
2. *Men and women are not the same.*
3. *A person's mother is always a woman.*
4. *Peter has a child, who is someone's mother.*
5. *Peter has a child named Paul.*
6. *Paul is a man.*

This background knowledge is represented in the $\mathcal{SHOIN}(\mathbf{D})$ description logic, which is the logical formalism underlying the Web Ontology Language OWL-DL [17]*, as the following axioms and assertions*[2]*:*

[2]OWL-DL is a W3C recommendation language for ontology representation in the Semantic Web. It is a syntactic variant of the $\mathcal{SHOIN}(\mathbf{D})$ description logic.

A_1: $\geq 2\ hasChild \sqsubseteq TaxCut$ A_2: $Man \sqcap Woman \sqsubseteq \bot$
A_3: $\exists motherOf.\top \sqsubseteq Woman$ A_4: $\exists hasChild.(\exists motherOf.\top)(Peter)$
A_5: $hasChild(Peter, Paul)$ A_6: $Man(Paul)$

These axioms and assertions are translated into the following first-order formulas:

F_1: $\forall x\colon ((\exists y_1 \exists y_2\colon (hasChild(x, y_1) \wedge hasChild(x, y_2) \wedge notSame(y_1, y_2))) \rightarrow TaxCut(x))$
F_2: $\forall x \forall y\colon ((Man(x) \wedge Woman(y)) \rightarrow (notSame(x, y) \wedge notSame(y, x)))$
F_3: $\forall x\colon ((\exists y : motherOf(x, y)) \rightarrow Woman(x))$
F_4: $\exists x\colon (hasChild(Peter, x) \wedge (\exists y\colon motherOf(x, y)))$
F_5: $hasChild(Peter, Paul)$
F_6: $Man(Paul)$

Accordingly, the "Tax-cut" problem can be formulated as a QA problem $\langle K, TaxCut(x)\rangle$ *on first-order logic, where* K *is the conjunction of the above six first-order formulas. Using the meaning-preserving Skolemization procedure given in* [6]*, this QA problem is transformed into a QA problem* $\langle Cs_0, TaxCut(x)\rangle$ *on* $ECLS_{\mathrm{F}}$*, where the clause set* Cs_0 *consists of the following extended clauses:*

C_1: $TaxCut(x) \leftarrow hasChild(x, y_1), hasChild(x, y_2), notSame(y_1, y_2)$
C_2: $notSame(x, y) \leftarrow Man(x), Woman(y)$
C_3: $notSame(x, y) \leftarrow Woman(x), Man(y)$
C_4: $Woman(x) \leftarrow motherOf(x, y)$
C_5: $hasChild(Peter, x) \leftarrow func(h_1, x)$
C_6: $motherOf(x, y) \leftarrow func(h_1, x), func(h_2, y)$
C_7: $hasChild(Peter, Paul) \leftarrow$
C_8: $Man(Paul) \leftarrow$

The clauses C_5 *and* C_6 *together represent the first-order formula* F_4*, where* h_1 *and* h_2 *are 0-ary function variables.*

The clause set Cs_0 *can be transformed by unfolding and definite-clause removal as described below:*

1. Cs_0 *is unfolded with respect to the predicate hasChild as follows:*
   (a) *The set* $D = \{C_5, C_7\}$ *is determined.*
   (b) *Unfolding is applied using* D *as follows:*
      (i) *The occurrence of* $hasChild(x, y_1)$ *in* C_1 *is selected and, by unfolding using* D*,* C_1 *is replaced with:*
         C_9: $TaxCut(Peter) \leftarrow func(h_1, y_1), hasChild(Peter, y_2), notSame(y_1, y_2)$
         C_{10}: $TaxCut(Peter) \leftarrow hasChild(Peter, y_2), notSame(Paul, y_2)$
      (ii) *The occurrence of* $hasChild(Peter, y_2)$ *in* C_9 *is selected and, by unfolding using* D*,* C_9 *is replaced with:*
         C_{11}: $TaxCut(Peter) \leftarrow func(h_1, y_1), func(h_1, y_2), notSame(y_1, y_2)$
         C_{12}: $TaxCut(Peter) \leftarrow func(h_1, y_1), notSame(y_1, Paul)$
      (iii) *The occurrence of* $hasChild(Peter, y_2)$ *in* C_{10} *is selected and, by unfolding using* D*,* C_{10} *is replaced with:*
         C_{13}: $TaxCut(Peter) \leftarrow func(h_1, y_2), notSame(Paul, y_2)$
         C_{14}: $TaxCut(Peter) \leftarrow notSame(Paul, Paul)$
2. *By definite-clause removal with respect to hasChild,* C_5 *and* C_7 *are removed. The resulting clause set is* $Cs_1 = \{C_2, C_3, C_4, C_6, C_8, C_{11}, C_{12}, C_{13}, C_{14}\}$.
3. *By unfolding with respect to the predicate notSame,* C_{11}-C_{14} *are replaced with:*

C_{15}: $TaxCut(Peter) \leftarrow func(h_1, y_1), func(h_1, y_2), Man(y_1), Woman(y_2)$
C_{16}: $TaxCut(Peter) \leftarrow func(h_1, y_1), func(h_1, y_2), Woman(y_1), Man(y_2)$

C_{17}: $TaxCut(Peter) \leftarrow func(h_1, y_1), Man(y_1), Woman(Paul)$
C_{18}: $TaxCut(Peter) \leftarrow func(h_1, y_1), Woman(y_1), Man(Paul)$
C_{19}: $TaxCut(Peter) \leftarrow func(h_1, y_2), Man(Paul), Woman(y_2)$
C_{20}: $TaxCut(Peter) \leftarrow func(h_1, y_2), Woman(Paul), Man(y_2)$
C_{21}: $TaxCut(Peter) \leftarrow Man(Paul), Woman(Paul)$
C_{22}: $TaxCut(Peter) \leftarrow Woman(Paul), Man(Paul)$

4. *By definite-clause removal with respect to notSame, C_2 and C_3 are removed.*
5. *By unfolding with respect to the predicate Man, C_{15}-C_{22} are replaced with:*

C_{23}: $TaxCut(Peter) \leftarrow func(h_1, Paul), func(h_1, y_2), Woman(y_2)$
C_{24}: $TaxCut(Peter) \leftarrow func(h_1, y_1), func(h_1, Paul), Woman(y_1)$
C_{25}: $TaxCut(Peter) \leftarrow func(h_1, Paul), Woman(Paul)$
C_{26}: $TaxCut(Peter) \leftarrow func(h_1, y_1), Woman(y_1)$
C_{27}: $TaxCut(Peter) \leftarrow func(h_1, y_2), Woman(y_2)$
C_{28}: $TaxCut(Peter) \leftarrow func(h_1, Paul), Woman(Paul)$
C_{29}: $TaxCut(Peter) \leftarrow Woman(Paul)$
C_{30}: $TaxCut(Peter) \leftarrow Woman(Paul)$

6. *By definite-clause removal with respect to Man, C_8 is removed.*
7. *By unfolding with respect to the predicate Woman, C_{23}-C_{30} are replaced with:*

C_{31}: $TaxCut(Peter) \leftarrow func(h_1, Paul), func(h_1, y_2), motherOf(y_2, y)$
C_{32}: $TaxCut(Peter) \leftarrow func(h_1, y_1), func(h_1, Paul), motherOf(y_1, y)$
C_{33}: $TaxCut(Peter) \leftarrow func(h_1, Paul), motherOf(Paul, y)$
C_{34}: $TaxCut(Peter) \leftarrow func(h_1, y_1), motherOf(y_1, y)$
C_{35}: $TaxCut(Peter) \leftarrow func(h_1, y_2), motherOf(y_2, y)$
C_{36}: $TaxCut(Peter) \leftarrow func(h_1, Paul), motherOf(Paul, y)$
C_{37}: $TaxCut(Peter) \leftarrow motherOf(Paul, y)$
C_{38}: $TaxCut(Peter) \leftarrow motherOf(Paul, y)$

8. *By definite-clause removal with respect to Woman, C_4 is removed.*
9. *By unfolding with respect to the predicate motherOf, C_{31}-C_{38} are replaced with:*

C_{39}: $TaxCut(Peter) \leftarrow func(h_1, Paul), func(h_1, y_2), func(h_1, y_2), func(h_2, y)$
C_{40}: $TaxCut(Peter) \leftarrow func(h_1, y_1), func(h_1, Paul), func(h_1, y_1), func(h_2, y)$
C_{41}: $TaxCut(Peter) \leftarrow func(h_1, Paul), func(h_1, Paul), func(h_2, y)$
C_{42}: $TaxCut(Peter) \leftarrow func(h_1, y_1), func(h_1, y_1), func(h_2, y)$
C_{43}: $TaxCut(Peter) \leftarrow func(h_1, y_2), func(h_1, y_2), func(h_2, y)$
C_{44}: $TaxCut(Peter) \leftarrow func(h_1, Paul), func(h_1, Paul), func(h_2, y)$
C_{45}: $TaxCut(Peter) \leftarrow func(h_1, Paul), func(h_2, y)$
C_{46}: $TaxCut(Peter) \leftarrow func(h_1, Paul), func(h_2, y)$

10. *By definite-clause removal with respect to motherOf, C_6 is removed.*

Since $C_{41} = C_{44}$, C_{44} can be removed. Similarly, since $C_{45} = C_{46}$, C_{46} can be removed. Since C_{42} and C_{43} have the same meaning, C_{43} can be removed. The resulting clause set is $Cs_2 = \{C_{39}, C_{40}, C_{41}, C_{42}, C_{45}\}$.

Next, by the functionality constraints imposed by func-atoms in their bodies, C_{39}, C_{40}, C_{41} and C_{42} are transformed into the following clauses, respectively:

C_{47}: $TaxCut(Peter) \leftarrow func(h_1, Paul), func(h_2, y)$
C_{48}: $TaxCut(Peter) \leftarrow func(h_1, Paul), func(h_2, y)$
C_{49}: $TaxCut(Peter) \leftarrow func(h_1, Paul), func(h_2, y)$
C_{50}: $TaxCut(Peter) \leftarrow func(h_1, y_1), func(h_2, y)$

Since $C_{45} = C_{47} = C_{48} = C_{49}$, the clauses C_{47}, C_{48} and C_{49} can be removed. Since y_1 and y do not appear in the head of C_{50} and no constraint is imposed by the body of it, the clause C_{50} can be replaced with:

C_{51}: $\quad TaxCut(Peter) \leftarrow$

The resulting clause set is $Cs_3 = \{C_{45}, C_{51}\}$. Since C_{45} and C_{51} have the same head and C_{51} is a unit clause, C_{45} can be removed. The resulting clause set is $Cs_4 = \{C_{51}\}$.

The original "Tax-cut" problem is thus equivalent to the QA problem $\langle\{C_{51}\}, TaxCut(x)\rangle$, from which the answer set $\{TaxCut(Peter)\}$ is directly obtained, i.e., Peter is the only person who gets discounted tax.

6. **Fundamental Differences from Existing Approaches.** This work differs from existing works on automated reasoning (e.g., [5, 18, 19, 20]), logic programming (e.g., [8, 11, 12]) and query answering for description logics (DLs) and for DLs with rules (e.g., [2, 9, 22, 23, 24]) in the following fundamental main points:

1. *Definite clauses vs. Arbitrary clauses:* Unfolding in this paper works in the extended space ECLS_F with multi-head clauses and existentially quantified function variables. The conventional unfolding for logic programs [11, 12], which has been extensively used for transformation of Prolog programs, is not applicable in this extended space, since it is devised in the usual restricted space of definite clauses, where there is no need for finding a definite-clause part of a given set of arbitrary clauses. The extended space ECLS_F is necessary for meaning-preserving Skolemization, which is in turn essential for solving QA problems on full first-order logic based on equivalent transformation (ET).
2. *Conventional Skolemization vs. Meaning-preserving Skolemization:* Meaning-preserving Skolemization is a necessary transformation process for converting an arbitrarily given first-order formula into a logically equivalent normal form, which can then be further equivalently simplified using ET rules. Existing approaches to automated reasoning with first-order formulas, e.g., [5, 18, 19, 20], use the classical Skolemization for conversion of given formulas into normal forms. The classical Skolemization cannot be used for ET-based QA-problem solving since it does not preserve the logical meaning of a first-order formula [5].
3. *Inference-based problem solving vs. ET-based problem solving:* The ET principle opens up the possibility of employing a very large class of transformation rules for problem solving – any rule whose application always results in meaning-preserving transformation can be used [3]. As a result, various types of transformation rules, with varying expressive power, can be introduced, e.g., rules with applicability conditions, possibly involving extralogical predicates, rules with execution parts, and multi-head rules [4]. By the ET principle, the correctness of a rule is verified individually, and is not threatened by the addition of any other rule – stepwise incremental algorithm construction and improvement are inherently supported [4, 21]. By contrast, usual approaches to problem solving use only certain specific inference rules (e.g., the resolution rule [20]). When computation by SLD resolution in logic programming [8], for example, is viewed in the ET framework, expansion of a node (generation of its children) in a search tree for finding SLD-refutations corresponds to an unfolding transformation step. Accordingly, computation in logic programming can be seen as computation using only one specific class of transformation rules, i.e., single-head general unfolding-based rules. By employment of such a restricted class of rules alone, it is often difficult to achieve effective computation control, in particular, for preventing infinite computation or for improving computation efficiency.

4. *Restricted subclasses of first-order logic vs. Full first-order logic:* Without ET-based problem solving, the range of possible problem-solving methods is restricted, making it difficult to devise an effective method for coping with QA problems on full first-order logic. Existing theories for solving QA problems deal only with QA problems on restricted subclasses of first-order logic, e.g., the definite-clause subclass [8], a family of description-logic (DL) subclasses [9], an integration of the DL language $\mathcal{SHOIN}(\mathbf{D})$ and DL-safe Horn rules [2], the fragment Description Horn Logic of first-order logic [22] (which is contained in the intersection of DL and Horn first-order logic), a hybrid integration of Datalog and the DL language $\mathcal{ALC}$ [23], and that of Datalog and the DL language $\mathcal{ALCNR}$ [24].

7. **Concluding Remarks.** This paper proposes equivalent transformation rules for unfolding and definite-clause removal in the $\mathrm{ECLS_F}$ space. The unfolding transformation rule consists of two steps: (i) find a set of definite clauses from a given clause set *Cs*, which may include multi-head clauses, and (ii) unfold clauses in *Cs* using the obtained definite clauses. The definite clauses obtained by the first step can be subsequently removed when they are no longer useful for unfolding and their head predicates do not occur in the body of any query clause of a QA problem under consideration. Many QA problems can be simplified by the proposed transformation rules. However, unfolding transformation and definite-clause removal alone are in general not sufficient for simplifying all QA problems thoroughly. They should be supplemented with other kinds of equivalent transformation rules, which will be discussed in subsequent papers.

This work provides an important step towards solving QA problems on first-order logic through equivalent transformation (ET). Using our ET-based method, a QA problem $\langle K, a\rangle$ on first-order logic can be solved as follows: (i) convert the first-order formula K by meaning-preserving Skolemization [6] into a set *Cs* of extended clauses in the $\mathrm{ECLS_F}$ space, (ii) transform *Cs* by simplification using unfolding and definite-clause removal, (iii) compute all representative models of the resulting clause set by bottom-up computation [15] or by satisfiability solving [14], and (iv) find the answer set $ans(K, a)$ by computing the intersection of the representative models of the resulting clause set and the set of all ground instances of a. This approach has many potential practical applications in many significant application domains, including QA problems on natural languages (e.g., Example 5.3) and the Semantic Web (e.g., Example 5.4). Using natural language processing, a QA problem on a natural language can often be translated into a QA problem on first-order logic, which can then be input to our method. Similarly, queries concerning OWL-based ontological background knowledge in the Semantic Web, possibly with rules defining relations between individuals represented in the Semantic Web Rule Language (SWRL) [1], can be converted into QA problems on first-order logic, which can be dealt with using our method.

Acknowledgment. This research was supported by the Collaborative Research Program 2012, Information Initiative Center, Hokkaido University, and by the National Research University Project of Thailand Office of Higher Education Commission.

REFERENCES

[1] I. Horrocks, P. F. Patel-Schneider, S. Bechhofer and D. Tsarkov, OWL rules: A proposal and prototype implementation, *Journal of Web Semantics*, vol.3, no.1, pp.23-40, 2005.

[2] B. Motik, U. Sattler and R. Studer, Query answering for OWL-DL with rules, *Journal of Web Semantics*, vol.3, no.1, pp.41-60, 2005.

[3] K. Akama and E. Nantajeewarawat, Formalization of the equivalent transformation computation model, *Journal of Advanced Computational Intelligence and Intelligent Informatics*, vol.10, no.3, pp.245-259, 2006.

[4] K. Akama, E. Nantajeewarawat and H. Koike, Program generation in the equivalent transformation computation model using the squeeze method, *Perspectives of System Informatics, Lecture Notes in Computer Science*, vol.4378, pp.41-54, 2007.
[5] C.-L. Chang and R. C.-T. Lee, *Symbolic Logic and Mechanical Theorem Proving*, Academic Press, 1973.
[6] K. Akama and E. Nantajeewarawat, Meaning-preserving Skolemization, *Proc. of 2011 International Conference on Knowledge Engineering and Ontology Development*, Paris, France, pp.322-327, 2011.
[7] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi and P. F. Patel-Schneider, *The Description Logic Handbook*, 2nd Edition, Cambridge University Press, 2007.
[8] J. W. Lloyd, *Foundations of Logic Programming*, 2nd Edition, Springer-Verlag, 1987.
[9] S. Tessaris, *Questions and Answers: Reasoning and Querying in Description Logic*, Ph.D. Thesis, Department of Computer Science, The University of Manchester, UK, 2001.
[10] J. Minker, *Foundations of Deductive Databases and Logic Programming*, Morgan Kaufmann Publishers, 1988.
[11] A. Pettorossi and M. Proietti, Transformation of logic programs: Foundations and techniques, *Journal of Logic Programming*, vol.19/20, pp.261-320, 1994.
[12] A. Pettorossi and M. Proietti, Synthesis and transformation of logic programs using unfold/fold proofs, *Journal of Logic Programming*, vol.41, pp.197-230, 1999.
[13] S. He, K. Akama and B. Li, Generation of specific solvers for query-answering problems with Skolem functions, *International Journal of Innovative Computing, Information and Control*, vol.8, no.10(A), pp.6613-6628, 2012.
[14] K. Akama and E. Nantajeewarawat, Correctness of solving query-answering problems using satisfiability solvers, *Technical Report*, Hokkaido University, 2012.
[15] K. Akama and E. Nantajeewarawat, A delayed splitting bottom-up procedure for model generation, *Proc. of the 25th Australasian Joint Conference on Artificial Intelligence, Lecture Notes in Computer Science*, vol.7691, Sydney, Australia, pp.481-492, 2012.
[16] K. Akama and E. Nantajeewarawat, Conjunction-based clauses for equivalent transformation of query-answering problems, *International Journal of Future Computer and Communication*, vol.1, pp.5-8, 2012.
[17] P. F. Patel-Schneider, P. Hayes and I. Horrocks, *OWL Web Ontology Language: Semantics and Abstract Syntax*, W3C Recommendation, www.w3.org/TR/owl-semantics, 2004.
[18] M. Fitting, *First-Order Logic and Automated Theorem Proving*, 2nd Edition, Springer-Verlag, 1996.
[19] M. Newborn, *Automated Theorem Proving: Theory and Practice*, Springer-Verlag, 2000.
[20] J. A. Robinson, A machine-oriented logic based on the resolution principle, *Journal of the ACM*, vol.12, pp.23-41, 1965.
[21] K. Miura, K. Akama and H. Mabuchi, Creation of ET rules from logical formulas representing equivalent relations, *International Journal of Innovative Computing, Information and Control*, vol.5, no.2, pp.263-277, 2009.
[22] B. N. Grosof, I. Horrocks, R. Volz and S. Decker, Description logic programs: Combining logic programs with description logic, *Proc. of the 12th International World Wide Web Conference*, Budapest, Hungary, pp.48-57, 2003.
[23] F. M. Donini, M. Lenzerini, D. Nardi and A. Schaerf, $\mathcal{AL}$-log: Integrating datalog and description logics, *Journal of Intelligent Information Systems*, vol.10, no.3, pp.227-252, 1998.
[24] A. Y. Levy and M.-C. Rousset, Combining Horn rules and description logics in CARIN, *Artificial Intelligence*, vol.104, no.1-2, pp.165-209, 1998.