

A Delayed Splitting Bottom-Up Procedure for Model Generation

Kiyoshi Akama¹ and Ekawit Nantajeewarawat²

¹ Information Initiative Center, Hokkaido University, Hokkaido, Japan
`akama@iic.hokudai.ac.jp`

² Computer Science Program, Sirindhorn International Institute of Technology
Thammasat University, Pathumthani, Thailand
`ekawit@siit.tu.ac.th`

Abstract. Meaning-preserving Skolemization is essential for development of a correct and efficient method of solving query-answering problems. It requires global existential quantifications of function variables, which in turn require an extended space of logical formulas. This paper proposes a bottom-up procedure for computing a set of models that sufficiently represents the set of all models of a given clause set in the extended formula space. Instantiations of function variables often result in generation of infinitely many models. To overcome the difficulty, a model-making pattern is introduced for representing a possibly infinite number of models, and such a pattern is split as late as possible. The proposed procedure provides a method for solving query-answering problems that include unrestricted use of universal and existential quantifications.

Keywords: Query-answering problems, automated reasoning, bottom-up computation, meaning-preserving Skolemization.

1 Introduction

A *query-answering problem* (QA problem) [6,8] 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 $\langle K, a \rangle$ 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. QA problems on definite clauses have been extensively discussed in logic programming [4]. QA problems on DLs have been discussed in [8]. Answering queries in deductive databases [5] 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. The answer to $\langle K, a \rangle$ can be equivalently formulated as $(\bigcap \text{Models}(K)) \cap \text{rep}(a)$, where $\bigcap \text{Models}(K)$ is the intersection of all models of K . We adopt the following three computation phases for computing the answer to $\langle K, a \rangle$: (i) convert K into a set Cs of clauses using Skolemization, (ii) use bottom-up computation to construct a set of models that sufficiently represents the set of all models of Cs , and (iii) find the intersection of the obtained models and $\text{rep}(a)$. The following fundamental issues need to be addressed:

1. How to preserve the meaning of a logical formula in a Skolemization process? Conventional Skolemization does not preserve the meaning of a formula [2]. In order to obtain meaning-preserving Skolemization, an extended formula space that allows existential quantifications of function variables is required.
2. How to compute models of an extended clause set? A clause set in the extended space contains existentially quantified global function variables. How to compute models of such an extended clause set has not been discussed in the literature.

A solution to the first problem has been provided by our recent work [1], in which a theory for extending a space of logical formulas by incorporation of function variables was developed and how meaning-preserving Skolemization could be achieved in the obtained extended space was shown. A procedure for converting a logical formula into a set of extended clauses on the extended space was also given in [1].

This paper addresses the second problem. A set of extended clauses may contain occurrences of function variables, which can be instantiated into infinitely many functions. Instantiations of function variables therefore often result in infinitely many models. Instead of generating models themselves, patterns of model construction, called *model-making patterns*, are generated. A model-making pattern is represented by a set of atoms with parameters. In bottom-up model generation, some model-making patterns may need to be split. To reduce the number of model-making patterns to be considered, a pattern is split only when necessary. A procedure for bottom-up generation of models based on such delayed pattern splitting is proposed in this paper.

To begin with, Section 2 explains the basic idea of meaning-preserving Skolemization and introduces the extended clause space. Section 3 presents our delayed splitting bottom-up procedure for model generation. Section 4 illustrates how the procedure works. Section 5 describes fundamental differences between this work and existing approaches. Section 6 provides concluding remarks.

2 Meaning-Preserving Skolemization on an Extended Clause Space

2.1 Need for Meaning-Preserving Skolemization

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. 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 [2].

In [1], 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. The basic idea of meaning-preserving Skolemization is to use existentially quantified function variables instead of usual Skolem functions. Function variables and extended conjunctive normal forms are introduced below.

2.2 Function Constants, Function Variables and *func*-Atoms

A usual function symbol in first-order logic denotes an unevaluated function; it is used for constructing a syntactically new term from existing terms (possibly recursively) without evaluating those existing terms. A different class of functions is used in the extended space. A function in this class is an actual mathematical function; it takes ground terms as input, and associates with them an output ground term. The input ground terms are evaluated for determining the output. 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

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

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 $\bar{f}(t_1, \dots, t_n) = t_{n+1}$.

2.3 An Extended Clause Space

A procedure for converting a first-order logical formula into an equivalent formula in an extended conjunctive normal form, called an existentially quantified conjunctive normal form (ECNF), is given in [1]. To define an ECNF, an extended clause is introduced.

Extended Clauses. An *extended clause* C is an extended formula of the form

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

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

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

The sets $\{a_1, \dots, a_n\}$ and $\{b_1, \dots, b_p, \mathbf{f}_1, \dots, \mathbf{f}_q\}$ are called the *left-hand side* and the *right-hand side*, respectively, of the extended clause C , denoted by $lhs(C)$ and $rhs(C)$, 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 $lhs(C)$ is called the *head* of C , denoted by $head(C)$, and the set $rhs(C)$ is also called the *body* of C , denoted by $body(C)$. 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.

Existentially Quantified Conjunctive Normal Forms. A formula in an *existentially quantified conjunctive normal form* (ECNF) is an extended formula of the form

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

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

An Extended Clause Space and QA Problems Thereon. The set of all ECNFs is referred to as the *extended clause space* ($ECLS_F$). By the above identification of an ECNF with a clause set, we often regard an element of $ECLS_F$ as a set of (extended) clauses. Given a QA problem $\langle K, a \rangle$ on first-order logic, the first-order formula K is converted by meaning-preserving Skolemization [1] into a clause set Cs in the $ECLS_F$ space. A QA problem $\langle Cs, a \rangle$ such that Cs is a clause set in $ECLS_F$ and a is a usual atom is called a *QA problem on $ECLS_F$* .

3 A Bottom-Up Procedure for Model Generation

After introducing the notions of parameterized terms and parameterized atoms (Section 3.1), how to split a pattern of ground atom sets by parameter splitting is described (Section 3.2). Solving equalities and inequalities with occurrences of parameterized terms is presented (Section 3.3). It is followed by a one-step bottom-up inference procedure (Section 3.4), which is used in our main procedure for model generation (Section 3.5).

3.1 Parameterized Terms and Parameterized Atoms

A parameter ρ takes the form $p : \text{except}(E)$, where p is a marker and E is a set of ground usual terms. It represents any arbitrary ground usual term that does not belong to E . When $E = \emptyset$, it represents any arbitrary ground usual term and the expression $p : \text{except}(E)$ is also written simply as p .

A *parameterized term* (for short, *P-term*) is defined as follows: (i) constants, usual variables, and parameters are P-terms, and (ii) if f is an m -ary function symbol and $t_1, \dots, t_m$ are P-terms, then $f(t_1, \dots, t_m)$ is a P-term. A *ground P-term* is a P-term with no occurrence of any usual variable. A *parameterized atom* (for short, *P-atom*) is an expression of the form $p(t_1, \dots, t_n)$, where p is an n -ary predicate symbol and $t_1, \dots, t_n$ are P-terms. It is called a *ground P-atom* when $t_1, \dots, t_n$ are ground P-terms. Let $\mathcal{G}_P$ denote the set of all ground P-atoms and $\text{pow}(\mathcal{G}_P)$ the power set of $\mathcal{G}_P$.

3.2 Splitting Ground P-Atom Sets by Parameter Splitting

Assume that ρ is a parameter $p : \text{except}(E)$ and t is a ground term such that $t \notin E$. A set of ground P-terms with occurrence of ρ may be split by splitting the parameter ρ into two cases: (i) the case when ρ represents t , and (ii) the case when it represents a usual ground term that does not belong to $E \cup \{t\}$.

Given $M \subseteq \text{pow}(\mathcal{G}_P)$, a parameter $\rho = p : \text{except}(E)$, and a ground term $t \notin E$, $\text{SPLIT}(M, \rho, t)$ is defined as the subset of $\text{pow}(\mathcal{G}_P)$ obtained from M by splitting ρ with respect to t as follows:

1. Let $M_\rho = \{m \mid (m \in M) \ \& \ (\rho \text{ occurs in } m)\}$.
2. Let ρ_{new} be a new parameter $p_{\text{new}} : \text{except}(E \cup \{t\})$, where p_{new} is a new marker.
3. Let $\sigma = \{\rho/t\}$ and $\sigma' = \{\rho/\rho_{\text{new}}\}$.
4. Let $\text{SPLIT}(M, \rho, t) = (M - M_\rho) \cup M_\rho\sigma \cup M_\rho\sigma'$.

3.3 Solving Equalities and Inequalities

Let Es be a set of equalities and inequalities, with occurrences of P-terms. $\text{SOLVE}(Es)$ is defined by

$$\text{SOLVE}(Es) = \text{SOLVE}(\emptyset, Es),$$

where for any set S of bindings and any set $\hat{Es}$ of equalities and inequalities, $\text{SOLVE}(S, \hat{Es})$ is defined as follows:

1. If $\hat{Es} = \emptyset$, then $\text{SOLVE}(S, \hat{Es}) = S$.
2. Assume that f and g are function symbols and $t_1, \dots, t_m, t'_1, \dots, t'_n$ are P-terms.
 - (a) If $\hat{Es} = \{(f(t_1, \dots, t_m) = g(t'_1, \dots, t'_n))\} \cup \tilde{Es}$, then:
 - i. If $f = g$ and $m = n$, then

$$\text{SOLVE}(S, \hat{Es}) = \text{SOLVE}(S, \{(t_1 = t'_1), \dots, (t_m = t'_m)\} \cup \tilde{Es}).$$

- ii. If $f \neq g$ or $m \neq n$, then $\text{SOLVE}(S, \hat{E}s) = \langle \text{FAIL} \rangle$.
- (b) If $\hat{E}s = \{(f(t_1, \dots, t_m) \neq g(t'_1, \dots, t'_n))\} \cup \tilde{E}s$, then:
 - i. If $f = g$ and $m = n$, then

$$\text{SOLVE}(S, \hat{E}s) = \text{SOLVE}(S, \{(t_i \neq t'_i)\} \cup \tilde{E}s),$$

where i is a selected index in $\{1, \dots, m\}$.

- ii. If $f \neq g$ or $m \neq n$, then $\text{SOLVE}(S, \hat{E}s) = \text{SOLVE}(S, \tilde{E}s)$.
- 3. If $\hat{E}s = \{(t_1 = t_2)\} \cup \tilde{E}s$, where t_1, t_2 are usual terms, then:
 - (a) If t_1 and t_2 are unifiable, then $\text{SOLVE}(S, \hat{E}s) = \text{SOLVE}(S \cup \sigma, \tilde{E}s\sigma)$, where σ is the most general unifier of t_1 and t_2 .
 - (b) If they are not unifiable, then $\text{SOLVE}(S, \hat{E}s) = \langle \text{FAIL} \rangle$.
- 4. If $\hat{E}s = \{(t_1 \neq t_2)\} \cup \tilde{E}s$, where t_1, t_2 are usual terms, then:
 - (a) If t_1 and t_2 are not unifiable, then $\text{SOLVE}(S, \hat{E}s) = \text{SOLVE}(S, \tilde{E}s)$.
 - (b) If t_1 and t_2 are equal, then $\text{SOLVE}(S, \hat{E}s) = \langle \text{FAIL} \rangle$.
 - (c) If t_1 and t_2 are not equal but they are unifiable, then $\text{SOLVE}(S, \hat{E}s) = \langle \text{OUTRANGE} \rangle$.
- 5. If ρ is a parameter $p : \text{except}(E)$ and $\hat{E}s = \{(\rho = \hat{t})\} \cup \tilde{E}s$, then:
 - (a) If $\hat{t}$ is the parameter ρ itself, then $\text{SOLVE}(S, \hat{E}s) = \text{SOLVE}(S, \tilde{E}s)$.
 - (b) If $\hat{t}$ is a usual variable v , then $\text{SOLVE}(S, \hat{E}s) = \text{SOLVE}(S \cup \{v/\rho\}, \tilde{E}s\{v/\rho\})$.
 - (c) If $\hat{t}$ is a ground term in E , then $\text{SOLVE}(S, \hat{E}s) = \langle \text{FAIL} \rangle$.
 - (d) If $\hat{t}$ is a ground term and $\hat{t} \notin E$, then $\text{SOLVE}(S, \hat{E}s) = \langle \text{SPLITREQ}, \rho, \hat{t} \rangle$.
 - (e) Otherwise $\text{SOLVE}(S, \hat{E}s) = \langle \text{OUTRANGE} \rangle$.
- 6. If ρ is a parameter $p : \text{except}(E)$ and $\hat{E}s = \{(\rho \neq \hat{t})\} \cup \tilde{E}s$, then:
 - (a) If $\hat{t}$ is a ground term in E , then $\text{SOLVE}(S, \hat{E}s) = \text{SOLVE}(S, \tilde{E}s)$.
 - (b) If $\hat{t}$ is a ground term and $\hat{t} \notin E$, then $\text{SOLVE}(S, \hat{E}s) = \langle \text{SPLITREQ}, \rho, \hat{t} \rangle$.
 - (c) Otherwise $\text{SOLVE}(S, \hat{E}s) = \langle \text{OUTRANGE} \rangle$.

Note that $\text{SOLVE}(S, \hat{E}s)$ is not uniquely determined, owing to nondeterministic selection of the index i in Case 2(b)i.

3.4 A One-Step Bottom-Up Inference Procedure

Next, a one-step bottom-up inference procedure is presented. It takes a subset M of $\text{pow}(\mathcal{G}_P)$ as input, and nondeterministically outputs one of the following results: (i) a set M' of ground atom sets with parameters, (ii) $\langle \text{NOCHANGE} \rangle$, (iii) $\langle \text{OUTRANGE} \rangle$. The procedure works as follows:

1. Select $m \in M$.
2. Select a clause C in Cs .
3. Let A be the set of all usual atoms in $\text{rhs}(C)$. Assuming that $A = \{a_1, \dots, a_n\}$, where $n \geq 0$, perform the following steps:
 - (a) Select $A' = \{a'_1, \dots, a'_n\} \subseteq m$.

- (b) Let EQ be the set of equalities and inequalities constructed from all eq -atoms and neg -atoms in $rhs(C)$.
- (c) Let $U = \text{SOLVE}(\{(a_1 = a'_1), \dots, (a_n = a'_n)\} \cup EQ)$. Then:
 - i. If U is a substitution, then let $\theta = U$.
 - ii. If $U = \langle \text{SPLITREQ}, \rho, t \rangle$, then let $M' = \text{SPLIT}(M, \rho, t)$ and stop with the output M' .
 - iii. If $U = \langle \text{FAIL} \rangle$, then stop with the output $\langle \text{NOCHANGE} \rangle$.
 - iv. If $U = \langle \text{OUTRANGE} \rangle$, then stop with the output $\langle \text{OUTRANGE} \rangle$.
- 4. Let F be the set of all $func$ -atoms in $rhs(C)$.
- 5. While $F \neq \emptyset$, perform the following steps:
 - (a) Select a $func$ -atom $\mathbf{f} = func(f, t_1, \dots, t_n, t_{n+1}) \in F$, where $n \geq 0$, such that $t_1\theta, \dots, t_n\theta$ are ground terms. If there is no such $func$ -atom in F , then stop with the output $\langle \text{OUTRANGE} \rangle$.
 - (b) Select a $func$ -atom $f(t_1\theta, \dots, t_n\theta, t') \in m$. If there is no such $func$ -atom in m , then let

$$M' = (M - \{m\}) \cup \{m \cup \{f(t_1\theta, \dots, t_n\theta, \rho_{\text{new}})\}\},$$

where ρ_{new} is a new parameter $p_{\text{new}} : \text{except}(\emptyset)$ with a new marker p_{new} , and stop with the output M' .

- (c) Let $U = \text{SOLVE}(\{(t_{n+1}\theta = t')\})$. Then:
 - i. If U is a substitution σ , then let $\theta' = \theta \circ \sigma$.
 - ii. If $U = \langle \text{SPLITREQ}, \rho, t \rangle$, then let $M' = \text{SPLIT}(M, \rho, t)$ and stop with the output M' .
 - iii. If $U = \langle \text{FAIL} \rangle$, then stop with the output $\langle \text{NOCHANGE} \rangle$.
 - iv. If $U = \langle \text{OUTRANGE} \rangle$, then stop with the output $\langle \text{OUTRANGE} \rangle$.
- (d) Remove $\mathbf{f}$ from F .
- 6. If $lhs(C\theta') \cap m \neq \emptyset$, then stop with the output $\langle \text{NOCHANGE} \rangle$.
- 7. Assume that $lhs(C\theta') = \{b_1, \dots, b_{n'}\}$, where $n' \geq 0$. Then let

$$M' = (M - \{m\}) \cup \{m \cup \{b_1\}\} \cup \dots \cup \{m \cup \{b_{n'}\}\}$$

and stop with the output M' .

3.5 A Delayed Splitting Bottom-Up Procedure

For any given subset M of $pow(\mathcal{G}_P)$, let $\text{POSS}(M)$ and $\text{DONE}(M)$ be defined as follows: (i) $\text{POSS}(M)$ is the set of all sets M' of ground atom sets with parameters possibly obtained when applying the above procedure to the input M ; (ii) $\text{DONE}(M) = \langle \text{SUCC} \rangle$ if $\langle \text{NOCHANGE} \rangle$ is the only possible output when applying the above procedure to the input M . Given a clause set Cs in ECLS_F , models of Cs , represented by a subset M of $pow(\mathcal{G}_P)$, are generated using a delayed splitting bottom-up procedure described below.

- 1. Initially, let $M = \{m_0\}$, where $m_0 = \emptyset$.

2. While $\text{Poss}(M) \neq \emptyset$, perform the following two steps:
 - (a) Select $M' \in \text{Poss}(M)$.
 - (b) $M := M'$.
3. If $\text{DONE}(M) = \langle \text{SUCC} \rangle$, then stop with M being the output, otherwise stop with a failure.

Now let $\langle Cs, a \rangle$ be a given QA problem on ECLS_F . To compute the answer to $\langle Cs, a \rangle$, we first compute a subset M of $\text{pow}(\mathcal{G}_P)$ using the delayed splitting bottom-up procedure above. Then the computed answer to $\langle Cs, a \rangle$ is the set $(\bigcap M) \cap \text{rep}(a)$. Note that although P-atoms may occur in the set M obtained from the delayed splitting bottom-up procedure, they do not belong to $\text{rep}(a)$ and therefore do not occur in the computed answer.

4 Examples

Three examples illustrating application of the presented bottom-up procedure are given below. It is assumed that a usual variable as well as a function variable begins with an asterisk.

Example 1. Let Cs consist of the following five clauses, where $*h$ is a 0-ary function variable and neq stands for “not equal”:

$$\begin{array}{ll}
 C_1: & q(*x) \leftarrow r(*x) \\
 C_3: & s(*x) \leftarrow \text{func}(*h, *x) \\
 C_5: & \leftarrow s(B) \\
 C_2: & r(*x) \leftarrow \text{func}(*h, *x) \\
 C_4: & \leftarrow s(*x), \text{neq}(*x, A), \text{neq}(*x, B)
 \end{array}$$

Assume that q -, r -, and s -atoms are usual atoms and neq -atoms are constraint atoms. Consider a QA problem $\text{prb}_1 = \langle Cs, q(*x) \rangle$. Let $M_0 = \{m_0\}$, where $m_0 = \emptyset$. The one-step inference procedure in Section 3.4 is applied successively with M_0 being the first input as follows:

1. M_0 is input to the procedure. The set m_0 is selected at Step 1. Suppose that C_3 is selected at Step 2. At Step 3, $\text{SOLVE}(\emptyset)$ is called and it returns the identity substitution. At Step 5b, the procedure constructs $m_1 = m_0 \cup \{*h(p_1)\} = \{*h(p_1)\}$, where p_1 is a parameter marker, and outputs $M_1 = \{m_1\}$.
2. M_1 is input to the procedure. The set m_1 is selected at Step 1. Suppose that C_3 is selected at Step 2. At Step 5c, $\text{SOLVE}(\{(*x = p_1)\})$ is called, with the output substitution being $\sigma = \{*x/p_1\}$. At Step 7, the procedure constructs $m_2 = m_1 \cup \{s(*x)\sigma\} = \{*h(p_1), s(p_1)\}$ and outputs $M_2 = \{m_2\}$.
3. M_2 is input to the procedure. The set m_2 is selected at Step 1. Suppose that C_4 is selected at Step 2. At Step 3, $\text{SOLVE}(\{(*x = p_1), (*x \neq A), (*x \neq B)\})$ is called and it returns $\langle \text{SPLITREQ}, p_1, A \rangle$; accordingly, m_2 is split into
 - $m_3 = \{*h(A), s(A)\}$, and
 - $m_4 = \{*h(p_2 : \text{except}(\{A\})), s(p_2 : \text{except}(\{A\}))\}$,
 where p_2 is a new marker. The procedure outputs $M_3 = \{m_3, m_4\}$.

4. M_3 is input to the procedure. Suppose that m_4 is selected at Step 1 and C_4 is selected at Step 2. At Step 3, $\text{SOLVE}(\{(*x = p_2 : \text{except}(\{A\})), (*x \neq A), (*x \neq B)\})$ is called, with $\langle \text{SPLITREQ}, p_2 : \text{except}(\{A\}), B \rangle$ being returned. So m_4 is split into
 - $m_5 = \{*h(B), s(B)\}$, and
 - $m_6 = \{*h(p_3 : \text{except}(\{A, B\})), s(p_3 : \text{except}(\{A, B\}))\}$,
 where p_3 is a new marker, and the procedure outputs $M_4 = \{m_3, m_5, m_6\}$.
5. M_4 is input to the procedure. Suppose that m_6 is selected at Step 1 and C_4 is selected at Step 2. At Step 3, $\text{SOLVE}(\{(*x = p_3 : \text{except}(\{A, B\})), (*x \neq A), (*x \neq B)\})$ is called and it returns the identity substitution (since the possibility of splitting the parameter $p_3 : \text{except}(\{A, B\})$ is excluded by the inequalities $(*x \neq A)$ and $(*x \neq B)$). At Step 7, since C_4 is a negative clause, m_6 is removed from M_4 and the procedure outputs $M_5 = \{m_3, m_5\}$.
6. M_5 is input to the procedure. Suppose that m_5 is selected at Step 1 and C_5 is selected at Step 2. At Step 3, $\text{SOLVE}(\{(B = B)\})$ is called and it returns the identity substitution. At Step 7, since C_5 is a negative clause, m_5 is removed from M_5 and the procedure outputs $M_6 = \{m_3\}$.
7. M_6 is input to the procedure. The set m_3 is selected at Step 1. Suppose that C_2 is selected at Step 2. At Step 3, $\text{SOLVE}(\emptyset)$ is called and it returns the identity substitution. At Step 5c, $\text{SOLVE}(\{(*x = A)\})$ is called and it returns the substitution $\sigma = \{*x/A\}$. At Step 7, the procedure constructs $m_7 = m_3 \cup \{r(*x)\sigma\} = \{*h(A), s(A), r(A)\}$ and outputs $M_7 = \{m_7\}$.
8. M_7 is input to the procedure. The set m_7 is selected at Step 1. Suppose that C_1 is selected at Step 2. At Step 3, $\text{SOLVE}(\{(*x = A)\})$ is called and it returns the substitution $\theta = \{*x/A\}$. At Step 7, the procedure constructs $m_8 = m_7 \cup \{q(*x)\theta\} = \{*h(A), s(A), r(A), q(A)\}$ and outputs $M_8 = \{m_8\}$.

Since $\text{POSS}(M_8) = \emptyset$ and $\text{DONE}(M_8) = \langle \text{SUCC} \rangle$, the main algorithm stops with M_8 being the output. The answer to prb_1 is $(\bigcap M_8) \cap \text{rep}(q(*x)) = \{q(A)\}$. $\square$

Example 2. Consider a QA problem $prb_2 = \langle Cs, q(*x) \rangle$, where Cs consists of the following four clauses, assuming that $*h$ is a 0-ary function variable:

$$\begin{array}{ll}
 C_1 : & q(*x), r(*x) \leftarrow s(*x), \text{func}(*h, *x) \\
 C_2 : & s(1) \leftarrow \\
 C_3 : & s(2) \leftarrow \\
 C_4 : & \leftarrow q(1)
 \end{array}$$

Note that C_1 is a multi-head clause. Again assume that q -, r -, and s -atoms are usual atoms. Let $M_0 = \{m_0\}$, where $m_0 = \emptyset$. The one-step bottom-up inference procedure is applied successively with M_0 being the first input as follows:

1. M_0 is input to the procedure. Suppose that C_2 is selected. The procedure constructs $m_1 = m_0 \cup \{s(1)\} = \{s(1)\}$ and outputs $M_1 = \{m_1\}$.
2. M_1 is input to the procedure. Suppose that C_3 is selected. The procedure constructs $m_2 = m_1 \cup \{s(2)\} = \{s(1), s(2)\}$ and outputs $M_2 = \{m_2\}$.
3. M_2 is input to the procedure. Suppose that C_1 is selected. At Step 3, supposing that $s(1)$ is selected, $\text{SOLVE}(\{(*x = 1)\})$ is called and it returns the substitution $\{*x/1\}$. At Step 5b, the procedure constructs $m_3 = m_2 \cup \{*h(p_1)\} = \{s(1), s(2), *h(p_1)\}$, where p_1 is a parameter marker, and outputs $M_3 = \{m_3\}$.

4. M_3 is input to the procedure. Suppose that C_1 is selected. At Step 3, supposing that $s(1)$ is selected, $\text{SOLVE}(\{(*x = 1)\})$ is called and it returns the substitution $\{*x/1\}$. At Step 5c, $\text{SOLVE}(\{(1 = p_1)\})$ is called and it returns $\langle \text{SPLITREQ}, p_1, 1 \rangle$. As a result, m_3 is split into
 - $m_4 = \{s(1), s(2), *h(1)\}$, and
 - $m_5 = \{s(1), s(2), *h(p_2 : \text{except}(\{1\}))\}$,
 where p_2 is a new marker. The procedure outputs $M_4 = \{m_4, m_5\}$.
5. M_4 is input to the procedure. Suppose that m_4 and C_1 are selected. At Step 3, supposing that $s(1)$ is selected, $\text{SOLVE}(\{(*x = 1)\})$ is called and it returns the substitution $\{*x/1\}$. At Step 5c, $\text{SOLVE}(\{(1 = 1)\})$ is called and it returns the identity substitution. At Step 7, the procedure construct
 - $m_6 = m_4 \cup \{q(1)\} = \{s(1), s(2), *h(1), q(1)\}$, and
 - $m_7 = m_4 \cup \{r(1)\} = \{s(1), s(2), *h(1), r(1)\}$,
 and outputs $M_5 = \{m_6, m_7, m_5\}$.
6. M_5 is input to the procedure. Suppose that m_6 and C_4 are selected. Since C_4 is a negative clause, m_6 is removed at Step 7. The procedure outputs $M_6 = \{m_7, m_5\}$.
7. M_6 is input to the procedure. Suppose that m_5 and C_1 are selected. At Step 3, supposing that $s(2)$ is selected, $\text{SOLVE}(\{(*x = 2)\})$ is called and it returns the substitution $\{*x/2\}$. At Step 5c, $\text{SOLVE}(\{(2 = p_2 : \text{except}(\{1\}))\})$ is called and it returns $\langle \text{SPLITREQ}, p_2 : \text{except}(\{1\}), 2 \rangle$. Then m_5 is split into
 - $m_8 = \{s(1), s(2), *h(2)\}$, and
 - $m_9 = \{s(1), s(2), *h(p_3 : \text{except}(\{1, 2\}))\}$,
 where p_3 is a new marker. The procedure outputs $M_7 = \{m_7, m_8, m_9\}$.
8. M_7 is input to the procedure. Suppose that m_8 and C_1 are selected. At Step 3, supposing that $s(2)$ is selected, $\text{SOLVE}(\{(*x = 2)\})$ is called and it returns the substitution $\{*x/2\}$. At Step 5c, $\text{SOLVE}(\{(2 = 2)\})$ is called and it returns the identity substitution. At Step 7, the procedure construct
 - $m_{10} = m_8 \cup \{q(2)\} = \{s(1), s(2), *h(2), q(2)\}$, and
 - $m_{11} = m_8 \cup \{r(2)\} = \{s(1), s(2), *h(2), r(2)\}$,
 and outputs $M_8 = \{m_7, m_{10}, m_{11}, m_9\}$.

Since $\text{POSS}(M_8) = \emptyset$ and $\text{DONE}(M_8) = \langle \text{SUCC} \rangle$, the main algorithm stops with the output M_8 . The answer to prb_2 is $(\bigcap M_8) \cap \text{rep}(q(*x)) = \emptyset$. $\square$

Example 3. Next, consider the “Tax-cut” problem discussed in [6]. This problem is to find all persons who can have discounted tax, with the knowledge consisting of the following statements: (i) Any person who has two children or more can get discounted tax. (ii) Men and women are not the same. (iii) A person’s mother is always a woman. (iv) Peter has a child, who is someone’s mother. (v) Peter has a child named Paul. (vi) Paul is a man. These six statements are represented by the following extended clauses:

- C_1 : $\text{TaxCut}(*x) \leftarrow \text{hasChild}(*x, *y), \text{hasChild}(*x, *z), \text{notSame}(*y, *z)$
- C_2 : $\text{notSame}(*x, *y) \leftarrow \text{Man}(*x), \text{Woman}(*y)$
- C_3 : $\leftarrow \text{notSame}(*x, *x)$
- C_4 : $\text{Woman}(*x) \leftarrow \text{motherOf}(*x, *y)$

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 fourth statement (i.e., “Peter has a child, who is someone’s mother”), where $*h_1$ and $*h_2$ are 0-ary function variables. The “Tax-cut” problem is then formulated as a QA problem $prb_3 = \langle Cs, TaxCut(*x) \rangle$, where Cs consists of the clauses C_1 – C_8 above.

Let $M_0 = \{m_0\}$, where $m_0 = \emptyset$. Taking M_0 as the first input, the one-step inference procedure is applied successively as follows:

1. In the first two application of the procedure, suppose that C_6 is selected. The obtained output is $M_1 = \{m_1\}$, where $m_1 = \{*h_1(p_1), *h_2(p_2)\}$ and p_1 and p_2 are parameter markers.
2. In the next six application of the procedure, suppose that the clauses C_5 , C_6 , C_7 , C_8 , C_4 , and C_2 are selected successively. The obtained output is $M_2 = \{m_2\}$, where $m_2 = \{hasChild(Peter, p_1), motherOf(p_1, p_2), hasChild(Peter, Paul), Man(Paul), Woman(p_1), notSame(Paul, p_1)\}$.
3. M_2 is input to the procedure. Suppose that C_3 is selected. At Step 3, $SOLVE(\{(*x = Paul), (*x = p_1)\})$ is called and it returns $\langle SPLITREQ, p_1, Paul \rangle$. Accordingly, m_2 is split into
 - $m_3 = m_2\sigma$, where $\sigma = \{p_1/Paul\}$, and
 - $m_4 = m_2\sigma'$, where $\sigma' = \{p_1/p_3 : except(\{Paul\})\}$ and p_3 is a new marker.
 The procedure outputs $M_3 = \{m_3, m_4\}$.
4. M_3 is input to the procedure. Suppose that m_3 and C_3 are selected. At Step 3, $SOLVE(\{(*x = Paul)\})$ is called and it returns the substitution $\{*x/Paul\}$. Since C_3 is a negative clause, m_3 is removed at Step 7. The procedure outputs $M_4 = \{m_4\}$.
5. M_4 is input to the procedure. Suppose that C_1 is selected. The procedure outputs $M_5 = \{m_5\}$, where $m_5 = m_4 \cup \{TaxCut(Peter)\}$.

Now $POSS(M_5) = \emptyset$ and $DONE(M_5) = \langle SUCC \rangle$. The main algorithm then stops with the output M_5 . The answer to the “Tax-cut” problem is thus $(\bigcap M_5) \cap rep(TaxCut(*x)) = \{TaxCut(Peter)\}$. $\square$

5 Fundamental Differences from Existing Approaches

This work differs from existing approaches for model generation, e.g., [3,7], and those for solving QA problems, e.g., [4,6,8], in the following main points:

1. *Use of meaning-preserving Skolemization*: Existing theories do not use meaning-preserving Skolemization. They use usual Skolemization, which does not preserve the meaning of a logical formula [2]. A model of a given formula is not necessarily a model of the formula obtained from it by usual Skolemization. Without meaning-preserving Skolemization, the range of possible processing methods is restricted, making it difficult to devise an effective solution for dealing with QA problems.

2. *Bottom-up computation with existentially quantified function variables:* Since usual Skolemization results in clauses with usual terms, previously existing theories do not use function variables. Resolution is used for usual proof methods [2,4], which deal with usual clauses without function variables. The use of function variables presents a new challenge to bottom-up computation. A fundamental distinction between our proposed algorithm and usual bottom-up proof methods is that our algorithm deals with instantiations of function variables whose scope covers an entire clause set, while the usual methods consider only instantiations of usual variables within an individual clause.
3. *Use of model-making patterns:* Instantiations of a function variable yield infinitely many potential model-generation cases to be considered, making it impossible to examine all cases individually. Model-making patterns classify an infinite number of model-generation cases, allowing one to handle all those cases using finite representation.

6 Concluding Remarks

This paper has proposed a bottom-up model-generation procedure in the extended space with function variables. To solve infiniteness caused by the instantiation of function variables, model-making patterns are used for model representation. Such patterns classify an infinite number of model-generation cases, making it possible to examine all those cases using finite representation. To reduce the number of model-making patterns to be considered, a pattern is split only when needed. The work provides a basis for construction of QA-problem solvers.

References

1. Akama, K., Nantajeewarawat, E.: Meaning-Preserving Skolemization. In: 2011 International Conference on Knowledge Engineering and Ontology Development, Paris, France, pp. 322–327 (2011)
2. Chang, C.-L., Lee, R.C.-T.: Symbolic Logic and Mechanical Theorem Proving. Academic Press (1973)
3. Hasegawa, R., Fujita, H., Koshimura, M.: MGTP: A Model Generation Theorem Prover—Its Advanced Features and Applications. In: Galmiche, D. (ed.) TABLEAUX 1997. LNCS, vol. 1227, pp. 1–15. Springer, Heidelberg (1997)
4. Lloyd, J.W.: Foundations of Logic Programming, 2nd edn. Springer (1987)
5. Minker, J.: Foundations of Deductive Databases and Logic Programming. Morgan Kaufmann Publishers (1988)
6. Motik, B., Sattler, U., Studer, R.: Query Answering for OWL-DL with Rules. Journal of Web Semantics 3, 41–60 (2005)
7. Schütz, H., Geisler, T.: Efficient Model Generation Through Compilation. In: McRobbie, M.A., Slaney, J.K. (eds.) CADE 1996. LNCS, vol. 1104, pp. 433–447. Springer, Heidelberg (1996)
8. Tessaris, S.: Questions and Answers: Reasoning and Querying in Description Logic. PhD Thesis, Department of Computer Science, The University of Manchester, UK (2001)