Side-Change Transformation

Kiyoshi Akama[1], Ekawit Nantajeewarawat[2] and Taketo Akama[3]
[1]*Information Initiative Center, Hokkaido University, Sapporo, Japan*
[2]*Computer Science, Sirindhorn International Institute of Technology, Thammasat University, Pathumthani, Thailand*
[3]*Modeleet Labs, Sapporo, Japan*

Keywords: Model-Intersection Problem, Extended Clause, Equivalent Transformation, Side-Change Transformation, Computation Control.

Abstract: Many logical problems, such as proof problems and query-answering problems, can be mapped into model-intersection (MI) problems, which constitute one of the largest and most fundamentally important classes of logical problems. To solve MI problems, many equivalent transformation rules have been employed. In this paper, we introduce a new transformation, called side-change transformation, and propose unfolding/side-change computation control, i.e., when neither unfolding nor definite-clause removal is applicable, an attempt is made to transform a given problem using side-change transformation so as to derive an equivalent problem to which unfolding is applicable. The correctness of side-change transformation is shown. While a resolution-based proof method increases problem size monotonically no matter what control is taken, a reduction of problem size can often be achieved by using the unfolding/side-change control.

1 INTRODUCTION

Proof problems and query-answering (QA) problems are two most important and largest classes of logical problems, and many solutions to their subclasses have been intensively studied. When we try to solve some of these problems on $\mathrm{FOL_c}$, i.e., the set of all first-order formulas with built-in constraint atoms, we bump up against the fact that none of these problem classes have been solved entirely so far. The conventional Skolemization (Chang and Lee, 1973; Fitting, 1996), by which neither the satisfiability nor the logical meaning of a formula in $\mathrm{FOL_c}$ can be preserved in general (Akama and Nantajeewarawat, 2015b), has been a major hindrance to solve problems in these two classes correctly.

To overcome the difficulty, new Skolemization, called meaning-preserving Skolemization (MPS), an extended clause space, called $\mathrm{ECLS_F}$, and model-intersection (MI) problems were introduced (Akama and Nantajeewarawat, 2011; Akama and Nantajeewarawat, 2015a). All proof problems and all QA problems on $\mathrm{FOL_c}$ have been mapped using MPS, with the answers to them being preserved, into MI problems on $\mathrm{ECLS_F}$ (Akama and Nantajeewarawat, 2015a; Akama and Nantajeewarawat, 2016a), thereby guaranteeing the correctness for solving a large class of logical problems. MI problems on $\mathrm{ECLS_F}$ constitute one of the largest classes of logical problems and are of fundamental importance.

To solve MI problems, we use many equivalent transformation (ET) rules. An appropriate strategy for applying ET rules is essential to solve logical problems efficiently. One strategy is to repeatedly apply unfolding and definite-clause removal (Akama and Nantajeewarawat, 2016b), by which a decrease of problem size can be expected. However, in the extended clause space $\mathrm{ECLS_F}$, unfolding is not necessarily applicable to an atom in the right-hand side of a clause, and computation may often reaches a state in which neither unfolding nor definite-clause removal can be applied.

To get out of such a state, we introduce in this paper a new transformation called *side-change transformation* on the extended space $\mathrm{ECLS_F}$, by which we try to transform a given problem in order to obtain an equivalent problem to which unfolding or definite-clause removal is applicable. The correctness of side-change transformation is shown.

Side-change transformation, which moves atoms to the other side of a clause, is a very simple transformation. It does not change problem size (the number of clauses and the number of atoms in the set of clauses representing a problem). However, side-

Akama, K., Nantajeewarawat, E. and Akama, T.
Side-Change Transformation.
DOI: 10.5220/0006934802370246
In *Proceedings of the 10th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K 2018) - Volume 2: KEOD*, pages 237-246
ISBN: 978-989-758-330-8

change transformation contributes to efficient computation when it is used together with extended unfolding transformation in the space $\mathrm{ECLS_F}$, which is larger than the clause space taken by the conventional theory (Akama and Nantajeewarawat, 2016b).

The rest of this paper is organized as follows: Section 2 introduces side-change transformation and explains its typical usage. Section 3 illustrates unfolding/side-change computation control for solving MI problems. Section 4 begins with a review of theoretical foundation concerning the extended clause space $\mathrm{ECLS_F}$ and the semantics of a set of extended clauses. It then formalizes side-change transformation and proves the correctness of this transformation. Section 5 presents some experiments. Section 6 concludes the paper.

2 SIDE-CHANGE TRANSFORMATION AND ITS USAGE

2.1 Side-Change Transformation

Let Cs be a clause set and p a predicate occurring in Cs. An atom whose predicate is p is referred to as a p-atom. The clause set Cs can be transformed by *side-change transformation* of p-atoms (also called side-change transformation for p) as follows:

1. Determine a new predicate $notp$ for p.
2. Move each p-atom in each clause to its opposite side in the same clause (i.e., from the left-hand side to the right-hand side and vice versa) with its predicate being changed from p to $notp$.

For example, a singleton clause set

$$\{(p(1,x), q(x,y) \leftarrow p(4,x), p(5,x), r(y))\}$$

is transformed by side-change transformation of p-atoms into

$$\{(notp(4,x), notp(5,x), q(x,y) \leftarrow notp(1,x), r(y))\}.$$

Let the clause set obtained from Cs by side-change transformation of p-atoms into $notp$-atoms be denoted by $\textsc{SideCh}(Cs, p, notp)$.

2.2 Sufficient Conditions for Enabling Unfolding Transformation

Let Cs be a set of clauses. To apply unfolding or definite-clause removal to Cs, we need to determine a set D of definite clauses and a predicate p such that (i) D is a subset of Cs, (ii) the head of each definite clause in D is a p-atom, and (iii) no p-atom appears in the left-hand side of any clause in $Cs - D$.

When unfolding is not applicable, side-change transformation is useful for deriving from Cs a new set of clauses to which unfolding or definite-clause removal can be applied. Two types of its usage that enable unfolding are described below.

2.2.1 Type 1

Let p be a predicate and $notp$ a new predicate determined for side-change transformation for p. Unfolding with respect to $notp$-atoms can be applied after side-change transformation for p if each clause C in Cs satisfies the following two conditions:

1. There is at most one p-atom in the right-hand side of C.
2. If there is exactly one p-atom in the right-hand side of C, then all user-defined atoms in the left-hand side of C are p-atoms.

Example 1. Consider the following clauses:

$$\begin{array}{ll} C_a: & q(x), p(5,x) \leftarrow r(x) \\ C_b: & \leftarrow p(x,y), r(y,w), s(y,w) \\ C_c: & p(x,z) \leftarrow p(x,y), r(y,z) \\ C_d: & p(x,w), p(x,2) \leftarrow p(x,y), t(w) \end{array}$$

Each of these clauses satisfies the two conditions above. By side-change transformation of p, these four clauses are transformed into:

$$\begin{array}{ll} C'_a: & q(x) \leftarrow r(x), notp(5,x) \\ C'_b: & notp(x,y) \leftarrow r(y,w), s(y,w) \\ C'_c: & notp(x,y) \leftarrow notp(x,z), r(y,z) \\ C'_d: & notp(x,y) \leftarrow notp(x,w), notp(x,2), t(w) \end{array}$$

We obtain the set $\{C'_b, C'_c, C'_d\}$ of definite clauses for the predicate $notp$, which can be used for unfolding with respect to $notp(5,x)$ in the right-hand side of the first clause C'_a. □

Assume that D_{notp} is the set of all definite clauses for $notp$ in Cs. If there is no $notp$-atom in the right-hand side of any clause in $Cs - D_{notp}$, then D_{notp} can be removed by definite-clause removal transformation.

2.2.2 Type 2

Assume that p and q are distinct predicates. Unfolding with respect to q-atoms can be applied after side-change transformation for p if each clause C in Cs satisfies the following two conditions:

1. There is at most one q-atom in the left-hand side of C.

2. If there is exactly one q-atom in the left-hand side of C, then (i) all other user-defined atoms in the left-hand side of C are p-atoms and (ii) no p-atom appears in the right-hand side of C.

Example 2. Consider the following four clauses:

C_e: $s(x,y) \leftarrow q(x,y)$
C_f: $q(x,y) \leftarrow r(y)$
C_g: $p(x,y), q(x,y) \leftarrow r(x), s(y)$
C_h: $p(x,y), p(x,z), q(x,y) \leftarrow t(y,z)$

Each of them satisfies the two conditions for Type 2. By side-change transformation of p, these clauses are changed into:

C'_e: $s(x,y) \leftarrow q(x,y)$
C'_f: $q(x,y) \leftarrow r(y)$
C'_g: $q(x,y) \leftarrow not\,p(x,y), r(x), s(y)$
C'_h: $q(x,y) \leftarrow not\,p(x,y), not\,p(x,z), t(y,z)$

We obtain the set $\{C'_f, C'_g, C'_h\}$ of definite clauses for the predicate q, which can be used for unfolding with respect to $q(x,y)$ in the right-hand side of C'_e. □

Assume that D_q is the set of all definite clauses for q in Cs. If there is no q-atom in the right-hand side of any clause in $Cs - D_q$, then D_q can be removed by definite-clause removal transformation.

2.3 Reduction of Problem Size

The resolution proof procedure is one of the most famous sound and complete proof procedures (Fitting, 1996). It uses two inference rules, i.e., the resolution inference rule (Robinson, 1965) and the factoring inference rule. Given a set Cs of clauses, the resolution proof procedure starts with Cs and repeatedly make inference by resolution or factoring to construct a sequence $[Cs_0, Cs_1, Cs_2, \ldots]$, where $Cs_0 = Cs$. When it reaches a clause set Cs_n that contains the empty clause ($\leftarrow$), the proof is completed. Each inference step increases the number of clauses by one.

The solution method for MI problems considered in this paper uses unfolding/side-change control, i.e., it attempts to apply transformation in the preference order of (i) definite-clause removal, (ii) unfolding, and (iii) side-change transformation. By definite-clause removal, the number of clauses decreases. Let p be a predicate and $number(p)$ denote the number of p-atoms in the right-hand sides of all clauses under consideration. By repeated application of unfolding with respect to p-atoms, we expect $number(p)$ to become zero and definite-clause removal can then be applied for removing the definition of p. By side-change transformation, we expect to obtain new clauses to which further unfolding is applicable. Appropriate combinations of these three types of transformation are often useful to reduce problem size.

C_1: $FM(x) \leftarrow FP(x)$ $\quad$ C_2: $FP(john) \leftarrow$
C_3: $FP(mary) \leftarrow$ $\quad$ C_4: $teach(john, ai) \leftarrow$
C_5: $St(paul) \leftarrow$ $\quad$ C_6: $AC(ai) \leftarrow$
C_7: $Tp(kr) \leftarrow$ $\quad$ C_8: $Tp(lp) \leftarrow$

C_9: $curr(x,z) \leftarrow exam(x,y), subject(y,z), St(x), Co(y), Tp(z)$
C_{10}: $mdt(x,y) \leftarrow curr(x,z), expert(y,z), St(x), Tp(z), FP(y), AC(w), teach(y,w)$
C_{11}: $mdt(x,y) \leftarrow St(x), NFP(y)$

C_{12}: $exam(paul, ai) \leftarrow$ $\quad$ C_{13}: $subject(ai, kr) \leftarrow$
C_{14}: $subject(ai, lp) \leftarrow$ $\quad$ C_{15}: $expert(john, kr) \leftarrow$
C_{16}: $expert(mary, lp) \leftarrow$

C_{17}: $AC(x) \leftarrow teach(mary, x)$
C_{18}: $\leftarrow AC(x), BC(x)$
C_{19}: $AC(x), BC(x) \leftarrow Co(x)$
C_{20}: $Co(x) \leftarrow AC(x)$
C_{21}: $Co(x) \leftarrow BC(x)$
C_{22}: $FP(x) \leftarrow NFP(x)$
C_{23}: $\leftarrow NFP(x), teach(x,y), Co(y)$
C_{24}: $teach(x,y), NFP(x) \leftarrow FP(x), func(f_0, x, y)$
C_{25}: $Co(y), NFP(x) \leftarrow FP(x), func(f_0, x, y)$

Figure 1: Background knowledge for the *mdt* problem on $\mathrm{ECLS_F}$.

A resolution proof procedure is a refutation method, i.e., it aims to show that a given clause set has no model. QA problems, on the other hand, is basically concerned with finding all models of a given clause set. The unfolding/side-change control can be used not only for proof problems but also for QA problems and MI problems.

3 COMPUTATION CONTROL WITH SIDE-CHANGE TRANSFORMATION

Control of rule application with side-change transformation is demonstrated by using a puzzle. When necessary, the reader is referred to the concept definitions in Section 4.

3.1 Problem Description

Consider the clause set Cs consisting of C_1–C_{25} in Fig. 1, which provides the background knowledge on $\mathrm{ECLS_F}$ for the *mayDoThesis* problem (for short, the *mdt* problem), modified from the original problem given in (Donini et al., 1998). All atoms appearing in Fig. 1 are user-defined atoms. The clauses C_9–C_{11}

C_{26}: $teach(john, ai) \leftarrow$
C_{27}: $AC(ai) \leftarrow$
C_{28}: $AC(x) \leftarrow teach(mary, x)$
C_{29}: $\leftarrow AC(x), BC(x)$
C_{30}: $AC(x), BC(x) \leftarrow Co(x)$
C_{31}: $Co(x) \leftarrow AC(x)$
C_{32}: $Co(x) \leftarrow BC(x)$
C_{33}: $\leftarrow NFP(x), teach(x, y), Co(y)$
C_{34}: $mdt(paul, mary) \leftarrow AC(x), teach(mary, x), Co(ai)$
C_{35}: $mdt(paul, john) \leftarrow AC(x), teach(john, x), Co(ai)$
C_{36}: $mdt(paul, x) \leftarrow NFP(x)$
C_{37}: $teach(mary, x), NFP(mary) \leftarrow func(f_0, mary, x)$
C_{38}: $teach(john, x), NFP(john) \leftarrow func(f_0, john, x)$
C_{39}: $Co(x), NFP(mary) \leftarrow func(f_0, mary, x)$
C_{40}: $Co(x), NFP(john) \leftarrow func(f_0, john, x)$

Figure 2: Clauses obtained by transformation of C_1–C_{25} in Fig. 1.

together provide the conditions for a student to do his/her thesis with a professor, where for any student s and any professor p, $mdt(s, p)$ is intended to mean "s may do his/her thesis with p."

Suppose that we want to find all professors with whom *paul* may do his thesis. This problem is formulated as a MI problem $\langle Cs, \varphi \rangle$, where φ is a mapping used for constructing the output answer from the intersection of all models of *Cs*, defined by

$$\varphi(G) = \{x \mid mdt(paul, x) \in G\}$$

for any set G of ground user-defined atoms. (The formal definition of a MI problem is deferred to Section 4.4.)

3.2 Inapplicability of Unfolding

Consider the clauses C_{26}–C_{40} in Fig. 2, which are obtained by equivalently transforming the clauses C_1–C_{25} in Fig. 1. Unfolding is not applicable to them due to the following reasons:

- Unfolding of the predicate *AC* is blocked since C_{30} contains an *AC*-atom and a *BC*-atom in its left-hand side.
- Unfolding of the predicate *Co* is blocked since C_{39} and C_{40} contain *Co*-atoms and *NFP*-atoms in their left-hand sides.
- Unfolding of the predicate *teach* is blocked since C_{37} and C_{38} contain *teach*-atoms and *NFP*-atoms in their left-hand sides.
- Unfolding of the predicate *mdt* is blocked since the answer to the problem is influenced by *mdt*-atoms.

As will be subsequently illustrated, unfolding with respect to *Co*-atoms and with respect to *teach*-atoms will be made applicable by side-change transformation for the predicate *NFP*.

3.3 Rule Application Control with Side-Change Transformation

With side-change transformation, the clause set *Cs* consisting of C_1–C_{25} in Fig. 1 is transformed as follows, where basic reduction rules given in (Akama and Nantajeewarawat, 2014) (such as removal of valid clauses, removal of subsumed clauses, and removal of duplicate atoms) are also used:

1. By (i) unfolding using the definitions of the predicates *FP*, *Tp*, *curr*, *subject*, *expert*, *St*, and *exam*, (ii) removing these definitions along with the definition of *FM* using definite-clause removal, and (iii) application of basic reduction rules, the clauses C_1–C_{25} are transformed into the clauses C_{26}–C_{40} in Fig. 2.
2. Side-change transformation for *NFP* enables (i) unfolding using the definitions of *Co* and *teach*, (ii) elimination of these definitions using definite-clause removal, and (iii) application of basic reduction rules.
3. Side-change transformation for *BC* enables (i) unfolding using the definition of *AC*, (ii) definite-clause removal, and (iii) application of basic reduction rules.
4. The resolution rule together with basic reduction rules are then applied.
5. Assume that *notNFP* is the predicate determined for side-change transformation for *NFP* at Step 2. By (i) side-change transformation for *notNFP*, (ii) unfolding of *NFP*, (iii) removal of independent *func*-atoms, and (iv) application of basic reduction rules, we obtain the clauses C_{41} and C_{42} given as follows:

   C_{41}: $mdt(paul, mary) \leftarrow$
   C_{42}: $mdt(paul, john) \leftarrow$

As a result, the MI problem $\langle Cs, \varphi \rangle$ in Section 3.1 is transformed equivalently into the MI problem $\langle \{C_{41}, C_{42}\}, \varphi \rangle$. Hence the answer to the original MI problem $\langle Cs, \varphi \rangle$ coincides with the answer to the simpler MI problem $\langle \{C_{41}, C_{42}\}, \varphi \rangle$, which can be readily determined as the set

$$\begin{aligned} &\varphi(\bigcap Models(\{C_{41}, C_{42}\})) \\ &= \varphi(\{mdt(paul, mary), mdt(paul, john)\}) \\ &= \{mary, john\}, \end{aligned}$$

where $Models(\{C_{41}, C_{42}\})$ denotes the set of all models of $\{C_{41}, C_{42}\}$.

4 CORRECTNESS THEOREM FOR SIDE-CHANGE TRANSFORMATION

To provide a theoretical basis for the correctness of side-change transformation, we first review the extended clause space ECLS_F, the semantics of an extended clause set, and the formal definition of a MI problem. We then show that side-change transformation preserves the answers to MI problems.

4.1 User-Defined Atoms, Constraint Atoms, and *func*-Atoms

We consider an extended formula space that contains three kinds of atoms, i.e., user-defined atoms, built-in constraint atoms, and *func*-atoms. A *user-defined atom* takes the form $p(t_1,\ldots,t_n)$, where p is a user-defined predicate and the t_i are usual 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 a predefined constraint predicate and the t_i are usual terms. Let $\mathcal{A}_\text{u}$ denote the set of all user-defined atoms, $\mathcal{G}_\text{u}$ the set of all ground user-defined atoms, $\mathcal{A}_\text{c}$ the set of all constraint atoms, and $\mathcal{G}_\text{c}$ the set of all ground constraint atoms.

A *func-atom* (Akama and Nantajeewarawat, 2011) 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 or an n-ary function variable, and the t_i are usual terms. It is a *ground func*-atom if f is a function constant and the t_i are ground usual terms.

4.2 Extended Clauses and an Extended Clause Space

An *extended clause* C 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 $\{a_1,\ldots,a_m,b_1,\ldots,b_n\} \subseteq \mathcal{A}_\text{u} \cup \mathcal{A}_\text{c}$ 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. The sets $\{a_1,\ldots,a_m\}$ and $\{b_1,\ldots,b_n,\mathbf{f}_1,\ldots,\mathbf{f}_p\}$ are called the *left-hand side* and the *right-hand side*, respectively, of the extended clause C, and are denoted by $\mathit{lhs}(C)$ and $\mathit{rhs}(C)$, respectively. When there is only one user-defined atom in $\mathit{lhs}(C)$, C is called an *extended definite clause*. When no confusion is caused, an extended clause is simply called a clause. The set of all extended clauses is denoted by ECLS_F. The *extended clause space* in this paper is the powerset of ECLS_F.

Let Cs be a set of extended clauses. Implicit existential quantifications of function variables and implicit clause conjunction are assumed in Cs. Function variables in Cs are all existentially quantified and their scope covers all clauses in Cs. With occurrences of function variables, clauses in Cs are connected through shared function variables. After instantiating all function variables in Cs into function constants, clauses in the instantiated set are totally separated.

4.3 Interpretations and Models

An *interpretation* is a subset of $\mathcal{G}_\text{u}$. A ground user-defined atom g is true under an interpretation I iff g belongs to I. Unlike ground user-defined atoms, the truth values of ground constraint atoms are predetermined independently of interpretations. Let TCON denote the set of all true ground constraint atoms, i.e., a ground constraint atom g is true iff $g \in \text{TCON}$. A ground *func*-atom $\mathit{func}(f,t_1,\ldots,t_n,t_{n+1})$ is true iff $f(t_1,\ldots,t_n) = t_{n+1}$.

Let $C = (a_1,\ldots,a_m \leftarrow b_1,\ldots,b_n,\mathbf{f}_1,\ldots,\mathbf{f}_p)$ be a ground clause, where $\{a_1,\ldots,a_m,b_1,\ldots,b_n\} \subseteq \mathcal{G}_\text{u} \cup \mathcal{G}_\text{c}$ and $\mathbf{f}_1,\ldots,\mathbf{f}_p$ are ground *func*-atoms. C is true under an interpretation I (in other words, *I satisfies C*) iff at least one of the following conditions is satisfied:

1. There exists $i \in \{1,\ldots,m\}$ such that $a_i \in I \cup \text{TCON}$.
2. There exists $i \in \{1,\ldots,n\}$ such that $b_i \notin I \cup \text{TCON}$.
3. There exists $i \in \{1,\ldots,p\}$ such that $\mathbf{f}_i$ is false.

An interpretation I is a *model* of a clause set $\mathit{Cs} \subseteq \text{ECLS}_\text{F}$ iff there exists a substitution σ for function variables that satisfies the following conditions:

1. All function variables occurring in Cs are instantiated by σ into function constants.
2. For any clause $C \in \mathit{Cs}$ and any substitution θ for usual variables, if $C\sigma\theta$ is a ground clause, then $C\sigma\theta$ is true under I.

For any $\mathit{Cs} \subseteq \text{ECLS}_\text{F}$, let $\mathit{Models}(\mathit{Cs})$ denote the set of all models of Cs.

4.4 MI Problems on the Extended Clause Space

A *model-intersection problem* (*MI problem*) on ECLS_F is a pair $\langle \mathit{Cs}, \varphi \rangle$, where $\mathit{Cs} \subseteq \text{ECLS}_\text{F}$ and φ is a mapping from $\mathit{pow}(\mathcal{G}_\text{u})$ to some set W. The mapping φ is called an *extraction mapping* (previously called an exit mapping). The answer to this problem, denoted by $\mathit{ans}_\text{MI}(\mathit{Cs},\varphi)$, is defined as $\varphi(\bigcap \mathit{Models}(\mathit{Cs}))$, where $\bigcap \mathit{Models}(\mathit{Cs})$ is the intersection of all models of Cs. Note that when $\mathit{Models}(\mathit{Cs})$ is the empty set, $\bigcap \mathit{Models}(\mathit{Cs}) = \mathcal{G}_\text{u}$.

The notion of an independent extraction mapping is next introduced; it is used for giving a sufficient condition for the correctness of side-change transformation. Given a set P of predicates, let $GAtoms(P)$ denote the set of all ground atoms in $\mathcal{G}_u$ the predicates of which belong to P. An extraction mapping φ is said to be *independent* of a set P of predicates iff for any $G_1, G_2 \subseteq \mathcal{G}_u$, if $(G_1 - G_2) \cup (G_2 - G_1) \subseteq GAtoms(P)$, then $\varphi(G_1) = \varphi(G_2)$.

4.5 Correctness Theorem for Side-Change Transformation

We show below that side-change transformation is answer-preserving.

Theorem 1. *Let $\langle Cs, \varphi \rangle$ be a MI problem on $ECLS_F$. Let p be a predicate occurring in Cs and $notp$ a new predicate determined for side-change transformation for p. If φ is independent of $\{p, notp\}$, then $ans_{MI}(Cs, \varphi) = ans_{MI}(\textsc{SideCh}(Cs, p, notp), \varphi)$.*

Proof: Assume that (i) φ is independent of $\{p, notp\}$, (ii) $Cs' = \textsc{SideCh}(Cs, p, notp)$, and (iii) π_p is a mapping that changes p-atoms into $notp$-atoms. Let m be an arbitrary model of Cs. Let a set $m' \subseteq \mathcal{G}_u$ be constructed from m by

$$m' = (m - GAtoms(\{p\})) \cup \{\pi_p(x) \mid x \in (GAtoms(\{p\}) - m)\}.$$

Obviously, m' is a model of Cs'. This construction determines a mapping ψ from $Models(Cs)$ to $Models(Cs')$. Obviously, ψ^{-1} is a mapping from $Models(Cs')$ to $Models(Cs)$ and $\psi \circ \psi^{-1}$ is an identity mapping. Hence ψ is a bijective mapping from $Models(Cs)$ to $Models(Cs')$. As a result,

- $\bigcap Models(Cs) - \bigcap Models(Cs') \subseteq GAtoms(\{p, notp\})$, and
- $\bigcap Models(Cs') - \bigcap Models(Cs) \subseteq GAtoms(\{p, notp\})$.

Since φ is independent of $\{p, notp\}$, it follows that $\varphi(\bigcap Models(Cs))$ is equal to $\varphi(\bigcap Models(Cs'))$. $\square$

5 EXPERIMENTS WITH SIDE-CHANGE TRANSFORMATION

5.1 The Agatha QA Problem

Consider the "Dreadsbury Mansion Mystery" problem (Pelletier, 1986), which is described as follows: Someone who lives in Dreadsbury Mansion killed Aunt Agatha. Agatha, the butler, and Charles live in Dreadsbury Mansion, and are the only people who live therein. A killer always hates his victim, and is never richer than his victim. Charles hates no one that Aunt Agatha hates. Agatha hates everyone except the butler. The butler hates everyone not richer than Aunt Agatha. The butler hates everyone Agatha hates. No one hates everyone. The problem is to find who is the killer.

F_1: $\exists x : (live(x, D) \wedge kill(x, A))$
F_2: $\forall x : (live(x, D) \leftrightarrow (eq(x, A) \vee eq(x, B) \vee eq(x, C)))$
F_3: $\forall x \forall y : (kill(x, y) \rightarrow hate(x, y))$
F_4: $\forall x \forall y : (kill(x, y) \rightarrow \neg richer(x, y))$
F_5: $\neg(\exists x : (hate(C, x) \wedge hate(A, x) \wedge live(x, D)))$
F_6: $\forall x : ((neq(x, B) \wedge live(x, D)) \rightarrow hate(A, x))$
F_7: $\forall x : ((\neg richer(x, A) \wedge live(x, D)) \rightarrow hate(B, x))$
F_8: $\forall x : ((hate(A, x) \wedge live(x, D)) \rightarrow hate(B, x))$
F_9: $\neg(\exists x : (live(x, D) \wedge \forall y : (live(y, D) \rightarrow hate(x, y))))$
F_{10}: $\forall x : (kill(x, A) \rightarrow killer(x))$

Figure 3: Background knowledge represented by first-order formulas.

5.2 Formalization by First-Order Formulas

Assume that eq and neq are predefined binary constraint predicates and for any ground usual terms t_1 and t_2, (i) $eq(t_1, t_2)$ is true iff $t_1 = t_2$, and (ii) $neq(t_1, t_2)$ is true iff $t_1 \neq t_2$. The background knowledge of this mystery is formalized as the conjunction of the first-order formulas in Fig. 3, where (i) the constants A, B, C, and D denote "Agatha," "the butler," "Charles," and "Dreadsbury Mansion," respectively, and (ii) for any terms t_1 and t_2, $live(t_1, t_2)$, $kill(t_1, t_2)$, $hate(t_1, t_2)$, and $richer(t_1, t_2)$ are intended to mean "t_1 lives in t_2," "t_1 killed t_2," "t_1 hates t_2," "t_1 is richer than t_2," respectively.

5.3 Meaning-Preserving Skolemization

Meaning-preserving Skolemization transforms a first-order formula equivalently into a set of extended clauses (Akama and Nantajeewarawat, 2011). Let MPS(E) denote the set of extended clauses resulting from applying meaning-preserving Skolemization to a given first-order formula E in FOL$_c$. MPS(E) is obtained from E by repeated subformula transformation and conversion into a clausal form.

Consider the formulas F_1–F_{10} in Fig. 3. Referring to the clauses C_1–C_{15} in Fig. 4, MPS$(F_1) = \{C_1, C_2\}$, MPS$(F_2) = \{C_3, C_4, C_5, C_6\}$, MPS$(F_3) =$

C_1: $live(x,D) \leftarrow func(f_0,x)$
C_2: $kill(x,A) \leftarrow func(f_0,x)$
C_3: $\leftarrow live(x,D), neq(x,A), neq(x,B), neq(x,C)$
C_4: $live(A,D) \leftarrow$
C_5: $live(B,D) \leftarrow$
C_6: $live(C,D) \leftarrow$
C_7: $hate(x,y) \leftarrow kill(x,y)$
C_8: $\leftarrow kill(x,y), richer(x,y)$
C_9: $\leftarrow hate(A,x), hate(C,x), live(x,D)$
C_{10}: $hate(A,x) \leftarrow neq(x,B), live(x,D)$
C_{11}: $richer(x,A), hate(B,x) \leftarrow live(x,D)$
C_{12}: $hate(B,x) \leftarrow hate(A,x), live(x,D)$
C_{13}: $\leftarrow hate(x,y), func(f_1,x,y), live(x,D)$
C_{14}: $live(y,D) \leftarrow live(x,D), func(f_1,x,y)$
C_{15}: $killer(x) \leftarrow kill(x,A)$

Figure 4: The initial state with extended clauses.

$\{C_7\}$, MPS$(F_4) = \{C_8\}$, MPS$(F_5) = \{C_9\}$, MPS$(F_6) = \{C_{10}\}$, MPS$(F_7) = \{C_{11}\}$, MPS$(F_8) = \{C_{12}\}$, MPS$(F_9) = \{C_{13}, C_{14}\}$, and MPS$(F_{10}) = \{C_{15}\}$. Let $K = F_1 \wedge F_2 \wedge \cdots \wedge F_9$. Then MPS$(K \wedge F_{10}) = \{C_1, C_2, \ldots, C_{15}\}$.

5.4 Simplification

We simplify the formalization in Fig. 3 for ease of computation and presentation. F_1 is simplified using F_2 into F_1' as follows:

$$\begin{aligned} F_1 &\equiv \exists x : (live(x,D) \wedge kill(x,A)) \\ &\equiv \exists x : ((eq(x,A) \vee eq(x,B) \vee eq(x,C)) \wedge \\ &\qquad kill(x,A)) \\ &\equiv \exists x : ((eq(x,A) \wedge kill(x,A)) \vee \\ &\qquad (eq(x,B) \wedge kill(x,A)) \vee \\ &\qquad (eq(x,C) \wedge kill(x,A))) \\ &\equiv \exists x : ((eq(x,A) \wedge kill(A,A)) \vee \\ &\qquad (eq(x,B) \wedge kill(B,A)) \vee \\ &\qquad (eq(x,C) \wedge kill(C,A))) \\ &\equiv kill(A,A) \wedge kill(B,A) \wedge kill(C,A) \\ &\equiv F_1' \end{aligned}$$

F_9 is simplified using F_2 into F_9' as follows:

$$\begin{aligned} F_9 &\equiv \neg(\exists x : (live(x,D) \wedge \\ &\qquad \forall y : (live(y,D) \rightarrow hate(x,y)))) \\ &\equiv \neg(\exists x : (live(x,D) \wedge \\ &\qquad (hate(x,A) \wedge hate(x,B) \wedge hate(x,C)))) \\ &\equiv F_9' \end{aligned}$$

Assume that

- $K = F_1 \wedge F_2 \wedge F_3 \wedge F_4 \wedge F_6 \wedge F_7 \wedge F_8 \wedge F_9$, and
- $K' = F_1' \wedge F_2 \wedge F_3 \wedge F_4 \wedge F_6 \wedge F_7 \wedge F_8 \wedge F_9'$.

The Agatha QA problem is $\langle K \wedge F_{10}, killer(x)\rangle$, and the simplified Agatha QA problem is $\langle K' \wedge F_{10}, killer(x)\rangle$. The set of clauses obtained from $K' \wedge F_{10}$ is shown in Fig. 5, where MPS$(F_1') = \{C_{16}\}$,

C_{16}: $kill(A,A), kill(B,A), kill(C,A) \leftarrow$
C_{17}: $\leftarrow live(x,D), neq(x,A), neq(x,B), neq(x,C)$
C_{18}: $live(A,D) \leftarrow$
C_{19}: $live(B,D) \leftarrow$
C_{20}: $live(C,D) \leftarrow$
C_{21}: $hate(x,y) \leftarrow kill(x,y)$
C_{22}: $\leftarrow kill(x,y), richer(x,y)$
C_{23}: $\leftarrow hate(A,x), hate(C,x), live(x,D)$
C_{24}: $hate(A,A) \leftarrow$
C_{25}: $hate(A,C) \leftarrow$
C_{26}: $richer(x,A), hate(B,x) \leftarrow live(x,D)$
C_{27}: $hate(B,x) \leftarrow hate(A,x), live(x,D)$
C_{28}: $\leftarrow hate(x,A), hate(x,B), hate(x,C), live(x,D)$
C_{29}: $killer(x) \leftarrow kill(x,A)$

Figure 5: The simplified initial state.

MPS$(F_2) = \{C_{17}, C_{18}, C_{19}, C_{20}\}$, MPS$(F_3) = \{C_{21}\}$, MPS$(F_4) = \{C_{22}\}$, MPS$(F_5) = \{C_{23}\}$, MPS$(F_6) = \{C_{24}, C_{25}\}$, MPS$(F_7) = \{C_{26}\}$, MPS$(F_8) = \{C_{27}\}$, MPS$(F_9) = \{C_{28}\}$, and MPS$(F_{10}) = \{C_{29}\}$.

5.5 Computation Control

We use priority ordering to control selection and application of ET rules. For example, assume that

- (subsumed) is an ET rule for elimination of subsumed clauses,
- (ud) is an ET rule for unfolding by a set of definite clauses or deletion of a set of definite clauses, and
- (rr n), where n is a nonnegative integer, is the resolution rule with a resolvent consisting of less than or equal to n atoms.

To represent a preference order among ET rules, a binary relation $>$ is used. ET rules together with a binary relation $>$ are referred to as prioritized ET rules or a prioritized ET rule set. The prioritized ET rules

$$\text{(subsumed)} > \text{(ud)} > \text{(rr 4)}$$

indicate that (subsumed) has priority over (ud) and (ud) has priority over (rr 4). The relation $>$ specifies computation control for rule selection at each computation step as follows:

- If (subsumed) is applicable, then apply it.
- If (subsumed) is not applicable and (ud) is applicable, then apply (ud).
- If (subsumed) and (ud) are not applicable and (rr 4) is applicable, then apply (rr 4).

For more detailed explanation of computation control and ET rules, the reader is referred to (Akama et al., 2018).

C_{30}: $hate(B,C) \leftarrow$
C_{31}: $hate(B,A) \leftarrow$
C_{32}: $killer(x) \leftarrow kill(x,A)$
C_{33}: $hate(x,y) \leftarrow kill(x,y)$
C_{34}: $hate(A,C) \leftarrow$
C_{35}: $hate(A,A) \leftarrow$
C_{36}: $\leftarrow kill(x,y), richer(x,y)$
C_{37}: $kill(A,A), kill(B,A), kill(C,A) \leftarrow$
C_{38}: $richer(B,A), hate(B,B) \leftarrow$
C_{39}: $\leftarrow kill(C,A)$

Figure 6: The computation state to which unfolding cannot be applied.

5.6 The State to Which Unfolding Is Not Further Applicable

Consider the simplified Agatha QA problem $\langle K' \wedge F_{10}, killer(x)\rangle$. Assume that

- (eq) is an ET rule for evaluating *eq*-atoms,
- (neq) is an ET rule for evaluating ground *neq*-atoms,
- (erase) is an ET rule for removing body atoms that are erasable in a clause, and
- (specAtom) is an ET rule for specializing a clause into a number of clauses.

By using the prioritized ET rules

(eq) > (neq) > (subsumed) > (erase) >
(specAtom) > (ud),

computation reaches the state in Fig. 6, where the clause $(richer(B,A), hate(B,B) \leftarrow)$ prevents unfolding with respect to *richer*-atoms and *hate*-atoms. Notice the two clauses in Fig. 6 that contain *richer*-atoms:

$\leftarrow kill(x,y), richer(x,y)$
$richer(B,A), hate(B,B) \leftarrow$

By side-change transformation with respect to *richer*, we obtain the following two clauses:

$notricher(x,y) \leftarrow kill(x,y)$
$hate(B,B) \leftarrow notricher(B,A)$

They make unfolding with respect to *richer* and/or *hate* applicable. Unfolding with respect to *notricher* and *hate* yields the following clauses:

$\leftarrow kill(B,A)$
$kill(A,A), kill(B,A), kill(C,A) \leftarrow$
$killer(x) \leftarrow kill(x,A)$
$\leftarrow kill(C,A)$

Resolution then resolves this state to enable further application of unfolding. All atoms except *killer*-atoms are removed by unfolding. Finally, we obtain a singleton clause set $\{(killer(A) \leftarrow)\}$ and reach the conclusion that Agatha is the answer.

Consequently, we add the side-change transformation rule (chSide) with the lowest priority, resulting in the prioritized ET rules

(eq) > (neq) > (subsumed) > (erase) >
(specAtom) > (ud) > (chSide),

in order to have a complete fully automatic computation from the initial state to the final result. We have a 62-step solution, in which (ud) is applied 30 times, (chSide) 1 time, (rr 5) 5 times, (subsumed) 16 times, (erase) 9 times, and (neq) 1 time.

C_{41}: $hate(B,C) \leftarrow$
C_{42}: $hate(B,A) \leftarrow$
C_{43}: $kill(B,A), kill(C,A) \leftarrow$
C_{44}: $hate(x,y) \leftarrow kill(x,y)$
C_{45}: $hate(A,C) \leftarrow$
C_{46}: $hate(A,A) \leftarrow$
C_{47}: $\leftarrow kill(x,y), richer(x,y)$
C_{48}: $\leftarrow hate(B,B)$
C_{49}: $richer(B,A), hate(B,B) \leftarrow$
C_{50}: $\leftarrow kill(C,A)$

Figure 7: The First Stop for Proof.

C_{51}: $\leftarrow kill(B,A)$
C_{52}: $\leftarrow kill(C,A)$
C_{53}: $kill(B,A), kill(C,A) \leftarrow$

Figure 8: The Second Stop for Proof.

5.7 Formalization of Proof by First-Order Formulas

Let C_{40} be the clause $(kill(B,A), kill(C,A) \leftarrow)$, and consider a proof problem $\langle K', C_{40}\rangle$, referred to as the simplified Agatha proof problem, which is to prove that $K' \models \{C_{40}\}$, i.e., neither B nor C killed A. Note that the answer to the simplified Agatha QA problem $\langle K', killer(x)\rangle$ being $\{A\}$ does not mean that neither B nor C killed A in any model. B and/or C may be a killer in some model.

5.8 Two Stops by Inapplicability of Unfolding

Consider the simplified Agatha proof problem $\langle K', \{C_{40}\}\rangle$. By application of unfolding, computation reaches the state in Fig. 7, which is very similar to the case of the simplified Agatha QA problem (Section 5.6), i.e., the clause $(richer(B,A), hate(B,B) \leftarrow)$ prevents unfolding with respect to *richer*-atoms and *hate*-atoms. By side-change transformation with respect to *richer*, *notricher*-atoms and *hate*-atoms can be unfolded. After removal of *notricher*-atoms and *hate*-atoms by (ud), we reach the state in Fig. 8. The clause $((kill(B,A), kill(C,A) \leftarrow)$ blocks further unfolding. By side-change transformation, we have:

$notkill(B,A) \leftarrow$
$notkill(C,A) \leftarrow$
$\leftarrow notkill(B,A), notkill(C,A)$

By unfolding with respect to *notkill*-atoms, we have an empty clause, i.e., it is proved that $\neg kill(B,A) \wedge \neg kill(C,A)$.

By experiments, we have found that there is a proof consisting of 53 steps, in which (ud) is applied 25 times, (chSide) 2 times, (subsumed) 14 times, (erase) 11 times, and (neq) 1 time.

5.9 A Solution with Forwarding

Consider the same simplified Agatha proof problem $\langle K', \{C_{40}\}\rangle$. We can also solve this proof problem using forwarding transformation in place of side-change transformation, where forwarding is an ET rule, denoted by (fwd), for removing atoms in the left-hand side of a clause by using a negative clause (Akama et al., 2018). Notice the blocker of unfolding in Fig. 7, i.e., $(richer(B,A), hate(B,B) \leftarrow)$, is changed into $(richer(B,A) \leftarrow)$ by forwarding with $(\leftarrow hate(B,B))$. Similarly, the blocker of unfolding in Fig. 8, i.e., $(kill(B,A), kill(C,A) \leftarrow)$, is changed into $(kill(B,A) \leftarrow)$ by forwarding with $(\leftarrow kill(C,A))$.

By the computation control

(eq) > (neq) > (subsumed) > (erase) >
(specAtom) > (ud) > (fwd),

we have a 55-step solution, in which (ud) is applied 24 times, (subsumed) 18 times, (fwd) 2 times, (erase) 10 times, and (neq) 1 time.

5.10 A Solution with Resolution

Consider again the proof problem $\langle K', \{C_{40}\}\rangle$. To solve this problem, resolution can be used in place of side-change transformation.

In case of Fig. 7, the resolvent of the clauses $(richer(B,A), hate(B,B) \leftarrow)$ and $(\leftarrow hate(B,B))$ is $(richer(B,A) \leftarrow)$. In case of Fig. 8, the resolvent of $(kill(B,A), kill(C,A) \leftarrow)$ and $(\leftarrow kill(C,A))$ is $(kill(B,A) \leftarrow)$.

By the computation control

(eq) > (neq) > (subsumed) > (erase) >
(specAtom) > (ud) > (rr 10),

we have a 57-step solution, in which (ud) is applied 24 times, (subsumed) 20 times, (rr 10) 2 times, (erase) 10 times, and (neq) 1 time.

5.11 A Solution using Resolution Without Unfolding

The proof problem $\langle K', \{C_{40}\}\rangle$ can also be solved without unfolding by mainly applying resolution through the prioritized ET rules

(eq) > (neq) > (subsumed) > (erase) >
(specAtom) > (rr 3) > (rr 8) > (rr 20).

By experiments, we know that there is a proof consisting of 43 steps, in which (rr 8) is applied 6 times, (rr 3) 6 times, (subsumed) 24 times, (erase) 6 times, and (neq) 1 time.

5.12 Applying Resolution to the Agatha QA Problem

Consider again the simplified Agatha QA problem $\langle K' \wedge F_{10}, killer(x)\rangle$. When we apply resolution to this QA problem, we have the final set of clauses in Fig. 9. By the clause $(killer(A) \leftarrow)$, we know that A is a killer. However, we do not know whether no one else is a killer. Compared with the solution by unfolding, resolution is not appropriate to provide a solution for QA problems.

Notice the three clauses $(kill(A,A) \leftarrow)$, $(\leftarrow kill(B,A))$, and $(\leftarrow kill(C,A))$ in Fig. 9, which show that A is a killer, while B and C are not. The clauses in Fig. 9 do not explicitly conclude that A is the only killer, which suggests that resolution alone is helpful to understand the solution but is not enough to show the exact answer to the given QA problem.

6 CONCLUSIONS

MI problems on the extended space $\mathrm{ECLS_F}$ constitute one of the largest classes of logical problems and are of fundamental importance. Since all proof problems on $\mathrm{FOL_c}$ and all QA problems on $\mathrm{FOL_c}$ can be mapped into MI problems on $\mathrm{ECLS_F}$, improvements to a solution method for MI problems on $\mathrm{ECLS_F}$ are essential to efficiently solve logical problems. We introduced side-change transformation on $\mathrm{ECLS_F}$ and proved the correctness of this transformation. We proposed unfolding/side-change computation control, by which we expect a reduction of problem size by decreasing the number of predicates in problem clauses. Such a reduction often contributes to more efficient computation. This computation strategy is in sharp contrast to the resolution proof procedure, which monotonically increases the problem size regardless of whatever control it may take.

C_{54}: $hate(B,x) \leftarrow live(x,D), kill(x,A)$
C_{55}: $hate(B,x) \leftarrow kill(A,x)$
C_{56}: $\leftarrow kill(x,C), kill(x,A), kill(x,B), live(x,D)$
C_{57}: $\leftarrow kill(x,B), kill(x,A), hate(x,C), live(x,D)$
C_{58}: $\leftarrow kill(x,A), hate(x,B), hate(x,C), live(x,D)$
C_{59}: $\leftarrow kill(C,x), kill(A,x)$
C_{60}: $\leftarrow kill(A,x), hate(C,x)$
C_{61}: $killer(A) \leftarrow$
C_{62}: $\leftarrow richer(A,A)$
C_{63}: $\leftarrow kill(B,A)$
C_{64}: $\leftarrow kill(A,B)$
C_{65}: $\leftarrow hate(A,B)$
C_{66}: $\leftarrow kill(B,B)$
C_{67}: $\leftarrow hate(B,B)$
C_{68}: $\leftarrow kill(C,A)$
C_{69}: $\leftarrow kill(C,C)$
C_{70}: $hate(B,C) \leftarrow$
C_{71}: $\leftarrow hate(C,C)$
C_{72}: $hate(B,A) \leftarrow$
C_{73}: $\leftarrow hate(C,A)$
C_{74}: $\leftarrow live(x,D), neq(x,A), neq(x,B), neq(x,C)$
C_{75}: $live(A,D) \leftarrow$
C_{76}: $live(B,D) \leftarrow$
C_{77}: $live(C,D) \leftarrow$
C_{78}: $\leftarrow kill(x,y), richer(x,y)$
C_{79}: $\leftarrow hate(A,x), hate(C,x)$
C_{80}: $\leftarrow hate(x,A), hate(x,B), hate(x,C), live(x,D)$
C_{81}: $hate(A,A) \leftarrow$
C_{82}: $hate(A,C) \leftarrow$
C_{83}: $hate(x,y) \leftarrow kill(x,y)$
C_{84}: $hate(B,x) \leftarrow hate(A,x)$
C_{85}: $richer(x,A), hate(B,x) \leftarrow live(x,D)$
C_{86}: $killer(x) \leftarrow kill(x,A)$
C_{87}: $richer(B,A) \leftarrow$
C_{88}: $kill(A,A) \leftarrow$

Figure 9: A final state of a solution by resolution.

ACKNOWLEDGEMENTS

This research was partially supported by JSPS KAKENHI Grant Numbers 25280078 and 26540110.

REFERENCES

Akama, K. and Nantajeewarawat, E. (2011). Meaning-Preserving Skolemization. In *3rd International Conference on Knowledge Engineering and Ontology Development*, pages 322–327, Paris, France.

Akama, K. and Nantajeewarawat, E. (2014). Equivalent Transformation in an Extended Space for Solving Query-Answering Problems. In *6th Asian Conference on Intelligent Information and Database Systems*, LNAI 8397, pages 232–241, Bangkok, Thailand.

Akama, K. and Nantajeewarawat, E. (2015a). A General Schema for Solving Model-Intersection Problems on a Specialization System by Equivalent Transformation. In *7th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management, Volume 2: KEOD*, pages 38–49, Lisbon, Portugal.

Akama, K. and Nantajeewarawat, E. (2015b). Function-variable Elimination and Its Limitations. In *7th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management, Volume 2: KEOD*, pages 212–222, Lisbon, Portugal.

Akama, K. and Nantajeewarawat, E. (2016a). Model-Intersection Problems with Existentially Quantified Function Variables: Formalization and a Solution Schema. In *8th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management, Volume 2: KEOD*, pages 52–63, Porto, Portugal.

Akama, K. and Nantajeewarawat, E. (2016b). Unfolding Existentially Quantified Sets of Extended Clauses. In *8th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management, Volume 2: KEOD*, pages 96–103, Porto, Portugal.

Akama, K., Nantajeewarawat, E., and Akama, T. (2018). Computation Control by Prioritized ET Rules. In *10th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management, Volume 2: KEOD*, Seville, Spain.

Chang, C.-L. and Lee, R. C.-T. (1973). *Symbolic Logic and Mechanical Theorem Proving*. Academic Press.

Donini, F. M., Lenzerini, M., Nardi, D., and Schaerf, A. (1998). $\mathcal{AL}$-log: Integrating Datalog and Description Logics. *Journal of Intelligent Information Systems*, 16:227–252.

Fitting, M. (1996). *First-Order Logic and Automated Theorem Proving*. Springer-Verlag, second edition.

Pelletier, F. J. (1986). Seventy-Five Problems for Testing Automatic Theorem Provers. *Journal of Automated Reasoning*, 2(2):191–216.

Robinson, J. A. (1965). A Machine-Oriented Logic Based on the Resolution Principle. *Journal of the ACM*, 12:23–41.