Learning Inference by Induction

Chiaki Sakama[1], Tony Ribeiro[2] and Katsumi Inoue[3]

[1] Wakayama University
sakama@sys.wakayama-u.ac.jp
[2] Institut de Recherche en Communications et Cybernetique de Nantes
tony.ribeiro@irccyn.ec-nantes.fr
[3] National Institute of Informatics
inoue@nii.ac.jp

Abstract. This paper studies *learning inference* by induction. We first consider the problem of learning logical inference rules. Given a set S of propositional formulas and their logical consequences T, the goal is to find deductive inference rules that produce T from S. We show that an induction algorithm **LF1T**, which learns logic programs from interpretation transitions, successfully produces deductive inference rules from input transitions. Next we consider the problem of learning non-logical inference rules. We address three case studies for learning abductive inference, frame axioms and conversational implicature by induction. The current study provides a preliminary approach to the problem of learning inference to which little attention has been paid in machine learning and ILP.

1 Introduction

Induction has been used for learning regularities or general rules hidden in data sets. Hypotheses generated by induction are used not only for explaining given evidences but for predicting new phenomena. Induction in these tasks realizes *learning knowledge*—knowing that something is true. Concept learning and data mining are of this type. On the other hand, little attention has been paid for *learning inference*—knowing the ways of thinking by humans. Learning inference is the problem of finding inference rules or thinking patterns by humans. We make a variety of inferences in our daily life. Those inferences are classified into two categories: logical inference and non-logical inference. Logical inference is used in classical or non-classical logics. Non-logical inference includes empirical laws or pragmatic inference. The primary interest of this paper is to argue the possibility of developing machine learning algorithms that can automatically acquire rules of inferences.

To learn logical inference rules, Sakama and Inoue [16] introduce a conceptual framework of *learning logics*. In this framework, they introduce a method of learning propositional logic inference rules inductively from input data that consist of formulas and their logical consequences. In the current paper, we implement learning logical inference rules using the **LF1T** induction algorithm [5, 14] and examine whether it successfully produces inference rules of propositional natural deduction. Next we apply the method to learning non-logical inference rules. We address three case studies, learning rules for abduction, frame axioms, and conversational implicature. We discuss related

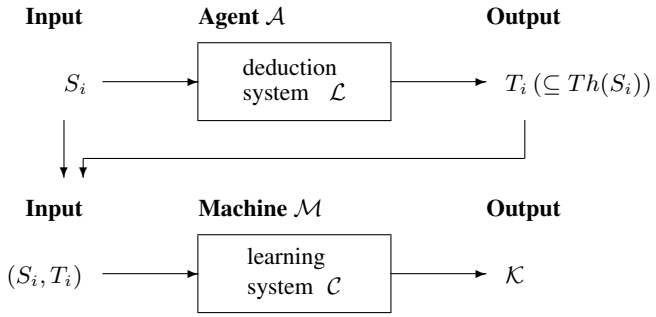

Fig. 1. Learning Logics [16]

issues and finally remark open questions. The rest of this paper is organized as follows. Section 2 provides a framework for learning logics and its implementation by **LF1T**. Section 3 addresses three cases of learning non-logical inference rules. Section 4 discusses related issues and Section 5 addresses final remarks.

2 Learning Logical Inference

2.1 Learning Logics

We review a conceptual framework for learning logics that is introduced in [16]. There are an agent $\mathcal{A}$ and a machine $\mathcal{M}$. The agent $\mathcal{A}$, which could be a human or a computer, is capable of deductive reasoning: it has a set $\mathcal{L}$ of axioms and inference rules in classical logic. Given a (finite) set S of formulas as an input, the agent $\mathcal{A}$ produces a (finite) set of formulas T such that $T \subseteq Th(S)$ where $Th(S)$ is the set of logical consequences of S. On the other hand, the machine $\mathcal{M}$ has no axiomatic system for deduction, while it is equipped with a machine learning algorithm $\mathcal{C}$. Given input-output pairs $(S_1, T_1), \ldots, (S_i, T_i), \ldots$ (where $T_i \subseteq Th(S_i)$) of $\mathcal{A}$ as an input to $\mathcal{M}$, the problem is whether one can develop an algorithm $\mathcal{C}$ which successfully produces an axiomatic system $\mathcal{K}$ for deduction. An algorithm $\mathcal{C}$ is *sound* wrt $\mathcal{L}$ if it produces an axiomatic system $\mathcal{K}$ such that $\mathcal{L} \vdash \mathcal{K}$. An algorithm $\mathcal{C}$ is *complete* wrt $\mathcal{L}$ if it produces an axiomatic system $\mathcal{K}$ such that $\mathcal{K} \vdash \mathcal{L}$ where $\vdash$ means the logical consequence under $\mathcal{L}$. Designing a sound and complete algorithm $\mathcal{C}$ is called a problem of *learning logics* (Figure 1). In this framework, an agent $\mathcal{A}$ plays the role of a teacher who provides training examples representing premises along with entailed consequences. The output $\mathcal{K}$ is refined by incrementally providing examples. Figure 1 describes a deduction system $\mathcal{L}$ while it could be a system of arbitrary logic, e.g. nonmonotonic logic, modal logic, fuzzy logic, as far as it has a formal system of inference.

Sakama and Inoue [16] provide a simple case study for learning deduction rules of propositional logic. They represent a formal system $\mathcal{L}$ of propositional logic using *metalogic programming* and show that deductive inference rules can be reproduced as meta-rules of logic programs. In this paper, we implement learning deductive inference

rules using the **LF1T** algorithm [5, 14] and examine whether it successfully produces inference rules of propositional natural deduction.

2.2 Learning from 1-step Transitions

Learning from 1-step transitions (**LF1T**) [5] is a framework for learning normal logic programs from transitions of interpretations. Here we apply **LF1T** for learning *definite logic programs*, a subclass of normal logic programs that do not contain negation as failure. Let $\mathcal{B}$ be the set of all ground atoms (Herbrand base) and P a (ground) definite logic program (or simply, a *program*) that consists of *rules* of the form:

$$a \leftarrow b_1, \ldots, b_n \qquad (n \geq 0) \tag{1}$$

where $a, b_1, \ldots, b_n$ are ground atoms from $\mathcal{B}$. For each rule R of the form (1), the atom a is the *head* (written $h(R)$) and the conjunction $b_1, \ldots, b_n$ is the *body* of the rule (written $b(R)$). The body is identified with the set of atoms $\{b_1, \ldots, b_n\}$. A rule with the empty body is a *fact*. A rule with variables represents the set of ground instances. A rule R_1 *subsumes* another rule R_2 (written $R_1 \leq R_2$) if $h(R_1) = h(R_2)$ and $b(R_1) \subseteq b(R_2)$. For two programs P_i and P_j, define $P_i \sqsubseteq P_j$ iff for any $R \in P_i$ there is $R' \in P_j$ s.t. $R \leq R'$.

LF1T produces a program from a pair of interpretations as follows.

Input: $E \subseteq 2^{\mathcal{B}} \times 2^{\mathcal{B}}$
Output: a program P such that $J = T_P(I)$ holds for any $(I, J) \in E$

where $T_P(I) = \{\, h(R) \mid R \in P \text{ and } b(R) \subseteq I \,\}$ [17]. A rule R is *consistent* with (I, J) if $b(R) \subseteq I$ implies $h(R) \in J$, otherwise, R is *inconsistent* with (I, J). A rule R is called an *anti-rule* with respect to a transition (I, J) if R is inconsistent with (I, J). A program P is *consistent* with (I, J) if every rule in P is consistent with (I, J). In **LF1T**, a positive example is input as a one-step state transition from I to J, which is given as a pair of Herbrand interpretations. **LF1T** outputs a single program that is consistent with all state transitions given in the input.

In this subsection, we use a *top-down* version of **LF1T** [14], which generates hypotheses by specialization from the most general rules until a program is consistent with all input state transitions. More precisely, **LF1T** starts with the initial program $P = \{\, a \leftarrow \mid a \in \mathcal{B}\}$. Then **LF1T** iteratively analyzes each transition (I, J). For each atom a that does *not* appear in J, **LF1T** produces an anti-rule:

$$a \leftarrow \bigwedge_{b_i \in I} b_i \,. \tag{2}$$

The rule (2) is an anti-rule wrt (I, J) because $\bigwedge_{b_i \in I} b_i \subseteq I$ but $a \notin J$. Any rule of P that subsumes such an anti-rule is inconsistent with the transition and must be revised. To this end, every rule R of P that subsumes (2) is *minimally specialized* by replacing R with the rules in $\{\, h(R) \leftarrow b(R) \wedge c_j \mid c_j \in \mathcal{B} \setminus I\}$ to make P consistent with the new transition by avoiding the subsumption of all anti-rules produced by (I, J). After such minimal specialization, P becomes consistent with the new transition while

Input : a set $\mathcal{B}$ of atoms and $E \subseteq 2^{\mathcal{B}} \times 2^{\mathcal{B}}$
Output : A definite logic program P such that $J = T_P(I)$ for any $(I, J) \in E$.

put $P := \{\, a \leftarrow \mid a \in \mathcal{B}\}$;
while $E \neq \emptyset$ do
  pick $(I, J) \in E$; $E := E \setminus \{(I, J)\}$;
  for each $a \in \mathcal{B} \setminus J$, put
    $R_a^I := a \leftarrow \bigwedge_{b_i \in I} b_i$;
    for any rule $R \in P$ that subsumes R_a^I, put
      $\mathcal{R} := \{\, h(R) \leftarrow b(R) \wedge c_j \mid c_j \in \mathcal{B} \setminus I \,\}$;
    Replace R with the rules of $\mathcal{R}$ and remove any rule $R' \in P$ that is subsumed by some rule in $\mathcal{R}$
return P

Fig. 2. top-down **LF1T**[14]

remaining consistent with all previously analyzed transitions. The algorithm is sketched in Figure 2. It is shown that **LF1T** produces a program P such that $J = T_P(I)$ for any $(I, J) \in E$ and minimal wrt the ordering $\sqsubseteq$ [14].[4]

In the next section, we use the **LF1T** induction algorithm for learning deduction rules and show experimental results.

2.3 Learning Deduction Rules by LF1T

In this section, we use **LF1T** as a learning system $\mathcal{C}$ in Figure 1. We assume a (propositional) *natural deduction system* $\mathcal{L}$ [13] represented by a *metalogic program* P. In P every propositional formula F is represented by a fact using a meta-predicate $hold$ as

$$hold(F). \tag{3}$$

Using the expression, *Modus Ponens* is represented as a *meta-rule* in P as follows:

$$hold(G) \leftarrow hold(F \rightarrow G), hold(F) \tag{4}$$

where F and G are variables representing any propositional formula.

Suppose that P contains the single rule of (4). Let p and q be propositional variables. Given the set $I = \{\, hold(p),\ hold(p \rightarrow q)\}$, it becomes $T_P(I) = \{\, hold(q)\,\}$. $T_P(I)$ represents a set of formulas that are deduced from I using Modus Ponens. In this way, a program P provides transitions (I, J) such that $J \subseteq T_P(I)$. Then, given (I, J) as input, our goal is to examine whether **LF1T** can reproduce correct inference rules of deduction represented by meta-rules in P. We use formulas represented by propositional variables (e.g. $p \rightarrow q$), rather than formulas represented by particular propositional constants (e.g. $human \rightarrow animal$). With this setting, produced rules represent relations among

[4] The result is shown for normal logic programs and is applied to their subclass of programs.

formulas written by propositional variables, which are instantiated by any propositional constants. To learn inference rules, an Herbrand base $\mathcal{B}$ is firstly fixed as a set of facts of the form (3). Inputs are then constructed as pairs $(I, J) \in E$ where $E \subseteq 2^{\mathcal{B}} \times 2^{\mathcal{B}}$. In what follows, we provide some results of experiments.

Let $\mathcal{B} = \{\, hold(p),\ hold(q),\ hold(r),\ hold(p \to r)\,\}$. We address the process of constructing a rule with the atom $hold(r)$ in the head.

Step 0: **LF1T** starts with the most general rule:

$$hold(r) \leftarrow . \tag{5}$$

Step 1: Suppose that the transition $(\emptyset, \emptyset)$ is given. The rule (5) is inconsistent with this transition (because $b((5)) = \emptyset \subseteq \emptyset$ but $h((5)) = hold(r) \notin \emptyset$), so that (5) is an anti-rule wrt $(\emptyset, \emptyset)$ and is (minimally) specialized into four rules by introducing an atom from $\mathcal{B}$:

$$hold(r) \leftarrow hold(p) \tag{6}$$
$$hold(r) \leftarrow hold(q) \tag{7}$$
$$hold(r) \leftarrow hold(r) \tag{8}$$
$$hold(r) \leftarrow hold(p \to r). \tag{9}$$

Those rules are consistent with the transition $(\emptyset, \emptyset)$.

Step 2: Suppose that the transition $(\{hold(p)\}, \{hold(p)\})$ is given. The rule (6) is inconsistent with this transition, so that (6) is specialized into three rules:

$$hold(r) \leftarrow hold(p), hold(q)$$
$$hold(r) \leftarrow hold(p), hold(r)$$
$$hold(r) \leftarrow hold(p), hold(p \to r).$$

These three rules are respectively subsumed by the rules (7), (8) and (9) in Step 1, hence removed. As a result, the rules (7), (8) and (9) remain.

Step 3: Suppose that the transition $(\{hold(q)\}, \{hold(q)\})$ is given. The rule (7) is inconsistent with this transition, and is removed after specialization. As a result of subsumption, three rules (8), (9) and the newly constructed rule (10) remain.

$$hold(r) \leftarrow hold(p), hold(q). \tag{10}$$

Step 4: Suppose that the transition $(\{hold(p \to r)\}, \{hold(p \to r)\})$ is given. The rule (9) is inconsistent with this transition, and is removed after specialization. As a result of subsumption, two rules are newly constructed:

$$hold(r) \leftarrow hold(p \to r), hold(p) \tag{11}$$
$$hold(r) \leftarrow hold(p \to r), hold(q). \tag{12}$$

Now four rules (8), (10), (11) and (12) remain.

Step 5: Suppose that the transition $(\{hold(p), hold(q)\}, \{hold(p), hold(q)\})$ is given. The rule (10) is inconsistent with this transition, and is removed after specialization. As a result of subsumption, three rules (8), (11) and (12) remain.

Step 6: Suppose that the transition $(\{hold(p \rightarrow r), hold(q)\}, \{hold(p \rightarrow r), hold(q)\})$ is given. The rule (12) is inconsistent with this transition, and is removed after specialization. As a result of subsumption, two rules (8) and (11) remain.

Step 7: Suppose that the transition $(\{hold(p \rightarrow r), hold(p)\}, \{hold(p \rightarrow r), hold(p), hold(r)\})$ is given. Two rules (8) and (11) are consistent with this transition and remain as they are.

The remaining two rules (8) and (11) are consistent with any other transitions (I, J) such that J represents logical consequences of I under a metalogic program P. Then **LF1T** produces those rules as output. The rule (8) represents **Repetition (Rep)** and (11) represents **Modus Ponens (MP)**. The input-output of **LF1T** is summarized in Table 1.[5]

Table 1. LF1T Input-Output

input	output
$(\emptyset, \emptyset)$	$hold(r) \leftarrow hold(p).$ $hold(r) \leftarrow hold(q).$ $hold(r) \leftarrow hold(r).$ $hold(r) \leftarrow hold(p \rightarrow r).$
$(\{hold(p)\}, \{hold(p)\})$	~~$hold(r) \leftarrow hold(p).$~~ $hold(r) \leftarrow hold(q).$ $hold(r) \leftarrow hold(r).$ $hold(r) \leftarrow hold(p \rightarrow r).$
$(\{hold(q)\}, \{hold(q)\})$	~~$hold(r) \leftarrow hold(q).$~~ $hold(r) \leftarrow hold(r).$ $hold(r) \leftarrow hold(p \rightarrow r).$ $hold(r) \leftarrow hold(p), hold(q).$
$(\{hold(p \rightarrow r)\}, \{hold(p \rightarrow r)\})$	$hold(r) \leftarrow hold(r).$ ~~$hold(r) \leftarrow hold(p \rightarrow r).$~~ $hold(r) \leftarrow hold(p), hold(q).$ $hold(r) \leftarrow hold(p \rightarrow r), hold(p).$ $hold(r) \leftarrow hold(p \rightarrow r), hold(q).$
$(\{hold(p), hold(q)\}, \{hold(p), hold(q)\})$	$hold(r) \leftarrow hold(r).$ ~~$hold(r) \leftarrow hold(p), hold(q).$~~ $hold(r) \leftarrow hold(p \rightarrow r), hold(p).$ $hold(r) \leftarrow hold(p \rightarrow r), hold(q).$
$(\{hold(p \rightarrow r), hold(q)\}, \{hold(p \rightarrow r), hold(q)\})$	$hold(r) \leftarrow hold(r).$ $hold(r) \leftarrow hold(p \rightarrow r), hold(p).$ ~~$hold(r) \leftarrow hold(p \rightarrow r), hold(q).$~~
$(\{hold(p \rightarrow r), hold(p)\}, \{hold(p \rightarrow r), hold(p), hold(r)\})$	$hold(r) \leftarrow hold(r).$ $hold(r) \leftarrow hold(p \rightarrow r), hold(p).$

[5] The experimental archive is found at http://www.wakayama-u.ac.jp/~sakama/ILP2015-short/

In the table, those inputs except the last one are of the form (I, I) that *does not* represent any change. Those examples are used for excluding rules describing incorrect transitions. By contrast, the last input is used for verifying whether the remaining rule represents correct transitions. Other results of experiments are addressed below.

- Given $\mathcal{B}=\{\, hold(p),\ hold(\neg p),\ hold(q),\ hold(\neg q),\ hold(p \to q),\ hold(q \to r),\ hold(p \to r)\,\}$, **LF1T** produces

$$hold(\neg p) \leftarrow hold(p \to q), hold(\neg q) \quad (\textbf{Modus Tollens (MT)})$$
$$hold(p \to r) \leftarrow hold(p \to q), hold(q \to r) \quad (\textbf{Hypothetical Syllogism (HS)})$$

- Given $\mathcal{B} = \{\, hold(p),\ hold(\neg p),\ hold(q),\ hold(\neg q),\ hold(p \vee q),\ hold(\neg p \vee \neg q),\ hold(r \vee s),\ hold(\neg r \vee \neg s),\ hold(p \to r), hold(q \to s)\,\}$, **LF1T** produces

$$hold(p) \leftarrow hold(p \vee q), hold(\neg q) \quad (\textbf{Disjunctive Syllogism (DS)})$$
$$hold(r \vee s) \leftarrow hold(p \vee q), hold(p \to r), hold(q \to s) \quad (\textbf{Constructive Dilemma (CD)})$$
$$hold(\neg p \vee \neg q) \leftarrow hold(\neg r \vee \neg s), hold(p \to r), hold(q \to s) \quad (\textbf{Destructive Dilemma (DD)})$$

- Given $\mathcal{B} = \{\, hold(p),\ hold(q),\ hold(r),\ hold(p \wedge q),\ hold(q \wedge r),\ hold(p \vee q),\ hold(q \vee r)\,\}$, **LF1T** produces

$$hold(p) \leftarrow hold(p \wedge q) \quad (\textbf{Conjunction Elimination (CE)})$$
$$hold(p \wedge q) \leftarrow hold(p), hold(q) \quad (\textbf{Conjunction Introduction (CI)})$$
$$hold(p \vee q) \leftarrow hold(p) \quad (\textbf{Disjunction Introduction (DI)})$$

The number of examples used for learning above rules depends on $\mathcal{B}$. To obtain correct inference rules, a sufficient number of input examples are needed. To see whether a produced rule R is a correct inference rule, input test data I to R and check whether R produces a correct output J satisfying $T_P(I) = J$ for a metalogic program P.

In this way, **LF1T** successfully produces inference rules of natural deduction. Note that produced rules are applied to any propositions. For instance, in Modus Ponens

$$hold(q) \leftarrow hold(p \to q), hold(p) \tag{13}$$

p and q are propositional variables that can be instantiated by any propositional constants. Moreover, since any propositional formula is named by a new propositional variable, we can replace propositional variables p, q, r, ... with new propositional variables F, G, H, ... representing arbitrary propositional formulas. The rule (13) is then interpreted as a rule between formulas:

$$hold(G) \leftarrow hold(F \to G), hold(F).$$

As such, **LF1T** produces *inference schemata* of natural deduction. Let $\mathcal{L}$ be a natural deduction system that has ten inference rules—**Rep**, **MP**, **MT**, **HS**, **DS**, **CD**, **DD**, **CE**, **CI**, and **DI**. Then we have the next result.

Proposition 2.1 *Let P be a metalogic program representing a propositional natural deduction system $\mathcal{L}$. Then there is a finite number of inputs (I, J) satisfying $T_P(I) = J$ such that* **LF1T** *can reproduce the set of inference rules of $\mathcal{L}$ from them.*

Proof. As we have seen in this section, the ten rules of $\mathcal{L}$ can be obtained by **LF1T**. The natural deduction system $\mathcal{L}$ consists of inference rules represented by a finite number $k\,(\geq 4)$ of propositional symbols. Since the number of pairs (I, J) of sets of formulas constructed by those symbols are finite, providing every pair (I, J) satisfying $T_P(I) = J$ will produce those inference rules written as meta-rules. □

There are natural deduction systems that have more inference rules, while they are realized using those ten rules presented above. For instance, inference rules of contradiction: $p \wedge \neg p \vdash \bot$ is realized by MP by identifying $\neg p \equiv (p \rightarrow \bot)$; and $\bot \vdash q$ is realized by MP as $\bot \wedge (\bot \rightarrow q) \vdash q$. Thus, once learning the MP rule (13) by **LF1T**, we can get $\bot \vdash q$ or $p \wedge \neg p \vdash \bot$ by instantiating a propositional variable p or q with $\bot$.

Replacement rules for logically equivalent formulas, such as *De Morgan's law*, *communication*, *association*, *distribution*, *transposition* and *double negation*, are represented as $hold(F) \leftarrow hold(G)$ and $hold(G) \leftarrow hold(F)$ for $F \equiv G$ in background knowledge. Background knowledge is combined with inference rules induced by **LF1T** to produce new rules of inference. For instance, the replacement rule $p \rightarrow q \equiv \neg p \vee q$ is represented as $hold(p \rightarrow q) \leftarrow hold(\neg p \vee q)$ and $hold(\neg p \vee q) \leftarrow hold(p \rightarrow q)$ in background knowledge. Then together with the MP rule (13), we can get the new rule $hold(q) \leftarrow hold(\neg p \vee q), hold(p)$ (which is also an instance of **DS**).

3 Learning Non-logical Inference Rules

In Section 2 we consider a logical system $\mathcal{L}$ that has axiomatic systems for inference. $\mathcal{L}$ provides correct training data (S, T) to a machine learning algorithm $\mathcal{C}$ in Figure 1, and **LF1T** successfully reproduces inference rules of $\mathcal{L}$ as a natural deduction system. On the other hand, axiomatic systems for inference do not always exist, especially for *non-logical* inference. In this case, a set of input-output pairs (or premise-consequence pairs) are not given from a teacher agent $\mathcal{A}$ in general, but can be implicitly hidden in log files of dynamic systems or in dialogues with unknown agents. A machine learning system $\mathcal{C}$ identifies those input-output relations automatically to produce a set of meta-theoretical inference rules for the domain or inference patterns of those agents. This section addresses three case studies for learning non-logical inference rules.

3.1 Abduction

A fallacy is an incorrect inference while often used in our daily life. For instance, the followings are well-known *logical fallacies* [19]:

Affirming a Disjunct: From $p \vee q$ and p, infer $\neg q$.
Affirming the Consequent: From $p \rightarrow q$ and q, infer p.
Denying the Antecedent: From $p \rightarrow q$ and $\neg p$, infer $\neg q$.

Affirming the consequent is also used for *abductive inference* [10]. In what follows, we address a case of learning inference rules for abduction.

Given background knowledge K and an observation $hold(G)$, abduction computes an explanation $hold(F)$ such that $hold(F \rightarrow G) \in K$. Unlike the case of learning deduction rules in Section 2.3, K, G and F are represented by propositional constants rather than propositional variables. This is because we assume no axiomatic system $\mathcal{L}$ in learning abduction rules. Training data are then provided as a particular background knowledge, an observation and its candidate explanations. The goal is to construct a general inference scheme of abduction from those data.

For example, let K be the background knowledge:

$$\begin{aligned} \textit{wet-grass} &\rightarrow \textit{wet-shoes} \\ \textit{rained} &\rightarrow \textit{wet-grass} \\ \textit{sprinkler-on} &\rightarrow \textit{wet-grass}. \end{aligned}$$

Using metalogic expression, the above rules are expressed as

$$K = \{\ hold(wg \rightarrow ws),\ \ hold(r \rightarrow wg),\ \ hold(s \rightarrow wg)\ \}$$

where ws, wg, r and s abbreviate $\textit{wet-shoes}$, $\textit{wet-grass}$, $\textit{rained}$ and $\textit{sprinkler-on}$, respectively.

In computing rules for abduction, a state transition is given as a pair (I, J) such that I contains an observation O and rules R from background knowledge K, and J is an explanation such that $R \cup J \vdash O$. For instance, $(I, J) = (\{hold(wg \rightarrow ws), hold(ws)\}, \{hold(wg)\})$ means that the observation $hold(ws)$ is explained by $hold(wg)$ using $hold(wg \rightarrow ws)$ in K. To reduce the hypotheses space, we set *abducibles* as $\mathcal{H} = \{hold(wg), hold(r), hold(s)\}$ and assume any explanation as an element of $\mathcal{H}$. Let $\mathcal{B} = K \cup \{hold(wg), hold(ws), hold(r), hold(s)\}$. We apply the top-down **LF1T** for atoms in $\mathcal{H}$. Given pairs of transitions, **LF1T** finally produces the following rules:

$$hold(wg) \leftarrow hold(wg) \tag{14}$$
$$hold(wg) \leftarrow hold(wg \rightarrow ws), hold(ws) \tag{15}$$
$$hold(r) \leftarrow hold(r) \tag{16}$$
$$hold(r) \leftarrow hold(r \rightarrow wg), hold(wg) \tag{17}$$
$$hold(r) \leftarrow hold(r \rightarrow wg), hold(wg \rightarrow ws), hold(ws) \tag{18}$$
$$hold(s) \leftarrow hold(s) \tag{19}$$
$$hold(s) \leftarrow hold(s \rightarrow wg), hold(wg) \tag{20}$$
$$hold(s) \leftarrow hold(s \rightarrow wg), hold(wg \rightarrow ws), hold(ws) \tag{21}$$

Explanations are represented by atoms in the heads. The rule (14) means that the explanation wg self-explains the observation wg. The rule (15) means that the explanation wg is produced from the background knowledge $wg \rightarrow ws$ and the observation ws. Each rule represents abduction for different explanations. To obtain general inference rules for abduction, we use *least generalization* of Horn clauses [9, 12]. In applying

least generalization, we first classify rules into equivalent classes based on the number of atoms in the body of a rule. Given a rule R, $|\, b(R)\,|$ represents the number of atoms in the body of R. Then define the equivalence class as

$$C_i = \{\, R : |\, b(R)\,| = i \,\}.$$

The rules (14)– (21) are classified into $C_1 = \{(14), (16), (19)\}$, $C_2 = \{(15), (17), (20)\}$, $C_3 = \{(18), (21)\}$. The rules in C_1 represent explanations as observations. The rules in C_2 represent explanations produced by one step of Affirming the Consequent. The rules in C_3 represent explanations produced by two steps of Affirming the Consequent. Abduction computes different explanations by $C_1 - C_3$, so state transitions (I, J) that are consistent with C_i are not always consistent with C_j $(j \neq i)$. For instance, the transition $(I, J) = (\{hold(r \rightarrow wg),\ hold(wg \rightarrow ws),\ hold(ws)\}, \{hold(r)\})$ is consistent with the rule (18), but is inconsistent with (15). So we compute rules $C_1 - C_3$ separately. The least generalization of C_2 is computed as follows. First, implication $p \rightarrow q$ is represented as a term $imp(p, q)$. Then the least generalization of (17) and (20) becomes

$$hold(F) \leftarrow hold(imp(F, wg)), hold(wg), hold(G)$$

where F and G are variables. After eliminating redundant atoms, it becomes[6]

$$hold(F) \leftarrow hold(imp(F, wg)), hold(wg) \tag{22}$$

which represents that any abducible F that implies wg is an explanation of the observation wg. Computing the least generalization of (22) and (15) and removing redundant atoms, we get

$$hold(F) \leftarrow hold(imp(F, G)), hold(G)$$

which represents that any abducible F that implies G is an explanation of the observation G. As such, computing least generalization of each set and removing redundant atoms, we obtain

$$hold(F) \leftarrow hold(F) \tag{23}$$
$$hold(F) \leftarrow hold(F \rightarrow G), hold(G) \tag{24}$$
$$hold(F) \leftarrow hold(F \rightarrow G), hold(G \rightarrow H), hold(H). \tag{25}$$

Each rule represents an inference rule of abduction. Among them, the rule (25) is obtained by a repeated application of (24), and the rule (23) is obtained by putting $F \equiv G$ in (24). As a result, we can pick the rule (24) as the inference rule of abduction.

The above example provides a simple case of learning abduction rules. On the other hand, there would be a case such that abduction is taken place together with deduction. For instance, suppose that a duplex system fails (sf) when two computers (c_1 and c_2) are down simultaneously. It is known that one of the computers does not work ($\neg c_1$) due to some trouble (tr). The situation is represented in the background knowledge

[6] An atom A occurring in $b(R)$ is *redundant* if $b(R) \setminus \{A\} \equiv_\theta b(R)$ where $\equiv_\theta$ is equivalence under θ-subsumption $\leq_\theta$, i.e., $R_1 \leq_\theta R_2$ iff $h(R_1\theta) = h(R_2)$ and $b(R_1\theta) \subseteq b(R_2)$ for some substitution θ.

as $K = \{\, tr,\ tr \rightarrow \neg c_1,\ \neg c_1 \wedge \neg c_2 \rightarrow sf \,\}$. Given the observation sf, a candidate explanation is $\neg c_2$, that is, c_2 does not work too. In this case, the next rule is produced

$$hold(\neg c_2) \leftarrow hold(tr), hold(tr \rightarrow \neg c_1), hold(\neg c_1 \wedge \neg c_2 \rightarrow sf), hold(sf)$$

representing that $\neg c_2$ is an explanation of the observation sf. By constructing a similar rule for $\neg c_1$, the next rule is produced by least generalization

$$hold(\neg F) \leftarrow hold(G), hold(G \rightarrow \neg H), hold(\neg F \wedge \neg H \rightarrow sf), hold(sf)$$

which means that if one of the two computers does not work by some reason, the system's failure is explained by the problem of another computer. The rule would be further generalized to

$$hold(F) \leftarrow hold(G), hold(G \rightarrow H), hold(F \wedge H \rightarrow K), hold(K) \tag{26}$$

The rule (26) represents an inference where abduction is taken place with deduction. To verify the correctness of produced rules, they are applied to other test cases and check whether those rules provide appropriate explanations and accurate predictions.

3.2 Frame Axiom

Applying **LF1T** to state transitions describing the world change could enable us to learning *frame axioms* [7]. Let us consider a block world such that there are three blocks a, b and c where a is on b, b and c are on a table t. The state is represented as

$$S = \{hold(on(a,b)),\ hold(on(b,t)),\ hold(on(c,t)),\ hold(clear(a)),\ hold(clear(c))\}$$

where $clear(x)$ means that there is nothing on x. After moving a on top of c the state changes into

$$T = \{hold(on(a,c)),\ hold(on(b,t)),\ hold(on(c,t)),\ hold(clear(a)),\ hold(clear(b))\}.$$

Now we have only one state transition (S, T) then a state change from S to T is deterministically described using a *bottom-up* version of **LF1T** [5] that produces a transition rule from S to T. A bottom-up **LF1T** for definite logic programs is sketched below.

Bottom-up **LF1T**(E: pairs of Herbrand interpretations, P: a definite logic program)

1. If $E = \emptyset$ then output P and stop;
2. Pick $(I, J) \in E$, and put $E := E \setminus \{(I, J)\}$;
3. For each $a \in J$, let
$$R_a^I \ := \ a \leftarrow \bigwedge_{b_i \in I} b_i;$$
4. If R_a^I is not subsumed by any rule in P, then $P := P \cup \{R_a^I\}$ and simplify P by removing all rules subsumed by R_a^I;
5. Return to 1.

Given the state transition $(I, J) = (S \cup \{move(a, c)\}, T)$, the bottom-up **LF1T** produces

$$hold(on(a, c)) \leftarrow conj(S), move(a, c) \tag{27}$$
$$hold(on(b, t)) \leftarrow conj(S), move(a, c) \tag{28}$$
$$hold(on(c, t)) \leftarrow conj(S), move(a, c) \tag{29}$$
$$hold(clear(a)) \leftarrow conj(S), move(a, c) \tag{30}$$
$$hold(clear(b)) \leftarrow conj(S), move(a, c) \tag{31}$$

where $conj(S)$ is the conjunction of atoms in S. Among them, rules (28), (29), and (30) describe transitions that do not change by the action of $move(a, c)$, since $conj(S)$ contains the same atoms appearing in the head of each rule.

Suppose moving c on top of a at the state S. Using the state transition $(I, J) = (S \cup \{move(c, a)\}, T)$, the bottom-up **LF1T** produces

$$hold(on(c, a)) \leftarrow conj(S), move(c, a) \tag{32}$$
$$hold(on(a, b)) \leftarrow conj(S), move(c, a) \tag{33}$$
$$hold(on(b, t)) \leftarrow conj(S), move(c, a) \tag{34}$$
$$hold(clear(c)) \leftarrow conj(S), move(c, a) \tag{35}$$

Among them, rules (33), (34), and (35) describe transitions that do not change by the action of $move(c, a)$. Computing the least generalization of (30) and (35) and removing redundant atoms, we get

$$hold(clear(x)) \leftarrow conj(S), move(x, y) \tag{36}$$

The rule (36) represents a frame axiom saying that moving a block x to y at the state S does not change the clearness of x. Computing the least generalization of (28) and (34) and removing redundant atoms, we get

$$hold(on(b, t)) \leftarrow conj(S), move(x, y) \tag{37}$$

The rule (37) represents a frame axiom saying that moving a block x to y at the state S does not change the location of b on the table. Furthermore, generalizing (29) and (37), we get

$$hold(on(z, t)) \leftarrow conj(S), move(x, y) \tag{38}$$

However, the rule (38) has the instance

$$hold(on(c, t)) \leftarrow conj(S), move(c, a)$$

which conflicts with the consequence of (32).[7]

Hence one can conclude that the rules (36) and (37) are valid frame axioms, while (38) is not a proper rule and is discarded. As such, frame axioms are successfully produced by induction.

[7] Here we assume the existence of a *state constraint* : $\forall x \forall y\, [hold(on(x, t)) \wedge hold(on(x, y)) \rightarrow y = t\,]$ asserting that if an object x is on a table t and x is on y then y is t.

3.3 Conversational Implicature

Non-logical inferences are also used in *pragmatics* [6]. In conversation or dialogue, the notion of *conversational implicature* [3] is known as a pragmatic inference to an implicit meaning of a sentence that is not actually uttered by a speaker. For instance, if a speaker utters the sentence "I have two children", it normally implicates "I do *not* have *more than* two children". This is called a *scalar implicature* which says that a speaker implicates the negation of a semantically stronger proposition than the one asserted. Given a collection of dialogues, a question is whether a machine learning system can automatically acquire pragmatic rules of inference that interpret implicit meaning behind utterance. To realize this, we assume a simple dialogue system that is able to converse with a human. Given a sentence S by a human, the system asks whether a sentence T that is semantically stronger than S is true or not. The human answers "yes" if it is true, and "no" otherwise. For instance, the following dialogue is taken place.

human: I have two children.
computer: Do you have three children?
human: No.
. . .
human: I have a car.
computer: Do you have two cars?
human: No.

The human's utterance is translated into factual knowledge as follows:

$$hold(have(child, s(s(0)))) \tag{39}$$
$$hold(\neg have(child, s(s(s(0))))) \tag{40}$$
$$hold(have(car, s(0))) \tag{41}$$
$$hold(\neg have(car, s(s(0)))) \tag{42}$$

Let (I, J) be a pair in which I represents the initial utterance of a human and J represents a reply by the human in response to a question by a computer. Then the above dialogue is represented by pairs:

$$(I, J) = (\{hold(have(child, s(s(0))))\}, \{hold(\neg have(child, s(s(s(0)))))\}),$$
$$(\{hold(have(car, s(0)))\}, \{hold(\neg have(car, s(s(0))))\}).$$

In this case, both I and J contain a single atom, so that the bottom-up **LF1T** simply constructs the transition rules as $A \leftarrow B$ where $A \in J$ and $B \in I$:

$$hold(\neg have(child, s(s(s(0))))) \leftarrow hold(have(child, s(s(0)))) \tag{43}$$
$$hold(\neg have(car, s(s(0)))) \leftarrow hold(have(car, s(0))) \tag{44}$$

The least generalization of (43) and (44) becomes

$$hold(\neg have(x, s(y))) \leftarrow hold(have(x, y)). \tag{45}$$

The rule (45) means if one says that the number of x he/she has is y, it implies that the number is not $y+1$. Suppose that the background knowledge contains the rule:

$$hold(\neg have(x,z)) \leftarrow hold(\neg have(x,w)), z \geq w \tag{46}$$

which says that if one does not have x that is w in number, then he/she does not have x more than w. Then, after learning (45), the next rule is deduced by (45) and (46):

$$hold(\neg have(x,z)) \leftarrow hold(have(x,y)), z \geq s(y) \tag{47}$$

which means that if one says that the number of x he/she has is y, it implies that he/she does not have it more than y. The rule (47) represents a rule of scalar implicature. On the other hand, if there is another dialogue such that

human: I have two dollars.
computer: Do you have three dollars?
human: Yes.

Then the next rule is constructed

$$hold(have(dollar, s(s(s(0))))) \leftarrow hold(have(dollar, s(s(0)))) \tag{48}$$

This is an exceptional case of scalar implicature. In the presence of (48), the rule (47) could be refined as

$$hold(\neg have(x,z)) \leftarrow hold(have(x,y)), z \geq s(y), x \neq dollar$$

or one could construct scalar implicature rules for individuals such that:

$$hold(\neg have(child,z)) \leftarrow hold(have(child,y)), z \geq s(y) \ \ (\text{from } (43) \text{ and } (46))$$
$$hold(\neg have(car,z)) \leftarrow hold(have(car,y)), z \geq s(y) \ \ (\text{from } (44) \text{ and } (46))$$

As such, scalar implicature rules are inductively constructed.

4 Discussion

New paradigms are emerging in machine learning such as *ontology learning* [18] and *representation learning* [1]. Also recent advances in robotics argue possibilities of robots' recognizing objects in the world, categorizing concepts, and associating names to them (*physical symbol grounding*) [2]. Once robots successfully learn concepts and associate symbols to them, the next step is to learn relations between concepts and logical or physical rules governing the world. According to Piaget's theory of cognitive development, children start learning concepts and symbols at age earlier than two (*pre-operational stage*), and begin to understand logical or rational thought at age around seven (*concrete operational stage*) [11]. Representation learning and physical symbol grounding aim at realizing machine learning at the level of the pre-operational stage. On the other hand, learning inference considered in this paper targets the problem of realizing machine learning at the level of the concrete operational stage.

In learning logical inferences, we show that the **LF1T** induction algorithm can reconstruct a system $\mathcal{L}$ of natural deduction. An interesting question is whether a machine learning algorithm can discover a *new* axiomatic system that is semantically equivalent to $\mathcal{L}$. It addresses the possibility of AI's discovering new logics that are unknown to human mathematicians. A logical formulation of conversational implicature is studied in [15] while, to the best of our knowledge, learning conversational implicature from dialogue has never been explored. We showed a simple case study of learning scalar implicature in conversation. If one develops an AI that learns and understands conversational implicature, it will realize an intelligent chat bot that can understand implicit meaning of humans' utterance. Learning non-logical inference also involves the problem of learning thinking patterns of individuals. It would be interesting to investigate whether a system can learn thinking patterns of people in particular regions or in particular professionals.

This paper realizes learning inference as induction of meta-rules. Induction in meta-theories are proposed in [4, 8]. Inoue $et\,al.$ [4] introduce *meta-level abduction* to invent predicates and apply it to finding physical skills. Muggleton $et\,al.$ [8] introduce *meta-interpretive learning* to invent relations by abduction and apply it to learning grammatical rules. These studies represent background knowledge as a meta-theory and abduce rules as meta-facts, while their goals are not learning inference rules. Induction or ILP has mostly been used for learning knowledge, while little study has been devoted to the topic of learning inference. The current study argues the possibility of using ILP for learning inference and serves as a step for opening the topic.

5 Conclusion

This paper studied learning inference by induction. We first addressed a method of learning deductive inference rules using the **LF1T** induction algorithm. We showed that **LF1T** successfully produces inference rules of propositional natural deduction as transition rules from premises to consequences. Secondly, we applied the method to learning non-logical inference rules. We showed that abductive inference rules are obtained from observations and explanations, frame axioms are computed by state changes, and scalar implicature rules in conversation could be learned from simple dialogues.

This is a preliminary research for learning inference rules by induction, and the proposed method will need further elaboration and extension in practice. Providing all possible transitions, **LF1T** will output production rules that are minimal with respect to subsumption. A limitation is that the number of possible transactions increases exponentially in proportion to the size of the Herbrand base. Further optimization is needed for learning inference rules from huge data. In this paper, **LF1T** is used for learning propositional inference rules in Section 2. For learning first-order inference rules, provide pairs of premises-consequences at the fact level and produce ground rules at first, then generalize those rules using ILP technique such as least generalization. Such a technique is directly applied to quantifier-free rules and further technique will be needed for learning quantified formulas. Applying the proposed framework to learning inference rules in other logics (e.g. probability/fuzzy logic), learning social rules in multia-

gent systems (e.g. negotiation), and learning strategic rules in games (e.g. chess) would be of interest.

References

1. Bengio, Y., Courville, A., Vincent, P.: Representation learning: a review and new perspectives. IEEE Transactions on Pattern Analysis & Machine Intelligence **35**, 1798–1828 (2013)
2. Coradeschi, S., Loutfi, A., Wrede, B.: A short review of symbol grounding in robotic and intelligent systems. KI - Kunstliche Intelligenz **27**, 129–136 (2013)
3. Grice, H. P.: Logic and conversation. In: Cole, P., Morgan, J. (eds.), Syntax and Semantics, 3: Speech Acts, 41–58, Academic Press (1975)
4. Inoue, K., Furukawa, K., Kobayashi, I., Nabeshima, H.: Discovering rules by meta-level abduction. Inductive Logic Programming, 19th International Conference, ILP 2009. LNCS (LNAI), vol. 5989, pp. 49–64, Springer, Heidelberg (2010)
5. Inoue, K., Ribeiro, T., Sakama, C.: Learning from interpretation transition. Machine Learning **94**, 51–79 (2014)
6. Levinson, S. C.: Pragmatics. Cambridge University Press (1983)
7. McCarthy, J., Hayes, P. J.: Some philosophical problems from the standpoint of artificial intelligence. In: Meltzer, B., Michie, D. (eds.), Machine Intelligence 4, pp. 463–502, Edinburgh University Press (1969)
8. Muggleton, S. H., Lin, D., Pahlavi, N., Tamaddoni-Nezhad, A.: Meta-interpretive learning: application to grammatical inference. Machine Learning **94**, 25–49 (2014)
9. Nienhuys-Cheng, S.-H., de Wolf, R.: Foundations of inductive logic programming. LNCS (LNAI), vol. 1228, Springer, Heidelberg (1997)
10. Peirce. C. S.: Collected Papers of Charles Sanders Peirce. Hartshorne, C., Weiss, P., Burks, A. W. (eds.), Harvard University Press (1958)
11. Piaget, J.: Main Trends in Psychology. London: Allen & Unwin (1973)
12. Plotkin, G. D.: A note on inductive generalization. In: Meltzer, B, Michie, D. (eds.), Machine Intelligence 5, pp. 153–63, Edinburgh University Press (1970)
13. Prawitz, D.: Natural Deduction: a proof-theoretical study. Dover Publications (2006)
14. Ribeiro, T., Inoue, K.: Learning prime implicant conditions from interpretation transition. Inductive Logic Programming, 24th International Conference, ILP 2015. LNCS (LNAI), vol. 9046, pp. 108–125, Springer, Heidelberg (2015)
15. Sakama, C., Inoue, K.: Abduction and conversational implicature. 12th International Symposium on Logical Formalizations of Commonsense Reasoning, AAAI Spring Symposium, Technical Report SS-15-04, pp. 130–133 (2015)
16. Sakama, C., Inoue, K.: Can machines learn logics? Artificial General Intelligence, 8th International Conference, AGI 2015. LNCS (LNAI), vol. 9205, pp. 341–351, Springer, Heidelberg (2015)
17. van Emden, M. H., Kowalski. R. A.: The semantics of predicate logic as a programming language. J. ACM **23**, 733–742 (1976)
18. Wong, W., Liu, W., Bennamoun, M.: Ontology learning from text: a look back and into the future. ACM Computing Surveys **44**, 20:1-20:36 (2012)
19. Woods, J., Irvine, A., Walton, D.: Argument: Critical Thinking, Logic and the Fallacies. Prentice-Hall, Toronto (2000)