A Case Study on Extracting the Characteristics of the Reachable States of a State Machine formalizing a Communication Protocol with Inductive Logic Programing

Dung Tuan Ho*, Min Zhang**, and Kazuhiro Ogata*

*Japan Advanced Institute of Science and Technology (JAIST)
{dung.ho,ogata}@jaist.ac.jp
**East China Normal University (ECNU)
zhmtechie@gmail.com

Abstract. A distributed system DS can be formalized as a state machine M and many desired properties of DS can be expressed as invariants of M. An invariant of M is a state predicate p of M such that p holds for all reachable states of M. To verify that DS enjoys a desired property, namely to prove that p is an invariant of M, we often need to find other invariants as lemmas, which is one of the most intellectual activities in Interactive Theorem Proving (ITP). For this end, our experiences on ITP tell us that it is useful to get better understandings of the reachable states R_M of M. We report on a case study in which Progol, an Inductive Logic Programming (ILP) system, has been used to extract the characteristics of the reachable states of a state machine formalizing Alternating Bit Protocol, a communication protocol. The case study demonstrates that ILP is useful to extract the characteristics of R_M.

Keywords: Alternating Bit Protocol, invariant, Inductive Logic Programming, Progol, state machine, reachable states

1 Introduction

A state machine consists of a set of states that includes the initial states and a binary relation over states. An element of the binary relation is called a transition. A distributed system DS can be formalized as a state machine M and many desired properties of DS can be expressed as invariants of M. An invariant of M is a state predicate p of M such that p holds for all reachable states of M. Reachable states of M are inductively[1] defined as follows: each initial state of M is reachable, and if a state s of M is reachable and (s, s') is an element of the binary relation of M, then s' is reachable.

To prove that p is an invariant of M, it suffices to to find an inductive invariant q of M such that $q(s) \Rightarrow p(s)$ for each state s of M. An inductive

[1] Note that "induction" is used to refer to two different meanings: one from ILP and the other from mathematical induction.

invariant q of M is a state predicate of M such that $q(s_0)$ holds for each initial state s_0 of M and $q(s) \Rightarrow q(s')$ holds for each element (s, s') of the binary relation of M. Note that an inductive invariant of M is an invariant of M but not vice versa.

Finding such inductive invariant q (or conjecturing a lemma q) is one of the most intellectual activities in ITP[2]. This activity requires human users to profoundly understand the system under verification or M formalizing the system to some extent. The users must rely on some reliable sources that let them get better understandings of the system and/or M to conduct the non-trivial task, namely *lemma conjecturing*. For this end, our experiences on ITP tell us that it is useful to get better understandings of the reachable state R_M of M. Some characteristics of R_M can be used to systematically construct a state predicate q_i that is a part of q.

States in M are characterized by some values that are called *observable values*. Based on our experiences on ITP, the characteristics of R_M are correlations among observable values of the elements (the reachable states of M) of R_M. Generally, the number of the elements of R_M is unbounded and then a huge number of reachable states are generated from M. The task of extracting correlations among a huge number of data (reachable states in our case) is the role of Machine Learning (ML).

We have conducted a case study in which Progol, an ILP system, has been used to extract the characteristics of the reachable states $R_{M_{\mathrm{ABP}}}$ of a state machine M_{ABP} formalizing Alternating Bit Protocol (ABP) that is a simplified version of Sliding Window Protocol used in TCP. The reasons why we have used ABP in the case study are that we have verified that ABP enjoys some desired property with ITP and extracted the characteristics of a $R_{M_{\mathrm{ABP}}}$. In the case study, we have made a comparison between the characteristics extracted by Progol and those manually extracted, demonstrating that Progol is able to extract many interesting characteristics of $R_{M_{\mathrm{ABP}}}$.

The rest of the paper is organized as follows. Sect. 2 describes what motivated us to extract the characteristics of R_M with ILP. Sect. 3 describes ABP, M_{ABP} and manually extracted characteristics of $R_{M_{\mathrm{ABP}}}$. Sect. 4 describes our method to extract the characteristics of R_M with ILP and reports on the case study. Sect. 5 mentions some related work. Sect. 6 concludes the paper and mentions some future direction.

2 Motivation

2.1 Systems Verification with Interactive Theorem Proving (ITP)

System verification is a research area aiming at rigorously checking if systems satisfy desired properties. ITP is a formal verification technique in which mathe-

[2] q may be in the form $q_1 \wedge \ldots \wedge q_n$. Each q_i may be called a lemma and is an invariant of M if q is an inductive invariant of M, although q_i may not be an inductive invariant of M.

matical models are made for systems and desired properties are treated as theorems of the mathematical models. State machines are used as such mathematical models. For example, it is possible to check if an e-commerce protocol satisfies the property that if an acquirer authorizes a payment, then both the buyer and seller concerned always agree on it [1]. Logics, theories, techniques and tools for theorem proving have been advanced a lot, e.g. logical decision procedures used in SMT [2]. However, some non-trivial interactions between human users and theorem provers are still needed to conduct proofs that non-trivial state machines enjoy non-trivial properties. One of the most intellectual activities in such interactions is to conjecture lemmas.

To formally verify that a system satisfies a desired property with ITP, the system is first formalized as a state machine M that is described in a formal specification language. A state predicate p is described in the same or a different specification language for the property. An interactive theorem prover is used to prove that p is an invariant of M. We use a proof score approach to systems verification called the OTS/CafeOBJ method[3], in which CafeOBJ, an algebraic specification language, is used as a specification language for M and p and also as an interactive theorem prover. There are three main activities in the OTS/CafeOBJ method to conduct ITP: application of simultaneous structural induction, case analysis and use of lemmas (including lemma conjecturing).

Let us consider a mutual exclusion protocol called TAS as an example. TAS written in an Algol-like language is shown in Fig. 1 (a). TAS uses *lock* to control processes such that there is at most one process in Critical Section (or at cs). Initially, *lock* is false and each process is in Remainder Section (or at rs). test&set(b) atomically sets b true and returns false if b is false, and just returns true otherwise. TAS is formalized as a state machine M_{TAS} whose transitions are depicted in Fig. 1 (b) and (c). The arrow on which $\mathrm{try}_i/[b = \mathrm{false}]$ is attached is interpreted as follows: if process i is at rs and b is false in a given state, then i moves to *cs* and b is set true. Note that transitions are declared in terms of equations in the OTS/CafeOBJ method. One desired property TAS should enjoy is the mutual exclusion property. Let $\mathrm{mx}(s, p, q)$ be $(\mathrm{pc}(s, p) = \mathrm{cs} \wedge \mathrm{pc}(s, q) = \mathrm{cs} \Rightarrow p = q)$, where s is a state, p, q are process identifiers and $\mathrm{pc}(s, x)$ is the location (rs or cs) where process x is in state s, and let $\mathrm{mx}(s)$ be $(\forall p, q \in \mathrm{Pid})\, \mathrm{mx}(s, p, q)$, where Pid is the set of all process identifications. To verify that TAS enjoys the property, all we have to do is to prove that $\mathrm{mx}(s)$ is an invariant of M_{TAS}.

Fig. 2 shows a snip of a proof tree that $\mathrm{mx}(s)$ is an invariant of M_{TAS}, although proofs are written as texts in the OTS/CafeOBJ method. Given a state s and a process identification r, $\mathrm{try}(s, r)$ is the state obtained by applying transition try_r in s, $\mathrm{exit}(s, r)$ is the state obtained by applying transition exit_r in s, and $\mathrm{lock}(s)$ is the Boolean value stored in variable *lock* in s. Simultaneous structural induction on s is first used to split the initial goal into three sub-cases. What to do for the three sub-cases is to show $\mathrm{mx}(s_0, p, q)$, $\mathrm{mx}(s, p, q) \Rightarrow \mathrm{mx}(\mathrm{try}(s, r), p, q)$ and $\mathrm{mx}(s, p, q) \Rightarrow \mathrm{mx}(\mathrm{exit}(s, r), p, q)$, respectively, where s_0 is an arbitrary initial

[3] Due to the space limitation, we do not explain the OTS/CafeOBJ method in detail. Please refer to [3, 4] for the OTS/CafeOBJ method

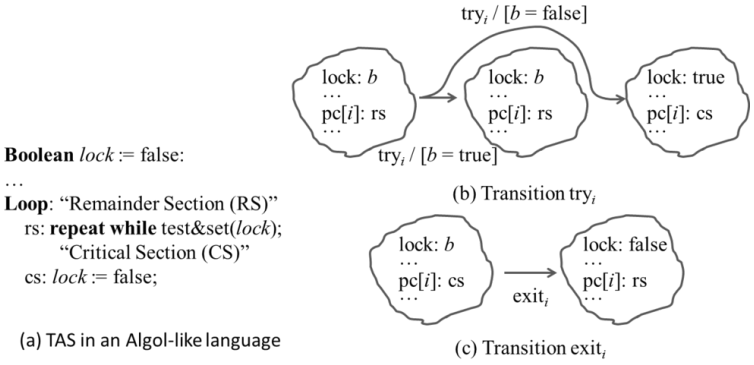

Fig. 1. TAS and a state machine M_{TAS} formalizing TAS

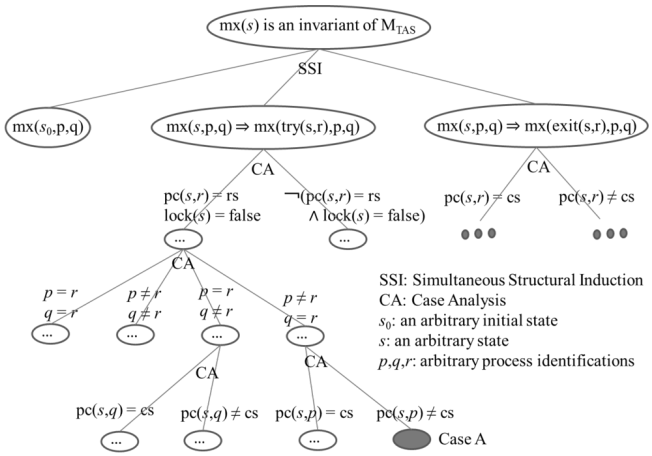

Fig. 2. A snip of a proof tree that mx(s) is an invariant of M_{TAS}

state, s is an arbitrary state, and p, q, r are arbitrary process identifications. Case analysis is then repeatedly used until what to show reduces either true or false. Any case in which what to show reduces true is discharged. For any case in which what to show reduces false, we need to conjecture lemmas. Let us consider the case marked Case A in Fig. 2 in which $\mathrm{mx}(s,p,q) \Rightarrow \mathrm{mx}(\mathrm{try}(s,r),p,q)$ reduces false. We can find some contradiction in the assumptions that characterize Case A: $\mathrm{pc}(s,r) = \mathrm{rs}$, $\mathrm{lock}(s) = \mathrm{false}$, $p \neq r$, $q = r$ and $\mathrm{pc}(s,p) \neq \mathrm{cs}$. We notice that $\mathrm{lock}(s) = \mathrm{false}$ contradicts $\mathrm{pc}(s,p) = \mathrm{cs}$, from which we conjecture the lemma: $\mathrm{pc}(s,p) = \mathrm{cs} \Rightarrow \mathrm{lock}(s)$. Let $\mathrm{lem1}(s,p)$ refer to the lemma (a state predicate). In Case A, $\mathrm{lem1}(s,p) \wedge \mathrm{mx}(s,p,q) \Rightarrow \mathrm{mx}(\mathrm{try}(s,r),p,q)$ reduces true, discharging Case A, provided that we prove that $(\forall p \in \mathrm{Pid})\, \mathrm{lem1}(s,p)$ is an invariant of M_{TAS}. The proof needs $\mathrm{mx}(s,p,q)$ as a lemma. This is why we use simultaneous structural induction.

The systematic way to conjecture lemmas may not work for larger systems than TAS because case analysis may have to be repeated too many times until what to show reduces either true or false and there may be too many assumptions to find any contraction in them. Fortunately, we have some experiences that lemmas can be effectively conjectured based on the characteristics of R_M. Therefore, it is worth extracting the characteristics of R_M for a given M. The characteristics are considered as knowledge about R_M that is huge and unbounded in general. The task of extracting knowledge from a large database is the role of ML. However, many classical machine-learning techniques only work for a database whose elements are expressed in propositional form, while our database consists of system states expressed in first-order form. There is the ML technique that can deal with first-order forms: Inductive Logic Programming.

2.2 Inductive Logic Programming (ILP)

ILP [5] is a research area staying at the intersection of ML and Logic Programming (LP) such that ILP inherits techniques and features from both sides. ILP's goal (also ML's goal) is to develop techniques and tools to induce the knowledge from a large amount of examples and to synthesize new knowledge from experience. Because ILP uses a LP language as the way to represent knowledge (called hypotheses) and examples, first-order formulas can be dealt with by ILP. This is important because there are many research areas whose domain knowledge needs to be expressed in first-order logic or a variant of first-order logic. This is also an important difference between ILP and classical ML techniques that use the limited knowledge representation mechanism such as propositional logic. Moreover, the representation mechanism in ILP makes the use of background knowledge easier and more efficient than the classical ML techniques. It is important because one of the well-established findings of artificial intelligence is that the use of background knowledge is essential for archiving intelligent behavior. In contrast to most other approaches to inductive learning, ILP is interested in properties of inference rules, in convergence of algorithms and in the computational complexity of procedures. Many ILP systems benefit from using the results of LP. ILP extends the theories and practice of computational logic by investigating induction rather than deduction as the basic mode of inference. Whereas LP theory describes deductive inference of logic formulae provided by users, ILP theory describes the inductive inference of logic programs from examples and background knowledge. In a general setting, the ILP's learning task is defined as follows. Given background knowledge B and examples E. The examples $E = E^+ \wedge E^-$ consists of positive examples E^+ such that $B \not\models E^+$ and negative examples E^- such that $B \wedge E^- \not\models \Box$. The aim is then to find a hypothesis H such that the hypothesis is complete with respect to the positive examples E^+, denoted $B \wedge H \models E^+$, and is consistency with respect to the negative examples E^-, denoted $B \wedge H \wedge E^- \not\models \Box$.

To formalize fully an ILP task, the relation between B, H, E^+ and E^- needs to be defined. This depends on several factors that include the chosen *LP language* and *LP semantics*. ILP methods often require some form of bias

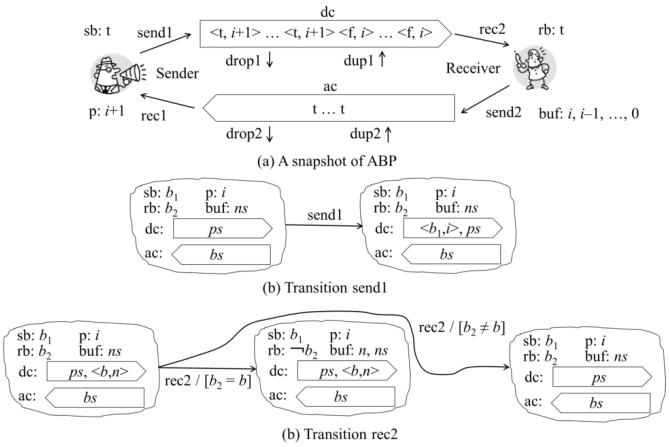

Fig. 3. ABP and part of a state machine M_{ABP} formalizing ABP

on the solution search space to restrict the computation to hypotheses. Forms of bias include *language bias* and *search bias* (e.g. top-down or bottom-up). A *Mode Declaration* is a form of language bias that specifies the syntactic form of the hypotheses that can be learned. It contains head declarations and body declarations that describe predicates that may appear, the desired input and output and number of instantiations (called *recall*).

Our purpose is to characterize R_M. However, it is not straightforward to generate unreachable states from a system specification M that can be used as negative examples, although positive examples that are reachable states and background knowledge can be obtained from the system specification. Fortunately, many learning modes for many kinds of learning purposes have been developed in Progol. One of them is learning from positive data only [6] that is most suitable mode for our purpose. This learning mode is based on a Bayesian framework such that an upper bound for expected error is calculated with respect to maximizing the Bayesian posterior probability when learning from positive examples only.

3 Verification of Alternating Bit Protocol

Alternating Bit Protocol (*ABP*) is a communication protocol that makes it possible to reliably deliver packets (expressed as natural numbers) to a receiver from a sender even under unreliable channels whose contents may be lost and duplicated. A snapshot of ABP is shown in Fig. 3 (a). Let Bool be the set of Boolean values, Nat be the set of natural numbers, Pair be the set of Bool-Nat pairs, PQueue be the set of queues of Pair, BQueue be the set of queues of Bool and List be the set of lists of Nat. $sb, rb \in$ Bool, $p \in$ Nat representing the packet to be delivered, $buf \in$ List in which the delivered packets are stored, $dc \in$ PQueue

and $ac \in$ BQueue. Note that t and f stand for true and false. There are six kinds of actions referred as $send_1$, rec_1, $send_2$, rec_2, dup_1, $drop_1$, dup_2, and $drop_2$. $send_1$ puts $\langle sb, p\rangle$ into dc. $send_2$ puts rb into ac. rec_1 gets the top b from ac if any, and complements sb and increments p if $sb \neq b$. rec_2 gets the top $\langle b, n\rangle$ from dc if any, and complements rb and adds n to buf if $rb = b$. $drop_1$ and $drop_2$ drop the top from dc and ac, respectively. dup_1 and dup_2 duplicate the top of dc and ac, respectively. ABP is formalized as a state machine M_{ABP}:

- Each state is characterized by six observable values. For each state s, $\mathrm{sb}(s)$, $\mathrm{rb}(s)$, $\mathrm{p}(s)$, $\mathrm{buf}(s)$, $\mathrm{dc}(s)$ and $\mathrm{ac}(s)$ represent sb, rb, p, buf, dc and ac, respectively. Their initial values are t, t, 0, nil (the empty list), the empty queue, and the empty queue, respectively.
- There are six kinds of transitions corresponding to the six kinds of actions, and the names of the six kinds of actions are also used as the names of the six kinds of transitions. For each state s, $\mathrm{send}_1(s)$, $\mathrm{rec}_1(s)$, $\mathrm{send}_2(s)$, $\mathrm{rec}_2(s)$, $\mathrm{dup}_1(s)$, $\mathrm{dup}_2(s)$, $\mathrm{drop}_1(s)$ and $\mathrm{drop}_2(s)$ denote the states obtained by applying the six kinds of actions in s, respectively. Transitions $send_1$ and rec_2 are depicted in Fig. 3 (b) and (c), respectively.

One property ABP should enjoy is called the reliable communication property that all packets up to p, the one currently being delivered (or $p-1$, the previous one) have been successfully delivered without any duplications nor any drops. All we have to do is to prove that the following state predicate (referred as $\mathrm{rcp}(s)$) is an invariant of M_{ABP} to verify that ABP enjoys the property: $(\mathrm{sb}(s) = \mathrm{rb}(s) \Rightarrow \mathrm{mklst}(\mathrm{p}(s)) = \mathrm{p}(s), \mathrm{buf}(s)) \wedge (\mathrm{sb}(s) \neq \mathrm{rb}(s) \Rightarrow \mathrm{mklst}(\mathrm{p}(s)) = buf(s))$, where $\mathrm{mklst}(0) = 0, \mathrm{nil}$ and $\mathrm{mklst}(n+1) = n+1, \mathrm{mklst}(n)$.

To prove that $rcp(s)$ is an invariant of M_{ABP}, we have first used simultaneous structural induction on s. Applying case analysis a couple of times to the induction case where transition rec_1 is taken into account, we have reached the case that corresponds to an arbitrary state s such that $\mathrm{ac}(s) = b10\ bf10$, $\mathrm{sb}(s) \neq b10$ and $\mathrm{rb}(s) \neq b10$, where $b10$ is an arbitrary one of Bool and $bf10$ is an arbitrary one of PQueue. The assumptions say that $\mathrm{ac}(s)$ is not empty and the top element of $\mathrm{ac}(s)$ is neither the same as $\mathrm{sb}(s)$ nor $\mathrm{rb}(s)$. The case could have been further split, but our experiences of theorem proving and understandings of ABP have told us that there seems to be some contradiction in the case. For any reachable state s, if $\mathrm{ac}(s)$ is not empty, then the top element of $\mathrm{ac}(s)$ must be the same as $\mathrm{sb}(s)$ and/or $\mathrm{rb}(s)$. Then, we have conjectured a lemma: $\mathrm{ac}(s) \neq \mathrm{empty} \Rightarrow (\mathrm{top}(\mathrm{ac}(s)) = \mathrm{sb}(s) \vee \mathrm{top}(\mathrm{ac}(s)) = \mathrm{rb}(s))$. The lemma can be used to discharge the case.

This is why we have graphically drawn the six state patterns shown in Fig. 4. We have conjectured several lemmas from the six state patterns to complete the proof that $\mathrm{rcp}(s)$ is an invariant of M_{ABP}. The six state patterns can be regarded as the characteristics of the reachable states of M_{ABP} and we have learned that the characteristics of R_M help us conjecture lemmas for the proof that a state predicate is an invariant of M. One of those lemmas is $(\forall ps_1, ps_2 \in \mathrm{PQueue})(\forall p_1, p_2, p_3 \in \mathrm{Pair})\,(dc(s) = ps_1@(p_1\ p_2\ ps_2) \wedge p_1 \neq p_2 \Rightarrow (p_3 \in ps_2 \Rightarrow$

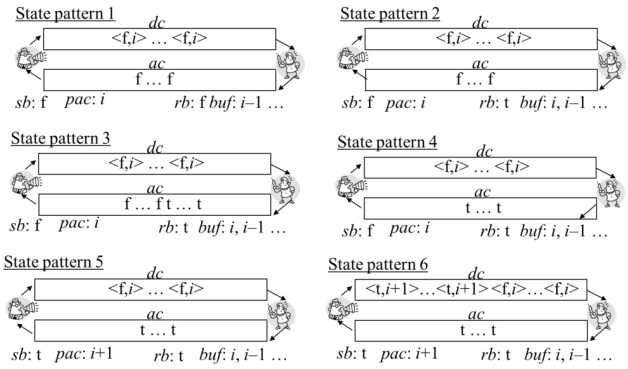

Fig. 4. Six state patterns of ABP

$p_2 = p_3) \wedge p_2 = \langle \mathrm{sb}(s), \mathrm{p}(s) \rangle)$ that can be conjectured from State pattern 6, where @ is the concatenation function of queue. In the next section, we report on a case study that Progol, an ILP system, has been used to extract the characteristics of the reachable states of M_{ABP} in the form of logic programs, which demonstrates that ILP is useful to extract the characteristics of R_M.

4 A Case Study

4.1 Method

The architecture of the proposed method that extracts the characteristics of R_M is depicted in Fig. 5. A system specification of M for the OTS/CafeOBJ method consists of a part of data structures and a part of a state machine. The data structure part is converted into Horn clauses that define types and predicates used as the background knowledge by an ILP system. The system specification for the OTS/CafeOBJ method is translated into another system specification of M that is suited for model checking with YAST [7], a specification translator. A bounded model checker (the Maude search command) is used to generate a set of reachable states from the system specification for model checking. The set of reachable states is used as E^+. This is how the sate machine part is used. The Maude search command can generate reachable states that satisfy a given condition. For example, states s such that neither $\mathrm{dc}(s)$ nor $\mathrm{ac}(s)$ is empty can be generated. The background knowledge, E^+ and mode declarations are fed into an ILP system and H is generated as the result. H is a set of Horn clauses that defines a predicate that can be regarded as the characteristics of R_M.

Among data structures used in the system specification of M_{ABP} for the OTS/CafeOBJ method are those for Bool, Nat, List, PQueue and BQueue. The data structure part also contains some functions on those data structures such as those that put an element into a queue and get the top from a queue if any. Such functions are converted into predicates defined in clauses. We may have to

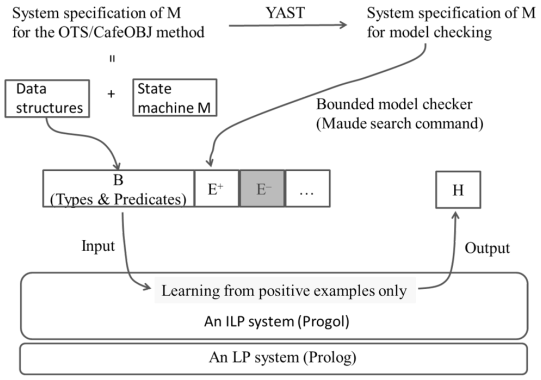

Fig. 5. Architecture of proposed method

use some more functions that are not explicitly used in the system specification such as gap0 and gap1 mentioned later to extract the better characteristics of R_M.

4.2 Experiments

We have conducted two experiments referred as Expt. 1 and Expt. 2. Both experiments have used about 10000 reachable states as positive examples, although each reachable state s_1 used in Expt. 1 fulfills the constraints that the $\mathrm{len}(\mathrm{dc}(s_1)) > 0$, $\mathrm{len}(\mathrm{ac}(s_1)) > 0$ and $\mathrm{len}(\mathrm{buf}(s_1)) > 0$, and each reachable state s_2 used in Expt. 2 fulfills the constraints that the $\mathrm{len}(\mathrm{dc}(s_2)) > 1$, $\mathrm{len}(\mathrm{ac}(s_2)) > 1$ and $\mathrm{dc}(s_2)) > 1$, $\mathrm{len}(\mathrm{ac}(s_2)) > 1$ and $\mathrm{len}(\mathrm{buf}(s_2)) >$, where len returns the number of elements in a given queue (or list). The constraints prevent from generating many trivial non-interesting clauses.

Each experiment produces a set of clauses such that the head of each clause is in the form $state(sb, p, rb, buf, dc, ac)$ that represents a state of M_{ABP} and the body of each clause captures some characteristics of the reachable states of M_{ABP}. Expt. 1 has used the following head mode declaration (referred as modeh1):

$$modeh(1, state(+bool, +pnat, +bool, +nlist, +pqueue, +bqueue))?$$

The recall number is 1 and each parameter of *state* is a single input variable. Exp. 2 has used the following head mode declaration (referred as modeh2):

$$modeh(1, state(+bool, +pnat, +bool, +nlist, \\ [p(+bool, +pnat) \mid +pqueue], [+bool \mid +bqueue]))?$$

where $p(b, n)$ represents $\langle b, n\rangle$ and and $[e \mid l]$ represents the list such that e is the head and l is the tail. The 5th parameter of *state* is not nil, and neither is the 6th parameter.

The following predicates are also used in body mode declarations:

toppqu(A,B) Pair B is on top of queue A of pairs
topbqu(A,B) Boolean value B is on top of queue A of Boolean values
mk(A,B) B is a ordered list of natural numbers from number A to 0
neg(A,B) Boolean value A is the complement of B
fst(A,B) A is the first element of pair B
snd(A,B) A is the second element of pair B
pred(A,B) Number B is the predecessor of number A
memp(A,B) Pair A is in queue B of pairs
memb(A,B) Boolean value A is in queue B of Boolean values

4.3 Evaluation

The set of clauses extracted as the hypothesis in Expt. 1 is as follows:

state(A,B,C,D,E,F):- toppqu(E,G),topbqu(F,A),mk(B,D), neg(A,C), fst(G,A), snd(G,B).
state(A,B,C,D,E,F):- toppqu(E,G),topbqu(F,C),mk(B,D),neg(A,C),fst(G,A), snd(G,B).
state(A,B,C,D,E,F):- toppqu(D,F),topbqu(E,A),pred(B,G), fst(F,H),snd(F,G), mk(G,C),neg(H,A).
state(A,B,C,D,E,F):- toppqu(D,F),topbqu(E,A),memp(p(A,B),D),pred(B,G), fst(F,A),snd(F,B),mk(G,C).

The set of clauses extracted as the hypothesis in Expt. 2 is as follows:

state(A,B,C,D,[p(A,B)|E],[A|F]):- neg(A,C),memp(p(A,B),E),memb(A,F).
state(A,B,C,D,[p(A,B)|E],[A|F]):- neg(A,C), memp(p(A,B),E),memb(C,F).
state(A,B,C,D,[p(E,F)|G],[C|H]):- neg(E,C),memp(p(A,B),E),memb(C,F).
state(A,B,A,C,[p(D,E)|F],[A|G]):- neg(A,D), succ(E,B),memp(p(A,B),F), memb(A,G).
state(A,B,C,D,[p(A,B)|E],[A|F]):- neg(A,G),memp(p(C,B),E), memb(A,F), neg(G,C).

Each clause obtained from these two experiments characterizes some aspects of reachable states. Considering Expt. 1, the first clause describes State pattern 2 and 3 such that $sb \neq rb$ (for $neq(A, C)$), sb is the top bit of ac (for $topbqu(F, A)$) and $\langle sb, p\rangle$ is the top packet of dc (for $topppqu(E, G)$, $fst(G, A)$, $snd(G, B)$). The second clause describes State pattern 4, is quite similar to the first clause, but rb is the top bit of ac (for $topbqu(F, C)$). The two remain clauses describe the case $sb = rb$. The third clause describes State pattern 5 and 6 such that the top packet $\langle b, n\rangle$ of dc is different from $\langle sb, p\rangle$, $b = rb$, $n = p - 1$ and mklst(n) is the same as buf. The last clause is quite similar to the third one but it describes State pattern 1 such that $\langle sb, p\rangle$ is the top pair of dc. This clause correctly describes almost all reachable states classified into State pattern 1 but does not

some states with $p = 0$. This is because $\mathbf{buf}(s) \neq$ nil for each s in the set of reachable states used as E^+ in Expt. 1, and then we do not think that this is a serious problem.

For characterizing $R_{M_{ABP}}$ in more detail, we have conducted Expt. 2 using modeh2 that specifies that the inputs used for both the 5th and 6th parameters of *state* are not empty. Because of each reachable state s used in Expt. 2 as a positive example such that $\mathrm{len}(\mathrm{dc}(s)) > 1$ and $\mathrm{len}(\mathrm{ac}(s)) > 1$, moreover, the bottom of each of two such inputs is not empty.. Therefore, we do not need to use *toppqu* and *topbqu* in the background knowledge. Each obtained clause describes the state patterns in more detail than Expt. 1. For instance, the first clause describes State pattern 2 and 3 in which *sb* is the top bit of *ac* and also appears in the bottom of *ac*. The second clause describes State pattern 3 in much more detail than the first one such that *rb* appears in the bottom of *ac*. The third clause describes State pattern 4 such that $sb \neq rb$ and *rb* appears in the top and bottom of *ac*. The fourth clause describes State pattern 6 such that $sb = rb$, $\langle \neg sb, p - 1 \rangle$ is the top of *dc* and $\langle sb, p \rangle$ appears in the bottom of *dc*. The last clause describes State pattern 1 such that $sb = rb$, $\langle sb, p \rangle$ appears in both the top and bottom of *dc*.

In ILP, the search space of this characterization task is restricted by both syntactic form (mode declaration) of target theories (predicate *state*) and the available predicates, functions and constant symbols (called vocabulary). The vocabulary strongly effects if the correct hypothesis can be successfully found. For this case study, State pattern 3 in Fig. 4 shows a characteristic in which *dc* consists of two different elements such that there exists only one position where the two adjacent elements are different in *dc*. Progol cannot learn this characteristic since the vocabulary provided by the system specification is not strong enough. If we use two more predicates gap0 and gap1 that checks if a given queue has no gap and at most one gap, respectively, as part of the vocabulary, Progol can learn this characteristics.

The experiments demonstrate that ILP is useful to extract some characteristics of $R_{M_{\mathrm{ABP}}}$ that can be found in the six state patterns in Fig. 4. It is beneficial to systematically extract such characteristics to conjecture lemmas for invariant verification with ITP described in Sect. 4. Although there are some characteristics that cannot be extracted by ILP if we only use the vocabulary directly obtained from the system specification of M_{ABP}, however, those characteristics can be extracted if we use the extra vocabulary gap0 and gap1. Hence, one piece of our future work is to come up with a way to discover such extra vocabulary, which is likely to be related to Predicate Invention [8], a research domain in ILP.

5 Related work

ML has been applied to find lemmas in ACL2 [9] that can calculate the similarity between the current proof and other proofs in a given proof library containing many existing proofs that have already proved. The most similar proofs will stay

at same group, and then the system will pick the lemmas in the proved proofs to generate the lemmas for current proof.

Furthermore, ILP has been successfully used in system verification with an integrated framework of model checking and ILP [10] such that ILP is used as a complementation part of model checking. A system specification suited for model checking is considered as system requirements but it is fail in some desired properties. To modify the specification, they provide some new requirements expressed in a modeling language, e.g. LTL such that the new specification will enjoy the properties. The new requirements are the hypotheses obtained from an ILP learning task that is formulized by the components generated from a model checker, i.e. counterexamples (a.k.a. negative examples), witnesses (a.k.a. positive examples) and background knowledge consisting of the system specification and the properties.

6 Conclusion

We have described how to conjecture lemmas based on the the characteristics of $R_{M_{\mathrm{ABP}}}$, suggesting that such characteristics are useful for lemma conjecture. We have then reported on a case study demonstrating that ILP is useful to extract some characteristics of $R_{M_{\mathrm{ABP}}}$. We have also realized that ILP requires some extra vocabulary to extract some other characteristics of $R_{M_{\mathrm{ABP}}}$.

References

1. Ogata, K. and Futatsugi, K.: Proof Score Approach to Analysis of Electronic Commerce Protocols, IJSEKE, 20(2): 253-287, 2010.
2. Kroening, D. and Strichman, O.: Decision Procedures: An Algorithmic Point of View, Springer, 2008.
3. Ogata, K. and Futatsugi, K.: Proof Scores in the OTS/CafeOBJ Method, 6th FMOODS, LNCS 2884, Springer, pp.170-184 (2003).
4. Ogata, K. and Futatsugi, K.: Some Tips on Writing Proof Scores in the OTS/CafeOBJ Method, Algebra, Meaning, and Computation, LNCS 4060, Springer, pp.596-615 (2006).
5. Muggleton, S., Raedt, L.: Inductive logic programming: Theory and methods. The Journal of Logic Programming 19-20: pp. 629-679 (1994)
6. Muggleton, S.:Learning from positive data, 6th ILP, LNCS 1314, Springer, pp.358-376 (1996)
7. Zhang, M., Ogata, K., Nakamura, M.: Translation of state machine from equational theories into rewrite theories with tool support. IEICE Transactions 94-D (2011) 976-988
8. Stahl, I.: Predicate invention in ILP - an overview, ECML-93, LNCS 667, Springer, pp.313-322 (1993)
9. Heras, J., Komendantskaya, E., Johansson, M., Maclean, E.: Proof-pattern recognition and lemma discovery in ACL2. In: LPAR-19. LNCS 8312 (2013) 389-406
10. Alrajeh, D., Russo, A., Uchitel, S., Kramer, J.: Integrating model checking and inductive logic programming, 21st ILP, 45–60, (2011).