Applications of Finite Linear Temporal Logic to Communication Protocols

Arvydas GUŠČIA, Stanislovas NORGĖLA[1]

Department of Computer Science, Vilnius University
Naugarduko 24, LT-03225, Vilnius, Lithuania

Abstract. Using finite linear temporal logic as a specification language for the communication protocols problems, we formalize an alternative bit with the aggregate manager task. The problem is described by formulas in which the degree of modal operators does not exceed two. This allows the described formalization to apply practically. Theoretically, the number of possible questions about aggregate work can be more than one million. Search of answers for concerned questions is transformed into appropriate linear temporal logic formula set satisfiability. The described formalism allows to automate the search of an answer. A positive answer, then and only then, when the set under consideration is satisfiable. Moreover, from the obtained model we can find a finite sequence of actions to be executed in order to achieve the goal. In addition, alternative bit protocol problem was described using PDDL. Experiments were made using the LPG-TD planner.

Keywords: knowledge representation, artificial intelligence planning, communication protocols, finite linear temporal logic.

[1]Corresponding Authors: e-mail Arvydas.Guscia@mif.stud.vu.lt, stasys.norgela@mif.vu.lt

1. Introduction.
At first, temporal logic was being created in order to formalize time tense in natural language. It occurred later that temporal logic can be successfully used as a specification language to solve the problems in informatics. One of the first uses of temporal logic was a formalization and verification of concurrent and distributed systems. The circle of application expanded later. Now temporal logic is used in such fields as program specification, temporal databases, knowledge representation and natural language.

Temporal logics are classified according to whether time is assumed to have a linear or branching structure. CTL is the most used of branching time logics. The formulas describe properties of computation trees. The tree is formed by designating a state in a transition graph as the initial state and then unwinding the structure into an infinite tree with the designated state at the root [4].

The complexity of satisfiability problem for the logics CTL (computation tree logic), CTL*, LTL (linear temporal logic) is PSPACE or EXP. The complexity of model checking problem for CTL* and LTL formulas is PSPACE. Only the model checking problem for formulas of CTL can be solved in polynomial time. That is why the most widely used algorithms for the search of goal is model checking for the CTL formulas. We can find problems discribed in LTL but inexpressible by CTL formulas. We will show that *finite* LTL can be successful used for applications. Model checking involves establishing that a temporal formula is satisfied in the set of models representing the problem. Typically, solutions of problems of communication protocols used model checking (see, G.H.Holzman [8]). An alternative approach is to find at least one model. We consider this second approach. We consider the application of finite LTL to the problem of communication protocols. The applications of satisfiability of formulas of LTL over finite structures to the problems of communication protocols are not examined.
Suppose that a finite LTL formula has a model. In this case, we can find at least one model using the tableau calculus [1]. If we find a cycle in the proof-search tree, then we have a model or it is a failure. This allows the described formalization to apply practically. Describe a communication protocol task by giving him *planning problem* [9] form.

The planning problem task is to describe a finite sequence of actions (actions order is fixed) which have to be performed in order to achieve goals. If the problem can't be solved, i.e., the goal can't be achieved by a finite sequence of actions and an initial condition, the answer is failure. The required finite sequence of actions can be extracted from a finite model of the problem under consideration. Finite linear temporal logic operators $\Box$, $\Diamond$, $\circ$ are used to describe the problem. Description consists of *initial conditions*, *actions* and the *goal*. Initial conditions are described

by using classical propositional logic formulas. The goal is described in a formula $\Diamond F$ - "sometime F will be true". F - a classical propositional logic formula. Actions are described in formulas $\Box(C \rightarrow do(a))$ - "at any time moment, if condition C is true, action "a" can be completed". C - a classical propositional logic formula, "a" - action which creates data changes. Data changes are described in formulas $\Box(do(a) \rightarrow oH)$. H - a classical propositional logic formula which is true at the next time moment. It is possible to describe the restrictions too. A restriction is additional data which accelerates the search of goal.

The use of finite linear temporal logic as a specification language for the problems of communication protocols has several advantages. An important aspect of finite linear temporal logic is its simple model of time and actions. We have a natural representation of a world that changes over time. This logic is more expressive than classical propositional logic. Note that finite quantified LTL is undecidable [5]. The validity problem for first-order linear temporal logic over *finite* time structures is not recursively axiomatizable. The search of answer is reduced to model search. Our task is based on the "planning as satisfiability" approach. An answer corresponds to a model of the problem specification.

The alternative bit protocol defines the relation between sender and receiver. Time to time sender sends data packets via the channel. Packet data can be deformed or even the packet can be lost. Packet contains sending data and control data. The control data is defined by using a packet number which can be equal to 1 or 0. What's why these packets are called alternative bit packets. At any time moment sender has packets which have to be sent. When the packet is sent, sender activates timer. If at the particular time moment sender doesn't get data which confirms that receiver has got the packet, then sender repeats the sending process by resending the same packet. But this time control data is altered. Bit is replaced by the opposite bit. If a sent packet control data bit is equal to an acknowledgement packet control data bit, then packet is sent without deformation.

Sender and receiver can exchange information with the aggregate manager about an aggregate functionality via the other channel - administrative channel which can't deform packet data. If sender sends the same packet k times (k normally equals 3) and doesn't get any information that receiver has got the packet, then sender aborts the sending process and informs the aggregate manager about situation. The manager can order sender to repeat the sending process (no more than k times) or to send the same packet via the administrative channel or to abort the sending process and wait for other orders.

Manager can ask receiver why the acknowledgement packet isn't sent.

Receiver's answer to the manager can be: a) I get packets and I send acknowledgement packets, for some reason sender doesn't get my acknowledgement packets b) there are temporal technical problems c) I don't get data packets from sender, but I don't know why. If receiver doesn't get packets from sender for a long time or suspects that packet data is deformed, then receiver informs the aggregate manager about situation.

The knowledge of the aggregate work is formed using a basic concept which is described in propositional variables (acknowledgement packet waiting time has ended, sender is ready to send and etc). What is time in our task? Firstly, time is discrete. At one time moment (or step), one particular action is performed (sending of the packet, stopping the timer, informing the manager and etc). All possible actions are present in the action list. These actions are performed when particular conditions are fulfilled. Step by step, when actions are performed, we get into various states. A graph represents all the possible states. The aggregate operation may be seen as a way to go through the graph nodes, regarding to available information. The number of steps is equal to the number of time moments. There are many situations (possible ways).

Questions about the aggregate operation are made by using the propositional variables as key words. Since there are 20 propositional variables, theoretically the number of possible questions can't be less than 2 to the power of 20, i.e., more than one million. So it is essential to automate the knowledge of the aggregate work, to make the artificial intelligence system which analyses work and provides answers to the questions of interest. S.Cerrito and others described a tableau calculus using which it is possible to get answers to the formalized alternative bit protocol problem queries [1,2].

Verification of aggregate specifications using the classical first order predicate logic formulas is described in documents [12, 13, 14]. Aggregate states, transition and output operators, their properties are described by formulas. Formulas are quite sophisticated: there are functional symbols, a set of individual constants is split into subsets that have no common elements, formulas contains predicates in which number of seats depends on discrete and continuous power (it can be large enough) of set of coordinates. The main formalization results were obtained in eighties of the last century by group of Kaunas technology researchers headed by H. Pranevičius. The method has not been realized. That is, it has not been programmed.

2. Alternative bit protocol problem realization using finite linear temporal logic.

The language of finite linear temporal logic considered in this paper extends classical propositional logic by means of the unary modal operators $\Box$ (always), $\Diamond$ (eventually), $\circ$ (next). $\Box A$ means that A is true now and will always be true, $\Diamond A$ that A is either true now or sometime in the future and $\circ A$ that A holds in the next state.

A temporal structure is a finite sequence of elements called states or time points. Interpretation M consists of states $s-0$, s_1, ..., s_n representation from N to set of literals. For every s_i, propositional variables subset $\nu(i)$ is assigned. The satisfiability relation $M_i \models F$ is inductively defined as follows:

- $M_i \models p$ if $p \in \nu(i)$, p is a propositional letter.
- $M_i \models A$ if A is not satisfiable.
- $M_i \models A \wedge B$ if $M_i \models A$ and $M_i \models B$.
- $M_i \models A \vee B$ if either $M_i \models A$ or $M_i \models B$.
- $M_i \models A \rightarrow B$ if either A is not satisfiable or $M_i \models B$.
- $M_i \models \Box A$ if for all $j \geq i$, $M_i \models A$.
- $M_i \models \Diamond A$ if there exists $j \geq i$ such that $M_i \models A$.
- $M_i \models \circ A$ if $M_{i+1} \models A$ and $i \neq n$; if $i = n$, then failure.

The *degree* of modal operators in a propositional formula of classical logic is 0. If the degree of modal operators in a formula A is n, then the *degree* of time operators in the formulas $\Box A$, $\Diamond A$, $\circ A$ is $n+1$.

The protocols are modeled as state-transition systems. Such structures are called Kripke structures.

Propositional variables describe statements which are elementary (can't be separated to simpler statements) and which formalizes the alternative bit protocol problem. Examples of propositional variables (the graph representing sender states has 20 nodes):

- *sending_bit* − true if and only if sender is working with the data packet which control bit is equal to 1.
- *received* − true if and only if receiver received the data packet.
- *timer* − true if and only if timer is activated.

States. We describe the ordering of events in time without introducing time explicitly. Using the propositional variables and Boolean connectives, we can build up formulas describing properties of states. We represent only some formulas describing the state of communication protocols:

- s_0: $sending\_bit, \neg ack\_received, \neg sent, \neg ready\_to\_send, \neg timeout, \neg timer, ack\_good$
  Access is gained when sender starts the sending process for the first time or when the correct acknowledgement packet is received or the packet is sent via the administrative channel. A number of sent packets is incremented by one. In this s_0 state the new packet is made (action $begin\_sending$). Moving to state s_1.

- s_3: $sending\_bit, \neg ack\_received, \neg sent, \neg ready\_to\_send, \neg timeout, timer, \neg ack\_good$
  Access is gained when timer is activated. Waiting for the acknowledgement packet. The acknowledgement packet can be received (then moving to state S9) or acknowledgement packet waiting time can end (then action $is\_timeout$ is performed, moving to state S4).

Actions. The transition relation can be expressed as a temporal logic formula. Actions which can be performed at any time moment if only conditions on the left implication side are fulfilled are described here. Examples of actions:

- $\Box(do\_send \rightarrow \circ(sent \wedge \neg ready\_to\_send))$
  Sending the packet via the channel.

- $\Box(is\_timeout \rightarrow \circ(timeout \wedge \neg timer \wedge \neg timer\_off))$
  Ending acknowledgement packet waiting time.

Channel. When sender is communicating with receiver via the channel, channel can be in many states. Possible data deformation or data loss situations are described using actions. Some examples: States of channel:

- $\neg sent, \neg ack\_sent$.
  In this state there is no sending via the channel.

- $sent, \neg(i = k), \neg(y = l)$.
  In this state the packet is sent via the channel.

Actions of channel:

- $\Box(((do\_receive \wedge sending\_bit) \rightarrow \circ(\neg sent \wedge receivedwedgereceiving\_bit)) \vee$
  $\vee((do\_receive \wedge \neg sending\_bit) \rightarrow \circ(\neg sent \wedge received \wedge \neg receiving\_bit)))$
  If the packet is sent, timer is activated and there is no packet loss $((i = k))$ and there is no packet deformation $((y = l))$, action $do\_receive$ is performed. After performance of action, at the next time moment, a control bit of received packet is equal to a control bit of sent packet, in other words, the packet isn't deformed. The packet is received by receiver ($received$) and the new packet isn't sent yet ($sent$).

We describe the states and the actions for receiver in a similar manner. The complete list of variables and actions can be obtained via home page of S.Norgėla: `http://www.mif.vu.lt/katedros/cs/`

Manager and administrative channel can be in many states because of performed actions. Logic of knowledge (epistemic logic) is used to formalize manager's operation. Logic of knowledge has two operators [], <>, what have following meaning: $[F]$ - *we know that* F, $< F >$ - *is possible, that we know, that* F. The operators satisfy equivalence $[F] \equiv \neg < \neg F >$.

3. Experiment

To accomplish an experiment, alternative bit protocol problem was described using PDDL [6]. Many actions such as "do_send" were transformed into actions of PDDL easily without needing more attention.

```
(:action do_send
:precondition(and(not(sent))(ready_to_send))
:effect(and(sent)(not(ready_to_send)))
.)
```

But, for example, action "do_select" which has "OR" operator in its effect field was transformed into more than one action. This is done to abolish "OR" operator since PDDL syntax does not allow disjunctive operators to appear in effect field. By abolishing the operator, we have actions with same preconditions but different effects so planner decides which action to use. Moreover, action "do_receive" is transformed into more than one action too. This is needed to avoid not only "OR" operator but also conditional effects. So if action has "OR" operator in its effect field or is using conditional effects, it is transformed into more than one action of PDDL. In addition, to implement packet deformation, packet loss and to count packets, fluents are needed. For example, functions "i" and "k" are implemented to achieve packet loss. When packet is sent via the channel, "i" value is incremented by one. When "i" value is equal to "k" value which is defined in problem file, init field, the packet is lost and "i" starts to count from zero. So packet loss, packet deformation frequencies can be defined in problem file to obtain different plans.

```
(:action do_receive_1
:precondition(and(timer)(sent)(sending_bit)(<(i)(k))(<(j)(1)))
:effect(and(received)(receiving_bit)(increase(i) 1)(increase(j)
1)(not(sent)))
)
```

```
(:action do_receive_2
:precondition(and(timer)(sent)(not(sending_bit))(<(i)(k))(<(j)(1)))
:effect(and(received)(not(receiving_bit))(increase(i) 1)(increase(j)
1)(not(sent)))
)
```

Basically, to make this experiment work, planners which support fluents and negative preconditions are needed. There are not so many planners which support these. LPG-TD planner [7] is chosen because it runs on Microsoft Windows. Experiment is performed by trying five times to send particular number of packets via the channel using different packet loss and packet deformation frequency. It is also taken into account that sender informs aggregate manager about failure after 3 attempts to send the same packet. Let's study experiment results (see table below). For example, if five times 1 packet is sent via the channel with packet loss and packet deformation frequency 1/2, then the average plan length needed to send 1 packet is 38 actions and average time needed to find the plan is 1.42 seconds. The table shows that if packet deformation and loss frequencies are higher, then bigger plans will be needed and, of course, plan length depends on number of packets needed to be sent. Results are the same as we had expected.

pakets/deformation, loss frequencies	1/2, 1/2	1/2, 1/4	1/4, 1/2	1/4, 1/4	1/8, 1/8
1	38a, 1.42s	28a, 2.10s	29a, 2.2s	8a, 0.05s	8a, 0.04s
2	53a, 18.18s	52a, 23s	52a, 20s	27a, 0.88s	15a, 0.08s
3	75a, 52.2s	75a, 82.3s	77a, 80.2s	60a, 0.5s	21a, 0.1s

Concluding remarks

The knowledge which represents relation between sender and receiver is described using finite linear temporal logic formulas. The core of computer-assisted answering system to questions about the alternative bit protocol aggregate operation at various time moments has been created. Questions are formulated in form of $\Diamond A$. Is there any possible situation that A is true? For example, can we access the situation where packet acknowledgement waiting time has ended and the acknowledgement packet isn't received? Question is described by formula $\Diamond(\neg timer \wedge \neg ack\_received)$. Software [3,11] is developed for this type of information processing. This software is available on the internet. Next step is to describe restrictions and to create proof-search tactic. To make the search more effective, the knowledge about impossible events and situations which are not needed to be considered must be added to the system.

References

[1] S.Cerrito, M.Cialdea Mayer, S.Praud. A tableau calculus for first order linear temporal logic over bound time structures, *Technical Report LRI*, N. 1207, 1999.

[2] S.Cerrito, M.Cialdea Mayer. Using linear temporal logic to model and solve planning problems, In: *Proceedings of the 8th International Conference on Artificial Intelligence: Methodology, Systems, Applications*, 141-152, 1998.

[3] M.Cialdea Mayer, C.Limongelli, A.Orlandini, V.Poggioni. Linear temporal logic as an executable semantics for planning languages, *Journal Logic, Language and Information*, 16, 63-89, 2007.

[4] E.M.Clarke, O.Grumberg, D.Peled. Model Checking, MIT Press, 2000.

[5] S.Cerrito, M.Cialdea Mayer, S.Praud, First Order Linear Temporal Logic over Finite Time Structures, *LNAI*, 1705, 62-76, 1999.

[6] M. Fox, D.Long. PDDL 2.1: an extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, 20, 61-124, 2003.

[7] A. Gerevini, A.Saetti, I. Serina. LPG-TD: a Fully Automated Planner for PDDL2.2 Domains (short paper), in *14th Int. Conference on Automated Planning and Scheduling (ICAPS-04), booklet of the system demo section*, Whistler, Canada, 2004.

[8] G.J.Holzman. The SPIN Model Checker, Addison-Wesley, p. 596, 2003.

[9] H.Kautz, B.Selman. Planning as satisfiability, In: *Proceedings of the 10th European Conf. in Artificial Intelligence*,Vienna, Austria, 360-363, 1992.

[10] H.Kautz, B.Selman. Unifying SAT-based and graph based planning, In: *Proceedings of the 16th International Joint Conference of Artificial Intelligence*, Stockholm, Sweden, 318- 325, 1999.

[11] H.Kautz, D.McAllester, B.Selman. Encoding plans in propositional logic, In: *Proceedings of the 4th International Conference on Knowledge Representation and Reasoning*, 374- 385, 1996.

[12] H.Pranevičius. Analysis and formalizations of complex systems, Kauno technologijos universitetas, 240 p, 2008. (in Lithuanian).

[13] H.Pranevicius, R.Ceponyte. Application of logic programming based for validation of computers network protocols aggregate specifications, *Automatic and computing technique*, 2, 22-27, 1992.

[14] H.Pranevicius, R.Miseviciene. Transformation of aggregate specifications to the predicate logic models, *The international workshop on harbour, maritime and multimodal logistics modelling & simulation.* Riga, 378-384, 2003.