Temporal logic with predicate λ-abstraction.

Alexei Lisitsa and Igor Potapov*
Department of Computer Science
University of Liverpool, U.K.
{alexei,igor}@csc.liv.ac.uk

Abstract

A predicate linear temporal logic $LTL_{\lambda=}$ without quantifiers but with predicate λ-abstraction mechanism and equality is considered. The models of $LTL_{\lambda=}$ can be naturally seen as the systems of pebbles (flexible constants) moving over the elements of some (possibly infinite) domain. This allows to use $LTL_{\lambda=}$ for the specification of dynamic systems using some resources, such as processes using memory locations, mobile agents occupying some sites, etc. On the other hand we show that $LTL_{\lambda=}$ is not recursively axiomatizable and, therefore, fully automated verification of $LTL_{\lambda=}$ specifications via validity checking is not, in general, possible. The result is based on computational universality of the above abstract computational model of pebble systems, which is of independent interest due to the range of possible interpretations of such systems.

1 Introduction

In this paper we consider a predicate linear temporal logic $LTL_{\lambda=}$ without quantifiers but with predicate λ-abstraction mechanism. The idea of predicate abstraction[1] goes back to M.Fitting who has proposed this as the general technique for obtaining the modal logics, which are, in a sense, intermediate between propositional and first-order. He suggested to extend a modal propositional logic L by adding relation symbols, flexible constants and the operator of predicate abstraction, but no quantifiers. The abstraction is used as a scoping mechanism. Simple example of what the abstraction can be used for is given by the following two formulae: $\Diamond\langle\lambda x.P(x)\rangle(c)$ and $\langle\lambda x.\Diamond P(x)\rangle(c)$. The first one says that P holds of what c designates in alternative world, while the second one says that at an alternative world P holds of what c designates in a current world.

Such an extension $L_{\lambda(=)}$ (both with and without equality) can be alternatively seen as very restricted fragment of corresponding first-order variant QL of L. It is proved in [2] that such extension when applied to $S5$ leads to the undecidable logic $S5_{\lambda=}$ but for many other classical modal logics L their extensions $L_{\lambda(=)}$ are still decidable.

We apply such an extension to the classical propositional linear time logic. The models of $LTL_{\lambda=}$ can be naturally seen as the systems of pebbles (flexible constants) moving over the elements of some (possibly infinite) domain. This provides an abstract view on dynamic systems using some resources, such as processes using memory locations, mobile agents occupying some sites, etc. Thus, despite being very restricted extension of propositional temporal logic, $LTL_{\lambda=}$ is suitable for specification of such systems. However, we show, as a main result of this paper, that $LTL_{\lambda=}$ is not only undecidable, but even is not recursively axiomatizable. It follows that automatic verification of $LTL_{\lambda=}$ specifications via validity checking is not, in general, possible. On the other hand, the result is based on the computational universality of the simple and abstract computational model of pebble systems, which may be of independent interest.

The paper is organised as follows. In the next section we present a syntax and semantics of $LTL_{\lambda=}$. In Section 3 we demonstrate the expressive power of $LTL_{\lambda=}$ by giving a range of examples of properties expressible in $LTL_{\lambda=}$ and discuss possible applications of $LTL_{\lambda=}$ for specifications of protocols. In Section 4 we present main ideas of modelling counter machines by pebble systems. In Section 5 we use these ideas for modelling Minsky machines computations in $LTL_{\lambda=}$ and prove the main result. In Section 6 we extend the main result to the case of logic with only future time modalities. In Section 7 it is shown that LTL_{λ} without equality but with countably many unary predicates is also not recursively axiomatizable. We conclude the paper by Section 8.

*Work partially supported by NAL/00684/G NF grant.

[1]The term "predicate abstraction" is already been used in the literature on verification in a quite a different sense, see for example [7]. To avoid any misunderstanding we will use the term "predicate λ-abstraction", or just "λ-abstraction".

Proceedings of the 12th International Symposium on Temporal Representation and Reasoning (TIME'05)
1530-1311/05 $20.00 © 2005 **IEEE**

2 Syntax and Semantics

The content of this section is an adaptation of the corresponding section of [2] to the case of temporal logic.

Let $\mathcal{V} = \{x, y, x_1, \ldots\}$ is an alphabet of *variables* and $\mathcal{C} = \{a, b, c, c_1, \ldots\}$ is an alphabet of *constant symbols*. For each n let $\mathcal{R}^n = \{P, R, R_1, \ldots\}$ is an alphabet of n-ary relational symbols. We refer to the tuple $\mathcal{L} = \langle \mathcal{V}, \mathcal{C}, \mathcal{R} \rangle$ as to the alphabet.

One may include or not an equality $=$ in the alphabet of binary relations symbols. In this paper we consider only the case with equality. A *term* is a constant symbol or a variable.

Definition 1 The set of $LTL_{\lambda=}$ -formulas (in the alphabet $\mathcal{L}$) and their free variables, are defined as follows.

1. If R is an n-ary relation symbol and $x_1, x_2, \ldots$ are variables, then $R(x_1, x_2, \ldots x_n)$ is a formula with $x_1, x_2, \ldots$ as its free variable occurences.
2. if φ is a formula, then $\neg\varphi$, $\Box\varphi$, $\Diamond\varphi$, $\bigcirc\varphi$, $\blacksquare\varphi$, $\blacklozenge\varphi$, $\odot\varphi$ are formulas. Free variable occurences are those of φ
3. If φ and ψ are formulas, then $(\varphi \wedge \psi)$, $(\varphi \vee \psi)$ and $(\varphi \rightarrow \psi)$ are formulas. Free variable occurences are those of φ together with those of ψ.
4. If φ is a formula, x is a variable, and t is a term, then $\langle \lambda x.\varphi \rangle(t)$ is a formula. Free variable occurences are those of φ, except for occurences of x, together with t if it is a variable.

A formula φ is called a *sentence* iff it does not have free variable occurences.

Formulae of $LTL_{\lambda=}$ are interpreted in first-order temporal models with the time flow isomorphic to the structure $\langle \mathbb{N}, \leq, succ \rangle$, where $\mathbb{N}$ is the set of natural numbers, $\leq$ is usual order relation on $\mathbb{N}$, and $succ$ is a successor operation on $\mathbb{N}$.

Definition 2 A *model* is a structure $\mathfrak{M} = \langle D, \mathcal{I} \rangle$, where:

1. D is a non-empty set, the domain;
2. $\mathcal{I}$ is an interpretation mapping that assigns:
   - to each constant symbol some function from $\mathbb{N}$ to D;
   - to each n-ary relation symbol some function from $\mathbb{N}$ to 2^{D^n} (the power set of D^n)
3. $\mathcal{I}(=)$ is the constant function assigning the equality relation on D to every moment of time from $\mathbb{N}$.

First-order (non-temporal) structures corresponding to each point of time will be denoted $\mathfrak{M}_n = \langle D, \mathcal{I}(n) \rangle$. Intuitively, the interpretations of $LTL_{\lambda=}$ -formulae are sequences of first-order structures, or *states* of $\mathfrak{M}$, such as $\mathfrak{M}_0, \mathfrak{M}_1, \ldots, \mathfrak{M}_n \ldots$.

An *assignment* in D is a function $\mathfrak{a}$ from the set $\mathcal{V}$ of individual variables to D. Thus we assume that (individual) variables of $LTL_{\lambda=}$ are *rigid*, that is assignments do not depend on the state in which variables are evaluated. In contrast, as follows from Definition 2 the constants are assumed to be non-rigid (flexible), that is their interpretations depend on moments of time (states). For a constant c we call element $I(c)(n)$ of D also a *designation* of c at the moment n. The assignment $\mathfrak{a}$ is extended to the assignment $(\mathfrak{a} * \mathcal{I}) : (\mathcal{C} \cup \mathcal{V}) \rightarrow D$ of all terms in the usual way:

1. For a variable x, $(\mathfrak{a} * \mathcal{I})(x, n) = \mathfrak{a}(x)$;
2. For a constant c, $(\mathfrak{a} * \mathcal{I})(c, n) = \mathcal{I}(c)(n)$.

If P is a predicate symbol then $P^{\mathcal{I}(n)}$ (or simply P^n if $\mathcal{I}$ is understood) is the interpretation of P in the state $\mathfrak{M}_n$.

Definition 3 The *truth-relation* $\mathfrak{M}_n \models^{\mathfrak{a}} \varphi$ (or simply $n \models^{\mathfrak{a}} \varphi$, if $\mathfrak{M}$ is understood) in the structure $\mathfrak{M}$ for the assignment $\mathfrak{a}$ is defined inductively in the usual way under the following semantics of temporal operators:

$n \models^{\mathfrak{a}} \bigcirc\varphi$ iff $n + 1 \models^{\mathfrak{a}} \varphi$;

$n \models^{\mathfrak{a}} \Diamond\varphi$ iff there is $m \geq n$ such that $m \models^{\mathfrak{a}} \varphi$;

$n \models^{\mathfrak{a}} \Box\varphi$ iff $m \models^{\mathfrak{a}} \varphi$ for all $m \geq n$;

$n \models^{\mathfrak{a}} \odot\varphi$ iff $n - 1 \models^{\mathfrak{a}} \varphi$;

$n \models^{\mathfrak{a}} \blacklozenge\varphi$ iff there is $0 \leq m \leq n$ such that $m \models^{\mathfrak{a}} \varphi$;

$n \models^{\mathfrak{a}} \blacksquare\varphi$ iff $m \models^{\mathfrak{a}} \varphi$ for all $0 \leq m \leq n$.

For the case of abstraction we have:

$n \models^{\mathfrak{a}} \langle \lambda x.\varphi \rangle(t)$ iff $n \models^{\mathfrak{a}'} \varphi$, where $\mathfrak{a}'$ coincide with $\mathfrak{a}$ on all variables except x and $\mathfrak{a}'(x) = (\mathfrak{a} * \mathcal{I})(t, n)$.

A formula φ is said to be *satisfiable* if there is a first-order structure $\mathfrak{M}$ and an assignment $\mathfrak{a}$ such that $\mathfrak{M}_0 \models^{\mathfrak{a}} \varphi$. If $\mathfrak{M}_0 \models^{\mathfrak{a}} \varphi$ for every structure $\mathfrak{M}$ and for all assignments $\mathfrak{a}$ then φ is said to be *valid*. Note that formulae here are interpreted in the initial state $\mathfrak{M}_0$.

We conclude this section by introducing an useful notation. Given a model $\mathfrak{M} = \langle D, \mathcal{I} \rangle$ and constant c denote by V_c^n the set of elements of D visited by c up to the moment n, that is $V_c^n = \{\mathcal{I}(m)(c) \mid 0 \leq m \leq n\}$. Then $V_c = \cup_{i \in \mathbb{N}} V_c^i$ is the set of elements of D visited by c in the model $\mathfrak{M}$.

3 Properties expressible in $LTL_{\lambda=}$

Despite being very restricted fragment of the first-order temporal logic the logic $LTL_{\lambda=}$
can express many non-trivial properties of its models. Because of the interpretation given to flexible constants (by a function from $\mathbb{N}$ to D) they can be though of as the *pebbles* moving over elements of the domain as the time goes by. Then, even in the absence of any other predicate symbols except equality, one can express dynamic properties of the system of pebbles evolving in time:

- $New(d) \Leftrightarrow \langle\lambda x.\bigcirc\Box\langle\lambda y.y \neq x\rangle(d)\rangle(d)$. The constant d has different designations (the pebble d occupies different positions) at the current and at any future moment of time.
- $Same(a,b) \Leftrightarrow \langle\lambda x.\langle\lambda y.x = y\rangle(a)\rangle(b)$. The constants a and b have the same designation now. (The pebbles are on the same place now).
- $SameInPast(a,d) \Leftrightarrow \langle\lambda x.\blacklozenge\langle\lambda y.y = x\rangle(d)\rangle(a)$. The constant a has the same designation as d had in the past.
- $NoChange(c) \Leftrightarrow \langle\lambda x.\bigcirc\langle\lambda y.x = y\rangle(c)\rangle(c)$. The constant c has the same designation at the current and next moments of time.
- $AlwaysReturn(a) \Leftrightarrow \Box\langle\lambda x.\langle\bigcirc\Diamond\langle\lambda y.x = y\rangle(a)\rangle(a)$. The pebble a always return to the place it occupies at any given moment of time.

Next, we present two examples of more complex formulae, which will play special role later on:

- $NextNew1(a,d,c) \Leftrightarrow \langle\lambda w.\langle\lambda x.\blacklozenge(\langle\lambda y.y = x\rangle(d) \wedge \bigcirc\langle\lambda v.(v = w)\rangle(d))\rangle(a)\rangle(c) \wedge \langle\lambda z.\bigcirc\langle\lambda t.(t = z)\rangle(a))\rangle(c)$
- $NextNew2(a,d) \Leftrightarrow \bigcirc\langle\lambda w.\odot\langle\lambda x.\blacklozenge(\langle\lambda y.y = x\rangle(d) \wedge \bigcirc\langle\lambda v.(v = w)\rangle(d))\rangle(a)\rangle(a)$

Both formulae express the same fact about the behaviour of pebbles a and d: the pebble a moves in the next moment of time to the position to which the pebble d has moved from the position which a is occupying now. In other words a moves in the same way as d did from the same position. The difference between two formulae is that NextNew1 does not use "last" operator $\odot$ but at the expense of an additional flexible constant.

In the above formulae no predicate symbols except equality was used. One example of the formula with additional binary predicate E is

$$\Box(\langle\lambda x.\langle\lambda y.(E(x,y) \leftrightarrow (\Box E(x,y) \wedge \blacksquare E(x,y)))\rangle(c_1)\rangle(c_2))$$

This formula says that predicate E, restricted to the pairs of elements of domain, visited by first and second constant, respectively, is rigid. But E may well have different interpretations at different moments of times on all other pairs of elements. This example illustrates also the *pebble locality* of properties expressible by $LTL_{\lambda=}$ formulae. Pebble locality of the property means the property depends only on the interpretations of predicate symbols on the elements of the domains ever visited by pebbles. It is easy to show that only such properties are expressible in $LTL_{\lambda=}$.

3.1 Pebble systems and agents using resources

The above metaphor of pebble system may also be seen as the very abstract model of computational processes (agents) using some resources. In such a model a pebble, or, indeed non-rigid constant c may be thought of as an computational process and elements of the domain as the abstract resources. Then, if at some moment of time a designation of c is an element x of the domain, one may understand it as *"c uses the resource x"*. To model the situation with processes, or agents using several resources at the same time, one may associate with an agent a *set* of flexible constants (pebbles). Another natural reading of the above situation may be *"the mobile agent c resides at the host x"*. Taking this point, the formulae of $LTL_{\lambda=}$ can be used to specify protocols, policies or requirements for agents operating within the common pool of resources. Pointing out this possibility, we restrict ourselves in this paper with the simple example of communicating protocol for mobile agents.

3.2 An example of communication protocol for Mobile Agents

Let us suppose the following scenario where a group of communicating mobile agents explore some hosts and transmit messages to each other. Because mobile agents can move autonomously from host to host, they cannot reliably know the location of their communication peer. Therefore, a practical communication protocol somehow must keep track of agent locations, allowing each agent to send messages to its peers without knowing where they physically reside.

There are many mobile agent tracking protocols, that use a forwarding pointers mechanism[1]. It means that each host on mobile agents migration path keeps a forwarding pointer to the next host on the path. The classical primitive for such protocol is based on *knowledge of each sender the target agent's home.* So messages are sent to the agent's home and forwarded to the target object along the forwarding pointers. Interesting alternative is a primitive *find a*

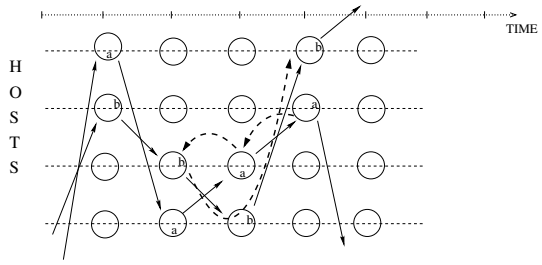

Figure 1. A mobile agent tracking protocols: the operation of sending message from agent a to agent b.

host, which was visited both by sender and receiver[2]. Using this primitive the messages are again forwarded to the target object along the forwarding pointers but from the host where the mobile agents migration paths intersect (see Figure 1).

We can specify the use of this primitive in $LTL_{\lambda=}$ as follows. For simplicity we assume that receiver always either do not move or move to the new host (never revisiting the hosts it already visited).

Let flexible constants s and r denote communicating mobile agents (sender and receiver, respectively) and m denotes the message. Then $LTL_{\lambda=}$ -formula

$$Same(s,m) \ \wedge$$

$$\bigcirc(\lambda z.\langle\blacklozenge\langle\lambda x.\blacklozenge\langle\lambda y.x = y \wedge y = z\rangle(r)\rangle(s)\rangle(m)) \ \wedge$$

$$\bigcirc\,\Box NextNew2(m,r)$$

describes the above protocol: at some moment of time message m is on the same hosts as s, then it moves along the path of r, starting from a host which both s and r have visited (and r done it no later than s).

It should be clear now, that $LTL_{\lambda=}$ is expressive enough to formulate also the correctness conditions for such protocols, like *once the message sent, it will be delivered eventually to a receiver*. Of course, one needs to specify some extra conditions which would guarantee correctness: *receiver must stop and wait in order to receive a message* (otherwise the message may always be behind the receiver). Or, one may specify the different speed of messages and agents, which would guarantee delivery even to the agents "on move". One way of doing this in $LTL_{\lambda=}$ is to specify that messages can move to the new host every round (discrete moment of time), while mobile agents can move only every *second* round. Thus, the proof of correctness of the above protocol may be reduced to the validity checking for some $LTL_{\lambda=}$ -formulae. We don't pursue a goal of automatic verification of protocols via validity checking (theorem proving) for $LTL_{\lambda=}$ -formulae in this paper, but rather demonstrate a related negative result on $LTL_{\lambda=}$ itself: it is highly undecidable and, therefore, fully automated verification based on validity checking of $LTL_{\lambda=}$ -formulae is not possible.

[2]We don't consider the issue of implementation of such a primitive and note that this can be done in various ways.

4 Minsky machines and their modelling by pebbled sets

In this section we use a well known model of Minsky machine to show universality of pebbled sets model. Informally speaking, Minsky machine is a two counter machine that can increment and decrement counters by one and test them for zero. It is known that Minsky machines represents a universal model of computations [6]. Being of very simple structure the Minsky machine are very useful for proving undecidability results (see for example [4, 5]).

It is convenient to represent a counter machine as a simple imperative program $\mathcal{M}$ consisting of a sequence of instructions labelled by natural numbers from 1 to some L. Any instruction is one of the following forms:

- l: ADD 1 to S_k; GOTO l';
- l: IF $S_k \neq 0$ THEN SUBTRACT 1 FROM S_k; GOTO l' ELSE GOTO l'';
- l: STOP.

where $k \in \{1,2\}$ and $l, l', l'' \in \{1,\ldots,L\}$.

The machine $\mathcal{M}$ starts executing with some initial non-negative integer values in counters S_1 and S_2 and the control at instruction labelled 1. We assume the semantics of all above instructions and of entire program is clear. Without loss of generality one can suppose that every machine contains exactly one instruction of the form l: STOP which is the last one ($l = L$). It should be clear that the execution process (run) is deterministic and has no failure. Any such process is either finished by the execution of L: STOP instruction or lasts forever.

As a consequence of the universality of such computational model the halting problem for Minsky machines is undecidable:

Theorem 1 ([6]) *It is undecidable whether a two-counter Minsky machine halts when both counters initially contain* 0.

We will use the following consequence of Theorem 1.

Corollary 1 *The set of all Minsky machines which begin with both counters containing* 0 *and do not halt is not recursively enumerable.*

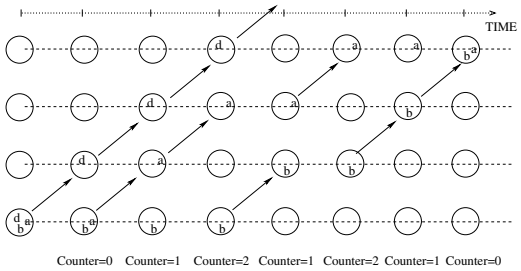

Figure 2. Counters encoding

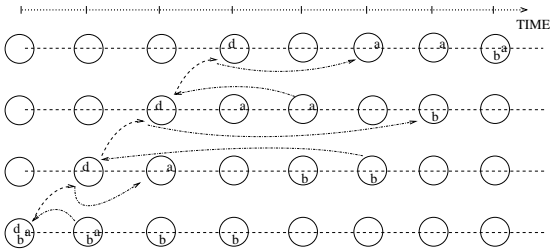

Figure 3. Pebbels a and b moves in the same way as d did from the same position.

Given any machine $\mathcal{M}$ (with initial values for the two counters) let us define its run $r^{\mathcal{M}}$ as a sequence of triples, or states of $r^{\mathcal{M}}$:

$$(l_1, p_1^0, p_2^0), (l_2, p_1^1, p_2^1), \ldots (l_{j+1}, p_1^j, p_2^j), \ldots$$

where l_j is the label of the instruction to be executed at jth step of computation, p_1^j and p_2^j are the nonnegative integers within the first and the second counters, respectively, after completion of jth step of computation. Depending on whether $\mathcal{M}$ stops or not $r^{\mathcal{M}}$ can be finite or infinite.

Henceforth we will consider only the computations of the Minsky machines started with both counters containing 0. Thus we always put $p_1^0 = 0$, $p_2^0 = 0$ and $l_1 = 1$.

4.1 Modelling Minsky machines by systems of pebbles

We will show our main result on non-r.e. axiomatizability of $LTL_{\lambda=}$ by modelling the computations of two counter Minsky machines in that logic. In fact we are going to model such machines by pebble systems and then just express required properties of such systems in the logic. In this subsection we explain the main idea of modelling, leaving all details of $LTL_{\lambda=}$ representation to the next section.

Given a pebble system with tree pebbles a, b, d. We denote the set of all elements that was visited by a pebble a (b) until the moment of time i by V_a^i (V_b^i). One may use then two pebbles, say a and b to model the counter's values as follows. We represent the counter's value at the moment i as the *cardinality* of the set $V_a^i \cap \overline{V_b^i}$. Increasing one of the sets of elements visited by a, or by b one may increase or decrease the counter value. Our modelling will ensure that $\forall i. V_b^i \subseteq V_a^i$. That means the counter's value at the moment i is in fact $card(V_a^i) - card(V_b^i)$ (see Figure 2).

Due to peculiarities of logical representation we confine the range of the elements visited by both pebbles to the set of elements visited by another special pebble d. We require d moves every time to the new element and we have $V_b^i \subseteq V_a^i \subseteq V_d^i$.

Let us show how to increase and decrease the cardinality of the set $V_a^i \cap \overline{V_b^i}$. Since the pebble d generates unique sequence of elements from the domain as the time goes by we can use this unique sequence for increasing of the cardinality of V_a^i or V_b^i by one.

Let pebble a (b) is on an element x of the domain. Since a is moving strictly along the path of d, the pebble d has visited the element x and moved to another element y. So in order to increase the cardinality of V_a^i (V_b^i) by one we need to move the pebble a (b) to the element y. In other words a (b) moves in the same way as d did from the same position.

We can increase (decrease) the value of counter by one or in other words increase (decrease) the cardinality of the set $C = V_a^i \cap \overline{V_b^i}$ by one if we increase the cardinality of the set V_a^i (V_b^i) by one according to the above procedure. Since there is a strict order of unique elements that we use for moving pebbles along the path of d we can easily test the emptiness of the counter or emptiness of the set C by checking if the pebble a and the pebble b are on the same element (see Figure 3).

5 Modelling of Minsky machines in $LTL_{\lambda=}$

In the translation of Minsky machines into formulae of $LTL_{\lambda=}$ we will use the formulae defined in the Section 3 and counters encoding method from the previous section.

5.1 Translation

Given a Minsky machine $\mathcal{M}$ defined by the sequence of instructions $c_1, \ldots c_L$ we define $LTL_{\lambda=}$ temporal formula $\chi^{\mathcal{M}}$ as follows.

Let $e_1, \ldots, e_L$ be flexible constants corresponding to instructions $c_1, \ldots, c_L$. Let e_0 and f be two additional constants. The intention is to model the fact "c_l is executed at the moment t" by coincidence of designations of e_l and f at the moment t. We denote by Q_l the formula expressing this fact: $\langle \lambda x. \langle \lambda y. x = y \rangle (e_l) \rangle (f)$. Since we assume c_L is the STOP instruction we will denote Q_L alternatively as Q_{stop}. Further we have five more constants for modelling counters: d, a_1, b_1, a_2, b_2.

Then, for every instruction c_l, except l : STOP, we define its translation $\chi(c_l)$ as follows:

A. An instruction of the form

l : ADD 1 to S_k; GOTO l';

is translated into the conjunction of the following formulae:

A1. $\Box(Q_l \to \text{NextNew}(a_k, d))$

A2. $\Box(Q_l \to \text{NoChange}(b_k))$

A3. $\Box(Q_l \to \text{NoChange}(a_{3-k}))$

A4. $\Box(Q_l \to \text{NoChange}(b_{3-k}))$

A5. $\Box(Q_l \to \bigcirc Q_{l'})$

Formulae A1–A4 ensure that in every temporal model $\mathfrak{M}$ for them, once we have $Q_l^n =$ **true** at a moment n, at the next moment the interpretation of the flexible constant a_k changes to a new value, while b_k, a_{3-k} and b_{3-k} keep their interpretation intact. The formula A5 describes switching truth values of propositions Q_i ($i \in \{1, \dots L\}$) and the aim here is to model the transition from the instruction which is executed to the next one.

B. An instruction of the form

l : IF $S_k \neq 0$ THEN SUBTRACT 1 FROM S_k; GOTO l' ELSE GOTO l'';

is translated into the conjunction of the following formulae:

B1. $\Box(Q_l \wedge \neg\text{Same}(a_k, b_k) \to \text{NoChange}(a_k))$

B2. $\Box(Q_l \wedge \neg\text{Same}(a_k, b_k) \to \text{NextNew}(b_k, d))$

B3. $\Box(Q_l \wedge \neg\text{Same}(a_k, b_k) \to \text{NoChange}(a_{3-k}))$

B4. $\Box(Q_l \wedge \neg\text{Same}(a_k, b_k) \to \text{NoChange}(a_{3-k}))$

B5. $\Box(Q_l \wedge \text{Same}(a_k, b_k) \to \text{NoChange}(a_k) \wedge \text{NoChange}(b_k) \wedge \text{NoChange}(a_{3-k}) \wedge \text{NoChange}(b_{3-k}))$

B6. $\Box(Q_l \wedge \neg Same(a_k, b_k) \to \bigcirc Q_{l'})$

B7. $\Box(Q_l \wedge Same(a_k, b_k) \to \bigcirc Q_{l''})$

Formulae B1–B4 ensure that, in every temporal model for them, once we have Q_l and the interpretations of a_k and b_k are different (meaning "k-st counter has non-zero value") the interpretation of b_k changes in the next moment of time, while interpretations of a_k, a_{3-k} and b_{3-k} still the same. Formula B5 ensures that, when Q_l and interpretations of a_k and b_k are the same (meaning "counter k has zero value") then interpretations $a_k, b_k, a_{3-k}, b_{3-k}$ should still the same in the next moment of time. Formulae B6 and B7 regulate the switching of truth values of Q_i ($i \in \{1, \dots L\}$).

Further, let the formula χ_0 be conjunction of the following formulae:

- $Q_0 \wedge \text{Same}(d, a_1) \wedge \text{Same}(a_1, b_1) \wedge \text{Same}(b_1, a_2) \wedge \text{Same}(a_2, b_2)$ At the initial moment of time the constants d, a_1, a_2, b_1, b_2 have the same designation.
- $\bigcirc(Q_1 \wedge Same(a_1, a_2) \wedge Same(a_2, b_1) \wedge Same(b_1, b_2) \wedge \neg Same(a_1, d))$ At the next moment of time Q_1 holds and d, a_1, a_2, b_1, b_2 have the same designations, while d has the different designation.
- $\Box New(d) \wedge \Box(\bigwedge_{1 \le i < j \le L} \neg\text{Same}(e_i, e_j))$, stating that d has different designations at different moments of time and $e_1, \dots e_L$ all have different interpretations.

Finally, let $\chi^{\mathcal{M}}$ be $\bigwedge_{i=1}^{L-1}(\chi(c_k))$ where $\mathcal{M}$ is a Minsky machine defined by the sequence of instructions $c_1, \dots, c_L$.

The formula $\chi_0 \wedge \chi^{\mathcal{M}}$ is intended to faithfully describe the computation of the machine $\mathcal{M}$ and the following lemma provides a formal justification for this.

Lemma 1 *A Minsky machine $\mathcal{M}$ produces an infinite run if, and only if, $\chi_0 \wedge \chi^{\mathcal{M}} \models \Box\neg Q_{stop}$.*

Proof

$\boxed{\Rightarrow}$ Let a machine $\mathcal{M}$ produces an infinite run $r^{\mathcal{M}} = (l_1, p_1^0, p_2^0), (l_2, p_1^1, p_2^1), \dots (l_{j+1}, p_1^j, p_2^j), \dots$, and a temporal structure $\mathfrak{M} = \langle D, \mathcal{I}\rangle$ is a model of $\chi_0 \wedge \chi^{\mathcal{M}}$. Straightforward induction on steps in $r^{\mathcal{M}}$ shows that, for all $j \ge 1$, the following relation between states of $\mathcal{M}$ and $\mathfrak{M}$ holds:

$l_j = l$ whenever $j \models Q_l$;

$p_1^j = |V_{a_1}^{j+1} \cap \overline{V_{b_1}^{j+1}}| = |V_{a_1}^{j+1}| - |V_{b_1}^{j+1}|$;

$p_2^j = |V_{a_2}^{j+1} \cap \overline{V_{b_2}^{j+1}}| = |V_{a_2}^{j+1}| - |V_{b_2}^{j+1}|$.

Since the run $r^{\mathcal{M}}$ is infinite we have $l_j \neq L$ for all $j \ge 1$, and therefore $j \models \neg Q_{stop}$ for all $j \ge 0$. Hence, $0 \models \Box\neg Q_{stop}$

$\boxed{\Leftarrow}$ By contraposition it is sufficient to show that if a machine $\mathcal{M}$ produces a finite run (halts) then $\chi_0 \wedge \chi^{\mathcal{M}} \wedge \Diamond Q_{stop}$ is satisfiable.

Let a machine, $\mathcal{M}$, halt and produce a finite run $r^{\mathcal{M}} = (l_1, p_1^0, p_2^0), \dots (l_{s+1}, p_1^s, p_2^s)$, $s \ge 0$. The final executed instruction is the STOP instruction, so we have $l_{s+1} = L$. Now, we construct a temporal structure $\mathfrak{M}_n = \langle D, \mathcal{I}(n)\rangle$ as follows. We let the domain D be a countable set. Then, for all $0 \le j \le s+1$, we ensure $j \models Q_l$ whenever $l_j = l$, and $j \models Q_{stop}$ for all $j > s+1$. Further we set $I(j)(d)$

(designations of d) to be different elements of the domain for all $j \geq 0$.

Further, we set

$0 \models \text{Same}(d, a_1) \wedge \text{Same}(a_1, b_1) \wedge \text{Same}(b_1, a_2) \wedge \text{Same}(a_2, b_2)$, and

$1 \models Q_1 \wedge Same(a_1, a_2) \wedge Same(a_2, b_1) \wedge Same(b_1, b_2) \wedge \neg Same(a_1, d)$.

Further define designations of a_1, a_2, b_1, b_2 for $2 \leq j \leq s$ inductively as follows:

- If the instruction with the label l_j is of the first form (ADD) then define $I(j)(a_k) = I(m+1)(d)$, where m is a such moment of time that $I(j-1)(a_k) = I(m)(d)$ and leave designations of the remaining constants the same as in $j-1$.
- If the instruction with label l_j is of the second form (SUBTRACT) and $(j-1) \models \neg Same(a_k, b_k)$ then define $I(j)(b_k) = I(m+1)(d)$, where m is a such moment of time that $I(j-1)(b_k) = I(m)(d)$ and leave designations of the remaining constants the same as in $j-1$.
- If the instruction with the label l_j is of the second form (SUBTRACT) and $(j-1) \models Same(a_k, b_k)$ then leave designations of all constants (a_k, b_k, k=1,2) the same as in $j-1$.

Finally, assume designations of a_1, b_1, a_2, b_2 to be arbitrary for all $j > s$.

It is easily seen that this overall construction provides a model for $\chi_0 \wedge \chi^{\mathcal{M}}$ and since $l_{s+1} = L$ one also has $s \models Q_{stop}$. Thus, $\chi_0 \wedge \chi^{\mathcal{M}} \wedge \Diamond Q_{stop}$ is satisfied in $\mathfrak{M}$.

□

Now from Theorem 1 and Lemma 1 our main result follows:

Theorem 2 *The set of valid formulas of $LTL_{\lambda=}$ is not recursively enumerable.*

6 Future time case

We have used both *past* and *future* time modalities in the above modelling of Minsky machines, so the non-axiomatizability result holds for the logic with both types of modalities. What about the case of logic with only future time? We show in this section that the future time fragment of $LTL_{\lambda=}$ augmented with the future time operator *until* is still non-recursively axiomatizable.

To get the syntax of future time temporal logic $LTL^F_{\lambda=}$ one should omit in the Definition 1 the clauses with the past time operators ($\bigodot$, $\blacklozenge$, $\blacksquare$) and add the following clause: If φ and ψ are formulas, then $(\varphi \,\mathcal{U}\, \psi)$ is also formula, read φ *until* ψ. Semantic of *until* operator is defined in usual way:

$$n \models^{\mathfrak{a}} \varphi \,\mathcal{U}\, \psi \quad \text{iff} \quad \text{there is } m \geq n \text{ such that } m \models^{\mathfrak{a}} \psi \text{ and } k \models^{\mathfrak{a}} \varphi \text{ for every } n \leq k < m;$$

Theorem 3 *The set of valid formulas of $LTL^F_{\lambda=}$ is not recursively enumerable*

Proof (sketch) As in the case of $LTL_{\lambda=}$ we use modelling of Minsky machines computations by formulas of $LTL^F_{\lambda=}$. The modelling is similar to that we have done above, so we restrict ourselves only to the demonstration of crucial points. The main idea is to specify some moment in the future as the starting point and then model computations step-by-step in a way very similar we have done before but moving *backward* in time. Let φ_{START} be a formula $New(d) \,\mathcal{U}\, \Box NoChange(d)$, saying that the pebble d moves to the new places until it stabilizes, i.e. interpretation $\mathcal{I}$ of d in any model of φ_{START} satisfies $\exists k \; (\forall i, j \;\; 0 \leq i < j \leq k \;\; (\mathcal{I}(i)(d) \neq \mathcal{I}(j)(d)) \wedge \forall l > k \;\; (\mathcal{I}(l)(d) = \mathcal{I}(k)(d)))$. The moment k we call starting point. Then, given a Minsky machine $\mathcal{M}$ we define its temporal translation as a formula $\rho_0 \wedge \rho^{\mathcal{M}}$ of $LTL^F_{\lambda=}$ with the following intended property: in any model $\mathfrak{M}$ of $\varphi_{START} \wedge \rho_0 \wedge \rho^{\mathcal{M}}$ with the starting point k, the states $\mathfrak{M}_k, \mathfrak{M}_{k-1}, \ldots \mathfrak{M}_0$ represent first $k+1$ states of the run $r^{\mathcal{M}}$ in the same way as $\mathfrak{M}_0, \ldots \mathfrak{M}_k, \ldots$ did in the proof of Lemma 1. To this end we introduce a "backward" analogue of the $NextNew(\_, \_, \_)$ formula, that is

$$BackNextNew(a, d, c) \Leftrightarrow$$

$$\bigcirc[\langle \lambda w. \langle \lambda x. \Diamond (\langle \lambda y. (y = x) \rangle (d) \;\wedge$$

$$\bigcirc \langle \lambda v. (v = w) \rangle (d)) \rangle (c) \rangle (a)] \;\wedge$$

$$\lambda z. \bigcirc (\lambda t. (t = z) \rangle (c)) \rangle (a)$$

The new translation $\rho^{\mathcal{M}}$ of a Minsky machine $\mathcal{M}$ is obtained from $\chi^{\mathcal{M}}$ defined in Section 5 replacing:

- each formula of the form $\Box(\Phi \rightarrow \Psi)$ where Ψ is not of the form $NextNew(\_, \_)$ and is not of the form $\bigcirc Q_t$ with the formula $(\bigcirc \Phi \rightarrow \Psi) \,\mathcal{U}\, (\bigcirc \Box NoChange(d))$;
- each formula of the form $\Box(\Phi \rightarrow \bigcirc Q_t)$ with the formula $(\bigcirc \Phi \rightarrow Q_t) \,\mathcal{U}\, (\bigcirc \Box NoChange(d))$; and
- each formula of the form $\Box(\Phi \rightarrow NextNew(x, d))$ with the formula $(\bigcirc \Phi \rightarrow BackNextNew(x, d, c)) \,\mathcal{U}\, (\bigcirc \Box NoChange(d))$

As an example, the formula B2 (Subsection 5.1) should be replaced with $(\bigcirc(Q_l \wedge \neg Same(a_k, b_k)) \rightarrow BackNextNew(b_k, d, c)) \,\mathcal{U}\, (\bigcirc \Box NoChange(d))$.

Further, the formula ρ_0 be the conjunction of the following formulae (it is appropriate modification of χ_0):

- $New(d)\,\mathcal{U}\,(Q_0 \wedge \text{Same}(d,a_1) \wedge \text{Same}(a_1,b_1) \wedge \text{Same}(b_1,a_2) \wedge \text{Same}(a_2,b_2) \wedge \Box NoChange(d))$
- $New(d)\,\mathcal{U}\,(Q_1 \wedge Same(a_1,a_2) \wedge Same(a_2,b_1) \wedge Same(b_1,b_2) \wedge \neg Same(a_1,d) \wedge \bigcirc \Box NoChange(d))$
- $\Box(\bigwedge_{1\le i<j\le L} \neg\text{Same}(e_i,e_j))$

Now we have an analogue of Lemma 1:

Lemma 2 *A Minsky machine $\mathcal{M}$ produces a finite run and stops by execution of $L: STOP$ instruction if, and only if, $\varphi_{START} \wedge \rho^{\mathcal{M}} \wedge \rho^0 \wedge Q_L$ is satisfiable formula, i.e. there is a model $\mathfrak{M}$ of $\varphi_{START} \wedge \rho^{\mathcal{M}} \wedge \rho^0$ such that $\mathfrak{M}_0 \models Q_L$*

The proof of the lemma is similar to that of Lemma 1. One should take into account, though, the opposite direction in time used for modelling computations.

By contraposition we have from Lemma 2, that $\mathcal{M}$ produces an infinite run, if and only if, $\varphi_{START} \wedge \rho^{\mathcal{M}} \wedge \rho^0 \to \neg Q_L$ is valid formula. The statement of theorem follows immediately $\Box$.

7 Encoding of a tiling problem

One of the anonymous referees of this paper has suggested an alternative proof of non-recursive axiomatizability of LTL^F_λ which does not require equality. The idea is to reduce well-known *recurrent tiling problem*[3] to the satisfiability problem for LTL^F_λ. However, this proof uses stronger assumption on the vocabulary of the logic. It establishes the result for the logic in the vocabulary with *countably many* unary predicates. Here it goes.

Let T be a finite set of square tiles. Every side of each tile $t \in T$ has a colour, namely $top(t), bot(t), lef(t), rt(t)$. Let $t_0 \in T$ be a distinguished tile. A recurrent tiling of (T,t_0) is a map $f: \mathbb{N}\times\mathbb{N} \to T$ such that $f(0,m) = t_0$ for infinitely many m, and for all $n,m \in \mathbb{N}$ we have $rt(f(n,m)) = lef(f(n+1,m))$ and $top(f(n,m)) = bot(f(n,m+1))$. It is known that the problem of whether there exists a recurrent tiling of a given (T,t_0) is Σ^1_1-hard (so, certainly not recursively enumerable).

For a given (T,t_0), let $H = \{(t,t') \in T\times T : rt(t) = lef(t')\}$ and $V = \{(t,t') \in T\times T : top(t) = bot(t')\}$. Consider the following formulas with only $\bigcirc, \Box, \Diamond$, a flexible constant d, and a unary relation symbol t for each $t \in T$.

1. $\Box\langle\lambda x.\Box \bigvee_{t\in T} (t(x) \wedge \bigwedge_{t'\ne t} \neg t'(x))\rangle(d)$
(unique tile everywhere)

2. $\Box\langle\lambda x.\Box \bigvee_{(t,t')\in H} (t(x) \wedge \bigcirc t'(x))\rangle(d)$
(horizontal colours match)

3. $\Box\langle\lambda x. \bigvee_{(t,t')\in V} [t(x) \wedge \bigcirc\langle y.t'(y)\rangle(d)]\rangle(d)$
(vertical colours along y-axis match)

4. $\Box\langle\lambda x.\bigcirc\langle\lambda y.\Box \bigvee_{(t,t')\in V} (t(x) \wedge \bigcirc t'(y))\rangle(d)\rangle(d)$
(remaining vertical colours match)

5. $\langle\lambda x.\Box\Diamond t_0(x)\rangle(d)$ (recurrent)

Suppose there is a recurrent tiling f of (T,t_0). For $n,m \in N$ write t_{nm} for $f(n,m)$. Take the domain D of a model to be N, and define $\mathcal{I}(d)(n) = n$. For each $n,m \in N$ and $t \in T$ we define $n \models t(m)$ iff $n \ge m$ and $t = t_{n-m,m}$. So $n+m \models t(m)$ iff $t = t_{nm}$. Now it is easy to check that all of the above formulas are true at time 0.

Conversely, assume that the formulas all hold at time 0 in some model. For each $m \in N$ let $d_m = \mathcal{I}(d)(m)$. For each $n,m \in N$, define $f(n,m) \in T$ to be the unique tile $t_{nm} \in T$ such that $n+m \models t_{nm}(d_m)$. This is well-defined by formula 1. Now it is straightforward to check that f is a recurrent tiling of (T,t_0).

So, the formulas are satisfiable iff (T,t_0) has a recurrent tiling. That gives a proof of the following theorem:

Theorem 4 *Let $\mathcal{L}$ be an alphabet including countably many unary predicate symbols. The set of valid formulas of LTL^F_λ in the alphabet $\mathcal{L}$, without equality and Until operator, is not recursively axiomatizable.*

8 Conclusion

We have considered the extension $LTL_{\lambda=}$ of classical propositional temporal logic PTL and shown that the logic is suitable for specifications of dynamic systems using some resources, such as processes using memory locations or mobile agents occupying some sites. Despite its simplicity $LTL_{\lambda=}$ proved to be not recursively axiomatizable, even when restricted to the future time fragment and to the fragment without equality, but with countably many unary predicates. The results indicate that fully automated verification of $LTL_{\lambda(=)}$ specifications via validity checking (theorem proving) is not, in general, possible. Identification of decidable fragments of $LTL_{\lambda(=)}$ (if any) is an interesting problem for further research. We leave the investigation of $LTL_{\lambda(=)}$ with restrictions on the number of predicates and flexible constants to the future work. Another interesting route towards verification is the design of efficient model checking procedures for $LTL_{\lambda=}$ and its fragments.

Acknowledgements. We would like to thank anonymous referees for their helpful suggestions. Special thanks to the referee who has proposed Theorem 4 and its proof.

References

[1] J. Cao, X. Feng, J. Lu, and S. K. Das. Mailbox-based scheme for mobile agent communications. *IEEE Computer*, 35(9):54–60, 2002.

[2] M. Fitting. Modal logic between propositional and first order. *Journal of Logic and Computation*, 12:1017–1026, 2002.

[3] D. Harel. Effective transformations on infinite trees, with applications to high undecidability, dominoes, and fairness. *Journal of the ACM*, 33:224–248, 1986.

[4] H. Huttel. Undecidable equivalence for basic parallel processes. In *Proceedings of TACS-94, Lecture Notes in Computer Scinece 789*, pages 454–464. Springer-Verlag, 1994.

[5] G. Kucherov and M. Rusinowitch. Undecidability of ground reducibility for word rewriting systems with variables. *Information Processing Letters*, 53:209–215, 1995.

[6] M. Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall International, 1967.

[7] T.Ball, B.Cook, Das, and S. Rajamani. Refining approximations in software predicate abstraction. In *Proceedings of Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 388–403. Springer-Verlag, 2004.