

CARLOS OLARTE
FRANK D. VALENCIA

Undecidability of Monadic First-Order Linear-Time Temporal Logic

Abstract. We prove the undecidability of the monadic fragment of Pnueli's First-Order Linear-Time Temporal Logic (FLTL) without function symbols nor equality. We show that, allowing quantification of flexible variables, the set of tautologies in this logic is not recursively enumerable. This justifies and clarifies the restriction on the quantification of variables in previous decidability results for this fragment. We also answer an open question raised in a previous work, namely, whether it is possible to get rid of the syntactic restrictions on the negation of formulae to obtain decidable fragments of FLTL.

Keywords: Linear Time Temporal Logic, Quantification of Flexible Variables, Decidability of Temporal Logic

1. Introduction

Temporal logics were introduced into computer science by Pnueli [Pnu77] and thereafter proven to be a good basis for the specification as well as for (automatic and machine-assisted) reasoning about concurrent systems.

It has been shown that First-Order Linear-Time Temporal Logic (henceforth referred to as FLTL) is *strongly incomplete*, i.e., it does not admit a finitistic sound and complete formal system [SH88, Aba90], or equivalently, the set of the tautologies of the logic is not recursively enumerable.

For this reason, several works in the literature address the problems of identifying decidable fragments of FLTL amenable for automatic verification and finding the minimum requirements in the logic to be incomplete (see e.g., [SH88, Mer92, Hus08, HWZ00a, DFL02]).

From Löwenheim we know that the *monadic* fragment without function symbols nor equality of First-Order Logic (FOL) is decidable (w.r.t. the satisfiability problem) [BGG01]. In [Mer92], it is shown that the corresponding monadic fragment of FLTL is also decidable. This is proven, as in FOL, by a reduction to the validity problem of propositional temporal logic by standard arguments.

In this paper we shall show that monadic FLTL without functions nor equality is *incomplete* in the above mentioned sense. This seemingly con-

Presented by **Name of Editor**; Received September 14, 2009

tradiictory statement w.r.t. [Mer92] arises from the fact that, unlike this work, [Mer92] disallows quantification over *flexible variables* (i.e., variables that may take on different values from one state to another). Our result, therefore, shows this restriction to be necessary for decidability.

Our undecidability result is proven by a reduction from the halting problem for Minsky machines [Min67] (two-counter machines), a standard Turing-complete formalism. We show that the monadic fragment of FLTL here considered is sufficient to simulate the behaviors of Minsky machines. More precisely, given any Minsky machine M , we can effectively construct a formula F_M that is valid if and only if M loops (it never halts). Since the complement of the set of halting Minsky machines is not recursively enumerable, the incompleteness of the monadic fragment of FLTL follows.

This result allows us also to answer an open question in [Val05], namely, if it is possible to drop the syntactic restriction on the negation of formulae to prove the decidability of the FLTL fragment there studied. Our undecidability result shows that this is not possible since with negation that logic would correspond to the FLTL here studied.

The undecidability result here presented was originally reported by the present authors in the conference paper [OV08] as an application of Universal Timed CCP (`utcc`), a formalism from the realm of Timed Concurrent-Constraint Programming (CCP) [Sar93]. In [OV08] the result was only stated but not proven and it relied on concurrency theory, particularly, concurrent constraint programming. Here we provide a direct proof of the statement using solely arguments of logic.

2. Preliminaries

In this section we shall briefly recall the syntax and semantics of Pnueli's First-Order Linear-Time Temporal Logic (FLTL) [MP91]. We shall also recall the notion of computations in a Minsky machine.

2.1. First-Order Linear-Time Temporal Logic

For our undecidability results it will suffice to consider formulae in FLTL with only two modalities: the *next* and *always* operators.

DEFINITION 2.1 (FLTL Syntax). *Given a first-order language \mathcal{L} , formulae in FLTL are built from the following syntax:*

$$F, G, \dots := p(\vec{t}) \mid F \wedge G \mid \neg F \mid \exists x F \mid \circ F \mid \square F.$$

where p is a predicate symbol in \mathcal{L} of arity $|\vec{t}|$.

Intuitively, the modalities $\circ F$ and $\square F$ state, respectively, that F holds *next* and *always*. Connectives like \forall, \Rightarrow , are assumed to be defined as syntactic abbreviations and will be used freely. We shall use the modality $\diamond F$ as an abbreviation of $\neg \square \neg F$, meaning that F eventually holds.

Semantics of FLTL. As done in Model Theory, the non-logical symbols of \mathcal{L} (predicate, function and constant symbols) are given meaning in an underlying \mathcal{L} -structure, or \mathcal{L} -model, $\mathcal{M}(\mathcal{L}) = (\mathcal{I}, \mathcal{D})$. This means, they are interpreted via \mathcal{I} as relations over a domain \mathcal{D} of the corresponding arity.

A *state* s is a mapping assigning to each variable x in \mathcal{L} a value $s[x]$ in \mathcal{D} . This interpretation is extended to \mathcal{L} -expressions in the usual way, for example, $s[f(x)] = \mathcal{I}(f)(s[x])$. We write $s \models_{\mathcal{M}(\mathcal{L})} p(\vec{t})$ if and only if $p(\vec{t})$ is true with respect to s in $\mathcal{M}(\mathcal{L})$.

The state s is said to be an x -variant of s' iff $s'[y] = s[y]$ for each $y \neq x$. This is, s and s' are the same except possibly for the value of the variable x .

We shall use σ, σ', \dots to range over infinite sequences of states. We say that σ is an x -variant of σ' iff for each $i \geq 0$, $\sigma(i)$ (the i -th state in σ) is an x -variant of $\sigma'(i)$.

Flexible and Rigid Variables. The set of variables is partitioned into *rigid* and *flexible*. For the rigid variables, the sequence of states σ must satisfy the *rigidity* condition: If x is rigid then for all state s, s' in σ , it holds $s[x] = s'[x]$. If x is a flexible variable then different states in σ may assign different values to x .

DEFINITION 2.2 (FLTL Semantics). We say that σ satisfies F in an \mathcal{L} -structure $\mathcal{M}(\mathcal{L})$, written $\sigma \models_{\mathcal{M}(\mathcal{L})} F$, if and only if $\langle \sigma, 0 \rangle \models_{\mathcal{M}(\mathcal{L})} F$ where:

$$\begin{array}{ll}
\langle \sigma, i \rangle \models_{\mathcal{M}(\mathcal{L})} \mathbf{true} & \\
\langle \sigma, i \rangle \not\models_{\mathcal{M}(\mathcal{L})} \mathbf{false} & \\
\langle \sigma, i \rangle \models_{\mathcal{M}(\mathcal{L})} p(\vec{t}) & \text{iff } \sigma(i) \models_{\mathcal{M}(\mathcal{L})} p(\vec{t}) \\
\langle \sigma, i \rangle \models_{\mathcal{M}(\mathcal{L})} \neg F & \text{iff } \langle \sigma, i \rangle \not\models_{\mathcal{M}(\mathcal{L})} F \\
\langle \sigma, i \rangle \models_{\mathcal{M}(\mathcal{L})} F \wedge G & \text{iff } \langle \sigma, i \rangle \models_{\mathcal{M}(\mathcal{L})} F \text{ and } \langle \sigma, i \rangle \models_{\mathcal{M}(\mathcal{L})} G \\
\langle \sigma, i \rangle \models_{\mathcal{M}(\mathcal{L})} \circ F & \text{iff } \langle \sigma, i+1 \rangle \models_{\mathcal{M}(\mathcal{L})} F \\
\langle \sigma, i \rangle \models_{\mathcal{M}(\mathcal{L})} \square F & \text{iff for all } j \geq i, \langle \sigma, j \rangle \models_{\mathcal{M}(\mathcal{L})} F \\
\langle \sigma, i \rangle \models_{\mathcal{M}(\mathcal{L})} \exists x F & \text{iff for some } x\text{-variant } \sigma' \text{ of } \sigma, \langle \sigma', i \rangle \models_{\mathcal{M}(\mathcal{L})} F
\end{array}$$

We say that F is valid in $\mathcal{M}(\mathcal{L})$ if and only if for all σ , $\sigma \models_{\mathcal{M}(\mathcal{L})} F$. F is said to be valid, denoted by $\models F$, if F is valid for every model $\mathcal{M}(\mathcal{L})$. We shall write $F \models G$ whenever $\models (F \Rightarrow G)$.

The semantics above corresponds to that in [MP91] where the interpretation of the variables is state dependent and the interpretation of the predicate and function symbols is *fixed*.

2.2. Minsky Machines

A two-counters or Minsky machine [Min67] is a well known Turing complete formalism. It is an imperative program consisting of a sequence of labeled instructions $(l_1, L_1); \dots; (l_m, L_m)$ which modify the values of two non-negative counters c_0 and c_1 .

The instructions, using counters c_n for $n \in \{0, 1\}$, are of three kinds:

- $(l_i : \text{HALT})$: Halts the machine.
- $(l_i : \text{INC}(c_n, l_j))$: Increments c_n and jumps to the instruction l_j .
- $(l_i : \text{DEC}(c_n, l_j, l_k))$: Tests if c_n is zero and then jumps to the instruction l_j . If c_n is not zero, it jumps to l_k .

A *configuration* of a Minsky machine is a tuple (l_i, v_0, v_1) where l_i is the label of the instruction to be executed and v_0 and v_1 the current value of the counters. Evolutions between such configurations are described by the reduction relation \longrightarrow_M in Figure 1. We shall use \longrightarrow_M^* to denote the reflexive and transitive closure of \longrightarrow_M .

In the sequel, without loss of generality, we assume that counters are initially set to zero and the machines *start* at the instruction l_1 . In other words, the initial configuration of any Minsky machine is of the form $(l_1, 0, 0)$.

We say that a Minsky machine M *halts* if the control reaches the location of a HALT instruction.

DEFINITION 2.3 (Minsky Machine Computations). *Let M be a Minsky machine with instructions $(l_1, L_1); \dots; (l_j; \text{HALT}); \dots (l_m, L_m)$. Let \longrightarrow_M be as in Figure 1. We say that M halts if there exists a derivation $(l_1, 0, 0) \longrightarrow_M^* (l_j, v_0, v_1) \not\rightarrow_M$.*

3. Encoding Minsky Machines into Monadic FLTL

In this section we show that given any Minsky Machine M , the monadic fragment of FLTL, without equality nor function symbols, suffices to effectively construct a formula F_M that faithfully describes the behavior of M . We shall assume a first-order signature with the monadic predicates $\text{out}(\cdot)$

$$\begin{array}{c}
 \text{M-INC} \frac{(l_i, \text{INC}(c_n, l_j)) \quad v'_n = v_n + 1 \quad v'_{1-n} = v_{1-n}}{(l_i, v_0, v_1) \longrightarrow_M (l_j, v'_0, v'_1)} \\
 \\
 \text{M-DEC} \frac{(l_i, \text{DEC}(c_n, l_j, l_k)) \quad v_n \neq 0 \quad v'_n = v_n - 1 \quad v'_{1-n} = v_{1-n}}{(l_i, v_0, v_1) \longrightarrow_M (l_k, v'_0, v'_1)} \\
 \\
 \text{M-DECJ} \frac{(l_i, \text{DEC}(c_n, l_j, l_k)) \quad v_n = 0}{(l_i, v_0, v_1) \longrightarrow_M (l_j, v_0, v_1)}
 \end{array}$$

 Figure 1. Reduction relation in Minsky machines. $n \in \{0, 1\}$

and **not-zero**(\cdot). Furthermore, we assume the propositional variables **isz** $_n$, **inc** $_n$, **dec** $_n$, **idle** $_n$, **zero** $_n$ for $n \in \{0, 1\}$ and **halt**.

The behavior of any Minsky machine can be simulated by the formulae $\mathcal{F}zero_n$, $\mathcal{F}not-zero_n$ and $\mathcal{F}ins$ in Figure 2. Let us give some intuitions about these formulae.

Counters. The formulae modeling the two counters c_0 and c_1 are obtained by replacing the subindex n by 0 and 1 respectively in the formulae $\mathcal{F}zero_n$ and $\mathcal{F}not-zero_n$. Roughly speaking, the formula $\mathcal{F}zero_n$ models the state $c_n = 0$ and $\mathcal{F}not-zero_n$ the state $c_n = k$ for $k > 0$.

State Zero: Once **zero** $_n$ holds, **isz** $_n$ must also hold. We can then use the propositional variable **isz** $_n$ to test if the counter is zero.

If the current instruction does not modify the value of c_n , **idle** $_n$ must hold (due to the formula $\mathcal{F}ins$) and then, the formula \circ **zero** $_n$ must also hold. This way, we model the fact that the counter remains in zero.

State Not-Zero: When an increment instruction is executed, **inc** $_n$ holds (due to the formula $\mathcal{F}ins$), and so does a formula of the form $H = \circ\exists a.(\text{not-zero}_n(a) \wedge \square(\text{out}(a) \Rightarrow F))$. In H , F is **zero** $_n$ if $c_n = 0$ (see $\mathcal{F}zero_n$) and **not-zero** $_n(x)$ otherwise (see $\mathcal{F}not-zero_n$). Intuitively, F represents the state immediately before the last increment instruction took place. This way, when a decrement instruction is performed, **out**(a) holds and so does F .

Consider now $\mathcal{F}not-zero_n$ which is of the form $\forall x. \text{not-zero}_n(x) \Rightarrow G$. As we explained before, a formula of the form $H = \circ\exists a.(\text{not-zero}_n(a))$ holds when an increment instruction is performed. Using H in conjunction with $\mathcal{F}not-zero_n$ we obtain an instantiation of the form $\exists a.(G[a/x])$ that represents the state $c_n = k + 1$. Notice that when $\exists a.(G[a/x])$ holds, **isz** $_n$ must not hold. Furthermore, similarly to the state zero, if the counter is

Counters	
$\mathcal{F}zero_n$	$= \text{zero}_n \Rightarrow \left(\begin{array}{l} \text{inc}_n \Rightarrow \circ \exists a. (\text{not-zero}_n(a) \wedge \\ \quad \square(\text{out}(a) \Rightarrow \text{zero}_n)) \\ \wedge \text{idle}_n \Rightarrow \circ \text{zero}_n \\ \wedge \text{isz}_n \end{array} \right)$
$\mathcal{F}not\text{-zero}_n$	$= \forall x. \text{not-zero}_n(x) \Rightarrow \left(\begin{array}{l} \text{inc}_n \Rightarrow \circ \exists b. (\text{not-zero}_n(b) \wedge \\ \quad \square(\text{out}(b) \Rightarrow \text{not-zero}_n(x))) \\ \wedge \text{dec}_n \Rightarrow \circ \text{out}(x) \\ \wedge \text{idle}_n \Rightarrow \circ \text{not-zero}(x) \\ \wedge \neg \text{isz}_n \end{array} \right)$
Instructions	
$\mathcal{F}ins$	$= \bigwedge_{1 \leq i \leq m} \text{out}(l_i) \Rightarrow \llbracket l_i : L_i \rrbracket_I \text{ where}$
$\llbracket l_i : \text{HALT} \rrbracket_I$	$= \text{halt}$
$\llbracket l_i : \text{INC}(c_n, l_j) \rrbracket_I$	$= \neg \text{halt} \wedge \text{inc} \wedge \neg \text{idle}_n \wedge \text{idle}_{1-n} \wedge \circ \text{out}(l_j)$
$\llbracket l_i : \text{DEC}(c_n, l_j, l_k) \rrbracket_I$	$= \text{isz} \Rightarrow (\text{idle}_n \wedge \circ \text{out}(l_j)) \wedge \\ \neg \text{isz} \Rightarrow (\neg \text{idle}_n \wedge \text{dec}_n \wedge \circ \text{out}(l_k)) \wedge \\ \text{idle}_{1-n} \wedge \neg \text{halt}$

Figure 2. Representation of a Minsky machine with instructions $(l_1 : L_1); \dots; (l_m : L_m)$. The subindex $n \in \{0, 1\}$.

not modified by the current instruction (idle_n holds), $\circ \text{not-zero}_n(a)$ must hold and then, the counter takes the same value in the next time interval.

Instructions. For the set of instruction $(l_1, L_1); \dots; (l_m, L_m)$ we assume a set of variables l_1, \dots, l_m . If the predicate $\text{out}(l_i)$ holds in a state, it means that the instruction l_i is executed. In the case of a halt instruction (l_i, HALT) , halt holds. For an increment or a decrement instruction $\neg \text{halt}$ holds.

The formula representing an increment operations $(l_i : \text{INC}(c_n, l_j))$ assures that inc_n holds. It also guarantees that idle_{1-n} holds while idle_n does not.

Finally, the formula representing a decrement instruction of the form $(l_i : \text{DEC}(c_n, l_j, l_k))$ tests if the counter c_n is zero. If this is the case, then it activates in the next time interval the instruction l_j . If isz_n does not hold, i.e. $c_n > 0$, dec_n must hold and the instruction l_k is activated in the next time unit.

The following definition introduces the formula \mathcal{F}_M that simulates the behavior of a Minsky machine M .

DEFINITION 3.1 (Encoding of a Minsky Machine). *Let M be a Minsky machine with instructions $(l_1 : L_1), \dots, (l_n : L_m)$. The encoding $\llbracket M \rrbracket$ is defined as the formula*

$$\mathcal{F}_M = \square(\mathcal{F}zero_0 \wedge \mathcal{F}not-zero_0 \wedge \mathcal{F}zero_1 \wedge \mathcal{F}not-zero_1 \wedge \mathcal{F}ins)$$

where $\mathcal{F}zero_0, \mathcal{F}not-zero_0, \mathcal{F}zero_1$ and $\mathcal{F}not-zero_1$ are obtained by replacing the sub-index n in the Equations in Figure 2.

3.1. Encoding of Numbers and Configurations

To show that the formula \mathcal{F}_M above faithfully describes the behavior of the machine M , we shall give first a suitable representation of numbers and configurations of M . This shall ease the forthcoming proofs.

As hinted at above, when an increment operations is performed, a formula of the form $H = \circ\exists a.(\text{not-zero}_n(a) \wedge \square(\text{out}(a) \Rightarrow F))$ must hold, where F represents the state immediately before the last increment instruction took place. Recall also that a decrement operation causes that $\text{out}(a)$ holds and so does F

We can then represent the state $c_n = k$, for $k > 0$, as a formula of the form $\exists a_1, \dots, a_k(F_1 \wedge \dots \wedge F_k \wedge \text{not-zero}_n(a_k))$ where F_1 is of the form $\square \text{out}(a_1) \Rightarrow \text{zero}_n$ and for $1 < i \leq k$, $F_i = \square \text{out}(a_i) \Rightarrow \text{not-zero}_n(a_{i-1})$. More precisely,

DEFINITION 3.2 (Representation of Numbers). *The FLTL formula representing the state $c_n = k$, notation $\llbracket c_n = k \rrbracket_N$, is defined as follows:*

$$\begin{aligned} \llbracket c_n = 0 \rrbracket_N &= \text{zero}_n \\ \llbracket c_n = 1 \rrbracket_N &= \exists a_1(\square \text{out}(a_1) \Rightarrow \text{zero}_n \wedge \\ &\quad \text{not-zero}_n(a_1)) \\ \dots & \\ \llbracket c_n = k \rrbracket_N &= \exists a_1, a_2, \dots, a_k(\square \text{out}(a_1) \Rightarrow \text{zero}_n \wedge \\ &\quad \square \text{out}(a_2) \Rightarrow \text{not-zero}_n(a_1) \wedge \\ &\quad \dots \\ &\quad \square \text{out}(a_k) \Rightarrow \text{not-zero}_n(a_{k-1}) \wedge \\ &\quad \text{not-zero}_n(a_k)) \end{aligned}$$

Using the previous definition of numbers, we define next the FLTL formula representing a configuration of a Minsky machine.

DEFINITION 3.3 (Encoding of Configurations). *Let M be a Minsky machine with instructions $(l_1; L_1), \dots, (l_m; L_m)$. Let $\llbracket \cdot \rrbracket_N$ be as in Definition 3.2 and*

\mathcal{F}_M be as in Definition 3.1. The encoding $\llbracket \cdot \rrbracket_C$ of a configuration of M is defined as

$$\llbracket (l_i, v_0, v_1) \rrbracket_C = \mathcal{F}_M \wedge \llbracket c_0 = v_0 \rrbracket_N \wedge \llbracket c_1 = v_1 \rrbracket_N \wedge \text{out}(l_i)$$

4. Monadic FLTL is Undecidable

We shall use the above construction to exhibit a formula that is valid if and only if the machine M loops (i.e., it never halts). This shall allow us to show that this fragment of FLTL is incomplete, i.e., its set of tautologies is not recursively enumerable.

We start by proving that computations of M are faithfully described by the formula \mathcal{F}_M in Definition 3.1.

LEMMA 4.1 (Soundness of the Encoding). *Let M be a Minsky machine with instructions $(l_1; L_1), \dots, (l_m; L_m)$, $\llbracket \cdot \rrbracket_C$ be as in Definition 3.3 and (l_i, v_0, v_1) be a configuration of M .*

If $(l_i, v_0, v_1) \rightarrow_M (l'_i, v'_0, v'_1)$ then $\llbracket (l_i, v_0, v_1) \rrbracket_C \models \neg \text{halt} \wedge \circ \llbracket (l'_i, v'_0, v'_1) \rrbracket_C$

Furthermore, if $(l_i, v_0, v_1) \not\rightarrow_M$, i.e., l_i is a HALT instruction, then it holds $\llbracket (l_i, v_0, v_1) \rrbracket_C \models \text{halt}$.

PROOF. First assume that $(l_i, v_0, v_1) \not\rightarrow_M$. Then $(l_i : L_i)$ is a HALT instruction and it is easy to see that $\llbracket (l_i, v_0, v_1) \rrbracket_C \models \text{halt}$ for any v_0, v_1 .

Assume now that $(l_i, v_0, v_1) \rightarrow_M (l'_i, v'_0, v'_1)$. Then, $(l_i : L_i)$ must be an increment or a decrement instruction. It is easy to see that for both cases $\llbracket (l_i, v_0, v_1) \rrbracket_C \models \neg \text{halt}$. Now we shall prove that $\llbracket (l_i, v_0, v_1) \rrbracket_C \models \circ \llbracket (l'_i, v'_0, v'_1) \rrbracket_C$ for both kind of instructions.

- Assume that $(l_i : L_i)$ is of the form $(l_i : \text{INC}(c_n, l_j))$. It must be the case that $l'_i = l_j$, $v'_n = v_n + 1$ and $v'_{1-n} = v_{1-n}$. Let $F = \text{out}(l_i) \Rightarrow \llbracket (l_i : L_i) \rrbracket_I$ be the subformula in $\llbracket (l_i, v_0, v_1) \rrbracket_C$ representing the encoding of the instruction l_i . We know that $\llbracket (l_i, v_0, v_1) \rrbracket_C \models \text{out}(l_i)$ and we can derive the following:

$$F \wedge \text{out}(l_i) \models \text{inc} \wedge \neg \text{idle}_n \wedge \text{idle}_{1-n} \wedge \circ \text{out}(l_j)$$

For the counter c_{1-n} , it is easy to see that

$$\mathcal{F}_M \wedge \text{idle}_{1-n} \wedge \llbracket c_{1-n} = v_{1-n} \rrbracket_N \models \circ \llbracket c_{1-n} = v_{1-n} \rrbracket_N$$

For the counter c_n we consider two cases: $v_n = 0$ and $v_n > 0$. For $v_n = 0$ we can derive the following

$$\begin{aligned} \mathcal{F}_M \wedge \mathbf{inc}_n \wedge \llbracket c_n = 0 \rrbracket_N &\models \circ \exists . a_1 (\mathbf{not_zero}(a_1) \wedge \\ &\quad \square \mathbf{out}(a_1) \Rightarrow \mathbf{zero}_n) \\ &\equiv \circ \llbracket c_n = 1 \rrbracket_N \end{aligned}$$

If $v_n = k$ for $k > 0$, then we have

$$\begin{aligned} \mathcal{F}_M \wedge \mathbf{inc}_n \wedge \llbracket c_n = k \rrbracket_N &\models \circ \exists . a_1, \dots, a_k, a_{k+1} (\\ &\quad \square \mathbf{out}(a_1) \Rightarrow \mathbf{zero}_n \wedge \\ &\quad \dots \\ &\quad \square (\mathbf{out}(a_{k+1}) \Rightarrow \mathbf{not_zero}_n(a_k)) \wedge \\ &\quad \mathbf{not_zero}_n(a_{k+1}) \\ &\equiv \circ \llbracket c_n = k + 1 \rrbracket_N \end{aligned}$$

We then conclude $\llbracket (l_i, v_0, v_1) \rrbracket_C \models \circ \llbracket (l'_i, v'_0, v'_1) \rrbracket_C$.

- Assume now that $(l_i : L_i)$ is of the form $(l_i : \mathbf{DEC}(c_n, l_j, l_k))$. We must consider two cases.

– If $c_n = 0$ then it must be the case that $v'_n = v_n$ and $l'_i = l_j$. We can derive the following

$$\begin{aligned} \mathcal{F}_M \wedge \llbracket c_n = 0 \rrbracket_N &\models \mathbf{isz}_n \wedge \mathbf{idle}_n \wedge \mathbf{idle}_{1-n} \wedge \circ (\mathbf{out}(l_j) \wedge \mathbf{zero}_n) \\ &\models \circ \llbracket c_n = 0 \rrbracket_N \end{aligned}$$

– If $c_n = k$ for some $k > 0$, then we derive

$$\mathcal{F}_M \wedge \llbracket c_n = k \rrbracket_N \models \neg \mathbf{isz}_n \wedge \mathbf{dec}_n \wedge \mathbf{idle}_{1-n} \wedge \circ \mathbf{out}(l_k)$$

Let $F' = \mathcal{F}_M \wedge \llbracket c_n = k \rrbracket_N \wedge \mathbf{dec}_n$. We derive the following

$$\begin{aligned} F' &\models \mathcal{F}_M \wedge \exists . a_1, \dots, a_k (\square \mathbf{out}(a_1) \Rightarrow \mathbf{zero}_n \wedge \\ &\quad \dots \\ &\quad \square \mathbf{out}(a_k) \Rightarrow \mathbf{not_zero}_n(a_{k-1}) \\ &\quad \wedge \mathbf{not_zero}_n(a_k) \wedge \circ \mathbf{out}(a_k)) \\ &\models \circ \exists . a_1, \dots, a_k (\square \mathbf{out}(a_1) \Rightarrow \mathbf{zero}_n \wedge \\ &\quad \dots \\ &\quad \square \mathbf{out}(a_{k-1}) \Rightarrow \mathbf{not_zero}_n(a_{k-2}) \\ &\quad \wedge \mathbf{not_zero}_n(a_{k-1}) \\ &\equiv \circ \llbracket c_n = k - 1 \rrbracket_N \end{aligned}$$

■

Using the previous lemma, we can show that a machine M produces an infinite run if and only if the formula $\mathcal{F}_M \Rightarrow \Box \neg \text{halt}$ is valid.

LEMMA 4.2. *A Minsky machine M loops (i.e. it never halts) if and only if*

$$\llbracket (l_1, 0, 0) \rrbracket_C \models \Box \neg \text{halt}$$

PROOF. Let $v_0^1 = v_1^1 = 0$ and $l^1 = l_1$.

(\Rightarrow) Assume that M produces an infinite run

$$(l^1, v_0^1, v_1^1) \longrightarrow_M (l^2, v_0^2, v_1^2) \dots \longrightarrow_M (l^n, v_0^n, v_1^n) \longrightarrow_M \dots$$

where for all $i \geq 1$, l^i is not a HALT instruction. From Lemma 4.1 we know that for all $i \geq 1$, $\llbracket (l^i, v_0^i, v_1^i) \rrbracket_C \models \circ \llbracket (l^{i+1}, v_0^{i+1}, v_1^{i+1}) \rrbracket_C$ and also that $\llbracket (l^i, v_0^i, v_1^i) \rrbracket_C \models \neg \text{halt}$. Therefore, for $i \geq 0$, it holds

$$\llbracket (l^1, v_0^1, v_1^1) \rrbracket_C \models \circ^i (\neg \text{halt})$$

where $\circ^0(F) \equiv F$ for any formula F . We then conclude $\llbracket (l_1, 0, 0) \rrbracket_C \models \Box \neg \text{halt}$.

(\Leftarrow) We proceed by contradiction. Assume that $\llbracket (l_1, 0, 0) \rrbracket_C \models \Box \neg \text{halt}$ and there exists $n \geq 1$ such that the machine produces a run

$$(l^1, v_0^1, v_1^1) \longrightarrow_M (l^2, v_0^2, v_1^2) \dots \longrightarrow_M (l^n, v_0^n, v_1^n) \not\longrightarrow_M$$

i.e., l^n is a HALT instruction. By Lemma 4.1 we know that

$$\llbracket (l^1, v_0^1, v_1^1) \rrbracket_C \models \circ^{n-1} \llbracket (l^n, v_0^n, v_1^n) \rrbracket_C \models \circ^{n-1} \text{halt} \models \diamond \text{halt}$$

thus a contradiction. ■

Since the set of looping Minsky machines (i.e. the complement of the halting problem) is not recursively enumerable, a finitistic axiomatization of monadic FLTL without equality nor function symbols would yield a non-recursively enumerable set of tautologies.

THEOREM 4.1 (Incompleteness). *There is not a sound and complete finitistic axiomatization for monadic FLTL without equality nor function symbols.*

PROOF. Directly from Lemma 4.2. ■

From the previous theorem, it follows that the *validity problem* in the above-mentioned monadic fragment of FLTL is undecidable.

5. Discussion and Related Work

As mentioned in the introduction, in [Mer92] it is proven that the validity problem of a monadic temporal logic formulae without equality nor function symbols is decidable. In the following we shall explain the seemingly contradiction of this statement with respect to our result in Theorem 4.1. First of all, the FLTL in [Mer92], referred to as TLV, differs from the FLTL here studied in that it disallows quantification of flexible variables. Let us elaborate why this restriction is important for the decidability of TLV.

The decidability of monadic TLV is proven in [Mer92] by showing that a monadic TLV formula F is valid if and only if F holds in all interpretations with universes of a certain finite cardinality. Then, the validity problem of F reduces to the validity problem in propositional temporal logic.

One may alternatively prove the decidability of monadic TLV by moving the existential quantifiers of F to the outermost position to find a prenex normal form formula $\exists \vec{x} \forall \vec{t} F'$ where F' is free of quantifiers. This way, one could reduce the validity problem of F to the validity problem of the propositional temporal formula F' similarly as it is done in monadic first-order logic [BGG01]. This strategy does not work for FLTL though. As it was pointed out in [Val05], moving a quantifier binding a flexible variable does not preserve satisfiability. To see this, consider for example the formula $F = (x = 42 \wedge \circ x \neq 42)$. If x is a flexible variable, notice that $\Box \exists x F$ is satisfiable whereas $\exists x \Box F$ is not. Notice also that if x is a rigid variable instead, $\Box \exists x F$ and $\exists x \Box F$ are both logically equivalent to **false**.

Let us now argue why the quantification of flexible variables is important in the construction of the formula \mathcal{F}_M in Definition 3.1. Recall that in \mathcal{F}_M , the formula $H = \exists a. \Box(\text{out}(a) \Rightarrow F)$ allows us to go back to the state immediately before the last increment instruction took place (described by F) when $\text{out}(a)$ holds. Assume that a were rigid and that $\text{out}(a)$ holds in the current time interval. This implies that F must hold in the current time interval and in all the subsequent time intervals. This clearly does not correspond to the expected behavior of the machine. Instead, if a is a flexible variable, the fact that $\text{out}(a)$ holds now does not imply that F must hold in the subsequent states.

Our decidability result then justifies the restriction on the quantification of flexible variable in [Mer92] to prove decidability of monadic TLV. This also allows us to answer an issue raised in a previous work. In [Val05] it is proven the decidability of the satisfiability problem of the negation-free fragment of FLTL. It was also suggested in [Val05] that one could get rid of the restriction of negation-free. Our results refutes this since by including

negation one can obviously define universal quantification (not present in the negation-free fragment of [Val05]) and then be able to reproduce the encoding of looping Minsky machines here presented.

Based on the undecidability result of $TLV(\emptyset)$ [SH88] (i.e. TLV with the empty set of predicates), [Mer92] proves an incompleteness result for monadic without equality and function symbols TLP logic. Unlike TLV , in TLP the interpretation of the predicates is flexible (state dependent) and all the variables are rigid. [Mer92] also relates undecidability results of n -adic fragments of TLP with undecidability results of $n + 1$ -adic fragments of TLV . Thus adding binary predicates turns TLV (and thus $FLTL$) strongly incomplete.

In [HWZ00b] the *monodic* fragment of $FLTL$ is introduced. A formula is monodic if every subformulae beginning with a temporal operator have at most one free variable. In this case the authors use a TLP-like semantics and conclude that the set of valid formulae in the 2-variable monodic fragment (i.e. monodic formulae with at most 2 distinct individual variables) is not recursively enumerable even considering finite domains in the interpretation. Nevertheless, validity in the fragment of 2-variable monodic formulae is decidable. In [DFL02] these results are extended by proving the undecidability for validity in the monodic monadic 2-variable with equality fragment of TLP. The work in [Hus08] extends the results in [HWZ00b] by showing the decidability of the monadic, monodic fragment of TLP with function symbols.

The formula F_M obtained from Definition 3.1 is clearly not monodic. It would be interesting to find the minimum number of distinct free variables that can occur in a $FLTL$ formula (or in a TLV -like logic) to obtain undecidability as in [DFL02] for the case of TLP.

References

- [Aba90] Martín Abadi. Corrigendum: The power of temporal proofs. *Theor. Comput. Sci.*, 70(2):275, 1990.
- [BGG01] E. Borger, E. Gradel, and Y. Gurevich. *The Classical Decision Problem*. Springer, 2001.
- [DFL02] Anatoli Degtyarev, Michael Fisher, and Alexei Lisitsa. Equality and monodic first-order temporal logic. *Studia Logica*, 72(2), 2002.
- [Hus08] Walter Hussak. Decidable cases of first-order temporal logic with functions. *Studia Logica*, 88(2):247–261, 2008.
- [HWZ00a] I. Hodkinson, F. Wolter, and M. Zakharyashev. Decidable fragments of first-order temporal logics, 2000.
- [HWZ00b] Ian M. Hodkinson, Frank Wolter, and Michael Zakharyashev. Decidable fragment of first-order temporal logics. *Ann. Pure Appl. Logic*, 106(1-3), 2000.

- [Mer92] Stephan Merz. Decidability and incompleteness results for first-order temporal logics of linear time. *Journal of Applied Non-Classical Logics*, 2(2), 1992.
- [Min67] Marvin L. Minsky. *Computation: finite and infinite machines*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1967.
- [MP91] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer-Verlag, 1991.
- [OV08] Carlos Olarte and Frank D. Valencia. The expressivity of universal timed CCP: Undecidability of monadic FLTL and closure operators for security. In *Proc. of PPDP 08*, 2008.
- [Pnu77] A. Pnueli. The temporal logic of programs. In *Proc. of the 18th IEEE Symposium on the Foundations of Computer Science (FOCS-77)*, pages 46–57. IEEE, IEEE Computer Society Press, 1977.
- [Sar93] Vijay A. Saraswat. *Concurrent Constraint Programming*. MIT Press, 1993.
- [SH88] Andrzej Szalas and Leszek Holenderski. Incompleteness of first-order temporal logic with until. *Theor. Comput. Sci.*, 57, 1988.
- [Val05] Frank D. Valencia. Decidability of infinite-state timed ccp processes and first-order ltl. *Theor. Comput. Sci.*, 330(3), 2005.

CARLOS OLARTE
INRIA and LIX, Ecole Polytechnique
Route de Saclay
91128 Palaiseau, France
`carlos.olarte@lix.polytechnique.fr`

FRANK D. VALENCIA
CNRS and LIX, Ecole Polytechnique
Route de Saclay
91128 Palaiseau, France
`frank.valencia@lix.polytechnique.fr`