Elective Temporal Logic

Ilaria Matteucci, G. Costa

Elective Temporal Logic*

Gabriele Costa
University of Pisa and
IIT-CNR
gabriele.costa@iit.cnr.it

Ilaria Matteucci
IIT-CNR
ilaria.matteucci@iit.cnr.it

ABSTRACT

In this paper we present a novel formalism for defining properties over linear execution traces, namely *elective temporal logic* (ETL). Differently from several other temporal logics, ETL is not dedicated to a specific time model, e.g. discrete time or real time. Hence, properties can be applied to each temporal context with no changes to the specified formulas. Moreover, the ETL denotational semantics is given through *elective functions*. In this way we map formulas into the characteristic functions of a set of accepted traces, i.e. the *valid* executions.

A further contribution of this work is an application of ETL to runtime monitoring. As a matter of fact, using a security monitor driven by an ETL formula, we can ignore irrelevant security actions performed by the guarded program reducing the monitor workload.

1. INTRODUCTION

Specifying the system requirements is a fundamental issue for the correct design and development of programs. For instance, verifying whether a certain program complies with a property can be automatically carried out if the property is defined using a formal specification language. In particular, the verification of collectives of security requirements, i.e. *security policies*, is commonly a complex process possibly involving different aspects and stages of the life cycle of a system. In part, security policies can be statically verified, e.g. at design time, in order to prevent the execution of faulty programs. Moreover, checks can be applied at deploy time for guaranteeing that the specification of a program meets the policies of the hosting platform. Finally, the execution of a program can be watched for monitoring its actual behaviour with respect to the security rules. All these operations can be considerably improved if the requirements are specified using a proper formalism.

9*Work partially supported by EU-funded project FP7-231167 CONNECT, by EU-funded project FP7-257930 ANIKETOS and by EU-funded project FP7-256980 NESSOS.

In this paper we present *elective temporal logic* (ETL), that is a novel formalism for the specification of programs properties. The most important differences between ETL and other temporal logics are (i) an *abstract* representation of time and (ii) its *elective semantics*.

Many temporal logics use a syntax that is devoted to the underlying time model, i.e. discrete time or continuous time. Usually, temporal logics formulas implicitly refer to the time model through some temporal operators. Temporal operators apply a temporal context to a formula in the sense that the property must hold only in certain time intervals or instants. Instead, ETL uses time expressions and time propositions that abstract from any time model. Indeed, the actual time model changes according to the definitions used by the time expressions semantics.

Elective symbols, firstly introduced in [6], are the building blocks of the ETL formulas. Using elective symbols instead of atomic propositions we can provide ETL with a denotational semantics that associates an *elective* function to each formula. Elective functions remove from a set, the function input, the elements that do not comply with the original formula.

A further contribution of this work is the investigation of the advantages deriving from the specification of security properties using ETL. In particular, we discuss the expressive power of ETL also in comparison with another, commonly used formalism, i.e. linear temporal logic. Moreover, we show an application of ETL to the development of lightweight security monitors. Also, we introduce the possibility of performing a quantitative evaluation of the computational costs deriving from the monitoring of a program for checking its compliance with a given formula.

This paper is structured as follows: Section 2 presents the ETL syntax, its semantics and the main features of the formalism; Section 3 introduces how to express properties over execution traces using ETL; Section 4 describes how ETL can be suitably used for the design of security monitors; In Section 5 we discuss the features of ETL in comparison with some related work and Section 6 concludes the paper. The proofs of the lemmata and theorems stated in the paper, together with some auxiliary results, are in the Appendix.

2. ELECTIVE TEMPORAL LOGIC

In this section we introduce ETL, its syntax and formal semantics. Furthermore, we give the definition of compliance of a trace with respect to an ETL formula.

2.1 Syntax

Given a denumerable set of actions Λ, an ETL formula is described by the following, abstract syntax.

$$\phi, \phi' ::= \top \mid \alpha_\tau \mid \neg\phi \mid \phi \wedge \phi' \mid \forall_{t:p(t)}.\phi$$

Basically, $\top$ is the formula satisfied by every trace, α_τ is an elective symbol for action $\alpha \in \Lambda$ performed at time τ, $\neg\phi$ is the negation of ϕ, $\phi \wedge \phi'$ is for logical conjunction and $\forall_{t:p(t)}.\phi$ is a universally quantified formula defined over a set of time instants.

The syntax defined above also uses time expressions (τ, τ') and time propositions (p, p'). Time expressions are needed for providing a formula ϕ with a temporal context. The syntax for time expressions is:

$$\tau, \tau' ::= 0 \mid t \mid \mathbf{s}(\tau)$$

Roughly, 0 stands for initial execution time, t is a time variable and $\mathbf{s}(\tau)$ denotes the successor of τ.

The substitution operator $[\tau/t]$ works as expected by replacing the instances of time variable t appearing in a time expression (we also extend it to time propositions and ETL formulas) with the time expression τ.

Time propositions declare a property over time instants. The syntax for time propositions is

$$p, p' ::= \tau R \tau' \mid p \wedge p' \mid p \vee p'$$

where $R \in \mathfrak{R}$. Roughly, $\mathfrak{R}$ is a set of binary relations. These relations may change according to the model of time, namely *time domain*, the system refers to. For instance, some of the binary relations that we will use in our examples are: $=$, $\neq$, $\leqslant$, $\geqslant$, $>$, $<$ and $\%$. Many of them are self-explanatory; we just give a definition for $\%$ according to the modulo operator in this way:

$$n\%m \;\Leftrightarrow\; n \bmod m = 0$$

Further details are provided in Section 2.2. We write $p(t)$ for emphasizing the presence of a free variable t and $p(\tau)$ for $p(t)[^{\tau}/_{t}]$.

We also introduce some abbreviations in order to have a more compact and readable notation.

$$\bot \stackrel{def}{=} \neg\top \qquad \phi \vee \phi' \stackrel{def}{=} \neg(\neg\phi \wedge \neg\phi')$$

$$\phi \rightarrow \phi' \stackrel{def}{=} \neg\phi \vee \phi' \qquad \exists_{t:p(t)}.\phi \stackrel{def}{=} \neg\forall_{t:p(t)}.\neg\phi$$

$$\phi \leftrightarrow \phi' \stackrel{def}{=} \phi \rightarrow \phi' \wedge \phi' \rightarrow \phi \qquad \mathbf{n} \stackrel{def}{=} \overbrace{\mathbf{ss}\cdots\mathbf{s}}^{n-\text{times}}(0)$$

$$\tau R t R' \tau' \stackrel{def}{=} (\tau R t) \wedge (t R' \tau')$$

We say that an ETL formula is *closed* when it contains no free time variables, that is, all the variables appearing in the formula are bound by a proper quantifier.

2.2 Semantics

In order to provide a formal semantics for ETL formulas we need to introduce some preliminary concepts.

The first step consists in providing both a time model and a formal semantics for time expressions. Time measurements are generated by an execution *clock*. Basically, a clock is a pair $\langle \prec, \mathcal{C} \rangle$ where $\prec$ is a strict partial order relation and $\mathcal{C} = \{n_0, n_1, \ldots\}$ is a denumerable, indexed set of time samples such that $n_i \prec n_j \Leftrightarrow i < j$ and n_0 is the bottom element. Moreover, $\mathcal{C} \subseteq \mathbb{T}$, where $\mathbb{T}$ is a time domain such that for all $R \in \mathfrak{R}$ there exists a corresponding relation $\mathbf{R} \in \mathbb{T} \times \mathbb{T}$. In the following we may use the same notation for both R and $\mathbf{R}$ where no risk of ambiguity arises. Then, we define a semantic function N mapping a time expression τ into an element of $\mathcal{C}$. Intuitively, the execution clock gives us a model of the system time.

We can now give the semantics for time expressions (with respect to a clock $\langle \prec, \mathcal{C} \rangle$).

$$\mathsf{N}(0) = n_0 \qquad \mathsf{N}(\mathbf{s}(\tau)) = \min\{n \in \mathcal{C} \mid n \succ \mathsf{N}(\tau)\}$$

Here we assume $\mathbb{N}_0$ to be the preferred time domain and, where not explicitly mentioned, we assume $\mathbb{T} = \mathbb{N}_0$. Nevertheless, all the following results apply to any time domain. The simplest clock that we can have in $\mathbb{N}_0$ is $\langle <, \mathbb{N}_0 \rangle$, that is a clock merely counting the performed actions (i.e., $n_i = i$). Using such a clock the semantics for time expressions can be simplified to

$$\mathsf{N}(0) = 0 \qquad \mathsf{N}(\mathbf{s}(\tau)) = 1 + \mathsf{N}(\tau)$$

An execution trace σ is an element of $\langle \Lambda \times \mathbb{T} \rangle^*$, that is a finite sequence of labelled actions. Each action of a trace has a timestamp denoting its execution time, e.g. $\alpha : n$ means that α happened at time n. The timestamps are generated by the execution clock. In symbols, if $\mathcal{C} = \{n_0, n_1, \ldots\}$ then each execution trace σ must be of the form $\sigma = (\alpha_0 : n_0)(\alpha_1 : n_1)\ldots$

We briefly define two useful operators over execution traces: $\sigma[n] = \alpha$ if and only if $\sigma = \sigma'(\alpha : n)\sigma''$ and $E(\sigma) = n \Leftrightarrow \sigma = \sigma'(\alpha : n)$. The actions composing a trace must be ordered according to their execution time. In other words, if $\sigma(\alpha : n)$ is a trace, then $n > E(\sigma)$.

We define the validity of time propositions in the following way:

$$\frac{\langle \mathsf{N}(\tau), \mathsf{N}(\tau') \rangle \in \mathbf{R}}{\vdash \tau\, R\, \tau'} \qquad \frac{\vdash p \quad \vdash p'}{\vdash p \wedge p'} \qquad \frac{\vdash p}{\vdash p \vee p'} \qquad \frac{\vdash p'}{\vdash p \vee p'}$$

The denotational semantics associates to each ETL formula a function $F : \Omega \rightarrow \Omega$, where Ω stands for the power set of $\langle \Lambda \times \mathbb{T} \rangle^*$, i.e. $2^{\langle \Lambda \times \mathbb{T} \rangle^*}$. The semantics function $[\![\cdot]\!]$ is recursively defined by the following rules.

$$[\![\top]\!] = \lambda A.A \qquad [\![\alpha_\tau]\!] = \lambda A.\{\sigma \in A : \sigma[\mathsf{N}(\tau)] = \alpha\}$$

$$[\![\phi \wedge \phi']\!] = [\![\phi]\!] \circ [\![\phi']\!] \qquad [\![\neg\phi]\!] = \lambda A.A \setminus [\![\phi]\!](A)$$

$$[\![\forall_{t:p(t)}.\phi]\!] = \bigcirc_{\vdash p(\mathbf{n})}^{n} [\![\phi[^{\mathbf{n}}/_{t}]]\!]$$

Basically, $\top$ denotes the identity function and α_τ is an elective symbol. Negation causes the complementation with respect to the argument set and the conjunction of two formulas corresponds to function composition. Finally, the universal quantifier denotes the (possibly infinite) composition of the functions obtained by instantiating the parameter t.

We say that two ETL formulas ϕ and ϕ' are *equivalent* (in symbols $\phi \equiv \phi'$) when they denote the same function.

We can now give the definition of compliance of an execution trace with respect to an ETL formula.

DEFINITION 1. *Given an execution trace σ and an ETL formula ϕ, we say that σ complies with ϕ (in symbols $\sigma \models_{\text{ETL}} \phi$) if and only if*

$$\sigma \in [\![\phi]\!](\Omega)$$

As expected, a trace σ satisfies a formula ϕ whenever the elective function for ϕ does not remove it from Ω.

3. APPROPRIATENESS OF ETL

In this section we discuss the appropriateness of ETL for expressing properties of execution traces. We start by comparing ETL and LTL. Then, we show how ETL can be used to express several properties of interest. Finally, we conclude the section with a discussion on the model checking problem for ETL formulas.

3.1 Encoding LTL in ETL

Here we show how formulas expressed using *linear temporal logic* (LTL) [15] can be translated into equivalent ETL ones. We start by briefly recalling the LTL syntax and semantics.

DEFINITION 2. *LTL formulas are all and only the formulas written according to the following, abstract syntax.*

$$\psi, \psi' ::= tt \mid \alpha \mid \neg\psi \mid \psi \wedge \psi' \mid \mathsf{X}\psi \mid \mathsf{G}\psi \mid \psi \,\mathsf{W}\, \psi'$$

Where tt denotes the formula that is always satisfied, α is an atomic proposition, $\neg\psi$ is for negation and $\psi \wedge \psi'$ is the conjunction of two sub formulas. The temporal operator $\mathsf{X}\psi$ means that ψ must be true at the next step, $\mathsf{G}\phi$ states that ϕ must be always true and $\psi \,\mathsf{W}\, \psi'$ says that ψ must hold until ψ' is valid or forever if ψ' never holds.

The compliance of an execution trace with respect to a LTL formula is defined as follows.

$$\begin{aligned}
&\langle \sigma, i\rangle \models_{\text{LTL}} tt \\
&\langle \sigma, i\rangle \models_{\text{LTL}} \alpha && \Leftrightarrow \sigma[i] = \alpha \\
&\langle \sigma, i\rangle \models_{\text{LTL}} \neg\psi && \Leftrightarrow \langle \sigma, i\rangle \not\models_{\text{LTL}} \psi \\
&\langle \sigma, i\rangle \models_{\text{LTL}} \psi \wedge \psi' && \Leftrightarrow \langle \sigma, i\rangle \models_{\text{LTL}} \psi \text{ and } \langle \sigma, i\rangle \models_{\text{LTL}} \psi' \\
&\langle \sigma, i\rangle \models_{\text{LTL}} \mathsf{X}\psi && \Leftrightarrow \langle \sigma, i+1\rangle \models_{\text{LTL}} \psi \\
&\langle \sigma, i\rangle \models_{\text{LTL}} \mathsf{G}\psi && \Leftrightarrow \text{for all } k \geqslant i\ \langle \sigma, k\rangle \models_{\text{LTL}} \psi \\
&\langle \sigma, i\rangle \models_{\text{LTL}} \psi \,\mathsf{W}\, \psi' && \Leftrightarrow \begin{cases} \langle \sigma, i\rangle \models_{\text{LTL}} \psi' \quad \text{or} \\ \langle \sigma, i\rangle \models_{\text{LTL}} \psi \text{ and } \langle \sigma, i+1\rangle \models_{\text{LTL}} \psi \,\mathsf{W}\, \psi' \end{cases}
\end{aligned}$$

In order to show that ETL is expressive enough to encode LTL formulas, we need to define a time context. According to the previous definitions, we use a discrete time model, i.e. $\mathbb{T} = \mathbb{N}_0$. Moreover, we assume $\{\leqslant, <\} \subseteq \mathfrak{R}$. The following rules define a translation function $\mathcal{T}$ that maps each LTL formula into an equivalent ETL specification.

$$\begin{aligned}
\mathcal{T}_\tau(tt) &= \top \\
\mathcal{T}_\tau(\alpha) &= \alpha_\tau \\
\mathcal{T}_\tau(\neg\psi) &= \neg\mathcal{T}_\tau(\psi) \\
\mathcal{T}_\tau(\psi \wedge \psi') &= \mathcal{T}_\tau(\psi) \wedge \mathcal{T}_\tau(\psi') \\
\mathcal{T}_\tau(\mathsf{X}\psi) &= \mathcal{T}_{\mathbf{s}(\tau)}(\psi) \\
\mathcal{T}_\tau(\mathsf{G}\psi) &= \forall_{t:t\geqslant\tau}.\mathcal{T}_t(\psi) \\
\mathcal{T}_\tau(\psi \,\mathsf{W}\, \psi') &= \exists_{t:t\geqslant\tau}.\Big(\mathcal{T}_t(\psi') \wedge \forall_{t':\tau\leqslant t'<t}.\mathcal{T}_{t'}(\psi)\Big) \\
&\quad \vee \forall_{t'':t''\geqslant\tau}.\mathcal{T}_{t''}(\psi)
\end{aligned}$$

The equivalence between a LTL formula and its translation into ETL is stated by the following theorem.

THEOREM 1. *For each LTL formula ψ and time instant i*

$$\langle \sigma, i\rangle \models_{\text{LTL}} \psi \iff \sigma \models_{\text{ETL}} \mathcal{T}_{\mathbf{i}}(\psi)$$

A main consequence of Theorem 1 is that ETL, with $\mathfrak{R} = \{\leqslant, <\}$, is at least expressive as LTL. However, as we shall discuss in the next section, adding further elements to $\mathfrak{R}$ increases the expressive power of ETL.

3.2 Property specification through ETL

Below we briefly overview on some property of interest for programs verification and security. Then, we formalise them using ETL and we also provide some example. Finally, we show how ETL can encode a class of properties that are not expressible using LTL.

Safety properties. Safety properties are one of the most important categories of properties for systems security. Roughly, a safety property says that unsafe configurations cannot be reached. Hence, having a formula ϕ that identifies an unsafe configuration we can express the property as

$$\forall_{t:t\geqslant\mathbf{0}}.\neg\phi$$

In words, the property says that, according to the time model, it is never the case that ϕ holds.

EXAMPLE 1. *Consider the case of an elevator. A safety property for it could be "never start moving when the door is open". Assuming we have the actions* `close_door` *and* `start_move`*, the corresponding ETL formula is*

$$\forall_{t:t\geqslant\mathbf{0}}.(\neg\texttt{close\_door}_t \rightarrow \neg\texttt{start\_move}_{\mathbf{s}(t)})$$

Mutual exclusion. The mutual exclusion problem arises when two concurrent agents need to access a single, shared resource. Again, we assume to have actions for resource lock and release (`lck_i` and `rls_i` with $i \in \{1, 2\}$). Thus, the property for preventing the access of the second agent while the first is using it (the other way round is symmetric) is

$$\forall_{t:t\geqslant \mathbf{0}}.((\texttt{lck\_1}_t \wedge \exists_{t':t'>t}.\texttt{lck\_2}_{t'}) \rightarrow \exists_{t'':t<t''<t'}.\texttt{rls\_1}_{t''})$$

This formula states that each valid computation containing two lock actions must also include a release action between them.

Example 2. *Imagine two transceivers using a shared communication channel. In order to avoid collisions each device must accomplish a transmission session, i.e.* `send` *and* `recv` *actions, before the other starts a new one. We model this property through the formula*

$$\forall_{t:t\geqslant \mathbf{0}}.((\texttt{send}_t \wedge \exists_{t':t'>t}.\texttt{send}_{t'}) \rightarrow \exists_{t'':t<t''<t'}.\texttt{recv}_{t''})$$

Chinese wall. Chinese wall is, in some sense, the dual of mutual exclusion. Indeed, it asserts that a certain actor can access only one of two sensitive resources. Whenever the actor access the first resource (`access_1`), the second one is definitely locked and vice versa. The corresponding ETL formula is

$$\begin{gathered}\exists_{t:t\geqslant \mathbf{0}}.\texttt{access\_1}_t \rightarrow \neg\exists_{t':t'>t}.\texttt{access\_2}_{t'} \\ \wedge \\ \exists_{t:t\geqslant \mathbf{0}}.\texttt{access\_2}_t \rightarrow \neg\exists_{t':t'>t}.\texttt{access\_1}_{t'}\end{gathered}$$

In words, this property states that the valid traces cannot contain accesses to both the resources. Nevertheless, there is no limitation to the number of access to a single resource.

Responsiveness. Responsiveness is often required in the interactions with servers or other systems that receive a request or a signal and answer with some acknowledgement. We can represent this property through the formula

$$\forall_{t:t\geqslant \mathbf{0}}.\texttt{req}_t \rightarrow \exists_{t':t<t'}.\texttt{ack}_{t'}$$

where `req` and `ack` are the actions for request and acknowledgement, respectively.

Another possible instance of this property is *bounded* responsiveness, that is, the server must answer within a certain time. The corresponding ETL formula is

$$\forall_{t:t\geqslant \mathbf{0}}.\texttt{req}_t \rightarrow \exists_{t':t<t'<t+\mathbf{k}}.\texttt{ack}_{t'}$$

where $\mathbf{k}$ is the answer time threshold.

Alternation. Sometimes one would like to check whether a system repeats a certain configuration in a given period of time. Provided we have a proper time relation, in ETL we can express these properties. Their general structure is

$$\forall_{t:t\%\mathbf{k}}.\phi_t$$

where $\mathbf{k}$ is the repetition period and ϕ_t is a formula identifying that a certain configuration starts at time t.

Example 3. *The* bluetooth *MAC protocol [13] uses sequences of 7, 625 μsec long time slots for the transmissions of the network orchestrator, i.e. the* master*, and the other devices, i.e. the* slaves*. The master uses the first of the 7 time slots, that is always reserved, for assigning the other slots to the slaves according to their requests. We specify this requirement using the formula*

$$\phi = \forall_{t:t\%\mathbf{7}}.\texttt{send}_t^M$$

where $\texttt{send}^M$ *is the action performed by the master sending a packet.*

Remarkably, this last kind of properties is generally not expressible through an LTL formula. This leads us to two interesting observations:

1. the relations in $\mathfrak{R}$ affect the ETL expressive power.
2. the fragment of ETL such that $\{<, \leqslant, \%\} \subseteq \mathfrak{R}$ is *strictly more expressive* than LTL.

3.3 Model checking ETL

The model checking problem consists of verifying whether a certain model $\mathcal{M}$ satisfies a formula. According to our assumptions, the behaviour of a system is represented by the execution traces it produces. Hence, we use a set-theoretic representation for statically modelling a system. In other words, given a set of traces A containing all the possible executions of a program and an ETL specification ϕ, the problem of model checking whether A complies with ϕ consists in verifying if A is a subset of the traces allowed by ϕ. This is stated by the following definition.

Definition 3. *(Model checking) Given a set of traces A and a formula ϕ we say that*

$$A \models_{\text{ETL}} \phi \iff A \subseteq [\![\phi]\!](\Omega)$$

Clearly, computing the set of traces that comply with ϕ starting from the set of all the possible traces Ω can be inefficient or even infeasible. However, we note that each ETL formula denotes a monotone function as stated below.

Lemma 1. *For each ETL formula ϕ and set of traces A*

$$[\![\phi]\!](A) \subseteq A$$

Hence, applying the function defined by ϕ to a certain set A corresponds to removing from A the traces that violate ϕ. The main consequence of Lemma 1 is the following theorem.

THEOREM 2. *For each ETL formula ϕ and traces set A*

$$A \models_{\text{ETL}} \phi \iff A = [\![\phi]\!](A)$$

Roughly, Theorem 2 asserts that the model checking problem for an ETL formula ϕ can be reduced to verifying whether ϕ denotes the identity function for a certain set A.

This method offers some advantage for the model checking procedure. Indeed, the standard behaviour of a model checker is to explore the model looking for a state that violates the property under investigation. Instead, an ETL specification can be directly compiled into a program having the same semantics of the formula. For instance, we can use ETL formulas as an intermediate language for specifying *proof checkers* for a single property. Proof checkers are used in several systems for implementing a dedicated verification procedure, e.g. for *proof carrying code* [14].

4. DESIGN OF SECURITY MONITORS

Here we apply ETL to the design of *runtime monitors* showing the advantages deriving from it.

4.1 Security monitors

In security applications a security monitor is a software agent that follows the execution of a target program and acts on it when detecting a security violation.

In general, we can represent a monitor for the property ϕ as a system component that receives a signal for each security-relevant operation that its target is going to perform. In order to be effective, a monitor must be non-bypassable, i.e. all the guarded actions produce a signal that triggers the monitor, and proactive, i.e. it can block its target before a violation takes place.

Schematically, a security monitor applying a policy ϕ works as follows. When an event (α) is generated by the target application, the monitor evaluates the system state and checks whether the new action extends the current trace to a valid one, i.e. $\sigma(\alpha : n) \models \phi$ where n is the current time. Here we consider the monitor state to be a pair containing the actions history σ and the action time n. If the validity check is successfully passed the monitor updates its state and allows the prosecution of the program, otherwise the program is stopped.

Since the monitor needs to prevent the actual execution of invalid actions, the guarded program is suspended during the monitor computation. As a consequence, monitoring results in a computational overhead mainly due to the effort of checking the compliance of a trace. Even though several efficient implementations have been proposed, e.g. see [8, 9], avoiding the compliance check when not necessary would be preferable.

4.2 Monitor optimisation through ETL

Here we show how to extend the structure of a security monitor in order to avoid unneeded policy checks.

Firstly, we introduce the *next* operator χ. Intuitively, χ takes a time instant n and an ETL formula ϕ and returns the first time value after n that is referred by ϕ or ∞ if no future event can affect ϕ. The χ function is recursively defined by the following rules.

- $\chi_n(\top) = \infty$
- $\chi_n(\alpha_\tau) = \begin{cases} \mathsf{N}(\tau) & \text{if } n < \mathsf{N}(\tau) \\ \infty & \text{otherwise} \end{cases}$
- $\chi_n(\neg\phi) = \chi_n(\phi)$
- $\chi_n(\phi \wedge \phi') = \min\{\chi_n(\phi), \chi_n(\phi')\}$
- $\chi_n(\forall_{t:p(t)}.\phi) = \min_{i:\vdash p(\mathbf{i})} \{\chi_n(\phi[^{\mathbf{i}}/_t])\}$.

Briefly, at any time instant $\top$ waits no successive action, α_τ is affected by an action performed a time τ provided that τ denotes a future moment in time while $\neg\phi$ and $\phi \wedge \phi'$ are straightforward. The rule for universally quantified formulas is slightly more tricky. In words, we state that the value returned by the χ operator is the minimum value of χ applied to the quantified formula instantiated over the elements satisfying p.

The following result outlines the advantage of including the χ operator in the monitoring process.

THEOREM 3. *For each finite trace σ and ETL formula ϕ if $\sigma \models_{\text{ETL}} \phi$, $E(\sigma) = n$ and $\chi_n(\phi) = n'$ then for all actions α and value m such that $n < m < n'$*

$$\sigma(\alpha : m) \models_{\text{ETL}} \phi$$

In words, Theorem 3 asserts that a security monitor can safely discard actions performed before the next, critical action.

Considering the previous discussion, a modified version of a security monitor can be used. Basically it uses the χ function for skipping unnecessary checks according to the following procedure. When an action is performed, the monitor verifies whether the current execution time is compliant with the expected time (variable x). If it is not the case, the monitor allows the action without checking the policy compliance. Otherwise, it works as a standard monitor and computes the expected time of the next action.

4.3 Monitoring overhead

A further consideration consists in using the χ function for statically analysing the impact of applying a monitor to a program. Security monitors always reduce the performances of a program due to the computational overhead needed for the security checks. Many efforts have been devoted to reducing the costs of the monitoring process, mainly by working on the optimisation of the monitoring systems. Here we

focus on a different aspect. Indeed, we aim at evaluating the impact of monitoring a certain property by computing a score that only depends on the formula used to specify the property. In other words, we want to assign to each ETL specification a overhead coefficient characterising the effort needed at runtime for the monitoring process independently of the actual implementation of the monitor.

Intuitively, the effort decreases when the ETL formula allows the monitor for skipping some actions, that is, the longer the monitor sleep time, the lower the system load. Moreover, this evaluation must take into account the length (in time units) of the interval, i.e. the execution time, during which the system monitors its target. Hence, given an n-long time interval and a ETL specification ϕ we compute the overhead score of ϕ using the following formula.

$$n \cdot \left(\sum_{i=0}^{n} \chi_i(\phi) - i \right)^{-1}$$

Increasing the value of n we can generalise this formula to

$$S(\phi) = \lim_{n \to \infty} n \cdot \left(\sum_{i=0}^{n} \chi_i(\phi) - i \right)^{-1} \quad (1)$$

In words, we consider the score of ϕ over a possibly infinite execution time. In this way we do not consider finite monitoring activities or formulas that cause the monitor to switch definitively off at a certain point.

EXAMPLE 4. *Consider the formula $\phi = \forall_t.\alpha_t$ and $\mathbb{T} = \mathbb{N}_0$. Intuitively, a monitor that checks the compliance of a program with ϕ needs to consider every single action coming from its target. In other words we have that for each i $\chi_i(\phi) = i + 1$. Hence, the score associated to ϕ is*

$$S(\phi) = \lim_{n \to \infty} n \cdot \left(\sum_{i=0}^{n} i + 1 - i \right)^{-1} = \lim_{n \to \infty} n \cdot \frac{1}{n} = 1$$

As expected, the monitor can never skip actions for effectively checking this policy.

EXAMPLE 5. *Consider again the scenario of Example 3. Assuming $\mathbb{T} = \mathbb{N}_0$, the score associated to this formula is*

$$S(\phi) = \lim_{n \to \infty} n \cdot \left(\sum_{i=0}^{n} 7 - i \bmod 7 \right)^{-1} \sim \lim_{n \to \infty} n \cdot \frac{1}{4n} = \frac{1}{4}$$

Clearly, in real-time systems, i.e. when $\mathbb{T} = \mathbb{R}_0^+$, we cannot directly apply (1). In order to adapt it to a continuous time model we need to replace the integer summation with an integral.

$$S(\phi) = \lim_{n \to \infty} n \cdot \left(\int_0^n \chi_i(\phi) - i \; di \right)^{-1} \quad (2)$$

Roughly, (2) is obtained from (1) by progressively reducing the size of the time intervals taken into account by the monitor. The following example shows the differences in evaluating the same formula under continuous or discrete time.

EXAMPLE 6. *Consider again the formula ϕ of Example 3. Now we assume $\mathbb{T} = \mathbb{R}_0^+$ and we evaluate again the score of ϕ obtaining*

$$S(\phi) = \lim_{n \to \infty} n \cdot \left(\int_0^n 7 - i \bmod 7 \; di \right)^{-1} = \lim_{n \to \infty} n \cdot \frac{2}{7n} = \frac{2}{7}$$

This result refers to an ideal monitor that is perfectly synchronised with the transmitting agents. However, our technique can be adapted in order to model more realistic cases. A possible approach is to introduce a time interval $\delta < 1$ that the monitor uses for synchronising with its target. In this way, when the monitor is approaching to the expected time i for the next action it starts listening a little in advance so that small synchronization errors can be prevented. The resulting equation is

$$S(\phi) = \lim_{n \to \infty} n \cdot \left(\int_0^n \chi_i(\phi) - i - \frac{\delta}{2} \; di \right)^{-1} \quad (3)$$

EXAMPLE 7. *Consider again the scenario of Example 3 (with $\mathbb{T} = \mathbb{R}_0^+$). We apply (3) for the calculation of $S(\phi)$ and we obtain the following result.*

$$S(\phi) = \lim_{n \to \infty} n \cdot \left(\int_0^n 7 - i \bmod 7 - \frac{\delta}{2} \; di \right)^{-1} = \frac{2}{7 - \delta}$$

5. RELATED WORK

Linear temporal logic [15] is one of the most used specification languages. As a matter of fact, many model checking tools, e.g. Spin [10] and NuSMV [7], use LTL-based frameworks for the specification of properties. Nevertheless, a LTL specification is evaluated according to its reference model, that provides a semantic interpretation to the formula. Since ETL comes with its own, denotational semantics, we can map each formula into a corresponding elective function. Then, elective functions can be verified, by applying them to a set of traces, with no need for a model checker.

Also monitoring environments can be driven by LTL specifications. Bauer et al. [4, 5] investigate the problem of performing and efficient run-time verification using LTL formulas. In particular, they show how to build a *minimal*, i.e. having as few states as possible, recognizer, i.e. a finite state automaton, for invalid traces. Although their automata are optimised in the number of states, they need to inspect every action composing a trace in order to check its validity. Instead, ETL-enabled monitors can be further optimised by preventing them from evaluating irrelevant actions. Moreover, we saw that ETL formulas can be evaluated in order to predict the computational effort deriving from monitoring them.

Metric interval temporal logic (MITL) is presented in [2]. Basically, MITL extends the LTL temporal operators, e.g.

W, with *time intervals*, i.e. non empty, convex subsets of $\mathbb{R}_0^+$. In this way, the scope of a temporal operator is relaxed on an arbitrary time interval. Moreover, the authors provide a semantic interpretation of MITL formulas using *timed automata* [1]. Unlike our proposal, MITL is dedicated to continuous-time models. Thus, MITL formulas can not be easily applied to discrete-time contexts, while ETL ones abstract from the specific time models.

In [12] the authors present a technique for the automatic synthesis of security controllers for a system, represented by a labelled transition system. Roughly, the controller is an agent, applied to its target through a proper process algebra operator, that is able to enforce a certain property. Actually, the property that is enforced depends on both the system and its security requirements, expressed using the μ-calculus [11]. This approach combines an operational model, i.e. the labelled transition system, and a logical formula for obtaining a security monitor. Similarly to some previous approaches, the resulting monitor needs to check every action performed by the guarded agent, while ETL-based monitors can ignore some of them.

An automata-based specification is presented in [3]. Usage automata are finite state automata that read programs traces. If a trace leads an automaton to a final, i.e. *offending*, state then it violates the policy represented by the automaton. A usage automaton is applied to a limited sequence of instructions, i.e. the scope of the policy, through a special operator, namely *policy framing*. Then usage automaton can be both model checked or translated into reference monitors. Basically, these automata work similarly to the others presented so far by reading the whole execution trace.

6. CONCLUSION

In this paper we have presented ETL, a novel formalism for defining temporal properties over execution traces. The main features of ETL are its elective semantics and its abstract time model. Elective semantics has been shown to provide several advantages for the formal verification of programs at both design-time and deploy-time. The abstract time model allows us to apply the same ETL formula to different temporal context with just an adjustment of the interpretation of the time relations. In particular, we have shown how to apply a unique specification to two different models using respectively a discrete and a continuous representation of time. Moreover, we discussed the possibility of designing a special reference monitor that uses ETL formulas for deciding whether to process a signal or not. The result is an improvement of the performances of the monitor the we can also measure analytically.

Further investigation of the verification process implemented using elective functions is the main goal for the future development of this work. As a matter of fact, here we applied elective functions directly to set of execution traces, that is an extensional, static representation of the behaviour of a program. Nevertheless, other models could be more convenient in order to have a more compact representation of the programs. For instance, the intensional representation of an agent can be obtained using a process algebra. However, the application of the elective functions needs to be revised for dealing with this kind of structures.

7. REFERENCES

[1] Rajeev Alur. Timed automata. In *Proceedings of the 11th International Conference on Computer Aided Verification*, CAV '99, pages 8–22, July 1999.

[2] Rajeev Alur, Tomás Feder, and Thomas A. Henzinger. The benefits of relaxing punctuality. *Journal of the ACM*, 43:116–146, January 1996.

[3] Massimo Bartoletti, Pierpaolo Degano, Gian Luigi Ferrari, and Roberto Zunino. Model checking usage policies. In Christos Kaklamanis and Flemming Nielson, editors, *Trustworthy Global Computing*, pages 19–35. Springer-Verlag, 2009.

[4] Andreas Bauer, Martin Leucker, and Christian Schallhart. Monitoring of real-time properties. In S. Arun-Kumar and Naveen Garg, editors, *FSTTCS 2006: Foundations of Software Technology and Theoretical Computer Science*, volume 4337 of *Lecture Notes in Computer Science*, pages 260–272. Springer Berlin / Heidelberg, 2006.

[5] Andreas Bauer, Martin Leucker, and Christian Schallhart. Runtime Verfication for LTL and TLTL. Technical Report TUM-I0724, Institut für Informatik, Technische Universität München, 2007.

[6] George Boole. The calculus of logic. *The Cambridge and Dublin Mathematical Journal*, 3, 1848.

[7] Alessandro Cimatti, Edmund M. Clarke, Fausto Giunchiglia, and Marco Roveri. NuSMV: A new symbolic model verifier. In *Proceedings of the 11th International Conference on Computer Aided Verification*, CAV '99, pages 495–499, July 1999.

[8] Gabriele Costa, Fabio Martinelli, Paolo Mori, Christian Schaefer, and Thomas Walter. Runtime monitoring for next generation Java ME platform. *Computers & Security*, July 2009.

[9] Lieven Desmet, Wouter Joosen, Fabio Massacci, Katsiaryna Naliuka, Pieter Philippaerts, Frank Piessens, and Dries Vanoverberghe. The S3MS .NET run time monitor: Tool demonstration. *Electronic Notes in Theoretical Computer Science*, 253(5):153–159, 2009.

[10] Gerard Holzmann. *Spin model checker, the: primer and reference manual*. Addison-Wesley Professional, first edition, 2003.

[11] Dexter Kozen. Results on the propositional mu-calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[12] Fabio Martinelli and Ilaria Matteucci. Through modelling to synthesis of security automata. *Electronic Notes in Theoretical Computer Science*, 179:31–46, July 2007.

[13] Riku Mettala. *Bluetooth Protocol Architecture (Version 1.0)*. Nokia Mobile Phones, September 1999.

[14] George C. Necula. Proof-carrying code. In *Proceedings of the 24th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, POPL '97, pages 106–119, 1997.

[15] Amir Pnueli. The temporal logic of programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, pages 46–57, November 1977.

Appendix

THEOREM 1. *For each LTL formula ψ and time instant i*

$$\langle \sigma, i\rangle \models_{\text{LTL}} \psi \iff \sigma \models_{\text{ETL}} \mathcal{T}_{\mathbf{i}}(\psi)$$

PROOF. By induction on the structure of ψ.

- $\psi = tt$. $\langle \sigma, i\rangle \models_{\text{LTL}} tt \iff \sigma \models_{\text{ETL}} \top$ (trivial)
- $\psi = \alpha$.

$$\langle \sigma, i\rangle \models_{\text{LTL}} \alpha \iff \sigma \models_{\text{ETL}} \alpha_{\mathbf{i}} \iff \sigma \in [\![\alpha_{\mathbf{i}}]\!](\Omega)$$

$$\iff \sigma \in \{\sigma' : \sigma'[\mathsf{N}(\mathbf{i})] = \alpha\} \iff \sigma[i] = \alpha$$

- $\psi = \neg\psi'$.

$$\langle \sigma, i\rangle \models_{\text{LTL}} \neg\psi' \iff \sigma \models_{\text{ETL}} \mathcal{T}_{\mathbf{i}}(\neg\psi')$$

$$\langle \sigma, i\rangle \models_{\text{LTL}} \neg\psi' \iff \langle \sigma, i\rangle \not\models_{\text{LTL}} \psi'$$

By induction we have

$$\langle \sigma, i\rangle \not\models_{\text{LTL}} \psi' \iff \sigma \not\models_{\text{ETL}} \mathcal{T}_{\mathbf{i}}(\psi') \iff \sigma \notin [\![\mathcal{T}_{\mathbf{i}}(\psi')]\!](\Omega)$$

$$\iff \sigma \in \Omega \setminus [\![\mathcal{T}_{\mathbf{i}}(\psi')]\!](\Omega)$$

- $\psi = \psi_1 \wedge \psi_2$. By induction we have

$$\langle \sigma, i\rangle \models_{\text{LTL}} \psi_1 \iff \sigma \models_{\text{ETL}} \mathcal{T}_{\mathbf{i}}(\psi_1)$$

and

$$\langle \sigma, i\rangle \models_{\text{LTL}} \psi_2 \iff \sigma \models_{\text{ETL}} \mathcal{T}_{\mathbf{i}}(\psi_2)$$

Hence

$$\langle \sigma, i\rangle \models_{\text{LTL}} \psi_1 \iff \sigma \in [\![\mathcal{T}_{\mathbf{i}}(\psi_1)]\!](\Omega)$$

and

$$\langle \sigma, i\rangle \models_{\text{LTL}} \psi_2 \iff \sigma \in [\![\mathcal{T}_{\mathbf{i}}(\psi_2)]\!](\Omega)$$

We conclude by observing that

$$\sigma \in [\![\mathcal{T}_{\mathbf{i}}(\psi_1)]\!](\Omega) \cap [\![\mathcal{T}_{\mathbf{i}}(\psi_2)]\!](\Omega) \iff$$

$$\sigma \in \Big([\![\mathcal{T}_{\mathbf{i}}(\psi_1)]\!] \circ [\![\mathcal{T}_{\mathbf{i}}(\psi_2)]\!]\Big)\Omega$$

- $\psi = \mathsf{X}\psi'$. A direct consequence of the inductive hypothesis.
- $\psi = \mathsf{G}\psi'$. From the definition of compliance for LTL we know that

$$\langle \sigma, i\rangle \models_{\text{LTL}} \mathsf{G}\psi \iff \text{ for all } j \geqslant i \langle \sigma, j\rangle \models_{\text{LTL}} \psi'$$

Similarly, for ETL we have that

$$\sigma \models_{\text{ETL}} \mathcal{T}_{\mathbf{i}}(\phi) \iff \sigma \models_{\text{ETL}} \forall_{t:t\geqslant\mathbf{i}}.\mathcal{T}_t(\phi') \iff$$

$$\sigma \in \bigcirc_{\vdash \mathbf{n}\geqslant\mathbf{i}}^{n} [\![\mathcal{T}_{\mathbf{n}}(\psi')]\!](\Omega)$$

Than, for any index $k \geqslant i$ we can apply the inductive hypothesis.

- $\psi = \psi_1 \,\mathsf{W}\, \psi_2$. Here we split the proof in two parts ($\Rightarrow$ and $\Leftarrow$).

($\Rightarrow$) We assume the premise $\langle \sigma, i\rangle \models_{\text{LTL}} \psi_1 \,\mathsf{W}\, \psi_2$ and we obtain

$$\langle \sigma, i\rangle \models_{\text{LTL}} \psi_1 \,\mathsf{W}\, \psi_2 \iff \begin{cases} \langle \sigma, i\rangle \models_{\text{LTL}} \psi_2 \text{ (1)} & \text{or} \\ \langle \sigma, i\rangle \models_{\text{LTL}} \psi_1 \text{ and} \\ \quad \langle \sigma, i+1\rangle \models_{\text{LTL}} \psi_1 \,\mathsf{W}\, \psi_2 \end{cases} \quad (2)$$

Hence, other two sub-cases arise

(1) Assuming $\langle \sigma, i\rangle \models_{\text{LTL}} \psi_2$ we must prove that

$$\sigma \models_{\text{ETL}} \exists_{t:t\geqslant\mathbf{i}}.\Big(\mathcal{T}_t(\psi_2) \wedge \forall_{t':\mathbf{i}\leqslant t'<t}.\mathcal{T}_{t'}(\psi_1)\Big) \vee$$

$$\forall_{t'':\mathbf{i}\leqslant t''}.\mathcal{T}_{t''}(\psi_1)$$

However, mapping t to $\mathbf{i}$ we reduce to

$$\sigma \models_{\text{ETL}} (\mathcal{T}_{\mathbf{i}}(\psi_2) \ \wedge \ \forall_{t':\mathbf{i}\leqslant t'<\mathbf{i}}.\mathcal{T}_{t'}(\psi_1)) \vee$$

$$\forall_{t'':\mathbf{i}\leqslant t''}.\mathcal{T}_{t''}(\psi_1)$$

that we can rewrite as

$$\sigma \models_{\text{ETL}} \mathcal{T}_{\mathbf{i}}(\psi_2) \vee \forall_{t'':\mathbf{i}\leqslant t''}.\mathcal{T}_{t''}(\psi_1)$$

Since, from the inductive hypothesis, we know that $\sigma \models_{\text{ETL}} \mathcal{T}_{\mathbf{i}}(\psi_2)$ the property is satisfied.

(2) Here we assume $\langle \sigma, i\rangle \models_{\text{LTL}} \psi_1$ and $\langle \sigma, i{+}1\rangle \models_{\text{LTL}} \psi_1 \,\mathsf{W}\, \psi_2$. Applying the inductive hypothesis we infer that $\sigma \models_{\text{ETL}} \mathcal{T}_{\mathbf{i}}(\psi_1)$ and $\sigma \models_{\text{ETL}} \mathcal{T}_{\mathbf{s(i)}}(\psi_1 \,\mathsf{W}\, \psi_2)$. From this we deduce that

$$\sigma \models_{\text{ETL}} \exists_{t:\mathbf{s(i)}\leqslant t}.\Big(\mathcal{T}_t(\psi_2) \wedge \forall_{t':\mathbf{s(i)}\leqslant t'<t}.\mathcal{T}_{t'}(\psi_1)\Big) \vee$$

$$\forall_{t'':\mathbf{s(i)}\leqslant t''}.\mathcal{T}_{t''}(\psi_1)$$

Other two cases arise. On one side, if

$$\sigma \models_{\text{ETL}} \forall_{t'':\mathbf{s(i)}\leqslant t''}.\mathcal{T}_{t''}(\psi_1)$$

since $\sigma \models_{\text{ETL}} \mathcal{T}_{\mathbf{i}}(\psi_1)$ we conclude that

$$\sigma \models_{\text{ETL}} \forall_{t'':\mathbf{i}\leqslant t''}.\mathcal{T}_{t''}(\psi_1)$$

On the other side, from

$$\sigma \models_{\text{ETL}} \exists_{t:\mathbf{s(i)}\leqslant t}.\Big(\mathcal{T}_t(\psi_2) \wedge \forall_{t':\mathbf{s(i)}\leqslant t'}.t\mathcal{T}_{t'}(\psi_1)\Big)$$

we can also deduce

$$\sigma \models_{\text{ETL}} \exists_{t:\mathbf{i}\leqslant t}.\Big(\mathcal{T}_t(\psi_2) \wedge \forall_{t':\mathbf{s(i)}\leqslant t'}.t\mathcal{T}_{t'}(\psi_1)\Big)$$

However, since we know that $\sigma \models_{\text{ETL}} \mathcal{T}_{\mathbf{i}}(\psi_1)$ we can conclude that

$$\sigma \models_{\text{ETL}} \exists_{t:\mathbf{i}\leqslant t}.\Big(\mathcal{T}_t(\psi_2) \wedge \forall_{t':\mathbf{i}\leqslant t'}.t\mathcal{T}_{t'}(\psi_1)\Big)$$

($\Leftarrow$) Here we assume $\sigma \models_{\text{ETL}} \mathcal{T}_{\mathbf{i}}(\psi_1 \,\mathsf{W}\, \psi_2)$ or, equivalently

$$\sigma \models_{\text{ETL}} \exists_{t:\mathbf{i}\leqslant t}.\Big(\mathcal{T}_t(\psi_2) \wedge \forall_{t':\mathbf{i}\leqslant t'}.t\mathcal{T}_{t'}(\psi_1)\Big) \vee$$

$$\forall_{t'':\mathbf{i}\leqslant t''}.\mathcal{T}_{t''}(\psi_1) \quad \text{(A)}$$

and we must prove

$$\langle \sigma, i\rangle \models_{\text{LTL}} \psi_1 \,\mathsf{W}\, \psi_2 \iff \begin{cases} \langle \sigma, i\rangle \models_{\text{LTL}} \psi_2 & \text{or} \\ \langle \sigma, i\rangle \models_{\text{LTL}} \psi_1 \text{ and} \\ \quad \langle \sigma, i+1\rangle \models_{\text{LTL}} \psi_1 \,\mathsf{W}\, \psi_2 \end{cases}$$

However, unfolding one element from the existen-

tial quantifier of (A) we obtain

$$\sigma \models_{\text{ETL}} \mathcal{T}_{\mathbf{i}}(\psi_2) \vee \exists_{t:\mathbf{s(i)}\leqslant t}.\Big(\mathcal{T}_t(\psi_2) \wedge \forall_{t':\mathbf{i}\leqslant t'}.t\mathcal{T}_{t'}(\psi_1)\Big)$$

$$\vee$$

$$\forall_{t'':\mathbf{i}\leqslant t''}.\mathcal{T}_{t''}(\psi_1) \quad \text{(B)}$$

That is, σ satisfies the disjunction of three predicates. If $\sigma \models_{\text{ETL}} \mathcal{T}_{\mathbf{i}}(\psi_2)$ the property is trivially satisfied by the inductive hypothesis. Then, we reduce to prove the second part of (B), that is

$$\sigma \models_{\text{ETL}} \exists_{t:\mathbf{s(i)}\leqslant t}.\Big(\mathcal{T}_t(\psi_2) \wedge \forall_{t':\mathbf{i}\leqslant t'}.t\mathcal{T}_{t'}(\psi_1)\Big) \vee$$

$$\forall_{t'':\mathbf{i}\leqslant t''}.\mathcal{T}_{t''}(\psi_1) \quad \text{(C)}$$

However, we note that (C) implies

$$\left.\begin{array}{c}\sigma \models_{\text{ETL}} \exists_{t:\mathbf{s(i)}\leqslant t}.\Big(\mathcal{T}_t(\psi_2) \wedge \mathcal{T}_{\mathbf{i}}(\psi_1) \wedge \\ \forall_{t':\mathbf{s(i)}\leqslant t'}.t\mathcal{T}_{t'}(\psi_1)\Big) \\ \vee \\ \mathcal{T}_{\mathbf{i}}(\psi_1) \wedge \forall_{t'':\mathbf{s(i)}\leqslant t''}.\mathcal{T}_{t''}(\psi_1)\end{array}\right\} \quad \text{(D)}$$

Moreover (D) implies that

$$\begin{array}{c}\sigma \models_{\text{ETL}} \exists_{t:\mathbf{s(i)}\leqslant t}.\mathcal{T}_{\mathbf{i}}(\psi_1) \\ \wedge \\ \exists_{t:\mathbf{s(i)}\leqslant t}.\Big(\mathcal{T}_t(\psi_2) \wedge \forall_{t':\mathbf{s(i)}\leqslant t'}.t\mathcal{T}_{t'}(\psi_1)\Big) \\ \vee \\ \mathcal{T}_{\mathbf{i}}(\psi_1) \wedge \forall_{t'':\mathbf{s(i)}\leqslant t''}.\mathcal{T}_{t''}(\psi_1)\end{array}$$

that is equivalent to (since t does not occur in $\mathcal{T}_{\mathbf{i}}(\psi_1)$)

$$\begin{array}{c}\sigma \models_{\text{ETL}} \mathcal{T}_{\mathbf{i}}(\psi_1) \\ \wedge \\ \exists_{t:\mathbf{s(i)}\leqslant t}.\Big(\mathcal{T}_t(\psi_2) \wedge \forall_{t':\mathbf{s(i)}\leqslant t'}.t\mathcal{T}_{t'}(\psi_1)\Big) \\ \vee \\ \mathcal{T}_{\mathbf{i}}(\psi_1) \wedge \forall_{t'':\mathbf{s(i)}\leqslant t''}.\mathcal{T}_{t''}(\psi_1)\end{array}$$

and to

$$\sigma \models_{\text{ETL}} \mathcal{T}_{\mathbf{i}}(\psi_1) \wedge \mathcal{T}_{\mathbf{s(i)}}(\psi_1 \,\mathsf{W}\, \psi_2)$$

that is the thesis.

□

LEMMA 1. *For each ETL formula ϕ and set of traces A*

$$[\![\phi]\!](A) \subseteq A$$

PROOF. We proceed by induction on ϕ.

- $\phi = \top$. Trivial.
- $\phi = \alpha_\tau$. A direct consequence of the definition of $[\![\alpha_\tau]\!]$.
- $\phi = \neg\phi'$. By definition $[\![\neg\phi']\!](A) = A \setminus [\![\phi']\!](A) \subseteq A$.
- $\phi = \phi_1 \wedge \phi_2$. By the inductive hypothesis we know that $\forall B.[\![\phi_1]\!](B) \subseteq B$ and $\forall C.[\![\phi_2]\!](C) \subseteq C$. Then, we set $B = A$ and $C = [\![\phi_1]\!](A)$ and we note that $[\![\phi_2]\!]([\![\phi_1]\!](A)) \subseteq [\![\phi_1]\!](A) \subseteq A$.
- $\phi = \forall_{t:p(t)}.\phi'$. By contradiction we assume that $[\![\phi]\!](A) \not\subseteq A$, that is $\exists\sigma.\sigma \in [\![\phi]\!](A) \wedge \sigma \notin A$. This implies that there exist B and m such that $\vdash p(\mathbf{m})$, $\sigma \notin B$ and $\sigma \in [\![\phi'[{}^{\mathbf{m}}/_t]]\!](B)$. However, applying the inductive hypothesis we find that for every A $[\![\phi'[{}^{\mathbf{m}}/_t]]\!](A) \subseteq A$.

□

THEOREM 2. *For each ETL formula ϕ and traces set A*

$$A \models_{\text{ETL}} \phi \iff A = [\![\phi]\!](A)$$

PROOF. We proceed by induction on the structure of ϕ.

- $\phi = \top$. Trivial.
- $\phi = \alpha_\tau$. Here we have $A \subseteq [\![\phi]\!](\Omega) = \{\sigma : \sigma[\mathsf{N}(\tau)] = \alpha\}$ from which the thesis follows.
- $\phi = \neg\phi'$. We note that $A \subseteq [\![\phi]\!](\Omega) = \Omega \setminus [\![\phi']\!](\Omega)$. However, by Lemma 1 we know that $[\![\phi']\!](A) \subseteq A$ from which we deduce $[\![\phi']\!](A) = \emptyset$ and then the thesis.
- $\phi = \phi_1 \wedge \phi_2$. Here we know that $A \subseteq [\![\phi_2]\!]([\![\phi_1]\!](A))$. By Lemma 1 we infer that both $[\![\phi]\!](A) \subseteq [\![\phi_1]\!](A)$ and $[\![\phi]\!](A) \subseteq [\![\phi_2]\!](A)$. Now we apply the inductive hypothesis to both ϕ_1 and ϕ_2 and we conclude observing that $[\![\phi_2]\!]([\![\phi_1]\!](A)) = [\![\phi_2]\!](A) = A$.
- $\phi = \forall_{t:p(t)}.\phi'$. By definition of compliance we know that $A \subseteq [\![\phi]\!](\Omega) = \bigcirc\limits_{\vdash p(\mathbf{n})}^{n} [\![\phi'[{}^{\mathbf{n}}/_t]]\!](\Omega)$. Moreover, by Lemma 1 we obtain that arbitrarily choosing a value k such that $\vdash p(\mathbf{k})$ we have $\bigcirc\limits_{\vdash p(\mathbf{n})}^{n} [\![\phi'[{}^{\mathbf{n}}/_t]]\!](\Omega) \subseteq [\![\phi'[{}^{\mathbf{k}}/_t]]\!](\Omega)$. Then we apply the inductive hypothesis to $\phi'[{}^{\mathbf{k}}/_t]$ and we find $[\![\phi]\!] = \bigcirc\limits_{\vdash p(\mathbf{n})}^{n} I_A^n$ where each I_A^i is a function behaving as the identity on A.

□

LEMMA 2. *Given an ETL formula ϕ and a trace σ such that $E(\sigma) = n$, then for each action α*

$$\left.\begin{array}{c}(\sigma \models_{\text{ETL}} \phi \ \wedge\ \sigma(\alpha : m) \not\models_{\text{ETL}} \phi) \\ \vee \\ (\sigma \not\models_{\text{ETL}} \phi \ \wedge\ \sigma(\alpha : m) \models_{\text{ETL}} \phi)\end{array}\right\} \Longrightarrow m \geqslant \chi_n(\phi)$$

PROOF. By induction on the structure of ϕ.

- $\phi = \top$. Trivial.
- $\phi = \alpha_\tau$. We note that, by definition, $(\sigma \models_{\text{ETL}} \phi \wedge \sigma(\alpha : m) \not\models_{\text{ETL}} \phi) \vee (\sigma \not\models_{\text{ETL}} \phi \wedge \sigma(\alpha : m) \models_{\text{ETL}} \phi)$ holds if and only if $\sigma \not\models_{\text{ETL}} \phi \wedge \sigma(\alpha : m) \models_{\text{ETL}} \phi$. Then, $\sigma(\alpha : m)[\mathsf{N}(\tau)] = \alpha$, but since for each $m' \neq m$ we have that $\sigma(\alpha : m)[m'] = \sigma[m']$ the only possibility is $m = \mathsf{N}(\tau) = \chi_n(\alpha_\tau)$.
- $\phi = \neg\phi'$. We have two symmetrical cases and we show the proof for the first one. Assuming by hypothesis that $\sigma \models_{\text{ETL}} \neg\phi \wedge \sigma(\alpha : m) \not\models_{\text{ETL}} \neq \phi$ holds, we deduce that $\sigma \not\models_{\text{ETL}} \phi \wedge \sigma(\alpha : m) \models_{\text{ETL}} \phi$ and we apply the inductive hypothesis to ϕ'. The thesis follows from $\chi_n(\neg\phi') = \chi_n(\phi')$

- $\phi = \phi_1 \wedge \phi_2$. Again we prove one of two cases. If $(\sigma \models_{\text{ETL}} \phi_1 \wedge \phi_2) \wedge (\sigma(\alpha : m) \not\models_{\text{ETL}} \phi_1 \wedge \phi_2)$ then we have that $\sigma \models_{\text{ETL}} \phi_1$ and $\sigma \models_{\text{ETL}} \phi_2$ from the left side. Moreover, from the right side, we have other two subcases: either $\sigma(\alpha : m) \not\models_{\text{ETL}} \phi_1$ or $\sigma(\alpha : m) \not\models_{\text{ETL}} \phi_2$. The two cases are symmetrical and we show only the first one. We assume $\sigma(\alpha : m) \not\models_{\text{ETL}} \phi_1$. Now, by the inductive hypothesis we infer that $\chi_n(\phi_1) \leqslant m$. By definition, we conclude observing that $\chi_n(\phi_1 \wedge \phi_2) \leqslant \chi_n(\phi_1) \leqslant m$.
- $\phi = \forall_{t:p(t)}.\phi'$. Here we have two possible cases:
  1. We assume $\sigma \models_{\text{ETL}} \phi \wedge \sigma(\alpha : m) \not\models_{\text{ETL}} \phi$. By definition of χ we know that
  $$\chi_n(\forall_{t:p(t)}.\phi') = \min_{i:\vdash p(\mathbf{i})} \{\chi_n(\phi'[^{\mathbf{i}}/_t])\}$$
  Moreover, by definition of compliance we can derive from $\sigma(\alpha : m) \not\models_{\text{ETL}} \phi$ that there exists some k such that $\vdash p(\mathbf{k})$ and $\sigma(\alpha : m) \not\models_{\text{ETL}} \phi'[^{\mathbf{k}}/_t]$. However, we also know that $\sigma \models_{\text{ETL}} \phi'[^{\mathbf{k}}/_t]$. Hence, we conclude noting that
  $$m \geqslant \chi_n(\phi'[^{\mathbf{k}}/_t]) \geqslant \min_{i:\vdash p(\mathbf{i})} \{\chi_n(\phi'[^{\mathbf{i}}/_t])\} = \chi_n(\phi)$$
  2. Here we assume $\sigma \not\models_{\text{ETL}} \phi \wedge \sigma(\alpha : m) \models_{\text{ETL}} \phi$. Similarly to the case above, we have that $\sigma \not\models_{\text{ETL}} \phi$ implies the existence of some k such that $\vdash p(\mathbf{k})$ and $\sigma \not\models_{\text{ETL}} \phi'[^{\mathbf{k}}/_t]$. Since $\sigma(\alpha : m) \models_{\text{ETL}} \phi'[^{\mathbf{k}}/_t]$ we conclude as before.

□

LEMMA 3. *For each ETL formula ϕ and for all m and n, if $m \geqslant n$ then $\chi_m(\phi) \geqslant \chi_n(\phi)$.*

PROOF. By induction over the structure of ϕ.

- $\phi = \top$. Trivial.
- $\phi = \alpha_\tau$. Then we have
  $$\chi_m(\alpha_\tau) = \begin{cases} \mathsf{N}(\tau) & \text{if } m < \mathsf{N}(\tau) \\ \infty & \text{otherwise} \end{cases}$$
  and
  $$\chi_n(\alpha_\tau) = \begin{cases} \mathsf{N}(\tau) & \text{if } n < \mathsf{N}(\tau) \\ \infty & \text{otherwise} \end{cases}$$
  Hence we have three cases: (1) $\mathsf{N}(\tau) < n$, (2) $n \leqslant \mathsf{N}(\tau) < m$ or (3) $m \leqslant \mathsf{N}(\tau)$. It's easy to check that the property holds for all of them.
- $\phi = \neg\phi'$. The property immediately follows from the inductive hypothesis.
- $\phi = \phi_1 \wedge \phi_2$. By definition we have that $\chi_n(\phi_1 \wedge \phi_2) = \min\{\chi_n(\phi_1), \chi_n(\phi_2)\}$. Applying the inductive hypothesis to both ϕ_1 and ϕ_2 we infer that $\chi_n(\phi_1 \wedge \phi_2) \leqslant \chi_n(\phi_1) \leqslant \chi_m(\phi_1)$ and $\chi_n(\phi_1 \wedge \phi_2) \leqslant \chi_n(\phi_2) \leqslant \chi_m(\phi_2)$. Thus $\chi_n(\phi_1 \wedge \phi_2) \leqslant \min\{\chi_m(\phi_1), \chi_m(\phi_2)\} = \chi_m(\phi_1 \wedge \phi_2)$.
- $\phi = \forall_{t:p(t)}.\phi'$. Here we have
  $$\chi_n(\forall_{t:p(t)}.\phi') = \min_{i:\vdash p(\mathbf{i})} \{\chi_n(\phi'[^{\mathbf{i}}/_t])\}$$
  and
  $$\chi_m(\forall_{t:p(t)}.\phi') = \min_{i:\vdash p(\mathbf{i})} \{\chi_m(\phi'[^{\mathbf{i}}/_t])\}$$
  Then there must be some k such that $\chi_m(\forall_{t:p(t)}.\phi') = \chi_m(\phi'[^{\mathbf{k}}/_t])$. Applying the inductive hypothesis we conclude that
  $$\chi_m(\forall_{t:p(t)}.\phi') = \chi_m(\phi'[^{\mathbf{k}}/_t]) \geqslant \chi_n(\phi'[^{\mathbf{k}}/_t]) \geqslant \chi_n(\forall_{t:p(t)}.\phi')$$

□

LEMMA 4. *For each ETL formula ϕ if $\chi_n(\phi) = m$ then for each $n < n^* < m$ holds that $\chi_{n^*}(\phi) = m$.*

PROOF. By induction over ϕ.

- $\phi = \top$. Trivial.
- $\phi = \alpha_\tau$. We have two cases. If $n \geqslant \mathsf{N}(\tau)$ then $\chi_n(\alpha_\tau) = \infty$ and the property is trivially satisfied. Otherwise we have that $m = \mathsf{N}(\tau)$ and the property holds by definition of χ.
- $\phi = \neg\phi'$. Trivially, by inductive hypothesis.
- $\phi = \phi_1 \wedge \phi_2$. By definition we have $\chi_n(\phi_1 \wedge \phi_2) = \min\{\chi_n(\phi_1), \chi_n(\phi_2)\}$. Lets call $m_1 = \chi_n(\phi_1)$ and $m_2 = \chi_n(\phi_2)$ and assume $m_1 \leqslant m_2$ (we do not show the other way round that is symmetric). We apply the inductive hypothesis to ϕ_1 and we obtain that if $n < n^* < m_1$ then $\chi_{n^*}(\phi_1) = m_1$. However, since $m_1 \leqslant m_2$ the property also holds for ϕ_2. Thus, we can conclude noting that $\chi_{n^*} = \min\{\chi_{n^*}(\phi_1), \chi_{n^*}(\phi_2)\} = m_1$.
- $\phi = \forall_{t:p(t)}.\phi'$. We instantiate the property to
  $$\chi_n(\phi) = \chi_n(\forall_{t:p(t)}.\phi') = \min_{i:\vdash p(\mathbf{i})} \{\chi_n(\phi'[^{\mathbf{i}}/_t])\} = m$$
  and we want to prove that for each $n < n^* < m$ holds that $\chi_{n^*}(\phi) = m$. Then, we show that assuming that there exists some n^* violating the property leads to a contradiction. Indeed, if such a n^* exists we have that $\chi_{n^*}(\phi) = m^* < m$. Hence, there must be some k such that $\chi_{n^*}(\phi'[^{\mathbf{k}}/_t]) = m^*$. From the inductive hypothesis we deduce that $\chi_n(\phi'[^{\mathbf{k}}/_t]) = m$. However, by Lemma 3 we know that $\chi_{n^*}(\phi'[^{\mathbf{k}}/_t]) \geqslant \chi_n(\phi'[^{\mathbf{k}}/_t])$, i.e. $m^* \geqslant m$.

□

THEOREM 3. *Let ϕ be an ETL formula and σ a trace such that $\sigma \models_{\text{ETL}} \phi$ and $E(\sigma) = n$ then for each m such that $n < m < \chi_n(\phi)$ and action α*
$$\sigma(\alpha : m) \models_{\text{ETL}} \phi$$

PROOF. By contradiction, we assume that under the hypothesis of the theorem there is some $m < \chi_n(\phi)$ such that $\sigma(\alpha : m) \not\models_{\text{ETL}} \phi$. However, by Lemma 2, we have that $\sigma \models_{\text{ETL}} \phi \wedge \sigma(\alpha : m) \not\models_{\text{ETL}} \phi$ implies that $m \geqslant \chi_n(\phi)$.

□