A Timed Graphical Interval Logic

***Abstract*—We define a graphical language for expressing timed requirements on concurrent systems. This formal language, called *Timed Graphical Interval Logic* (TGIL), is inspired by realtime extensions of Dillon's et al Graphical Interval Logic and can be used as an alternative to timed extensions of temporal logic. We define the semantics of TGIL as a set of timed traces—using a dense time semantics—and illustrate its use in formal verification by describing a method for generating an observer from a TGIL specification.**

I. Introduction

An issue limiting the adoption of model-checking technologies by the industry is the ability, for non-experts, to express their requirements using the low-level languages used by model-checkers. In this paper, we define a Timed Graphical Interval Logic (TGIL), that is a formal graphical notation for expressing the timing constraints and behavioral properties of a reactive system.

Engineers frequently use diagrams to explain the behavior of a system or to describe desired scenarios. Nonetheless, such drawings usually suffer from the same drawbacks than requirements described using natural language: they can be ambiguous or misleading; they are not precise enough (do not cover all the cases); they are not amenable to automated transformation; . . . In this work, we take our inspiration from an existing graphical notation, the Graphical Interval Logic (GIL) of Dillon et al [4], and extends it with two operators for expressing timing constraints. We show, with a simple example, that TGIL is more expressive than another realtime extension of GIL [6] that was already proposed.

Together with the definition of a formal semantics for TGIL, the main contribution of this work is to describe a method for generating "an observer" from a TGIL specification. These observers can be used in conjunction with a model-checking tool to check that the specification is valid on a given system.

Our main motivation in the design of TGIL was to define the semantics of a set of realtime specification patterns [2] using a graphical notation. This set of patterns extends the specification language of Dwyer et al. [5] with the ability to express hard, realtime constraints commonly found in the analysis of real-time systems. For example, the timed pattern "Present A after B within $[d_1, d_2)$" express the requirement that event A must occur within d_1 units of time (u.t.) of the first occurrence of B, if any, but not later than d_2. The semantics of patterns has already been defined using Metric Temporal Logic (MTL), a timed extension of linear temporal logic [7]. The idea is to provide an alternative formal definition based on TGIL.

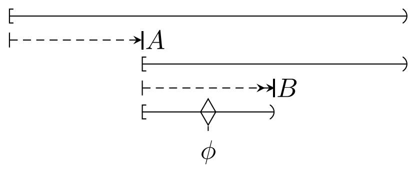
We believe that this new approach may ease the work of engineers that are not trained with formal verification techniques. Moreover, our experience shows that being able to confront different definitions for the same pattern, using contrasting approaches, is useful for teaching patterns.

The remainder of the paper is as follows. Next, we describe the graphical notation for TGIL and define an equivalent textual syntax. In Section III, we define the semantics of TGIL using a satisfaction relation over timed traces. Before concluding, we study the expressiveness of our logic and describe a method for using TGIL as the property specification language in a model-checking problem.

II. Timed Graphical Interval Logic

We consider the problem of expressing behavioral properties and timing constraints on the execution of a reactive system. We assume that the execution of the system can be described using a combination of events and time delays. (We use letters $A, B, \ldots$ to denote a predicate on events.) Events should be understood as instantaneous actions involved in the evolution of the system: it can be an observable transition in the system; a process that changes its state; etc.

TGIL can be viewed as a real-time extension of the Graphical Interval Logic (GIL) of Dillon et al. [4]. An example of GIL diagram—that is also in TGIL—is given below.



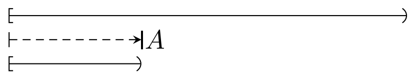
A TGIL specification, or formula, is a diagram that should be read from top to bottom and from left to right. Our example depicts three main notions used in TGIL. For each notion, we briefly describe its graphical notation and propose an equivalent textual syntax.

1) Execution Context: every formula is expressed with respect to an execution context, displayed with a straight line $\vdash\!\!\longrightarrow$, that represents a portion of an execution trace—a time interval—where the property is evaluated. The initial, top-most context symbolizes the whole execution trace, that is the time interval $[0, +\infty[$.

In this short abstract we only consider contexts that are "closed at their beginning and open at their end." This convention, that is also followed in GIL, simplify the presentation and may help avoid problematic examples, such as Zeno behaviors. Nonetheless, our notation can be extended to handle unrestricted types of contexts.

2) Search: formulas and sub-contexts are built from searches, that define instants matching a given constraint in the current context; searches are displayed with a dashed arrow $\vdash\dashrightarrow$ and are decorated with (a predicate on) events. In our example, the first search starts from the beginning of the initial context (thus at time 0) and define the first time instant in the context where an event A occurs (say t_A). The second execution context is defined by the result of this search; it starts at time t_A.

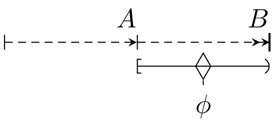
A search can be combined with a context in order to define a sub-context: from a context and a search point, say S, we can define the context located after S (as we do in our example) or the context located before it (as shown with the diagram below). We use the notation $[\rightarrow A|\,\phi\rangle$ for describing the first kind of context and the notation $\langle\phi|\rightarrow A]$ for the other kind, where ϕ is a TGIL specification.



There are two kind of searches in our graphical notation: a weak search ($\vdash\dashrightarrow$) and a strong search ($\vdash\dashrightarrow\!\!\!\rightarrow$) version. With a strong search, $[\twoheadrightarrow A|\,\phi\rangle$, the formula is false if we fail to find an event matching A in the current context (if the search fails). At the opposite, with a weak search, the formula is true if the search fails. We show how to define the weak search version as a derived operator in our logic.

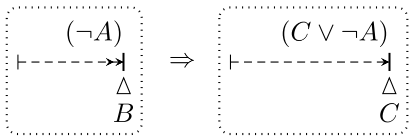
3) Formulas: a TGIL specification associates properties to contexts and search points in the diagram. For instance, the last (bottom) element in our example states that the formula ϕ should be true somewhere/sometimes in the context $[t_A, t_B[$, where t_A and t_B are the dates associated to the pair of events (A, B) matched in the previous searches. In our case, ϕ can be a formula—expressed using another TGIL diagram—or simply a predicate on events. (The instant where ϕ is true is materialized by the diamond, like in a search.)

As a consequence, our running example can be interpreted as follows: look for the first occurrence of an event A. If there is any then find the following occurrence of B. If no such occurrence exists the property is false. Finally, find an event, "in-between", where ϕ holds. For concision, we omit intermediate contexts when they can be inferred from the diagram. Thus, our example can be equivalently drawn:



The textual equivalent of this requirement could be written $[\rightarrow A|\,\langle(\Diamond\,\phi)|\twoheadrightarrow B]\rangle$.

TGIL also provides a construction for expressing "punctual properties"—depicted using a triangle under a search point—that is a property relevant at this given instant in a context. For example, the first (boxed) part of the TGIL diagram below states that, at the instant A is false, then B is true (since it is a strong search, it also states that B must eventually happens). The textual equivalent of this formula is $[\twoheadrightarrow(\neg A)|\,B\rangle$.

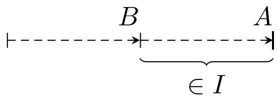


As we show in the previous diagram, TGIL formulas can be combined using boolean connectives and grouping—the graphical equivalent of parentheses—drawn using a boxed rectangle.

4) Timing Constraints: finally, TGIL provides two operators for adding timing constraints on formulas: an operator that bounds the delay between two instants; and an operator that restrict a context to a given time interval.

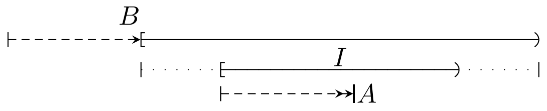
In the following, we use the symbol I as a shorthand for the time interval $[d_1, d_2)$.

The first operator, called *time length*, uses a "curly braces" notation, illustrated below. It states that the delay between the two instants materialized by the end of the brace is in I.



We can use this operator to constrain the length (time duration) of a context. In particular, when the two instants are the boundaries of the same search—as it is the case in our example—we use the textual notation $[\rightarrow_I A|\ldots\rangle$ to state that the length of the search $\rightarrow A$ is in I. This restricted operator, denoted *Len*(I) in [6], is the only timing operator that was considered in a previous timed extension of GIL.

The second operator, *time restriction*, limits an execution context to a given time interval I (relatively to the starting point of the context). This operator is drawn using a combination of dotted and solid line. We give, as an example, the specification for the Present pattern described in Sect. I.



This formula can be interpreted as follows: assume that I_C is the time interval corresponding to the current execution context. If t_A is the instant of the first occurrence of A in I_C then search the first occurrence of B in the context $I_C \cap [t_A + d_1, t_A + d_2)$. If I_C is bounded, its time restriction may be empty, in which case all posterior searches will fail.

To the best of our knowledge, the time restriction operator is totally new in the context of GIL. This operator is necessary to define the pattern Present A after B within I that we described in the introduction. Indeed, equipped with the *Len* operator alone, we can only detect if the first A following a B is within I.

III. Formal Semantics

The semantics of a TGIL formula, ϕ, is defined as the set of *timed traces* that holds for ϕ. A timed trace σ is a (possibly infinite) sequence of events and duration $d(\delta)$ with $\delta \in \mathbb{Q}^+$. In the following, we use ϵ to denote the empty trace and, given a finite trace σ and a—possibly infinite—trace σ', we denote $\sigma\sigma'$ the concatenation of σ and σ'. This operation is associative.

The *duration* of a finite trace σ, denoted $\Delta(\sigma)$, is the sum of all its time delays. We extend this definition to infinite traces by defining $\Delta(\sigma)$ as the limit of $\Delta(\sigma_i)$ where σ_i are growing prefixes of σ.

A TGIL specification can only be satisfied by execution traces that are either finite or that cannot block the passing of time: we say that a trace σ is *well-formed* if and only if $\mathsf{dom}(\sigma)$ is finite or $\Delta(\sigma) = \infty$. With this restriction, we avoid problematic behaviors, such that an infinite number of events can occur in finite time, without forbidding time divergence. More generally, we consider execution traces up-to "time equivalence", $\equiv$, that is the largest congruence such that the traces $d(\delta_1 + \delta_2)$ and $d(\delta_1)d(\delta_2)$ are equivalent. This relation preserves duration and guarantees that a well-formed trace is only equivalent to other well-formed traces.

We consider a finite set of propositional variables, $A, B, \ldots$, that denote "atomic properties" of events $\omega \in \Omega$. We use the expression $\omega \in A$ to denote that the proposition A is true for ω. (By extension, we should also use A to denote a predicate over propositional variables.)

We define the semantics of TGIL using our textual syntax. Besides the propositional fragment, the main operators are the punctual formula (A), the left and right searches; the sometimes modality ($\Diamond$) and time restriction ($\backslash_I$).

$$\begin{aligned} \phi \quad ::= & \neg\phi \mid \phi_1 \vee \phi_2 \mid A \\ & \mid [\twoheadrightarrow_I A \mid \phi\rangle \mid \langle\phi \mid \twoheadrightarrow_I A] \mid (\Diamond\, \phi) \mid (\backslash_I\, \phi) \end{aligned}$$

We use the satisfaction relation $\sigma \models \phi$ to denote that the formula ϕ holds for σ. In this definition, we use σ_I to denote the sub-trace of σ restricted to the time interval I and the notation $\sigma \equiv A\sigma$ as a shorthand for the condition $(\sigma \equiv \omega\sigma') \wedge (\omega \in A)$.

$$\begin{array}{lll} \sigma \models \neg\phi & \text{iff} & \text{not } \sigma \models \phi \\ \sigma \models \phi_1 \vee \phi_2 & \text{iff} & (\sigma \models \phi_1) \vee (\sigma \models \phi_2) \\ \sigma \models A & \text{iff} & \sigma \equiv A\sigma' \\ \sigma \models [\twoheadrightarrow_I A \mid \phi\rangle & \text{iff} & \sigma \equiv \sigma_1 A \sigma_2 \wedge A \notin \sigma_1 \\ & & \wedge\, \Delta(\sigma_1) \in I \wedge \sigma_2 \models \phi \\ \sigma \models \langle\phi \mid \twoheadrightarrow_I A] & \text{iff} & \sigma \equiv \sigma_1 A \sigma_2 \wedge A \notin \sigma_1 \\ & & \wedge\, \Delta(\sigma_1) \in I \wedge \sigma_1 \models \phi \\ \sigma \models (\Diamond\, \phi) & \text{iff} & \exists \sigma_1, \sigma_2 \,.\, \sigma \equiv \sigma_1\sigma_2 \wedge \sigma_2 \models \phi \\ \sigma \models (\backslash_I\, \phi) & \text{iff} & \sigma_I \models \phi \end{array}$$

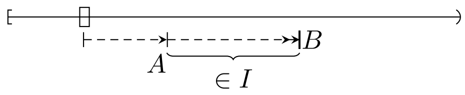
This definition is quite similar to the satisfaction relation for Linear Temporal Logic (LTL). In particular, TGIL is an instance of a linear time logic, in the sense that it cannot be used to reason on "several possible timelines" simultaneously.

IV. Derived Operators and Synthesis of Observers

We can define additional logical operators that are useful for defining properties. The "true" formula, $\top$, can be encoded by any tautology, such as $A \vee \neg A$. Another example of derived formula is $(\backslash_I\, (\Diamond\, \phi))$, which defines a property that is satisfied if ϕ holds sometimes in I. We use the notation $(\Diamond_I\, \phi)$, for this derived operator, to stress the direct relationship with MTL. The dual of the sometimes operator, $(\Box_I\, \phi) \overset{\text{def}}{=} \neg(\Diamond_I\, (\neg\phi))$, holds for the traces such that ϕ is always true in I.

It is possible to derive the weak search operator from the strong search version. Indeed, the formula $[\rightarrow_I A \mid \phi\rangle$ is true if and only if $[\twoheadrightarrow_I A \mid \phi\rangle$ is true or the search for A fails in the context $\backslash_I$. Put another way, if we can find A in $\backslash_I$ then $[\twoheadrightarrow_I A \mid \phi\rangle$ should be true: $[\rightarrow_I A \mid \phi\rangle \overset{\text{def}}{=} (\Diamond_I\, A) \Rightarrow [\twoheadrightarrow_I A \mid \phi\rangle$.

We already showed how to use TGIL for defining the semantics of the $\mathsf{Present}$ pattern (see [2] for a complete catalog of timed patterns). We now look at an example of *response pattern*, used to express "cause–effect" relationship, such as the fact that a triggering event must be followed by a response in a bounded time. The pattern A $\mathsf{leadsto}$ B within I holds for all timed traces where every occurrence of A is followed by an occurrence of B within I (we only consider the first occurrence of B after A). Alternatively, we can define the semantics of the $\mathsf{leadsto}$ pattern with the diagram:



that is with the formula: $(\Box\, [\rightarrow A \mid [\twoheadrightarrow_I B \mid \top\rangle\rangle)$. Using our satisfaction relation "as a substitute" for a proof system, we can show that this definition is equivalent to the following, optimized formula, that uses one less search: $(\Box\, (\neg A \vee [\twoheadrightarrow_I B \mid \top\rangle))$.

Next, we describe a method for using TGIL as the property specification language for model-checking. We follow an observer-based approach, meaning that the relationship between a model and its specification is interpreted as the composition of the model with an observer of its behavior. More precisely, we consider systems defined using Time Transition Systems (TTS), an extension of Time Petri Nets with data variables and priorities. (See e.g. [1] for the semantics of TTS.) TTS models can be checked using *selt*, an SE-LTL model-checker provided in the *Tina* toolbox (http://projects.laas.fr/tina/).

The idea is to synthesize a TTS model (an observer) from a TGIL specification; to generate the state space of the system composed with its observer; and to test the satisfaction of a simple reachability property. Due to space limitation, we only show the result of applying our method on a specific example, the $\mathsf{Present}$ pattern: $[\rightarrow B \mid (\backslash_I\, [\twoheadrightarrow A \mid \top\rangle)\rangle$. The model-checking problem for TGIL, in its entirety, is undecidable. Nonetheless, the method that we describe here could be applied to any "positive" formulas, that is formulas without negation.

We define some conventions used when defining the observers for the formula $[\rightarrow B \mid (\backslash_{[d_1,d_2)}\, [\twoheadrightarrow A \mid \top\rangle)\rangle$. The observer will be a Time Petri Net without places, see Fig.1 (this net is composed with transitions in the observed system, that may have associated places). The observer uses boolean

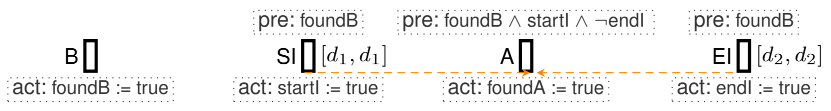


Figure 1. TTS observer for the pattern Present A after B within $[d_1, d_2)$

variables to encode the "state" of every operator in the formula: foundB is true after the search for B succeeds (the top operator is $[\rightarrow B| \ldots \rangle$); startI is true after the beginning of the time restriction $(\backslash_I\ )$ (d_1 u.t. after foundB is true) while endI is true after d_2 u.t.; foundA is true after the search for A succeeds.

In our encoding, each variable is set by a distinct transition. Transitions have a precondition, pre, that is a predicate over variables. The precondition should be true for the transition to be enabled. Symmetrically, each transition has an action, act, that is evaluated when the transition is fired. In the observer of Fig. 1, SI and EI are transitions that belong to the observer, whereas A and B are transitions that will be composed with the "events" A and B in the observed system. We also make use of priorities (dashed arrows between transitions) in order to give the precedence to transitions belonging to the observer.

The property holds if the search for B fails or if we find an A while the predicate (foundB$\wedge$startI$\wedge\neg$endI) is true. Therefore, to check if the pattern holds for the system, it is enough to check the reachability property (we express the property in LTL): $\big(\Diamond\, \text{foundB}\big) \Rightarrow \big(\Diamond\, \text{foundA}\big)$.

V. Related Work and Contributions

We have defined the semantics—as well as both graphical and textual notations—for TGIL, an extension of the Graphical Interval Logic (GIL) of Dill et al [4]. Another real-time extension of the Graphical Interval Logic, called RTGIL, has been proposed by Dillon et al. [6]. RTGIL extends GIL by adding the equivalent of our "time length" search operator, $[\rightarrow_I A|\, \phi\rangle$. In comparison, TGIL is more expressive since it provides an operator for time constrained search, $(\backslash_I\ \phi)$, that is not derivable in RTGIL. For example, the timed pattern present A after B within I can be expressed in TGIL, but not in RTGIL.

Other works propose graphical notations for expressing behavioral properties. Most of these proposals are based on informal diagrammatic notations, such as UML, or are not concerned with verification.

Apart from the work on GIL, that we mentioned extensively, Alfonso et al. [3] define Visual Timed event Scenarios (VTS), a graphical language to define complex requirements using annotations on a partial order of event. It is possible to express timing constraints using VTS but some simple requirements cannot be expressed, such as the fact that a given event, say A, should be true for a duration of d. This requirement corresponds to the formula $(\Diamond_{[0,+\infty)}\,(\Box_{[0,d)}\, A))$ in TGIL. Concerning tooling, another reference is the TimeEdit tool [9], that is based on timeline diagrams. TimeEdit specifications can be compiled into Büchi automata—just like LTL—and used with the Spin model-checker. Nonetheless, timeline diagrams do not directly support the definition of timing constraints.

The usefulness of TGIL goes beyond the definition of timed patterns. It is also a good candidate to replace timed extensions of temporal logic and study their decidable fragments. For example, the timed search operator of TGIL is reminiscent of the $\triangleright_I$ operators defined in the State Clock Logic (SCL) of Raskin and Schobbens [8], a decidable, realtime extension of PTL.

In this short abstract, we have defined a timed extension of Dillon's et al Graphical Interval Logic (TGIL) that is more expressive than previous proposals. The semantics of TGIL can be easily defined using an equivalent "textual notation" that may facilitate (such as e.g. proving the consistency of partial proof systems) We show how to apply TGIL for the definition of a realtime specification language and give an example of the synthesis of an observer. A limitation of this approach is that TGIL is essentially a linear time logic (it expresses constraints on traces), whereas some properties may require a branching time extension. In future work, we plan to enrich the logic with more expressive timing constraints and to study their interaction with a branching time variant of TGIL. We also plan to extend our compilation of diagrams into TTS observers to a larger subset of TGIL.

References

[1] N. Abid, S. Dal Zilio, and D. Le Botlan. Verification of Real-Time Specification Patterns on Time Transitions Systems. Technical Report 11365, LAAS, 2011. `hal-00593963`

[2] N. Abid, S. Dal Zilio, and D. Le Botlan. A Real-Time Specification Patterns Language. Technical Report 11364, LAAS, 2011. `hal-00593965`

[3] A. Alfonso, V. Braberman, N. Kicillof and A. Olivero. Visual Timed Event Scenarios. In *Proc. of 26th International Conference on Software Engineering*, 2004.

[4] L.K. Dillon, G. Kutty, L.E. Moser, P.M. Melliar-Smith and Y.S. Ramakrishna. A Graphical Interval Logic for Specifying Concurrent Systems. In ACM Transactions on Software Engineering and Methodology. 1994.

[5] M. B. Dwyer, G. S. Avrunin, and J. C. Corbett. Patterns in property specifications for finite-state verification. In *Proc. of ICSE*, 1999.

[6] L. E. Moser, P. M. Melliar-Smith, Y. S. Ramakrishna, G. Kutty and L. K. Dillon. The Real-Time Graphical Interval Logic Toolset. In *Proc. of Computer-Aided Verification*, 1996.

[7] J. Ouaknine and J. Worrell. On the decidability and complexity of metric temporal logic over finite words. In *Logical Methods in Computer Science*, vol. 3, 2007.

[8] J. Raskin and P. Schobbens. State Clock Logic: a Decidable Real-Time Logic. In Proc. of HART, LNCS vol. 1201, 1996.

[9] M. H. Smith, G. Holzmann, and K. Etessami. Events and Constraints: a Graphical Editor for Capturing Logic Requirements of Programs. In *Proc. of RE'01—International Symposium on Requirements Engineering*, 2001.