CAV'06, LNCS 4144, pp 263-277, 2006

Allen Linear (Interval) Temporal Logic –Translation to LTL and Monitor Synthesis–

Grigore Roşu[1] * and Saddek Bensalem[2]

[1] Department of Computer Science, University of Illinois at Urbana-Champaign, USA
[2] VERIMAG, 2 Avenue de Vignate, 38610 Gieres, France

Abstract. The relationship between two well established formalisms for temporal reasoning is first investigated, namely between Allen's interval algebra (or Allen's temporal logic, abbreviated ATL) and linear temporal logic (LTL). A discrete variant of ATL is defined, called Allen linear temporal logic (ALTL), whose models are ω-sequences of timepoints. It is shown that any ALTL formula can be linearly translated into an equivalent LTL formula, thus enabling the use of LTL techniques on ALTL requirements. This translation also implies the NP-completeness of ATL satisfiability. Then the problem of monitoring ALTL requirements is investigated, showing that it reduces to checking satisfiability; the similar problem for unrestricted LTL is known to require exponential space. An effective monitoring algorithm for ALTL is given, which has been implemented and experimented with in the context of planning applications.

1 Introduction

Allen's interval algebra, also called Allen's temporal logic (ATL) in this paper, is one of the best established formalisms for temporal reasoning [5]. It is frequently used in AI, especially in planning. Linear temporal logic (LTL) [8] is successfully applied in program verification, temporal databases, and related domains. Despite the widespread use of both ATL and LTL, there is no formal and systematic investigation of their relationship. This paper makes a step in this direction. To have a semantic basis for such a relationship, we define a discrete variant of ATL, called Allen linear temporal logic (ALTL), whose syntax and complexity of satisfiability are the same as for ATL, but whose models resemble those of LTL.

We show that ALTL can be linearly encoded into a subset of LTL. This encoding yields the NP-completeness of the satisfiability problem for an ATL (proposed in [4]) slightly richer than the original one proposed by Allen. On the practical side, this result allows us to use the plethora of techniques and analysis tools developed for LTL on requirements (or compatibilities) expressed using ATL. Since ATL is *the* logic of planning, and since validation and verification (V&V) of complex plans for systems with decisional autonomy is highly desirable, if not crucial, in many applications, this automated translation into LTL potentially enables us to use well-understood V&V techniques and tools in a domain lacking (but in need of) them. Further, it may also support the suggestion made in [2] that LTL can be itself seriously regarded as a suitable formalism for temporal reasoning in AI, and particularly in planning. There are, however, complexity aspects that cannot be ignored (some of them pointed in this paper).

* Supported by NSF grants CCF-0234524, CCF-0448501, and CNS-0509321.

The importance of monitoring in planing cannot be overestimated. For example, an autonomous rover whose execution plans have been rigorously verified may still fail for reasons such as hardware or operating system failures, unexpected terrain in an unknown environment, etc. Having monitors to check online the execution of plans step by step and to trigger recovery code in case of violations is of crucial importance. It is the challenge of generating efficient monitors from planning requirements that motivated the work in this paper. We argue that a blind use of monitoring algorithms for LTL to monitor ALTL formulae is not feasible even on small ALTL formulae; then we give a special-purpose monitoring algorithm for ALTL which only needs to call a boolean satisfiability checker at each step if synchronous monitoring is desired, or at the end of the monitoring session if asynchronous monitoring is acceptable, or anywhere in between, for example at specific relevant events, such as synchronization points. Since checking satisfiability of a formula is a simpler problem than synchronous monitoring (a synchronous monitor should report violation right away if the formula is not satisfiable), the algorithm proposed in this paper is asymptotically optimal. This result is particularly interesting because, for unrestricted LTL, it is known that any monitor (synchronous or not) needs exponential space [10]. The proposed monitoring algorithm has been implemented and experimented with in the context of planning for autonomous rovers.

Preliminaries. We assume the reader familiar with Linear Temporal Logic (LTL) [8]. We here only recall some basics and introduce our notation. LTL is interpreted in "flows of time", modeled as strict linear orders $(T, <)$, where T is a nonempty set of "time points".The LTL language consists of propositional symbols (p_0, p_1, $\cdots$), boolean operators ($\neg$ and $\wedge$), and temporal operators $\mathcal{U}$ ("until") and $\circ$ ("next"), and LTL formulae follow the common syntax $\varphi ::= p \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \;\mathcal{U}\; \varphi_2 \mid \circ\,\varphi$. LTL models are triples $M = (T, <, v)$ such that $(T, <)$ is a strict total order (a flow of time) and v is a map called valuation associating with each variable p a set $v(p) \subseteq T$ of time points (where p is supposed to be true). The satisfaction relation $M \models \varphi$ is defined as in [8]. Other important temporal operators, such as $\Diamond$(eventually) and $\Box$ (always), are expressible using $\mathcal{U}$ as $\Diamond\varphi = \text{true}\,\mathcal{U}\,\varphi$ (φ will eventually hold) and $\Box\varphi = \neg\Diamond\neg\varphi$ (φ will always hold). $\Diamond$ can also be expressed in terms of $\Box$, namely $\Diamond\varphi = \neg\Box\neg\varphi$. In this paper we only need the $\{\Box, \Diamond\}$-fragment of LTL (without $\circ$ and $\mathcal{U}$). Since $\Diamond$ and $\Box$ can be defined in terms of each other, we take the liberty to call this fragment $\mathsf{LTL}_{\Box}$ (could have also called it $\mathsf{LTL}_{\Diamond}$). The "satisfiability problem" for a formula φ is concerned with whether there is some model M such that $M \models \varphi$. The satisfiability problem of LTL formulae is PSPACE-complete, while the satisfiability of $\mathsf{LTL}_{\Box}$ is NP-complete [11].

2 Allen (Linear) Temporal Logic - ATL (ALTL)

***Allen Temporal Logic** (ATL)* [1] is specified as a framework to deal with incomplete relative temporal information, such as "event A is before or overlaps event B". Allen takes the *interval* as the primitive temporal quantity and introduces 13 (mutually exclusive) basic binary relations between any two intervals,

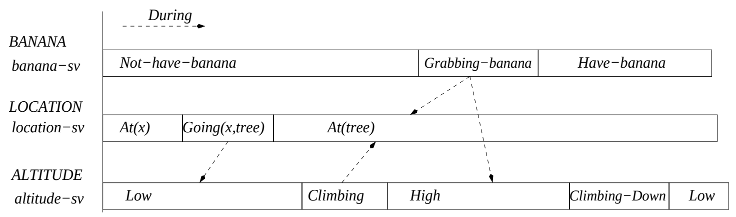

Fig. 1. Attributes and compatibilities

with the following intuitive meaning: $Equals(i, j)$ holds iff i and j consist of the same time points; $Meets(i, j)$ (or $MetBy(j, i)$) holds iff j starts *immediately* after i; $Before(i, j)$ (or $After(j, i)$) holds iff i starts and ends before j, but there is also some proper time elapsed between the end of i and the beginning of j; $Overlaps(i, j)$ (or $OverlappedBy(j, i)$) holds iff i starts strictly before j starts, they have some common time points, and i ends strictly before j ends; $Contains(i, j)$ (or $During(j, i)$) holds iff j starts strictly after i starts and terminates strictly before i terminates; $Starts(i, j)$ (or $StartedBy(j, i)$) holds iff i and j start together but j continues (strictly) after i ends; dually, $Ends(i, j)$ (or $EndedBy(j, i)$) holds iff i and j terminate together but j starts strictly before i starts. Constraints among intervals, also called requirements or *compatibilities*, are given as boolean combinations of such relations on intervals. In (model-)theoretical works on ATL, time is assumed to flow continuously, typically *not* at an enumerable rate (e.g., timepoints can be rational or real numbers). Following this model, we formally define the semantics of these interval relations in Definition 4; then we propose a time-discrete variant of ATL, in which the time-points are enumerable.

ATL is extensively used in AI planing to formalize and reason about concurrency and temporal extent. In AI planning, intervals can represent both action instances and the states of various attributes or components of a system. Attributes whose states change over time are called *state variables*, each being possibly regarded as a concurrent thread. The history of states of a state variable over a period of time is called a *timeline* and is typically partitioned into *intervals*, where an interval is a set of contiguous timepoints in which the corresponding state variable satisfies some property of interest. A *compatibility* then determines necessary correlations among various behaviors of parts of the system in order for a plan to be legal. One appealing aspect of ATL in this domain is that compatibilities can be elegantly depicted using an intuitive graphical notation (see Figure 1), that allows planning specialists to develop surprisingly large and complex specifications in a short time.

Example 1. We use McCarthy's classic monkey/banana planning problem as a running example. A monkey is at location "x", the banana is hanging from the tree. The monkey is at height "Low", but if it climbs the tree then it will be at height "High", same as the banana. Available actions are: "Going" from a place to another, "Climbing" (up) and "Climbing Down", and "Grabbing" banana.
Attributes. BANANA has one state variable "*Banana-sv*" saying if the monkey has the banana or not. LOCATION has one variable "*Location-sv*" for the location of the monkey. ALTITUDE has one variable "*Altitude-sv*" for the height.

Compatibilities. Now we can consider some *compatibilities* for the intervals corresponding to these attributes, also depicted in Figure 1:

– *Have-banana* ("H_b") requires *Grabbing-banana* ("G_b") which requires *Not-have-banana* ("N_{hb}"). *Grabbing-banana* is performed while *High* and *At(tree)*.
– *At(tree)* ("$@(tree)$") requires going from the location *"x"* to the tree which requires *At(x)* ("$@(x)$"). *Going(x,tree)* ("$G(x, tree)$") is performed while *Low*.
– *High* ("H") requires *Climbing* ("C") which requires *Low* ("L"), and *Climbing-Down* ("C_D") requires *High*. *Climbing* is performed while *At(tree)*.

These compatibilities can be formally specified in ATL as follows:

$$\begin{aligned}&\mathit{Meets}(N_{hb}, G_b) \wedge \mathit{Meets}(G_b, H_b) \wedge \mathit{During}(G_b, @(tree)) \wedge \mathit{During}(G_b, H) \wedge\\ &\mathit{Meets}(@(x), G(x, tree)) \wedge \mathit{Meets}(G(x, tree), @(tree)) \wedge \mathit{During}(G(x, tree), L) \wedge\\ &\mathit{Meets}(L, C) \wedge \mathit{Meets}(C, H) \wedge \mathit{Meets}(H, C_D) \wedge \mathit{Meets}(C_D, L) \wedge \mathit{During}(C, @(tree)).\end{aligned}$$

Let us consider the subformula consisting of the first four conjuncts above (first line), and suppose that an unexpected "flying monkey" wants the banana. It climbs the tree, but it cannot reach for the banana. Being a flying monkey, it jumps for the banana, grabs it while gliding when it is still *High* and *At(tree)*, but as it glides it leaves the tree location. Supposing that it leaves the tree location at the same time it changes the status from *Grabbing-banana* to *Have-banana*, one can notice that the third conjunct is violated. Indeed, G_b must hold *during* $@(tree)$, meaning that there must be some (non-zero) periods of time in which the monkey was at the tree location before and after grabbing the banana.

It is often useful to state that some propositions hold all the time or eventually *during* an interval. For example, assume one more state predicate, *hungry*, saying whether the monkey is hungry or not, and assume that we want to state that monkeys should grab and have bananas only if they are hungry and do not already have bananas. This can be done with the following additional conjunct:

$$\mathit{Occurs}(hungry, N_{hb}) \wedge \mathit{Holds}(hungry, G_b) \wedge \mathit{Holds}(hungry, H_b) \qquad \square$$

There are different views on how intervals should be modeled in different time flows. A common interpretation is that the intervals are ordered pairs of distinct points in $\mathbb{Q}$ or $\mathbb{R}$. For simplicity, it is convenient to use semantics where intervals are arbitrary convex non-empty subsets of time points of an arbitrary time flow.

Definition 1. *If $\mathcal{P}$ is a set of* ***atomic propositions*** *and $\mathcal{I}$ is a set of* ***intervals****, then an* ***Allen temporal logic formula over $\mathcal{P}$ and $\mathcal{I}$****, or an $\mathsf{ATL}(\mathcal{P},\mathcal{I})$-****formula*** *or even just a* ***formula*** *when $\mathcal{P}$ and $\mathcal{I}$ are understood from context, is any boolean combination of* ***basic formulae*** *of the form $\mathit{Equals}(i, j)$, $\mathit{Before}(i, j)$, $\mathit{After}(i, j)$, $\mathit{Overlaps}(i, j)$, $\mathit{OverlappedBy}(i, j)$, $\mathit{Meets}(i, j)$, $\mathit{MetBy}(i, j)$, $\mathit{Contains}(i, j)$, $\mathit{During}(i, j)$, $\mathit{Starts}(i, j)$, $\mathit{StartedBy}(i, j)$, $\mathit{Ends}(i, j)$, $\mathit{EndedBy}(i, j)$, $\mathit{Holds}(p, i)$, $\mathit{Occurs}(p, i)$, where $i, j \in \mathcal{I}$ and $p \in \mathit{Bool}(\mathcal{P})$.*

$\mathit{Bool}(\mathcal{P})$ is the set of boolean propositions over variables in $\mathcal{P}$. Interestingly, the original formulation of ATL [1] did not include *Holds* and *Occurs*; motivated by practical reasons, they were added later in [4]. To define a formal semantics of ATL we need to first define an appropriate notion of model.

Definition 2. *Let $(T, <)$ be a strict total order. The relation $<$ is tacitly extended to a strict partial order on subsets of T, namely $X < Y$ iff $x < y$ for all $x \in X$ and $y \in Y$. Also, by abuse of notation, we may write just x instead $\{x\}$; thus, $x < Y$ means that $x < y$ for all $y \in Y$. For $x, y \in T$ let (x, y) be the set $\{z \in T \mid x < z < y\}$. A subset C of T is* ***$<$-convex****, or simply* ***convex****, iff $(x, y) \subseteq C$ for any $x, y \in C$.*

In $\mathbb{R}$, for example, the convex sets are precisely the intervals. Recall that intervals in $\mathbb{R}$ can be open or closed on any of their ends, and that they may be bound by $-\infty$ or $+\infty$ at their left or right ends, respectively.

Definition 3. *A* ***$(\mathcal{P}, \mathcal{I})$-interval model*** *(or simply an* ***interval model*** *when $\mathcal{P}$ and $\mathcal{I}$ are understood) is a structure $\mathcal{M} = (T, <, v, \sigma)$, where $(T, <)$ is a strict total order (modeling the intended flow of time), $v : \mathcal{P} \to 2^T$ is a valuation map assigning to each atomic proposition $p \in \mathcal{P}$ a set of time points $v(p)$ (in which the proposition is assumed to be true), and σ is a map that associates with every interval $i \in \mathcal{I}$ a non-empty convex subset $\sigma(i)$ of T. We may also refer to $(\mathcal{P}, \mathcal{I})$-interval models as models of ATL$(\mathcal{P}, \mathcal{I})$.*

We are now ready to give the formal semantics of ATL.

Definition 4. *An interval model $\mathcal{M} = (T, <, v, \sigma)$* ***satisfies****: Equals(i, j) iff $\sigma(i) = \sigma(j)$; Before(i, j) or After(j, i) iff there is some $t \in T$ such that $\sigma(i) < t < \sigma(j)$; Overlaps$(i, j)$ or OverlappedBy(j, i) iff $\sigma(i) \cap \sigma(j) \neq \emptyset$ and there are some $t_i \in \sigma(i)$ and $t_j \in \sigma(j)$ such that $t_i < \sigma(j)$ and $\sigma(i) < t_j$; Meets(i, j) or MetBy(j, i) iff $\sigma(i) < \sigma(j)$ and there is no $t \in T$ such that $\sigma(i) < t < \sigma(j)$; Contains$(i, j)$ or During(j, i) iff there are some $t_i, t'_i \in \sigma(i)$ such that $t_i < \sigma(j) < t'_i$; Starts(i, j) or StartedBy(j, i) iff $\sigma(i) \subset \sigma(j)$, there is no $t_j \in \sigma(j)$ such that $t_j < \sigma(i)$, but there is some $t_j \in \sigma(j)$ such that $\sigma(i) < t_j$; Ends(i, j) or EndedBy(j, i) iff $\sigma(i) \subset \sigma(j)$, there is no $t_j \in \sigma(j)$ such that $\sigma(i) < t_j$, but there is some $t_j \in \sigma(j)$ such that $t_j < \sigma(i)$; Holds(p, i) iff $\sigma(i) \subseteq v(p)$; and Occurs(p, i) iff $\sigma(i) \cap v(p) \neq \emptyset$ iff $\neg$Holds$(\neg p, i)$. Satisfaction is defined as usual on boolean combinations of ATL formulae. We use the notation $\mathcal{M} \models_{ATL} \varphi$ to denote the fact that the interval structure $\mathcal{M}$ satisfies the ATL formula φ.*

Therefore, *Holds*(p, i) is satisfied iff p holds at any time point in i, while *Occurs*(p, i) is satisfied iff p holds at some time point in i. The propositions p used in *Holds* and *Occurs* may hold at various random timepoints, so they cannot be replaced by intervals. The NP-completeness of the satisfiability problem for ATL without *Holds* [12] gives us immediately the NP-hardness of our ATL with *Holds* above. We will show in the next section that it is actually NP-complete.

In many practical applications of interest, time elapses at a discrete and enumerable rate. We next define a variant of Allen temporal algebra in which the support of the interval models are ω-sequences of time points, that is, linear (infinite) sequences $t_1 < t_2 < t_3 < \cdots < t_n < \cdots$. We write these strict total orders compactly as $t_1 t_2 t_3 \ldots t_n \ldots$. We call the new logic ***Allen Linear Temporal Logic*** (ALTL). Note that ALTL has the same syntax as ATL and its satisfaction relation is defined like in ATL, but that its models are structures of the form $\mathcal{M} = (t_1 t_2 \ldots, v, \sigma)$, where $t_1 t_2 \ldots$ are ω-sequences of time points and σ maps intervals in $\mathcal{I}$ into non-empty convex sets $\sigma(i)$ of $T = \{t_1, t_2, \ldots\}$ (with the expected strict total ordering $<$ defined as $t_m < t_n$ iff $m < n$). It is easy to see that the convex sets of T are either finite sets of the form $\{t_m, t_{m+1}, \ldots, t_n\}$ for some $0 < m \leq n$, or infinite sets of the form $\{t_m, t_{m+1}, \ldots\}$ for some $0 < m$.

3 Linear Translation of ALTL into LTL

We next define an automatic encoding of ALTL into $\mathsf{LTL}_\Box$. Note that the models of ALTL differ from those of LTL in that they contain a concrete interpretation for each interval. Therefore, in order to establish a semantic relationship between the models of the two logics, we need to first add syntactic support for "intervals" to LTL. A simple way to do this is to add an atomic propositional symbol $\in_i$ to the syntax of LTL for each interval $i \in \mathcal{I}$, with the intuition that a time point is in the interval i in a model of ALTL if and only if the proposition $\in_i$ holds in that time point in the corresponding model of LTL. Moreover, we need to also capture, via corresponding LTL formulae, the fact that intervals are interpreted into non-empty convex sets in ALTL models.

Definition 5. *Let $\mathcal{P}_\mathcal{I}$ be the set of atomic propositions $\mathcal{P} \cup \{\in_i \mid i \in \mathcal{I}\}$ and let $\Psi_\mathcal{I}$ be the set of LTL formulae $\{\psi_i \mid i \in \mathcal{I}\}$ over propositions in $\mathcal{P}_\mathcal{I}$, where ψ_i is the formula $\Diamond\!\in_i \wedge \neg\Diamond(\in_i \wedge \Diamond(\neg\!\in_i \wedge \Diamond\!\in_i))$ for each $i \in \mathcal{I}$.*

The following establishes the relationship between models of ALTL and of LTL:

Proposition 1. *There is a bijection between $(\mathcal{P}, \mathcal{I})$-interval models and models of $\mathsf{LTL}(\mathcal{P} \cup \{\in_i \mid i \in \mathcal{I}\})$ that satisfy $\Psi_\mathcal{I}$.*

Proof. Let $\mathcal{M} = (T, <, v, \sigma)$ be a tuple where $(T, <)$ is an ω-sequence, v is a map $\mathcal{P} \to 2^T$, and σ is a map $\mathcal{I} \to 2^T$; what $\mathcal{M}$ is missing to be a model of $\mathsf{ALTL}(\mathcal{P}, \mathcal{I})$ is the requirements that $\sigma(i)$ is non-empty and convex for any $i \in \mathcal{I}$. Then we can build a model $\mathcal{N} = (T, <, u)$ of $\mathsf{LTL}(\mathcal{P} \cup \{\in_i \mid i \in \mathcal{I}\})$, where $u(p) = v(p)$ for all $p \in \mathcal{P}$ and $u(\in_i) = \sigma(i)$ for all $i \in I$. Conversely, for any model $\mathcal{N} = (T, <, u)$ of $\mathsf{LTL}(\mathcal{P} \cup \{\in_i \mid i \in \mathcal{I}\})$ one can build a tuple $\mathcal{M} = (T, <, v, \sigma)$, where v is the restriction of u to $\mathcal{P}$ and $\sigma(i)$ is defined as $u(\in_i)$ for any $i \in \mathcal{I}$. What is left to prove is that $\sigma(i)$ is non-empty and convex for any $i \in \mathcal{I}$ if and only if $\mathcal{N} \models_{\mathsf{LTL}} \Psi_\mathcal{I}$. First, note that, for any $i \in \mathcal{I}$, $\sigma(i) \neq \emptyset$ is equivalent to $\mathcal{N} \models_{\mathsf{LTL}} \Diamond\!\in_i$. Second, since $\sigma(i)$ is convex if and only if there are no time points t_m, t_n, t_k with $0 < m < n < k$ such that $t_m, t_k \in \sigma(i)$ and $t_n \notin \sigma(i)$, one deduces that $\sigma(i)$ is convex if and only if $\mathcal{N} \models_{\mathsf{LTL}} \neg\Diamond(\in_i \wedge \Diamond(\neg\!\in_i \wedge \Diamond\!\in_i))$. Therefore, $\sigma(i)$ is non-empty and convex for each $i \in \mathcal{I}$ if and only if $\mathcal{N} \models_{\mathsf{LTL}} \Psi_\mathcal{I}$. $\Box$

Definition 6. *We let $[\cdot]$ define the bijection above, that is, if $\mathcal{M}$ is a $(\mathcal{P}, \mathcal{I})$-interval model then $[\mathcal{M}]$ is the corresponding model of $\mathsf{LTL}(\mathcal{P} \cup \{\in_i \mid i \in \mathcal{I}\})$ satisfying $\Psi_\mathcal{I}$, defined as in the proof of Proposition 1.*

We are now ready to define the first part of our syntactic encoding of ALTL formulae into LTL formulae.

Definition 7. *Let $[\cdot]$ be the function taking formulae φ in $\mathsf{ALTL}(\mathcal{P},\mathcal{I})$ into formulae $[\varphi]$ in $\mathsf{LTL}(\mathcal{P} \cup \{\in_i \mid i \in \mathcal{I}\})$ defined inductively as follows: $[\neg\varphi]$ is $\neg[\varphi]$; $[\varphi_1 \wedge \varphi_2]$ is $[\varphi_1] \wedge [\varphi_2]$; $[Equals(i,j)]$ is $\Box(\in_i \Leftrightarrow \in_j)$; $[Before(i,j)]$ and $[After(j,i)]$ are $\Diamond(\in_i \wedge \Diamond(\neg\in_i \wedge \neg\in_j \wedge \Diamond\in_j))$; $[Meets(i,j)]$ and $[MetBy(j,i)]$ are $\Diamond(\in_i \wedge \Diamond\in_j \wedge \neg\Diamond(\in_i \wedge \in_j) \wedge \neg\Diamond(\neg\in_i \wedge \neg\in_j \wedge \Diamond\in_j))$; $[Overlaps(i,j)]$ and $[OverlappedBy(j,i)]$ are $\Diamond(\in_i \wedge \neg\in_j \wedge \Diamond(\in_i \wedge \in_j \wedge \Diamond(\neg\in_i \wedge \in_j)))$; $[Contains(i,j)]$ and $[During(j,i)]$ are $\Diamond(\in_i \wedge \neg\in_j \wedge \Diamond(\in_i \wedge \in_j \wedge \Diamond(\in_i \wedge \neg\in_j)))$; $[Starts(i,j)]$ and $[StartedBy(j,i)]$ are $\Box(\in_i \Rightarrow \in_j) \wedge \neg\Diamond(\in_j \wedge \neg\in_i \wedge \Diamond\in_i) \wedge \Diamond(\in_j \wedge \neg\in_i)$; $[Ends(i,j)]$ and $[EndedBy(j,i)]$ are $\Box(\in_i \Rightarrow \in_j) \wedge \Diamond(\in_j \wedge \neg\in_i) \wedge \neg\Diamond(\in_j \wedge \in_i \wedge \Diamond(\in_j \wedge \neg\in_i))$; $[Holds(p,i)]$ is $\Box(\in_i \Rightarrow p)$; and $[Occurs(p,i)]$ is $[\neg Holds(\neg p,i)]$, that is, $\Diamond(\in_i \wedge p)$.*

Example 2. Let us consider again the subformula

$$Meets(N_{hb},G_b) \wedge Meets(G_b,H_b) \wedge During(G_b,@(tree)) \wedge During(G_b,H)$$

of the formula that characterizes the compatibilities of the monkey/banana problem (see Example 1), to illustrate how to encode an ALTL formula into an equivalent $\mathsf{LTL}_\Box$ one. Its encoding is:

$$\begin{array}{l}\Diamond(\in_{N_{hb}} \wedge \Diamond\in_{G_b} \wedge \neg\Diamond(\in_{N_{hb}} \wedge \in_{G_b}) \wedge \neg\Diamond(\neg\in_{N_{hb}} \wedge \neg\in_{G_b} \wedge \Diamond\in_{G_b}))\wedge \\ \Diamond(\in_{G_b} \wedge \Diamond\in_{H_b} \wedge \neg\Diamond(\in_{G_b} \wedge \in_{H_b}) \wedge \neg\Diamond(\neg\in_{G_b} \wedge \neg\in_{H_b} \wedge \Diamond\in_{H_b}))\wedge \\ \Diamond(\in_{@(tree)} \wedge \neg\in_{G_b} \wedge \Diamond(\in_{@(tree)} \wedge \in_{G_b} \wedge \Diamond(\in_{@(tree)} \wedge \neg\in_{G_b})))\wedge \\ \Diamond(\in_H \wedge \neg\in_{G_b} \wedge \Diamond(\in_H \wedge \in_{G_b} \wedge \Diamond(\in_H \wedge \neg\in_{G_b}))) \wedge (\bigwedge_{i\in\mathcal{I}} \psi_i),\end{array}$$

where $\mathcal{I} = \{N_{hb}, H_b, H, G_b, @(tree)\}$ and ψ_i is $\Diamond\in_i \wedge \neg\Diamond(\in_i \wedge \Diamond(\neg\in_i \wedge \Diamond\in_i))$. As expected, the LTL encoding of the entire formula in Example 1 is very large. $\Box$

The companion report [9] shows a rewriting implementation of this encoding.

Theorem 1. *Given an $\mathsf{ALTL}(\mathcal{P},\mathcal{I})$ formula φ and a $(\mathcal{P},\mathcal{I})$-interval model $\mathcal{M}$, then $\mathcal{M} \models_{\mathsf{ALTL}} \varphi$ iff $[\mathcal{M}] \models_{\mathsf{LTL}} [\varphi]$.*

Proof. Structural induction on φ. If φ has the form $\neg\varphi_1$ then $\mathcal{M} \models_{\mathsf{ALTL}} \varphi$ is equivalent to saying that it is *not* the case that $\mathcal{M} \models_{\mathsf{ALTL}} \varphi_1$, which, by the induction hypothesis and Definition 7, is equivalent to saying that $[\mathcal{M}] \models_{\mathsf{LTL}} [\varphi]$. The case where φ has the form $\varphi_1 \wedge \varphi_2$ is similar. What is left to show is that the property holds when φ is any of the interval relations. Let us discuss only one of them, for example $Meets(i,j)$. Suppose that $\mathcal{M} = (T,<,v,\sigma)$ and recall that $\sigma(i)$ is non-empty for any interval i. By Definition 4, $\mathcal{M} \models_{\mathsf{ALTL}} Meets(i,j)$ iff $\sigma(i) < \sigma(j)$ and there is so $t \in T$ such that $\sigma(i) < t < \sigma(j)$. By the way $[\mathcal{M}]$ is built and because ψ_i and ψ_j ensure the non-emptiness and the convexity of the trace fragments in which $\in_i$ and $\in_j$ hold, This is equivalent to saying that $\in_j$ holds *strictly* after $\in_i$, i.e., the $\Diamond(\in_i \wedge \Diamond\in_j \wedge \neg\Diamond(\in_i \wedge \in_j) \wedge ...)$; part of $[Meets(i,j)]$, and that there is no period of time following $\in_i$ that appears before $\in_j$ in which neither $\in_i$ nor $\in_j$ holds, i.e., the $\Diamond(...\neg\Diamond(\neg\in_i \wedge \neg\in_j \wedge \Diamond\in_j))$ part of $[Meets(i,j)]$. The result can be proved similarly for the other intervals. $\Box$

Our goal next is to reduce the satisfiability problem for ALTL to $\mathsf{LTL}_\Box$ satisfiability, known to be an NP-complete problem [11]. Theorem 1 gives us only half of the result, namely that if a formula φ is satisfiable in ALTL then the formula $[\varphi]$ is satisfiable in $\mathsf{LTL}_\Box$. To get the other half, one could define a slightly different translation of ALTL formulae, namely one that would also include the conjunction of the formulae in $\Psi_{\mathcal{I}}$. The problem with that is, however, that $\mathcal{I}$ can be infinite, meaning that the generated LTL formula would be infinite. Fortunately, only the intervals that explicitly appear in φ need to be taken into account, thus making our transformation finite:

Definition 8. *For an ALTL$(\mathcal{P},\mathcal{I})$ formula φ, let $\mathcal{I}_\varphi$ be the finite set of intervals appearing in φ and let $\langle\varphi\rangle$ be the formula $[\varphi] \wedge \bigwedge \Psi_{\mathcal{I}_\varphi}$ in LTL$(\mathcal{P} \cup \{\in_i \mid i \in \mathcal{I}_\varphi\})$.*

Corollary 1. *Given a formula φ in ALTL$(\mathcal{P},\mathcal{I})$, the following are equivalent: (1) φ is satisfiable in ALTL$(\mathcal{P},\mathcal{I})$; (2) $\langle\varphi\rangle$ is satisfiable in LTL$(\mathcal{P} \cup \{\in_i \mid i \in \mathcal{I}_\varphi\})$; and (3) $\langle\varphi\rangle$ is satisfiable in LTL$(\mathcal{P} \cup \{\in_i \mid i \in \mathcal{I}\})$.*

Proof. Since a model over more atomic propositions can be also regarded as a model over fewer propositions, it is immediate that *3.* implies *2.*. By Theorem 1, any model of φ in ALTL$(\mathcal{P},\mathcal{I})$ yields a model of $[\varphi]$ in LTL$(\mathcal{P} \cup \{\in_i \mid i \in \mathcal{I}\})$ that satisfies $\Psi_{\mathcal{I}}$. Therefore, *1.* implies *3.*. To show that *2.* implies *1.*, by Proposition 1 it suffices to show that any model in LTL$(\mathcal{P} \cup \{\in_i \mid i \in \mathcal{I}_\varphi\})$ satisfying $\Psi_{\mathcal{I}_\varphi}$ can be extended, by just adding appropriate valuations for the additional atomic propositions to assure that the satisfaction of φ is not affected, to a model in LTL$(\mathcal{P} \cup \{\in_i \mid i \in \mathcal{I}\})$ satisfying $\Phi_{\mathcal{I}}$. This can be done many different ways. One straightforward model extension is to require that each proposition in $\{\in_i \mid i \in \mathcal{I} - \mathcal{I}_\varphi\}$ holds in precisely one (arbitrary) time point.

Corollary 2. *The satisfiability problem for ALTL is NP-complete.*

Proof. By Corollary 1, an ALTL formula φ is satisfiable iff $\langle\varphi\rangle$ is satisfiable as an LTL formula. Since $\langle\varphi\rangle$ can be generated linearly in the size of the φ and since LTL-satisfiability is NP-complete, ALTL-satisfiability is also NP-complete.

4 Monitoring ALTL

It is known that *any* monitoring algorithm for LTL-formulae requires space exponential in the size of the monitored formula [10] in the worst case. Can we find better monitoring algorithms for ALTL? We first argue empirically that a blind use of monitoring algorithms for LTL may be unfeasible in large applications and then propose an ALTL-specific monitoring algorithm which avoids the exponential-space complexity of monitoring LTL-formulae. More precisely, we give a monitoring algorithm for ALTL which only requires space (it needs to store its current state only) that is linear in the size of the input formula and whose most expensive task is to check the satisfiability of a *boolean* formula that is incrementally smaller (in the sense that some of its variables are irreversibly replaced by true or false) with each event received from the monitored system, and which initially has precisely the size of the original ALTL formula.

Let us first describe informally the "monitoring problem" for a logic whose models are (finite or infinite) traces. Given a formula ξ of size n and a "running system" abstracted by its incrementally emitted events (or abstract states encoded by the atomic propositions that "hold" in them) t_1, t_2, ..., the problem is to report when a bad prefix is reached, that is, when a finite trace $t_1 t_2 .. t_m$

is encountered such that there is no infinite trace $t_1t_2...t_mt_{m+1}...$ that satisfies ξ. We here assume that storing the events is *not* an option, because their number can grow arbitrarily large. Indeed, m can be large enough so that even an algorithm that is linear in the continuously increasing execution trace at each emitted event (e.g., one that traverses the trace backwards, like the one in [10]) can become easily more impractical than one just exponential in the formula but constant in the trace (e.g., when one generates an automata monitor from it, like in [3]). One can (non-trivially) formalize the monitoring problem for a logic as a decision problem, but this is rather intricate and beyond our scope here. Here we limit ourselves to the informal problem description above and conclude that ALTL-monitoring is asymptotically as expensive as ALTL-satisfiability:

(a) in any logic, monitoring is a harder problem than satisfiability;
(b) for any ALTL-formula ξ, we give a monitoring algorithm which is not more expensive than checking the satisfiability of ξ.

One can readily see that monitoring is harder than satisfiability: a monitor for ξ reports violation on the empty trace iff ξ is not satisfiable. Since ALTL-satisfiability is NP-complete (Corollary 2), any monitoring algorithm for ALTL is expected to be worst-case exponential in practice. However, as in many other similar situations, this does not necessarily mean that the problem of monitoring ALTL formulae is not practical. We next briefly discuss an immediate algorithm based on the translation to LTL, and then give an algorithm specific to ALTL that avoids the complexity of monitoring LTL and which seems quite efficient in practice. The next section discusses an experiment where the ALTL formula is large enough that the LTL-based monitoring algorithms cannot handle it.

The transformation in Section 3 suggests using a general purpose monitoring algorithm for LTL (e.g., the one in [3]), to monitor the LTL formula obtained linearly from the ALTL formula. We have experimented with this technique and have succeeded to generate, unfortunately huge, LTL monitors only for relatively small ALTL formulae. For example, for the ALTL formula in Example 2, which is a subformula of the ALTL formula in Example 1, the generated monitor had more than 60,000 edges, while the algorithm ran out of memory trying to generate an LTL monitor for the entire ALTL formula in Example 1; and that is just a toy example. The reason for our failure to generate monitors following this approach is the intermediate Büchi automata generator from LTL formulae; the LTL monitors in [3] are obtained pruning the corresponding Büchi automata (which can be exponential), by removing portions of them related to liveness – only the safety fragment of a formula is monitorable. The interested reader is encouraged to check [9] for more details on this unsuccessful approach.

We next give a monitoring algorithm for ALTL *not* based on general monitoring algorithms for LTL. The idea is to regard the ALTL formula φ as a *boolean proposition* in which the interval relations are regarded as special "dynamic" variables. For each interval relation we generate a little state machine, which has two special states, true and false. These state machines are shown in Figure 2. We also add a top-level conjunct consisting of precisely one special variable for each interval that appears in φ; these latter variables correspond, intuitively, to the formulae ψ_i in Definition 5. The monitoring algorithm works as follows: (1)

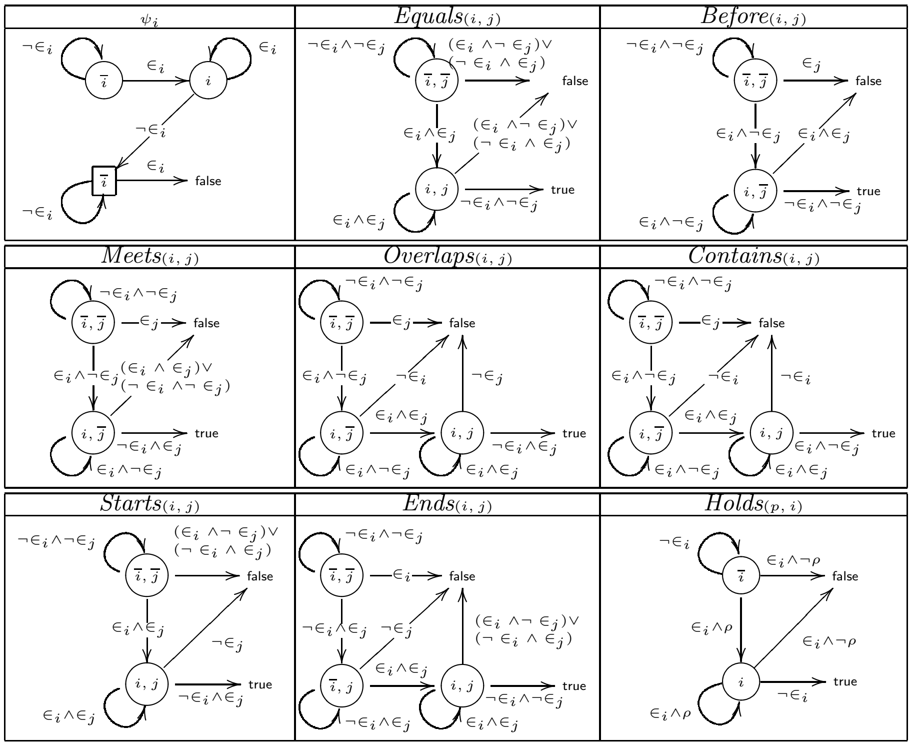

Fig. 2. State machines are run synchronously by the monitor with each event.

generate all the state machines in Figure 2 (left-top state is initial); (2) let ξ be the boolean proposition obtained from φ as above; (3) run a *boolean* satisfiability checker on ξ and stop with "error" if ξ not satisfiable; (4) otherwise, for the next event t received from the monitored system, run all the state machines one step according to t (take that deterministic edge which is satisfied by t); (5) modify the formula ξ by replacing each variable whose corresponding state machine is in a state true or false by the corresponding truth value; (6) goto step (3).

Let us briefly discuss the state machines. The ones for ψ_i ensure that intervals are contiguous (convex); some intervals can be unbounded. The next seven state machines correspond to the relations on intervals. Let us discuss the one for $Meets(i,j)$. One starts with the initial state $\overline{i},\overline{j}$ (neither in i nor in j), and there it stays as far as one does not enter any of the intervals. If while in this state the monitored program enters the interval j, that is, if $\in_j$ holds, then the relation $Meets(i,j)$ is obviously violated (interval i cannot be empty). Otherwise, if the interval i but not j is entered, then the machine moves to state $i,\overline{j}$ where it waits until either i is left and j is entered in which case it returns true, or otherwise until i is left without entering j or i and j overlap, when it returns false. The machine for $Holds(p,i)$ checks that p holds during the interval i.

Example 3. Let us consider again the monkey/banana formula in Example 2,

$(Meets(N_{hb}, G_b) \wedge Meets(G_b, H_b) \wedge During(G_b, @(tree)) \wedge During(G_b, H))$, and consider an execution trace which starts with the abstract events $t_1 = \{\in_{N_b}\}$, $t_2 = \{\in_{N_b}, \in_{@(tree)}\}$, $t_3 = \{\in_{G_b}, \in_{@(tree)}, \in_H\}$, $t_4 = \{\in_{H_b}, \in_H\}$, ..., where an abstract event formed of a set of atomic propositions is an event in which all those, and only those propositions hold. This execution trace corresponds to the "flying monkey" scenario at the end of Example 1.

Let us simulate the execution of the ALTL monitoring algorithm above on this example. There are nine state machines like in Figure 2 necessary, four corresponding to each of the four interval relations and five corresponding to each interval appearing in the formula. The boolean formula ξ is just a conjunction of the corresponding nine variables. All one needs to do is to run the nine state machines on the execution trace, update the boolean proposition and then check for satisfiability after each event. After the first three events, the five ψ_i formulae will be in some intermediate (not false) states, and the four machines corresponding to the interval relations will be in the states true, $(G_b, \overline{H_b})$, $(@(tree), G_b)$, and (G_b, H), respectively, so the formula is still satisfiable. However, when the event t_4 is processed, the machine corresponding to $During(G_b, @(tree))$, or to $Contains(@(tree), G_b)$, transits to false, invalidating the boolean proposition. □

Example 4. Consider now the ALTL formula $\neg Before(i, j)$ and a two-event trace $\{\in_i\}\{\}$. The monitoring algorithm above sets the machine corresponding to $Before(i, j)$ to state $\bigcirc\!\!\!\!\!\!\;\;(i, \overline{j})$ after processing $\{\in_i\}$ and then to state true after processing $\{\}$, causing the monitor to report "error" before any event containing $\in_j$ is seen. Note that $\{\in_i\}\{\}$ is indeed a bad prefix for $\neg Before(i, j)$ ($\in_j$ must hold eventually in any interval model of ALTL). Therefore, our monitoring algorithm for ALTL detects bad prefixes as soon as they appear. □

Note that the state machines corresponding to ψ_i will intercept any violation of the convexity of intervals. If any of the convexities of intervals is violated, that is, if an interval starts, then it is interrupted and then started again, then the monitoring algorithm above returns "error", because the observed trace cannot even be continued into an interval model; one can easily modify the algorithm to return a different type of error in such situations. Note also that these state machines for ψ_i do not have a true state: there is no way to decide by means of monitoring that ψ_i holds, because this is a property of an infinite trace; by monitoring, one can only detect the safety fragment of the inherent ALTL property "intervals are non-empty and convex", namely the break of their convexity. Therefore, the formulae ψ_i can only detect violations of the monitored formula: their corresponding variables can only be transformed into false, never into true. If in a particular application there are external factors implying the well-formedness of intervals, then one can drop the variables (and the machines) corresponding to ψ_i (and thus be able to also detect formula validations online).

Theorem 2. *The monitoring algorithm for ALTL above is correct.*

Proof. Thanks to the machines corresponding to ψ_i, we can assume the well-formedness of intervals in the proof. Consider some finite trace $\tau = t_1 t_2 ... t_m$

that is well-formed wrt intervals, i.e., it can be the prefix of some interval model of ALTL. Let us first prove that for any interval relation, its corresponding state machine is in state false after processing τ iff τ is a bad prefix of that interval relation. We only show it for one relation, say $\mathit{Before}(i,j)$; the others are similar. Note that τ is a bad prefix of $\mathit{Before}(i,j)$ iff τ contains (some event satisfying) $\in_j$ before or at the same time with $\in_i$. Since the state machine of $\mathit{Before}(i,j)$ reaches the state false iff $\in_j$ is seen before in_j or if $\in_j$ and $\in_i$ are seen together as part of an event, and since the machines corresponding to ψ_i ensure the contiguity of intervals, we can conclude that τ is a bad prefix of $\mathit{Before}(i,j)$ iff the corresponding machine of $\mathit{Before}(i,j)$ is in state false after processing τ.

Let us next prove that for any interval relation, the corresponding machine is in state true after processing τ iff τ is a good prefix of that relation, in the sense that for any infinite trace ϕ such that $\tau\pi$ is an interval model of ALTL, it is the case that $\tau\pi$ satisfies that relation. As above, let us just prove it for $\mathit{Before}(i,j)$. Note that the machine of $\mathit{Before}(i,j)$ can be in state true after processing τ iff τ contains no event satisfying $\in_j$ and contains some event satisfying $\in_i$ followed by one which does not satisfy $\in_i$. This is equivalent to saying that any interval model of the form $\tau\pi$ (recall that intervals have non-empty interpretations in interval models) satisfies $\mathit{Before}(i,j)$.

Let us now consider any ALTL formula φ and a finite trace τ as above such that the ξ formula maintained by the algorithm is satisfiable after processing τ. If φ has the form $\varphi_1 \wedge \varphi_2$ then τ is a bad (good) prefix of φ iff it is a bad (good) prefix of φ_1 or (and) φ_2. If φ has the form $\neg\varphi_1$ then τ is a bad (good) prefix of φ iff it is a good (bad) prefix of φ_1. Therefore, in order to test whether τ is a bad prefix of φ one only needs to know whether it is a bad prefix of φ's interval relations, that is, if their corresponding state machines are in their corresponding false or true states after processing τ. The satisfiability checking of ξ after each event ensures that violations are reported as early as possible. $\square$

If one is not interested in reporting ALTL property violations as early as possible, then one can run the satisfiability checker less frequently, say once every 100 events, or even just once at the end of the monitoring session, and thus significantly reduce the runtime overhead. If minimal runtime overhead is highly desirable, since the formula ξ to check for satisfiability changes incrementally by irreversibly transforming some of its variables into true or false, to achieve a minimal runtime overhead one can use an incremental SAT solver.

5 Experiment

Implementation. We have implemented a prototype monitor generation tool, called ALTL2Monitor. It implements the monitoring algorithm presented in the previous section using the SAT solver zChaff [7] for satisfiability checking.

Case Study. Our case study is a simplified version of an exploration rover (Gromit, at Nasa Ames). The mission of the robot is to visit a number of way-points, into an initially unknown rough environment, while monitoring interesting targets on its path. The robot continuously takes pictures of the terrain in

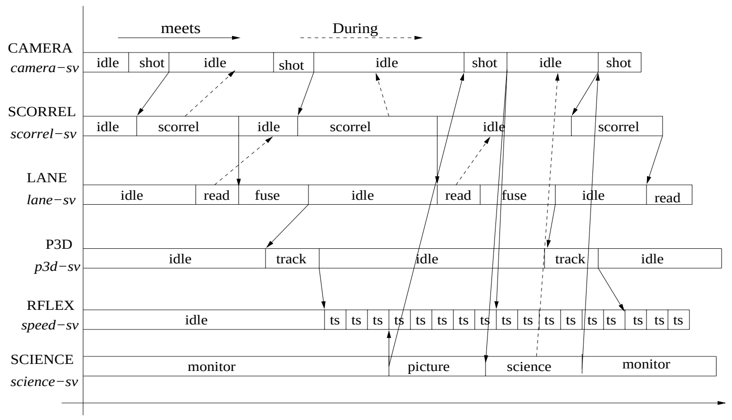

Fig. 3. Partial Gromit Model: Attributes and compatibilities

front of it, performs a stereo correlation to extract a cloud of 3D points, merges these points in its model of environment and starts this process again. In parallel, it continuously considers its currents position, the next waypoint to visit, the obstacles in the model of the environment built and produces a trajectory. These two interdependent cyclic processes are synchronized. Last, a third process interrupts whenever an interesting rock has been detected. The functional layer of Gromit is implemented using functional modules (for more details see [6]). For each of them we shall consider the "visible" state variables of interest:
- Rflex is the module interfaced with the low-level speed controller. It has a state variable for the position of the robot, each interval representing a specific robot position, and another one for the speed passed to the wheels controller.
- Camera shoots a pair of stereo calibrated images and saves them. It has one state variable representing the camera status (taking picture, or idle).
- SCorrel produces and stores a stereo correlated image. It has a state variable representing the Scorrel process (performing stereo correlation, or idle).
- Lane builds a model of the environment by aggregating clouds of 3D points produced by Scorrel. It services two requests: read in an internal buffer and fuse the read. Lane has one state variable for the model building process.
- P3D is a rover navigation software. It produces an arc trajectory which is translated in a speed reference, to try to reach a waypoint. P3D has a variable for its state (idle or computing the speed) and one for the waypoints to visit.
- Science. This module monitors a particular condition of interest to scientist (such as detecting a rock with particular features). When such a condition arises while the robot is moving toward a waypoint, it stops and takes a picture of the rock. It has one state variable for its state (monitoring interesting rock or idle).

Figure 3 shows some temporal relations representing a simplified version of the actual Gromit Rover.

Results. Due to intellectual property restrictions, we did not have access to the execution platform of the Gromit Rover. However, the CNRS Laboratory LAAS (at Toulouse, France) provided[1] us with a file formalizing some of the compat-

[1] We warmly thank Felix Ingrand for help.

ibilities as an ATL formula of more than 100 interval relations, as well as with a set of one hundred traces generated by Gromit Rover execution platform. We applied our prototype ALTL2Monitor off-line to check these traces; the checking took negligible time. However, the satisfiability checker was applied only once at the end of the monitoring session of each trace, because we expected the traces to be correct, which was indeed the case.

6 Conclusion

We presented Allen linear temporal logic (ALTL), an automated translation of ALTL into LTL, a monitor synthesis algorithm for ALTL, as well as a real-life experiment. While LTL can be a suitable logic for AI and planning, we also believe that ALTL can be a suitable logic for certain program verification efforts. Its simplicity, neutrality and visual interpretation cannot be ignored. We plan to apply our ALTL monitoring prototype to the autonomous embedded system iRobot ATRV of the LAAS Laboratory.

References

1. J. Allen. Towards a general theory of actions and time. *Artificial Intelligence*, 23(2):123–154, 1984.
2. D. Calvanese, G. De Giacomo, and M. Y. Vardi. Reasoning about actions and planning in LTL action theories. In *KR*, pages 593–602, 2002.
3. M. D'Amorim and G. Roşu. Efficient monitoring of omega-languages. In *CAV'05*, volume 3576 of *LNCS*, pages 364–378. Springer, July 2005.
4. M. Ghallab and A.M. Alaoui. Managing efficiently temporal relations through indexed spanning trees. In *IJCAI*, pages 1297–1303, 1989.
5. A.A. Krokhin, P. Jeavons, and P. Jonsson. Reasoning about temporal relations: The tractable subalgebras of Allen's interval algebra. *J. ACM*, 50(5):591–640, 2003.
6. S. Lacroix, A. Mallet, D. Bonnafous, G. Bauzil, S. Fleury, M. Herrb, and R. Chatila. Autonomous rover navigation on unknown terrains, functions and integration. *International Journal of Robotics Research*, 2003.
7. M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *Design Automation Conference (DAC'01)*, June 2001.
8. A. Pnueli. The temporal logic of programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, New York, 1977. IEEE.
9. G. Roşu and S. Bensalem. Allen linear (interval) temporal logic – translation to LTL and monitor synthesis. Technical Report UIUCDCS-R-2006-2681, University of Illinois at Urbana-Champaign, January 2006.
10. G. Roşu and K. Havelund. Rewriting-based techniques for runtime verification. *J. of Automated Software Engineering*, 12(2):151–197, 2005.
11. A. P. Sistla and E. M. Clarke. The complexity of propositional linear temporal logics. *J. ACM*, 32(3):733–749, 1985.
12. M. Vilain, H. Kautz, and P. van Beek. Constraint propagation algorithms for temporal reasoning: a revised report. In *Readings in Qualitative Reasoning about Phisical Systems.* Morgan Kaufmann, Los Altos, CA, 1989.