Non-monotonic Temporal Logics that Facilitate Elaboration Tolerant Revision of Goals

Chitta Baral and **Jicheng Zhao**
Department of Computer Science and Engineering
Arizona State University
Tempe, Arizona 85281-8809, U.S.A.
{*chitta,jicheng*}*@asu.edu*

Abstract

Temporal logics are widely used in specifying goals of agents. We noticed that when directing agents, humans often revise their requirements for the agent, especially as they gather more knowledge about the domain. However, all existing temporal logics, except one, do not focus on the revision of goals in an elaboration tolerant manner. Thus formal temporal logics that can allow elaboration tolerant revision of goals are needed. As non-monotonic languages are often used for elaboration tolerant specification, we propose to explore non-monotonic temporal logics for goal specification. Recently, a non-monotonic temporal logic, N-LTL, was proposed with similar aims. In N-LTL, goal specifications could be changed via strong and weak exceptions. However, in N-LTL, one had to a-priori declare whether exceptions will be weak or strong exceptions. We propose a new non-monotonic temporal logic, that not only overcomes this, but is also able to express exception to exceptions, strengthen and weaken preconditions, and revise and replace consequents; all in an elaboration tolerant manner.

Introduction

An important component of autonomous agent design is goal specification. Often goals of agents are not just about or not necessarily about reaching one of a particular set of states, but also about satisfying certain conditions imposed on the trajectory, or the execution structure. Besides, reactive agents with maintenance goals may not have a particular set of final states to reach. Thus the use of temporal logics and temporal connectives to specify goals has been suggested in the autonomous agent community and planning community (Barbeau, Kabanza, & St-Denis 1995; Bacchus & Kabanza 1998; Giacomo & Vardi 1999; Niyogi & Sarkar 2000; Pistore & Traverso 2001). In the decision theoretic planning community, suggestions have been made to use temporal logics in specifying non-Markovian rewards (Bacchus, Boutilier, & Grove 1996; 1997; Thiebaux *et al.* 2006).

However, in many domains such as in a human-robot interaction domain like a rescue and recovery situation, goals once specified may need to be further updated, revised, partially retracted, or even completely changed. This could be because at the time of initially specifying the goal, the user did not have complete information about the situation, or he was in haste and hence he did not completely think through the whole situation, and as the situation unveiled physically or in the user's mind, he had to make changes to his specification. The following example illustrates these points.

Suppose John has an agent in his office that does errands for him. John may ask the agent to bring him coffee. But soon after he may remember that the coffee machine was broken the day before. He is not sure if the machine is fixed or not. He then revises his directive to the agent telling it that if the coffee machine is still broken then a cup of tea would be fine. Just after that he gets a call from a colleague who says that he had called a coffee machine company and asked them to deliver a new coffee machine. Then John calls up the agent and tells it that if the new coffee machine is already there then it should bring him coffee. (Note that the old coffee machine may still be broken.) He also remembers that he takes more sugar with his tea and that the tea machine has various temperature settings. So he tells the agent that if it is going to bring tea then it should bring him an extra pack of sugar and set the tea machine setting to "very hot" right before getting the tea.

One may wonder why does not John in the above example give a well thought out directive at the start without making further changes after that. As we mentioned earlier, some of it is because he lacked certain information, such as a new coffee machine having been ordered; in another case he had forgotten about the coffee machine being broken, and since he takes tea less often, he had also initially forgotten about the extra sugar.

This is not an isolated case. In rescue and recovery situations with robots being directed by humans, there is often so much chaos together with the gradual trickling of information and misinformation that the human supervisors may have to revise their directives to the robots quite often.

To make changes to the specification, one can of course retract the earlier specification and give a completely new specification. However, that may cost precious time in terms of communication and formulation of the new specification, and may not be even appropriate, as the agent may already have started acting based on the earlier specification. A language that can be revised in an elaboration tolerant manner is necessary.

This raises the question of choosing a goal specification language that can be revised or elaborated easily. As McCarthy says in (McCarthy 1998), a natural language would be more appropriate. However, there is still a need of a formal language, sometimes as an intermediary between a natural language and the machine language and other times as a goal specification language. Considering the necessity and usefulness of temporal logics in specifying trajectories in standard planning and in specifying non-Markovian rewards in decision theoretic planning, to remain upward compatible with existing work in these directions, we prefer to stay with the temporal connectives in temporal logics. The question then is: What kind of temporal logic will allow us easy revision of specifications?

In other aspects of knowledge representation, the use of non-monotonic logics for elaboration tolerant representation (McCarthy 1998) is often advocated and for reasons similar to our example: Intelligent entities need to reason and make decisions with incomplete information and in presence of additional information they should be able to retract their earlier conclusions. Thus a non-monotonic temporal logic could be a good candidate for our purpose. Looking back at the literature, although there have been many proposals for non-monotonic logics, we could find only a few works (Fujiwara & Honiden 1991; Saeki 1987; Baral & Zhao 2007) on non-monotonic temporal logics. The first two are not appropriate for our purpose, as they do not aim for elaboration tolerant revision of specifications. The language N-LTL proposed in the last one (Baral & Zhao 2007) has many limitations such as it only allows strong exceptions and weak exceptions but does not allow arbitrary revising or retracting existing sub-formulas. Besides, when there is an exception in N-LTL, it must be predefined whether it is a weak exception or a strong exception. These limitations restrict the ability of N-LTL to specify goals in an evolving scenario.

The main objective of this paper is to develop an appropriate non-monotonic temporal goal specification language that allows elaboration tolerant revision of goal specifications. Thus, we develop the language ER-LTL, which is based on LTL (Manna & Pnueli 1992). Each ER-LTL program is composed of a set of rules of the form

$$\langle h : [r](f_1 \rightsquigarrow f_2)\rangle \tag{1}$$

The symbol h is referred to as the head of the rule and Rule 1 states that, normally, if formula f_1 is true, then the formula f_2 should be true, with exceptions given by rules with r in their heads and this rule is an exception to a formula labeled by h. In ER-LTL, we also take a similar approach as in N-LTL and use Reiter's idea of a surface non-monotonic logic (Reiter 2001), sentences of which get compiled into sentences of a more tractable logic and thus avoid increase in complexity. We use the idea of completion when rules about exceptions are given for the same precondition. We will show that with simple rules such as Rule 1, we are able to express various ways to revise goals. This includes specification of exceptions to exceptions, strengthening and weakening of preconditions, and revision and replacement of consequents.

The rest of the paper is organized as follows: We first give background on goal specification using LTL. We then propose the syntax and semantics of the new language ER-LTL. We then illustrate with examples how ER-LTL can be used in specifying goals and revising them in an elaboration tolerant manner. Finally we discuss related works and conclude.

Background: Goal Representation Using LTL

Syntactically, LTL formulas are made up of propositions, propositional connectives $\vee$, $\wedge$, and $\neg$, and future temporal connectives $\bigcirc$, $\Box$, $\Diamond$ and U. We now formally define truth values of temporal formulas with respect to trajectories. A *trajectory* is an infinite sequence of states.

Definition 1 Let $\langle p\rangle$ be an atomic proposition, $\langle f\rangle$ an LTL formula.

$$\langle f\rangle \quad ::= \quad \langle p\rangle | (\langle f\rangle \wedge \langle f\rangle) \mid (\langle f\rangle \vee \langle f\rangle) \mid \neg\langle f\rangle \mid \bigcirc\langle f\rangle \mid \Box\langle f\rangle \mid \Diamond\langle f\rangle \mid (\langle f\rangle \mathsf{U} \langle f\rangle)$$

Definition 2 Let σ given by $s_0, s_1, \ldots, s_k, s_{k+1}, \ldots$ be a trajectory, p denote a propositional formula, s_j ($j \geq 0$) denote a state in σ, and f and f_is ($i = 1, 2$) denote LTL formulas.

- $(j, \sigma) \models p$ iff p is true in s_j.
- $(j, \sigma) \models \neg f$ iff $(j, \sigma) \not\models f$.
- $(j, \sigma) \models (f_1 \vee f_2)$ iff $(j, \sigma) \models f_1$ or $(j, \sigma) \models f_2$.
- $(j, \sigma) \models (f_1 \wedge f_2)$ iff $(j, \sigma) \models f_1$ and $(j, \sigma) \models f_2$.
- $(j, \sigma) \models \bigcirc f$ iff $(j + 1, \sigma) \models f$.
- $(j, \sigma) \models \Box f$ iff $(k, \sigma) \models f$, for all $k \geq j$.
- $(j, \sigma) \models \Diamond f$ iff $(k, \sigma) \models f$, for some $k \geq j$.
- $(j, \sigma) \models (f_1 \mathsf{U} f_2)$ iff there exists $k \geq j$ such that $(k, \sigma) \models f_2$ and for all i, $j \leq i < k$, $(i, \sigma) \models f_1$. □

As usual, we denote $(\neg f_1 \vee f_2)$ as $(f_1 \Rightarrow f_2)$. Often (Bacchus & Kabanza 1998) planning with respect to LTL goals is formalized with the assumption that there is complete information about the initial state, and the actions are deterministic. Let Φ be a transition function that defines transitions between states due to actions. $\Phi(s_i, a_k) = s_j$ iff the action a_k transitions the agent from state s_i to state s_j. Let s be a state designated as the initial state, and let $a_1, \ldots, a_n$ be a sequence of deterministic actions whose effects are described by a domain description. *The trajectory corresponding to s and $a_1, \ldots, a_n$* is the sequence $s_0, s_1, \ldots$, that satisfies the following conditions: $s = s_0$, $s_{i+1} = \Phi(s_i, a_{i+1})$, for $0 \leq i \leq n - 1$, and $s_{j+1} = s_j$, for $j \geq n$. A sequence of actions $a_1, \ldots, a_n$ is a *plan* from the initial state s for the LTL goal f, if $(0, \sigma) \models f$, where σ is the trajectory corresponding to s and $a_1, \ldots, a_n$.

Let f_1, and f_2 be two LTL formulas. We define that $f_1 \models f_2$ if for any index i and trajectory σ, $(i, \sigma) \models f_1$ implies $(i, \sigma) \models f_2$. LTL is monotonic as for any LTL formulas f_1, f_2, and f_3, whenever $f_1 \models f_2$, we have $(f_1 \wedge f_3) \models f_2$.

ER-LTL

We now present the non-monotonic temporal logic ER-LTL that is based on LTL; ER stands for "*E*xceptions and *R*evisions". We first define the syntax and semantics of the language.

Syntax

Definition 3 (ER-LTL program) Let G, R, and P be three disjoint sets of *atoms*. Let g be the only atom in G. Let $\langle r \rangle$ be an atom in R, $\langle p \rangle$ be an atom in P. An *ER-LTL formula* $\langle f \rangle$ is defined recursively as:

$$\langle f \rangle ::= \langle p \rangle \mid (\langle f \rangle \wedge \langle f \rangle) \mid (\langle f \rangle \vee \langle f \rangle) \mid \neg\langle f \rangle \mid \bigcirc \langle f \rangle \mid \Box\langle f \rangle \mid \Diamond\langle f \rangle \mid (\langle f \rangle \mathsf{U} \langle f \rangle) \mid [\langle r \rangle](\langle f \rangle \rightsquigarrow \langle f \rangle)$$

An *ER-LTL rule* is of the form $\langle h : [r](f_1 \rightsquigarrow f_2) \rangle$, where $h \in G \cup R$, $r \in R$, and f_1 and f_2 are two ER-LTL formulas; h is referred to as the *head*, and $[r](f_1 \rightsquigarrow f_2)$ as the *body* of the rule. An *ER-LTL program* is a finite set of ER-LTL rules. □

We allow the symbols $\top$ and $\bot$ as abbreviations for propositional formulas that evaluate to *true* and *false* respectively. For example, for atom $q \in P$, $q \vee \neg q$ is abbreviated as $\top$, and $q \wedge \neg q$ is abbreviated as $\bot$.

In an ER-LTL program, rules with head g express the initial goal which may later be refined.

In comparison to LTL, $[r](f_1 \rightsquigarrow f_2)$ is the only new constructor in ER-LTL. We refer to f_1 as the precondition and f_2 as the consequent of this formula. It states that normally if the precondition f_1 is true, then the consequent f_2 needs to be satisfied, with the exceptions specified via r. The conditions denoting the exceptions labeled by r are defined using other rules. When those exceptions are presented in the program, we might need to satisfy other goals instead of f_2. If the sub-formula is preceded with a head atom $h \in R \cup G$ as in $\langle h : [r](f_1 \rightsquigarrow f_2) \rangle$, it further states that this sub-formula is an exception to formulas labeled by h.

We now define several auxiliary definitions that will be used in defining the semantics of ER-LTL.

Definition 4 (Atom dependency) Let T be an ER-LTL program. Let h_1, h_2 be atoms in $R \cup G$. Atom h_1 *depends* on h_2 in T if there is a rule in T such that h_2 occurs in the body of the rule while h_1 is the head of the rule. The dependency relation is transitive. □

Example 1 *Consider the following rules:*

$$\langle r_1 : [r_2](p \rightsquigarrow ((\Box q) \rightsquigarrow r)) \rangle \quad (2)$$

$$\langle r_1 : [r_2]((\Diamond p \vee [r_3](\Box q \rightsquigarrow (p \,\mathsf{U}\, q))) \rightsquigarrow (\Diamond q)) \rangle \quad (3)$$

Rule 2 is not a syntactically valid ER-LTL rule. $((\Box q) \rightsquigarrow r)$ *in it is not a valid ER-LTL formula. It should be preceded by a label in R. Rule 3 is a valid ER-LTL rule. With respect to a program consisting of Rule 3,* r_1 *depends on* r_2 *and* r_3*.*

Definition 5 (Loop-free) An ER-LTL program is *loop-free* if in the formula, no atom in R depends on itself. □

Definition 6 (Leaf) In an ER-LTL program, an atom is called a *leaf* if it does not depend on any atom in R. □

Semantics

We now define a translation from ER-LTL to LTL so as to relate the semantics of ER-LTL to the semantics of LTL. We use a similar technique as in N-LTL to capture temporal relations among different rules to combine them to be one temporal formula. The preconditions of rules are dealt with as branches in a tree. Atom r_1 depends on r_2 states that r_2 should be fully expanded before r_1.

Definition 7 Given a finite loop-free ER-LTL program T, we translate it to an LTL formula $Tr(T)$ as follows:

1. For each sub-formula in T of the form $[r_t](l_{t0} \rightsquigarrow f_{t0})$ where for all rules with r_t in the head:

   $\langle r_t : [r_{t1}](l_{t1} \rightsquigarrow f_{t1}) \rangle \cdots \langle r_t : [r_{tk}](l_{tk} \rightsquigarrow f_{tk}) \rangle$

   we have that r_{ti} $(1 \leq i \leq k)$ are leaf atoms, l_{ti}, f_{ti} $(0 \leq i \leq k)$ are LTL formulas.

   (a) If $[r_t](l_{t0} \rightsquigarrow f_{t0})$ is not preceded with ":", we replace the formula $[r_t](l_{t0} \rightsquigarrow f_{t0})$ with $(l_{t0} \wedge \neg l_{t1} \wedge \cdots \wedge \neg l_{tk} \Rightarrow f_{t0}) \wedge (l_{t0} \wedge l_{t1} \Rightarrow f_{t1}) \wedge \cdots \wedge (l_{t0} \wedge l_{tk} \Rightarrow f_{tk})$. The resulting program is still called T;

   (b) If $[r_t](l_{t0} \rightsquigarrow f_{t0})$ is preceded with ":" and it is in a rule of the form $\langle r_v : [r_t](l_{t0} \rightsquigarrow f_{t0}) \rangle$ where $r_v \in G \cup R$, we replace the rule with:

   $\langle r_v : [r_t](l_{t0} \wedge \neg l_{t1} \wedge \cdots \wedge \neg l_{tk} \rightsquigarrow f_{t0}) \rangle$

   $\langle r_v : [r_t](l_{t0} \wedge l_{t1} \rightsquigarrow f_{t1}) \rangle \cdots \langle r_v : [r_t](l_{t0} \wedge l_{tk} \rightsquigarrow f_{tk}) \rangle$

   The resulting program is still called T.
2. Repeat Step 1 until it can no longer be applied further.
3. Suppose $\langle g : [r_i](l_i \rightsquigarrow f_i) \rangle$ ($0 \leq i \leq n$) are all rules with the head g. We define $Tr(T)$ as $\wedge_{i=0}^{n}(l_i \Rightarrow f_i)$. □

Example 2 *An ER-LTL program T is given as follows*[1]*:*

$$\langle g : [r_1](bird \rightsquigarrow fly) \rangle \quad (4)$$

$$\langle r_1 : [r_2](penguin \rightsquigarrow \neg fly) \rangle \quad (5)$$

$$\langle r_1 : [r_3](wounded \rightsquigarrow \top) \rangle \quad (6)$$

$$\langle r_2 : [r_4](flying\_penguin \rightsquigarrow fly) \rangle \quad (7)$$

After the first processing of step 1 of Definition 7, we get the set of rules:

$$\langle g : [r_1](bird \rightsquigarrow fly) \rangle$$
$$\langle r_1 : [r_2](penguin \wedge \neg flying\_penguin \rightsquigarrow \neg fly) \rangle$$
$$\langle r_1 : [r_3](wounded \rightsquigarrow \top) \rangle$$
$$\langle r_1 : [r_2](penguin \wedge flying\_penguin \rightsquigarrow fly) \rangle$$

After the second processing of step 1, we get the set of rules:

$$\langle g : [r_1](bird \wedge \neg penguin \wedge \neg wounded \rightsquigarrow fly) \rangle$$
$$\langle g : [r_1](bird \wedge penguin \wedge \neg flying\_penguin \rightsquigarrow \neg fly) \rangle$$
$$\langle g : [r_1](bird \wedge wounded \rightsquigarrow \top) \rangle$$
$$\langle g : [r_1](bird \wedge penguin \wedge flying\_penguin \rightsquigarrow fly) \rangle$$

[1]Here and in a later example we use the flying bird example that has been used a lot in the non-monotonic reasoning literature. This is only for quick illustration purposes, and not to suggest that our language is an alternative to traditional non-monotonic languages. There has been significant progress in the research on non-monotonic reasoning and we are not ready to claim our language as an alternative. Our claim is only with respect to non-monotonic temporal logics, which have not been explored much.

Finally, based on step 3, we get $Tr(T)$ and then simplify it to: $(bird \wedge \neg penguin \wedge \neg wounded \Rightarrow fly) \wedge (bird \wedge penguin \wedge \neg flying\_penguin \Rightarrow \neg fly) \wedge (bird \wedge penguin \wedge flying\_penguin \Rightarrow fly)$.

ER-LTL in Goal Specification

Loop-free ER-LTL programs have the following property.

Proposition 1 *Given a loop-free ER-LTL program T, $Tr(T)$ is an LTL formula.*

Given this property, we can define when a plan satisfies an ER-LTL program.

Definition 8 Let T be a loop-free ER-LTL program, $\sigma = s_0, s_1, \cdots, s_k, \cdots$ be a trajectory, and i be an index of σ. $(i, \sigma) \models T$ in ER-LTL if $(i, \sigma) \models Tr(T)$ with respect to LTL. □

We say an ER-LTL program T is equivalent to an LTL formula T' if $Tr(T)$ and T' are equivalent in LTL. For any LTL formula G, we can have an ER-LTL program T such that $Tr(T)$ and G are equivalent.

To plan with an ER-LTL goal T, we find plans for LTL formula $Tr(T)$. When T is updated to $T \cup T'$, we need to find plans for LTL formula $Tr(T \cup T')$.

Definition 9 (Entailment) Given that T_1 and T_2 are loop-free ER-LTL programs, $T_1 \models T_2$ if $Tr(T_1) \models Tr(T_2)$ in LTL. □

Proposition 2 *The entailment in Definition 9 is non-monotonic.*

This implies that it is possible that a plan satisfies an ER-LTL program $T_1 \cup T_2$ but not T_1. It should be noted that the opposite is also true.

Example 3 *Consider the following two ER-LTL rules:*

$$\langle g : [r_1](\top \rightsquigarrow \Box p)\rangle \quad (8)$$

$$\langle r_1 : [r_2](\Diamond q \rightsquigarrow \Diamond q)\rangle \quad (9)$$

Let T_1 be a program consisting of Rule 8, and T_2 be a program consisting of Rule 8 and Rule 9. $Tr(T_1) = \Box p$ while $Tr(T_2) = \Diamond q \vee \Box p$. It is easy to see that $T_1 \models T_1$, while $T_2 = T_1 \cup \{Rule\ 9\}$ and $T_2 \not\models T_1$. Thus the entailment relation defined in Definition 9 in ER-LTL is non-monotonic.

Exceptions and Revisions in ER-LTL

We now illustrate the application of ER-LTL in modeling exceptions and revisions. We start with the modeling of exceptions that happen mainly because the user has incomplete information about the domain, the domain has been changed after the initial goal is given, or the user does not have a clear specification for the agent initially.

Exceptions

We first consider the differences between weak exceptions and strong exceptions in goal specification.

Weak Exception and Strong Exception: Strong exceptions are to refute the default conclusion when exceptions happen; Weak exceptions are to render the default inapplicable. In terms of goal specification, suppose $f_1 \wedge f_2$ is the initial goal we have, after having the weak exception on f_1, we do not know whether sub-goal f_1 should be true or not, we thus can remove the sub-formula f_1 from the existing specification. On the other hand, if we have a strong exception on f_1, we should conclude that f_1 is no longer true, and cannot be true. Thus, we need to have $\neg f_1$ as a part of the revised goal specification. Let us consider the following example, again, for simplicity, given with respect to the birds flying scenario.

Example 4 *We know that birds normally fly. Penguins are birds that do not fly. We do not know whether wounded birds fly or not.*

The initial statement can be written as Rule 4 in Example 2. It is equivalent to the LTL formula $bird \Rightarrow fly$. If we append Rule 5 to Rule 4, the program is equivalent to the LTL formula $((bird \wedge \neg penguin) \Rightarrow fly) \wedge ((bird \wedge penguin) \Rightarrow \neg fly)$. If we append Rule 6 about wounded birds to Rule 4, we have a program that is equivalent to the LTL formula $((bird \wedge \neg wounded) \Rightarrow fly)$.

This example shows that when we need a strong exception, we can specify the negation of the initial consequents explicitly as in Rule 5. When we need a weak exception, we can simply say as in Rule 6 that under the exception, no consequents are needed.

Exception to Exception: We illustrate the way we deal with exceptions to exception by the following example.

Example 5 *We know that birds normally fly. Penguins are birds that do not fly. However, a flying penguin is a penguin that can fly.*

We write the initial statement as Rule 4 in Example 2. Later, Rule 5 and Rule 7 are appended. Rule 7 is an exception to the exception stated in Rule 5. The program consisting of the three rules is equivalent to the LTL formula $(bird \wedge (\neg penguin \vee flying\_penguin) \Rightarrow fly) \wedge (bird \wedge penguin \wedge \neg flying\_penguin \Rightarrow \neg fly)$.

Revision: Change User Intentions

We are also able to deal with various revisions in ER-LTL. In ER-LTL, we split the requirements to preconditions and consequents such that we may have goals as "if some conditions are satisfied, the agent should satisfy some goals". We now list a few approaches of revising preconditions and consequents. They help to revise any part of the initial ER-LTL goal. In the following, we consider a simple example where the initial ER-LTL program is:

$$\langle g : [r_1](f_1 \rightsquigarrow f_2)\rangle \quad (10)$$

where f_1 and f_2 are two LTL formulas.

Changing Consequents: We start with exploring ways to change consequents in the goal.

Example 6 *To Rule 10, if we append the ER-LTL rule*

$$\langle r_1 : [r_2](f_1 \rightsquigarrow f_3)\rangle, \quad (11)$$

where f_3 is an LTL formula, the revised program is equivalent to the LTL formula $((f_1 \wedge \neg f_1) \Rightarrow f_2) \wedge ((f_1 \wedge f_1) \Rightarrow f_3)$, or $f_1 \Rightarrow f_3$. Now the consequent has changed from f_2 to f_3.

We can change the consequent to be stronger or weaker than the initial specification, or we can revise it to one that is different from the initial specification. We can make similar revisions for preconditions.

Changing Preconditions: We now list a few examples illustrating how to change preconditions in a goal specification.

Example 7 (Making Preconditions Stronger) *Suppose we want to refine the goal given as Rule 10 by having a new precondition f_3 together with f_1. We refine the program by appending the rule:*

$$\langle r_1 : [r_2](\neg f_3 \rightsquigarrow \top)\rangle. \tag{12}$$

The new formula states that if $\neg f_3$ is satisfied, then the goal is satisfied naturally. The refined program is equivalent to the LTL formula $(f_1 \wedge f_3) \Rightarrow f_2$.

Example 8 (Making Preconditions Weaker) *Suppose we want to refine the goal given as Rule 10 so that under a new condition f_3, we also need to satisfy the consequent f_2. This refinement will weaken the precondition f_1. We refine the program by appending the rule:*

$$\langle g : [r_1](f_3 \rightsquigarrow f_2)\rangle. \tag{13}$$

The new program is equivalent to the LTL formula $(f_1 \vee f_3) \Rightarrow f_2$.

Example 9 (Changing Preconditions) *Suppose we want to refine the goal given as Rule 10 so as to change the precondition f_1 to f_3. We can do this by appending the following rules*

$$\langle r_1 : [r_2](f_1 \rightsquigarrow \top)\rangle$$
$$\langle g : [r_3](f_3 \rightsquigarrow f_2)\rangle$$

to the program consisting of Rule 10. The new program is equivalent to the LTL formula $((f_1 \wedge \neg f_1) \Rightarrow f_2) \wedge (f_1 \wedge f_1 \Rightarrow \top) \wedge (f_3 \Rightarrow f_2)$, which can be simplified as $f_3 \Rightarrow f_2$.

Revision after Revision: We now consider an example that needs further revision after the first revision.

Example 10 *In Example 6, the revised program is equivalent to the LTL formula $f_1 \Rightarrow f_3$. If we want to further revise the consequent to f_4, and make the program equivalent to $f_1 \Rightarrow f_4$, we can add the rule $\langle r_2 : [r_3](f_1 \rightsquigarrow f_4)\rangle$ to the existing program.*

Nested Revision: Nested revisions are also common when we introduce a new goal to the domain while not clear about the preconditions and consequents of the new goal. We need rules that specify that the preconditions and consequents will be given later. We illustrate this by the following example.

Example 11 *Suppose the initial ER-LTL program is*

$$\langle g : [r_1](\top \rightsquigarrow f_1)\rangle. \tag{14}$$

Suppose now we know in addition to f_1 some thing more needs to be done; but we do not yet know what. We can append the following rule to accommodate that possibility:

$$\langle r_1 : [r_2](\top \rightsquigarrow f_1 \wedge [r_3](\top \rightsquigarrow \top))\rangle, \tag{15}$$

It will allow us to add additional requirements later.

Thus ER-LTL enables us in revising goals of an agent in an elaboration tolerant manner. We now elaborate on how we can represent the evolution of John's requirement that we introduced in the Introduction section.

Representing John's Requirements in ER-LTL

Example 12 *John can specify his initial goal in ER-LTL as:*

$$\langle g : [r_0](\top \rightsquigarrow \Diamond(\mathit{coffee} \wedge \Diamond \mathit{back}))\rangle. \tag{16}$$

It is equivalent to the LTL formula $\Diamond(\mathit{coffee} \wedge \Diamond \mathit{back})$. It states that the agent needs to get a cup of coffee and then come back.

After realizing that the coffee machine might be broken, John can refine his goal by adding the following two rules:

$$\langle r_0 : [r_1](\top \rightsquigarrow \Diamond([r_2](\top \rightsquigarrow \mathit{coffee}) \wedge \Diamond \mathit{back}))\rangle \tag{17}$$
$$\langle r_2 : [r_3](\mathit{broken} \rightsquigarrow \mathit{tea})\rangle \tag{18}$$

Rule 17 now allows the sub-formula about coffee in the initial goal to be further refined. The overall specification is now equivalent to the LTL formula $\Diamond((\neg \mathit{broken} \Rightarrow \mathit{coffee}) \wedge (\mathit{broken} \Rightarrow \mathit{tea}) \wedge \Diamond \mathit{back})$. Notice that John did not have to retract his previous goal and give a new goal; neither did he have to change the earlier specification; he just had to add to his previous specification and the semantics of the language takes care of the needed change. This is an example of "elaboration tolerance" of a language.

Later, after knowing from a colleague that a new coffee machine might be installed, John can give the agent a new command by adding one more rule to the existing goal:

$$\langle r_3 : [r_4](\mathit{newMachine} \rightsquigarrow \mathit{coffee})\rangle \tag{19}$$

The overall goal is now equivalent to the LTL formula $\Diamond((\mathit{broken} \wedge \neg \mathit{newMachine} \Rightarrow \mathit{tea}) \wedge (\neg(\mathit{broken} \wedge \neg \mathit{newMachine}) \Rightarrow \mathit{coffee}) \wedge \Diamond \mathit{back})$.

Finally, John can give the agent a new command by adding the following rule.

$$\langle r_3 : [r_5](\neg \mathit{newMachine} \rightsquigarrow (\mathit{hot} \wedge \bigcirc(\mathit{tea} \wedge \mathit{sugar})))\rangle$$

The overall goal is now equivalent to the LTL formula: $\Diamond((\mathit{broken} \wedge \neg \mathit{newMachine} \Rightarrow (\mathit{hot} \wedge \bigcirc(\mathit{tea} \wedge \mathit{sugar})) \wedge (\neg(\mathit{broken} \wedge \neg \mathit{newMachine}) \Rightarrow \mathit{coffee})) \wedge \Diamond \mathit{back})$.

Note that in this example, we illustrated the way of expanding and revising the goal in an elaboration tolerant manner by introducing a different consequent, weakening the requirements, introducing exceptions to exceptions, and introducing nested exceptions.

Strengthen and Weakening in ER-LTL

In this section, we consider how the new rules added to the program affect the existing ER-LTL program.

In ER-LTL, with the introduction of preconditions and consequents, we branch on preconditions. Given a loop-free ER-LTL program, adding a new rule with head g corresponds to adding a new branch to the tree composed of the branches. Adding a new rule with head $r \in R$ corresponds to adding a new branch, or revising existing branches. A new tree is introduced whenever there is a nested rule. We can refine the tree by adding or removing some branches based on the new rules added to the program. For example, given an ER-LTL rule of the form:

$$\langle r_1 : [r_2](f_1 \rightsquigarrow f_2)\rangle$$

Another rule with head r_2 of the form

$$\langle r_2 : [r_3](f_3 \rightsquigarrow f_4)\rangle$$

introduces a new branch if $f_3 \not\models f_1$. Otherwise, the existing branch on f_2 is removed and the new branch on f_4 is added.

With the different rules added, we may make the goal easier or more difficult to satisfy:

Definition 10 *Given two ER-LTL programs T_1 and T_2, if $T_1 \cup T_2 \models T_1$, we call T_2 a strengthening of T_1; if $T_1 \models T_1 \cup T_2$, we call T_2 a weakening of T_1.*

Proposition 3 *(a) A rule of the form*

$$\langle h : [r](f \rightsquigarrow \top)\rangle$$

is a weakening of any ER-LTL program, where $h \in R \cup G$, $r \in R$, and f is a well defined ER-LTL formula.
(b) A rule of the form

$$\langle g : [r](f_1 \rightsquigarrow f_2)\rangle$$

is a strengthening of any ER-LTL program, where $g \in G$, $r \in R$, and f_1 and f_2 are well defined ER-LTL formulas.

After weakening a program, the new ER-LTL program is satisfied by more policies and after strengthening a program, the new ER-LTL program is satisfied by fewer policies.

Conclusion and Future Work

In this paper, we develop ER-LTL, a non-monotonic temporal logic that can be used for specifying goals which can then be revised in an elaboration tolerant manner. We borrowed the idea of completion and exception from logic programming and the idea of a surface non-monotonic logic, sentences of which can be translated to sentences in a monotonic logic, from Reiter. Our approach of extending LTL can be used to extend other monotonic temporal logics such as CTL and CTL*.

Earlier we mentioned the existing three non-monotonic temporal logics and their relationship with our work. We elaborate a bit more on how our work compares with N-LTL (Baral & Zhao 2007). Similar to N-LTL, each ER-LTL program is composed of a set of rules. Labels are used to combine multiple rules to be a formula and denote the relations of multiple temporal formulas. But we have a richer syntax and semantics than N-LTL, and have a different way of computing completion. N-LTL does not have the $\rightsquigarrow$ construct, and there goal specifications could only be changed via strong and weak exceptions; but one had to a-priori declare whether an exception will be weak or strong exception. ER-LTL not only overcomes this, but is also able to express exceptions to exceptions, strengthening and weakening of preconditions, and revise and replace consequents; all in an elaboration tolerant manner.

One aspect of our research which we could not fit in these pages is progressing of goals in ER-LTL. This is important because, as an agent executes some actions and satisfies some sub-goals, the remaining goal for the agent changes. In terms of future work, the approach we use in syntactic formula revision is not restricted to temporal formulas. Its implications vis-a-vis existing non-monotonic logics, belief revision mechanisms, and formalizing natural language discourses needs to be explored.

References

Bacchus, F., and Kabanza, F. 1998. Planning for temporally extended goals. *Annals of Math and AI* 22:5–27.

Bacchus, F.; Boutilier, C.; and Grove, A. 1996. Rewarding behaviors. In *AAAI 96*, 1160–1167.

Bacchus, F.; Boutilier, C.; and Grove, A. 1997. Structured solution methods for non-markovian decision processes. In *AAAI 97*, 112–117.

Baral, C., and Zhao, J. 2007. Non-monotonic temporal logics for goal specification. In *IJCAI-07*, 236–242.

Barbeau, M.; Kabanza, F.; and St-Denis, R. 1995. Synthesizing plan controllers using real-time goals. In *IJCAI*, 791–800.

Fujiwara, Y., and Honiden, S. 1991. A nonmonotonic temporal logic and its kripke semantics. *J. Inf. Process.* 14(1):16–22.

Giacomo, G. D., and Vardi, M. 1999. Automata-theoretic approach to planning for temporally extended goals. In *ECP*, 226–238.

Manna, Z., and Pnueli, A. 1992. *The temporal logic of reactive and concurrent systems: specification*. Springer Verlag.

McCarthy, J. 1998. Elaboration tolerance. In *Common Sense*.

Niyogi, R., and Sarkar, S. 2000. Logical specification of goals. In *Proc. of 3rd international conference on Information Technology*, 77–82.

Pistore, M., and Traverso, P. 2001. Planning as model checking for extended goals in non-deterministic domains. In *IJCAI'01*, 479–486.

Reiter, R. 2001. *Time, concurrency and processes*. MIT. chapter Knowledge in action: Logical Fundations for specifying and implementing dynamical systems, 149–183.

Saeki, M. 1987. Non-monotonic temporal logic and its application to formal specifications (in japaneese). *Transactions of IPS Japan* 28(6):547–557.

Thiebaux, S.; Gretton, C.; Slaney, J.; Price, D.; and Kabanza, F. 2006. Decision-theoretic planning with non-markovian rewards. *Journal of AI Research, pages 17–74, 2006.* 25:17–74.