Non-monotonic Temporal Logics for Goal Specification

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

Abstract

One of the main ways to specify goals of agents is to use temporal logics. Most existing temporal logics are monotonic. However, in representing goals of agents, we often require that goals be changed non-monotonically. For example, the initial goal of the agent may be to be always in states where p is true. The agent may later realize that under certain conditions (exceptions) it is ok to be in states where p is not true. In this paper, we propose a simple extension of LTL, which we call N-LTL, that allows non-monotonic specification of goals. We study properties of N-LTL. We also consider a translation from N-LTL to logic programs and study the relationship between N-LTL and logic programs.

1 Introduction and Motivation

In specifying goals of agents [Bacchus and Kabanza, 1998; Niyogi and Sarkar, 2000; Baral *et al.*, 2001; Baral and Zhao, 2004; 2006] many have used temporal logics such as linear temporal logic LTL, branching time temporal logics CTL*, π-CTL*, and their extensions. However, all these logics are monotonic. In specifying goals of agents, often there is a need of being able to specify goals non-monotonically. For example, initially, an agent may be given a goal of having p true through the trajectory while reaching s. Later, the agent may decide to weaken its goal so that in certain exceptional cases p does not have to be true. One way to do that would be for the agent to replace its original goal by a revised goal. Coming up with a completely new revised goal, or obtaining the revised goal by doing surgery on the original goal specification violate the principle of elaboration tolerance [McCarthy, J., 1988]. But if we use existing temporal logics, we do not have a choice. What we need is a goal specification language that allows us to update the goal specification by simply adding new statements to the original specification. Such a goal specification language would be non-monotonic.

Another motivation for having a non-monotonic goal specification language that allows easy updating through adding is that we may not want to give the agent a directive that is too specific, too complicated, and that takes into account all possible exceptions, from the very beginning. Besides we may not even know all the exceptions initially. A good non-monotonic goal specification language should allow us to specify a simple goal initially and should allow us to refine it by adding new exceptions.

Although non-monotonic versions of various modal logics [McDermott, 1982] have been proposed in the past, so far we have come across only two[1] [Fujiwara and Honiden, 1991; Saeki, 1987] non-monotonic versions of temporal logics. The first extends auto-epistemic logic with temporal operators and does not explore issues such as elaboration tolerant representation of exceptions and weak exceptions. The second has semantics issues that are mentioned in the first.

In this paper we start with developing a non-monotonic temporal logic based on the linear temporal logic LTL and then briefly mention non-monotonic versions of a branching time temporal logic. In designing our non-monotonic versions of temporal logics, we keep focus on our overall aim of having non-monotonic goal languages. So rather than follow the path of non-monotonic modal logics and auto-epistemic logic we focus on specific aspects of knowledge representation that need non-monotonicity and borrow some specific techniques that allow such non-monotonicity.

One of the important use of non-monotonicity is the ability to express normative statements such as "normally q's have the property p." This resonates well with our need of non-monotonic goal languages as we may need to specify that "normally a state should satisfy the property p". Accompanying normative statements we have various kinds of exceptions. For example, consider the age old normative statement "birds normally fly". One kind of exception (called "strong exceptions") to such a statement is that "penguins are birds that do not fly". Another kind of exception, referred to as weak exceptions, is that "injured birds are weak exceptions to the normative statement about birds flying;" as for wounded birds we do not know whether they fly or not. There is a need of similar exceptions with respect to goal specifications. A normative goal specification may specify that "normally a state should satisfy the property p". Strong exceptions may be states that satisfy some specific conditions, while weak exceptions may be other states that satisfy some other specific conditions.

[1] There are several other papers, such as [Giannotti *et al.*, 1999; Stein and Morgenstern, 1994; MacNish and Fallside, 1990], which from their titles seem like dealing with non-monotonic temporal logics. However, they are not quite related.

To accommodate the above we introduce two special notations[2]

- $[r]\phi$
- $[[r]]\phi$

The intuitive meaning of the first one is that normally ϕ holds in a state and the label r lists the weak exceptions. The intuitive meaning of the second one is that normally ϕ holds in a state and the label r lists the strong exceptions.

The role of r here is similar to the role of labeling defaults and normative statements when representing them in logic programming. There, often the label is used as a parameter with respect to the ab predicates.

Since the non-monotonicity in goal languages is not due to having incomplete knowledge about the states, but rather due to the specifier not quite precisely knowing what she wants, we do not use operators such as the negation as failure operator 'not' from logic programming. Here the issue is different from inferring or assuming negation by default.

On the other hand we borrow the idea behind program completion in logic programming to specify and interpret the conditions listed corresponding to the label r. Thus there may be a set of conditions written as

$$\langle r : \psi_1 \rangle \quad \ldots \quad \langle r : \psi_k \rangle$$

that specify the strong exception or weak exception conditions with respect to r. Given the above, the overall condition associated with r becomes $\psi_1 \vee \ldots \vee \psi_k$. One is allowed to add additional conditions. For example, if $\langle r : \psi_{k+1} \rangle$ is added to the above set then the overall condition associated with r becomes $\psi_1 \vee \ldots \vee \psi_k \vee \psi_{k+1}$.

We now illustrate the above with respect to an example.

Example 1 *Suppose initially we want to maintain p true while reaching for s. We know beforehand that our aim to maintain p is not strict; it is just that we do not know yet, under what conditions truth of p may not be necessary. After a while, we realize that when q is true we may not need to have p true.*

The initial goal can be written in our language as $\langle g : (\Box[r]p) \wedge \Diamond s \rangle$. It says that we should maintain p while reaching s. If the strong exception r happens in some states, we may not need to maintain p in those states. The weak exception r is then specified as $\langle r : q \rangle$.

To informally illustrate how non-monotonicity is manifested in the above example, we first define when a language is monotonic.

Definition 1 A logic L together with a query language Q and entailment relation $\models$ is monotonic if for all T, T' in L and t in Q, $T \models t$ implies $T \cup T' \models t$. □

With respect to Example 1 let T be $\{\langle g : (\Box[r]p) \wedge \Diamond s \rangle\}$, $T' = \{\langle r : q \rangle\}$. Intuitively, $T \models \Box p$ and $T \cup T'$ is equivalent to $\Box(p \vee q) \wedge \Diamond s$ in LTL; thus $T \cup T' \not\models \Box p$. This illustrates the non-monotonicity of our proposed language. We now move to the formal definition of our language and its semantics.

The rest of this paper is organized as follows: Since our first non-monotonic goal language N-LTL is based on LTL, we first give an overview of the temporal logic LTL. Then, in Section 3, we propose the syntax and semantics of a new language N-LTL, together with its properties. In Section 4, we study the relationship between N-LTL and logic programs. We then briefly discuss how one can use a similar approach to develop non-monotonic versions of other temporal logics such as CTL*.

2 Background: Goal Representation Using LTL

In this section, we summarize the linear temporal logic LTL and what it means for a sequence of actions to be a plan with respect to a goal in 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 2 Let $\langle p \rangle$ be an atomic proposition, $\langle f \rangle$ an LTL formula.

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

□

Definition 3 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, and f and f_is ($i = 1, 2$) denote LTL formulas.

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

Often [Bacchus and Kabanza, 1998; Baral *et al.*, 2001] planning with respect to LTL goals are 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 transits 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.

Since truth of LTL formulas are defined with respect to a reference state and a trajectory made up of an infinite sequence of states, to define the correctness of a plan consisting

[2]Unlike traditional exceptions and weak exceptions, we want the specifier to have pre-decided control over whether a particular goal fragment could have a weak exception or an exception, but not both.

of a finite sequence of actions with respect to an initial state and an LTL goal, the finite sequence of states obtained by applying the plan to the initial state is appended with the last state infinitely.

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 $j \geq n$.

Definition 4 (Plans with respect to LTL goals) The sequence of actions $a_1, \ldots, a_n$ is a *plan* from the initial state s for the LTL goal f, if $(s, \sigma) \models f$, where σ is the trajectory corresponding to s and $a_1, \ldots, a_n$. □

The role of LTL in specifying planning goals has been well studied and examples of that can be found in [Bacchus and Kabanza, 1998; Niyogi and Sarkar, 2000; Baral *et al.*, 2001].

3 N-LTL: A Non-monotonic Extension of LTL

In this section, we extend LTL to capture non-monotonic requirements in specifying a goal. We call the new language N-LTL which stands for *non-monotonic LTL*. We first define the syntax and semantics of the language.

3.1 Syntax

While designing the language we asked ourselves two questions.

- If syntactically the goal is one temporal formula, how can we revise it to have new goals by just adding to the original formula?
- How to refer to one part of a specification in another part of the specification?

For the first we borrow ideas from Reiter's approach to reason about actions [Reiter, 2001] where he compiles his specification to classical logic. While the classical logic part is monotonic, reasoning with respect to the specification language is non-monotonic and the non-monotonicity is achieved through the compilation process. For the second we borrow ideas from logic programming. Similar to a logic program consisting of a set of rules, we consider each N-LTL formula to be a set of rules and we use rule labels such as r in Example 1 to link these rules into one temporal formula.

Definition 5 Let $\{g\}$, R, and P be three disjoint sets of atoms. Let $\langle r \rangle$ be an atom in R, $\langle p \rangle$ be an atom in P, $e \in \{g\} \cup R$. $\langle f \rangle$ is a formula defined below:

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

An N-LTL formula is a set of rules $\langle e : f \rangle$, Where e is the *head*, and f is the *body* of the rule $\langle e : f \rangle$. □

In an N-LTL formula, g is a special symbol that stands for the final goal formula. R is the set of labels to be used to define corresponding strong exceptions and weak exceptions. A formula f defines the conditions of atoms in R or the conditions of g. Intuitively, $[\langle r \rangle](\langle f \rangle)$ means that normally f is true, with the weak exceptions denoted through r. $[[\langle r \rangle]](\langle f \rangle)$ means that normally f is true, with the strong exceptions denoted through r. The weak and strong exception conditions corresponding to r are defined through other rules.

Definition 6 (Atom Dependency) Let $\langle e_1 : f_1 \rangle$ be a rule in an N-LTL formula. If $e_2 \in R$ occurs in the body of $\langle e_1 : f_1 \rangle$, then e_2 *depends* on e_1. The dependency relation is transitive. □

Definition 7 (Loop Free) An N-LTL formula is *loop free* if in the formula, no atom in R depends on itself. □

Example 2 *The following formula*

$$\langle g : [r_1](p) \mathsf{U} q \rangle$$
$$\langle r_1 : [r_2](s) \rangle$$
$$\langle r_2 : [r_1](t) \rangle$$

is not loop free as r_1 depends on r_2 and r_2 depends on r_1.

An *N*-LTL specification is a loop free N-LTL formula.

3.2 Semantics of N-LTL Specifications

As we mentioned earlier, we define the semantics of N-LTL specifications by following the approach taken by Reiter to reason about actions: we compile N-LTL specifications to LTL theories.

Definition 8 Given a loop-free N-LTL formula (i.e., an N-LTL specification) T, we translate it to an LTL formula $Tr(T)$ as follows:

1. Let $\langle e : f_1 \rangle, \langle e : f_2 \rangle, \cdots, \langle e : f_n \rangle$ be all the rules in T with e in the head, $e \in \{g\} \cup R$. We construct a formula $f_1 \vee f_2 \vee \cdots \vee f_n$, and we call it $E(e)$. We do this for any atom e if the set of rules with e in the head is not empty.
2. If atom a_1 depends on atom a_2, and $E(a_1)$ is defined, we replace any occurrence of $[a_1](f)$ in $E(a_2)$ with $f \vee E(a_1)$. The revised formula is still called $E(a_2)$.
3. If atom a_1 depends on atom a_2, and $E(a_1)$ is defined, we replace any occurrence of $[[a_1]](f)$ in $E(a_2)$ with $E(a_1)$. The revised formula is still called $E(a_2)$.
4. We do Step 2 and Step 3 recursively until no atoms e depending on g while $E(e)$ is not empty occurs in $E(g)$.
5. Finally, in $E(g)$, we replace all remaining $[r](f)$ and $[[r]](f)$ with f. The revised goal formula is $Tr(T)$.

□

Example 3 *Let us consider the N-LTL formula T as follows:*

$$\langle g : (\Diamond [r_1](p)) \wedge [r_3](q) \rangle$$
$$\langle r_1 : [[r_2]](v) \rangle$$
$$\langle r_1 : \Box t \rangle$$
$$\langle r_2 : s \rangle$$

According to the definition, we know initially $E(g) = \Diamond[r_1](p) \wedge [r_3](q)$, $E(r_1) = [[r_2]](v) \vee \Box t$, and $E(r_2) = s$. By replacing the formulas according to the dependence relations, we have $E(g) = \Diamond(p \vee [[r_2]](v) \vee \Box t) \wedge [r_3](q) = \Diamond(p \vee s \vee \Box t) \wedge [r_3](q)$. There is no rules with r_3 as the head. Thus $E(g) = \Diamond(p \vee s \vee \Box t) \wedge q$. Further, $Tr(T) = \Diamond(p \vee s \vee \Box t) \wedge q$.

Loop-free N-LTL formulas (also called N-LTL specification) have the following properties.

Proposition 1 *Given an N-LTL specification T, $Tr(T)$ is a well defined LTL formula.*

Given this property, we can define when a plan satisfies an N-LTL goal.

Definition 9 Let T be an N-LTL specification. Given a state s, and a trajectory $\sigma = s_0, s_1, \cdots, s_k, \cdots$, $(s, \sigma) \models T$ in N-LTL if $(s, \sigma) \models Tr(T)$ with respect to LTL. □

Definition 10 (Plans with respect to N-LTL goals) Let T be an N-LTL specification. A sequence of actions $a_1, \cdots, a_n$ is a *plan* from the initial state s for the N-LTL goal T, if σ is the trajectory corresponding to s and $a_1, \cdots, a_n$, and $(s, \sigma) \models T$ in N-LTL. □

N-LTL specifications that do not have any strong exceptions (but may have weak exceptions) always make the goal weaker by accepting more plans when adding new rules.

Proposition 2 *For any N-LTL specification T and $T \cup T'$, if P is a plan with respect to T which does not have any strong exceptions (but may have weak exceptions), then P is a plan with respect to $T \cup T'$.*

Temporal logics such as LTL have a different property when we consider an LTL goal formula to be a set of temporal formulas and add new formulas to an existing theory. If P is a plan with respect to an LTL goal $T \cup T'$ then P is also a plan with respect to T. In LTL, adding more formulas reduces (or at best leaves it unchanged) the set of plans satisfying it while in N-LTL that does not have strong exceptions, adding more rules enlarges the set of plans satisfying it (or at least leaves it unchanged). This is not the case when the N-LTL theory has strong exceptions.

Example 4 *Consider the problem in Example 1. The initial N-LTL formula is the specification T given as:*

$$\langle g : \Box[r_1](p) \wedge \Diamond s \rangle$$

According to the definition, since there are no rules with head r_1, we have $(s_j, \sigma) \models T$ in N-LTL iff $(s_j, \sigma) \models \Box p \wedge \Diamond s$ in LTL.

Suppose we add T', as given below, to it:

$$\langle r_1 : q \rangle$$

According to the definition, we know $E(g) = (\Box[r_1](p)) \wedge \Diamond s$ and $E(r_1) = (q)$. Further revision makes $E(g) = \Box(p \vee q) \wedge \Diamond s$. Thus, $(s_j, \sigma) \models (T \cup T')$ in N-LTL iff $(s_j, \sigma) \models \Box(p \vee q) \wedge \Diamond s$ in LTL.

We can check that any plan satisfies LTL formula $\Box p \wedge \Diamond s$ also satisfies $\Box(p \vee q) \wedge \Diamond s$. T does not have strong exceptions, and Proposition 2 holds here.

Now, let us define what do we mean when we say that N-LTL is non-monotonic. LTL is monotonic since $T \models t$ implies $T \cup T' \models t$ where T and T' are two sets of LTL temporal formulas and t is an LTL temporal formula. We consider the following entailment in N-LTL.

Definition 11 (Entailment) $T \models t$ if $Tr(T) \models t$, where T is an N-LTL specification and t is an LTL formula. □

Proposition 3 *The entailment in Definition 11 is non-monotonic.*

Let us consider one example to illustrate the use of N-LTL.

Example 5 *One professor asks his robot to make a photocopy of one document and fetch a cup of coffee. However, before the robot goes out the office, the professor finds out that the coffee is sold out. No plan can satisfy the goal given to the robot. The professor would now like to weaken the goal. Following are three possibilities:*

1. *He would be happy with a cup of tea instead;*
2. *He just needs the copy of the document and is willing to forget about the coffee;*
3. *The robot may come back to his office with the document copied and go about looking for the coffee later.*

If the professor was using N-LTL and from past experience knows that he may have to revise his goal, especially with respect to coffee, he can express his initial goal as:

$$\langle g : \Diamond(([r]coffee) \wedge copy \wedge \Diamond office) \rangle$$

It is equivalent to satisfying $\Diamond(coffee \wedge copy \wedge \Diamond office)$ in LTL.

Later on, according to the new conditions and new alternatives, the professor may revise his original goal by adding one of the following three rules:

1. *Adding the new rule $\langle r : tea \rangle$ which makes the overall goal equivalent to $\Diamond((coffee \vee tea) \wedge copy \wedge \Diamond office)$ in LTL;*
2. *Let $\top$ denote $true$ and $\bot$ denote $false$. Adding the new rule $\langle r : \top \rangle$ which makes the overall goal equivalent to $\Diamond((coffee \vee \top) \wedge copy \wedge \Diamond office)$ in LTL, which is equivalent to $\Diamond(copy \wedge \Diamond office)$ in LTL;*
3. *Adding the new rule $\langle r : \Diamond(coffee \wedge \Diamond office) \rangle$ which makes the overall goal equivalent to $\Diamond(coffee \wedge copy \wedge \Diamond office) \vee \Diamond(copy \wedge \Diamond office \wedge \Diamond(coffee \wedge \Diamond office))$ in LTL. Now it allows the agent to get the document copied first before waiting for the coffee.*

If we want to get tea instead of coffee, we should know initially that the sub-task of getting coffee can be replaced by a different goal.[3] *Now, the initial formulation is*

$$\langle g : \Diamond(([[r]]coffee) \wedge copy \wedge \Diamond office) \rangle$$

When we add a new goal $\langle r : tea \rangle$. The goal afterwards is equivalent to $\Diamond(tea \wedge copy \wedge \Diamond office)$ in LTL.

Note that "copy" must be satisfied in this domain as the professor does not want to weaken this condition. While the robot is on its way of getting things done, some other exceptions may happen. With N-LTL, we may further refine part of the goal.

Note that if a rule in the formula has a substring $[[r_1]](f)$, then removing rules $\langle r_1 : \bot \rangle$ may affect the semantics of the formula. If there is no substring $[[r_1]](f)$ in any part of the formula, then we can remove rules $\langle r_1 : \bot \rangle$ without affecting the semantics of the formula.

[3] We made this design decision consciously. Thus unlike in logic programming where the same default may have strong exceptions and weak exceptions, here we empower the initial goal specifier to make that decision beforehand.

3.3 Complexity of N-LTL

The complexity of various problems (planning etc.) with respect to N-LTL specifications is at the same level of the polynomial hierarchy as with respect to LTL specifications. This is because, the loop-free criteria and the fact that there are polynomial number of special symbols like $[r]$ and $[[r]]$ ensure that an N-LTL specification can be converted to an LTL specification in polynomial time and the increase in size is also polynomial.

4 Relation to Logic Programs

We now consider the relation between N-LTL goals and logic programs, especially with respect to planning. Use of temporal domain knowledge in planning was first proposed in [Bacchus and Kabanza, 1998]. Planning with temporal logic LTL has also been studied in [Neljanko and Niemelä, 2001; Son *et al.*, 2002]. In this section, we translate a domain description and a goal specified in N-LTL to a logic program. We consider the relations of stable models of the translated programs and the original planning problems. In the following translation, for $i \neq j$, even if s_i and s_j in the trajectory are referring to the same state, we assign them different labels.

Definition 12 Let σ given by $s_0, s_1, \cdots, s_k, \cdots$ be a trajectory, p denote an atom, s_j denote a state, f, f_1, and f_2 denote LTL formulas that also take elements in R as atoms. We now define an encoding of σ, called $encoding(\sigma, T)$, where T is an N-LTL formula. The encoding consists of the following parts:

1. For any i such that $0 \leq i$, there is a rule
$$next(s_i, s_{i+1}) \leftarrow .$$
2. For any $0 \leq i < j$, there is a rule
$$future(s_i, s_j) \leftarrow .$$
3. If p is true in state s, there is a rule
$$holds(s_i, p) \leftarrow .$$
where s_i is the ith state in the trajectory and it corresponds to the state s.
4. The following rules corresponding to $\neg$, $\vee$, and $\wedge$ are in the program
$$\begin{aligned} holds(s, \neg f) &\leftarrow \textit{not } holds(s, f). \\ holds(s, f_1 \vee f_2) &\leftarrow holds(s, f_1). \\ holds(s, f_1 \vee f_2) &\leftarrow holds(s, f_2). \\ holds(s, f_1 \wedge f_2) &\leftarrow holds(s, f_1), holds(s, f_2). \end{aligned}$$
5. The following rules corresponding to temporal operators $\bigcirc$, $\Diamond$, and $\Box$ are in the program:
$$\begin{aligned} holds(s, \bigcirc f) &\leftarrow holds(s_1, f), next(s, s_1). \\ holds(s, \Diamond f) &\leftarrow holds(s_1, f), future(s, s_1). \\ holds(s, \Box f) &\leftarrow \textit{not } holds(s, \Diamond \neg f). \end{aligned}$$
6. The following rules corresponding to the temporal operator U are in the program:
$$\begin{aligned} nfirst(s, s_2, f) &\leftarrow future(s, s_1), \\ &\quad future(s_1, s_2), holds(s_1, f). \\ first(s, s_2, f) &\leftarrow \textit{not } nfirst(s, s_2, f), \\ &\quad future(s, s_2), holds(s_2, f). \\ nholds(s, f_1 \mathsf{U} f_2) &\leftarrow first(s, s_2, f_2), future(s, s_1), \\ &\quad future(s_1, s_2), holds(s_1, \neg f_1). \\ holds(s, f_1 \mathsf{U} f_2) &\leftarrow \textit{not } nholds(s, f_1 \mathsf{U} f_2). \end{aligned}$$

□

Definition 13 We encode an N-LTL formula T into a logic program, denoted by $LP(T)$ as follows:

For each rule $\langle e : f \rangle$ in T, we create a list of the atoms from R (the set of rule labels) that occur in f. If one atom occurs multiple times in f, we include it as many times in the list. We now revise f to a set of new formulas $simplify(f)$:

- For any atom r_k in the list, if it is not in the head of any rule, then we replace all occurrences of $[r_k](f_k)$ or $[[r_k]](f_k)$ with f_k;
- Otherwise, suppose the ith atom in the list is e_j. If it occurs in the formula f as $[[e_j]](f_j)$, then we replace this sub-formula $[[e_j]](f_j)$ in f with e_j.
- Suppose a list of atoms $e_{i1}, e_{i2}, \cdots, e_{il}$ remain in formula f. We calculate a Cartesian product of l sets, where the sth set is $\{e_{is}, f_{is}\}$ and the sth atom e_{is} occurs in the formula f as $[e_{is}](f_{is})$. We replace each occurrence $[e_{is}](f_{is})$ by either e_{is} or f_{is} according to its value in one element of the Cartesian product. Thus we have 2^l different combinations. We consider each combination as one instance of $simplify(f)$, the revised formula of f.

We then add the rule

$$holds(s, e) \leftarrow holds(s, simplify(f)).$$

into $LP(T)$ for each of formula in $simplify(f)$. □

Example 6 *Suppose one rule in an N-LTL goal T is as follows:*

$$\langle r : [r_1](f) \wedge [r_2](h) \wedge [[r_1]](s) \wedge [[r_3]](t) \rangle$$

There are rules in T with the head r_1 and also rules with the head r_2. There are no rules in T with the head r_3. We need to add the following 4 rules into $LP(T)$:

$$\begin{aligned} holds(s, r) &\leftarrow holds(s, f \wedge h \wedge r_1 \wedge t). \\ holds(s, r) &\leftarrow holds(s, f \wedge r_2 \wedge r_1 \wedge t). \\ holds(s, r) &\leftarrow holds(s, r_1 \wedge h \wedge r_1 \wedge t). \\ holds(s, r) &\leftarrow holds(s, r_1 \wedge r_2 \wedge r_1 \wedge t). \end{aligned}$$

Note that in the translated program, we have an atom $holds(s, f \wedge r_2 \wedge r_1 \wedge t)$. r_1 and r_2 are not atoms in the transition system. They are defined by other rules.

Example 7 *Consider the N-LTL goal as follows:*

$$\begin{aligned} \langle g : h \rangle \\ \langle g : \Diamond[r_1](f) \rangle \\ \langle r_1 : h \rangle \\ \langle r_1 : \Box t \rangle \end{aligned}$$

It is known that this formula is equivalent to the LTL formula $h \vee \Diamond(h \vee \Box t \vee f)$. *We have the following rules in* $LP(T)$*:*

$$\begin{aligned} holds(s, g) &\leftarrow holds(s, h). \\ holds(s, g) &\leftarrow holds(s, \Diamond f). \\ holds(s, g) &\leftarrow holds(s, \Diamond r_1). \\ holds(s, r_1) &\leftarrow holds(s, h). \\ holds(s, r_1) &\leftarrow holds(s, \Box t). \end{aligned}$$

Consider a trajectory $\sigma = s_0, s_1, s_1, \cdots, s_1, \cdots$ *where only* t *is true in state* s_1. *No fluent is true in state* s_0. *In the program* $encoding(\sigma, T)$, *we know* $holds(s_2, \Box t)$ *is true in all its stable models. Thus we have* $holds(s_2, r_1)$ *in all stable models of* $encoding(\sigma, T) \cup LP(T)$. *Further, according to Item 5 in Definition 12, since there is a rule* $holds(s_0, \Diamond r_1) \leftarrow holds(s_0, r_1), future(s_0, s_1)$. *we know* $holds(s_1, \Diamond r_1)$. *Thus* $holds(s_0, g)$ *is true in all stable models of the program.*

Proposition 4 *Given a trajectory* $\sigma = s_0, s_1, \cdots, s_n, \cdots$. *Let* T *be an N-LTL specification.* $(s_0, \sigma) \models T$ *in N-LTL iff* $encoding(\sigma, T) \cup LP(T) \models holds(s_0, g)$.

5 Non-monotonic Extension of CTL*, π-CTL*, and P-CTL*

Different from LTL, in CTL* [Emerson and Srinivasan, 1989; Emerson, 1990], π-CTL* [Baral and Zhao, 2004], and P-CTL* [Baral and Zhao, 2006], formulas are categorized as state formulas and path formulas. As a consequence, the corresponding non-monotonic languages should be defined with respect to path formulas and state formulas respectively. We now define the way of extending the definition of non-monotonic temporal logics for CTL*, called N-CTL*. We base the language N-CTL* on CTL*.

We refer readers to [Emerson and Srinivasan, 1989; Emerson, 1990] for the syntax and semantics of CTL*. There are two kinds of formulas in CTL*: state formulas and path formulas. Normally state formulas are properties of states while path formulas are properties of paths. Recall that the symbols A and E are the branching time operators meaning 'for all paths' and 'there exists a path' respectively.

Definition 14 Let $\{g\}$, R_{pf}, R_{sf} and P be four disjoint sets of atoms. Let $\langle r_{pf} \rangle$ be an atom in R_{pf}, $\langle r_{sf} \rangle$ be an atom in R_{sf}, $\langle p \rangle$ be an atom in P, $e \in \{g\} \cup R_{sf}$. Let $\langle sf \rangle$ denote a state formula, and $\langle pf \rangle$ denote a path formula.

$$\begin{aligned} \langle sf \rangle ::= {} & \langle p \rangle \mid \langle sf \rangle \wedge \langle sf \rangle \mid \langle sf \rangle \vee \langle sf \rangle \mid \neg\langle sf \rangle \mid \\ & \mathsf{E}\langle pf \rangle \mid \mathsf{A}\langle pf \rangle \mid [\langle r_{sf} \rangle]\langle sf \rangle \mid [[\langle r_{sf} \rangle]]\langle sf \rangle \end{aligned}$$

$$\begin{aligned} \langle pf \rangle ::= {} & \langle sf \rangle \mid \langle pf \rangle \vee \langle pf \rangle \mid \neg\langle pf \rangle \mid \langle pf \rangle \wedge \langle pf \rangle \mid \bigcirc \langle pf \rangle \mid \\ & \langle pf \rangle \ \mathsf{U} \ \langle pf \rangle \mid \Diamond\langle pf \rangle \mid \Box\langle pf \rangle \mid [\langle r_{pf} \rangle]\langle pf \rangle \mid \\ & [[\langle r_{pf} \rangle]]\langle pf \rangle \end{aligned}$$

An N-CTL* formula is a set of rules $\langle e : sf \rangle$, or $\langle r_{pf} : pf \rangle$. Each of e or r_{pf} is the *head* of the rule, and each of sf or pf is the *body* of the rule $\langle e : sf \rangle$ or $\langle r_{pf} : pf \rangle$. □

The semantics of N-CTL* is defined in a way similar to the definition of the semantics of N-LTL. We now illustrate the usefulness of N-CTL* through two examples.

Example 8 *The initial goal is to require that* p *be true until* q *is reached. However, it is realized that in some domain, this goal is too strong as no plan can always have* p *until reaching* q. *If in some states in the main path, all possible trajectories will have* p *true in the future, then we may consider it as a strong exception and may allow not to have* p. *The initial goal can be represented as* $\langle g : [r](p)\mathsf{U}q \rangle$ *in N-CTL*. *This goal is equivalent to* $p\mathsf{U}q$ *in CTL*. *It can be further refined by adding one rule about the weak exception* r *as* $\langle r : \mathsf{A}\Diamond p \rangle$. *The revised goal is equivalent to* $(p \vee \mathsf{A}\Diamond p)\mathsf{U}q$, *which is equivalent to* $(\mathsf{A}\Diamond p)\mathsf{U}q$.

Example 9 *Initially, the goal is to make sure that in most trajectories starting from the initial state,* $\Diamond p$ *is true. However, later on, we may require that once* $\Diamond q$ *is satisfied by some trajectories, those trajectories are considered as exceptional ones and we do not require them to satisfy* $\Diamond p$. *The initial goal is represented as:* $\langle g : \mathsf{A}[r](\Diamond p) \rangle$. *Later, the goal can be weakened by adding the weak exception rule* $\langle r : (\Diamond q) \rangle$.

The initial goal is equivalent to the CTL formula* $\mathsf{A}\Diamond p$ *and the revised goal is equivalent to* $\mathsf{A}(\Diamond p \vee \Diamond q)$.

6 Conclusion and Future Work

In this paper, we proposed a non-monotonic temporal logic based on LTL and hinted at a branching time non-monotonic temporal logic based on CTL*. We motivated the need for such non-monotonic temporal logics from the point of view of needing ways to express goals that can be changed in an elaboration tolerant manner. We presented several properties of our non-monotonic goal language N-LTL and briefly discussed its encoding using logic programming.

One possible controversial design decision that we made is to empower the specifier to decide whether weak exceptions or strong exceptions are allowed. This is in contrast to use of defaults in standard non-monotonic logics where the normative statement itself does not refer to strong exception or weak-exception, but rather they are added separately. Further study to compare the two approaches is an immediate future work.

7 Acknowledgement

This work was supported by a contract from ARDA and NSF grant 0412000. We thank the anonymous reviewers for their comments, especially for pointing us to the earlier work [Fujiwara and Honiden, 1991].

References

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

[Baral and Zhao, 2004] Chitta Baral and Jicheng Zhao. Goal specification in presence of non-deterministic actions. In *Proceedings of ECAI'04*, pages 273–277, 2004.

[Baral and Zhao, 2006] Chitta Baral and Jicheng Zhao. Goal specification, non-deterministic, and quantifying over policies. In *AAAI-06*, 2006.

[Baral *et al.*, 2001] Chitta Baral, Vladik Kreinovich, and Raul Trejo. Computational complexity of planning with temporal goals. In *IJCAI-01*, pages 509–514, 2001.

[Emerson and Srinivasan, 1989] E. Allen Emerson and Jai Srinivasan. Branching time temporal logic. In J.W. de Bakker, W. P. de Roever, and G. Roezenberg, editors, *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, pages 123–172. Springer-Verlag, Berlin, 1989.

[Emerson, 1990] E. Allen Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 997–1072, 1990.

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

[Giannotti *et al.*, 1999] Fosca Giannotti, Giuseppe Manco, Mirco Nanni, and Dino Pedreschi. On the effective semantics of nondeterministic, nonmonotonic, temporal logic databases. *Lecture Notes in Computer Science*, 1584:58–72, 1999.

[MacNish and Fallside, 1990] Craig MacNish and Frank Fallside. Temporal reasoning: a solution for multiple agent collision avoidance. In *IEEE International Conference on Robotics and Automation*, volume 1, pages 494–499, 1990.

[McCarthy, J., 1988] McCarthy, J. Mathematical logic in artificial intelligence. *Daedalus*, 117(1):297–311, 1988.

[McDermott, 1982] Drew McDermott. Non-monotonic logic II: Nonmonotonic modal theories. *Journal of the ACM*, 29:33–57, 1982.

[Neljanko and Niemelä, 2001] Keijo Neljanko and Ilkka Niemelä. Bounded LTL model checking with stable models. In *LPNMR*, pages 200–212, 2001.

[Niyogi and Sarkar, 2000] Rajdeep Niyogi and Sudeshna Sarkar. Logical specification of goals. In *Proc. of 3rd international conference on Information Technology*, pages 77–82, 2000.

[Reiter, 2001] Ray Reiter. *Knowledge in action : logical foundations for specifying and implementing dynamical systems*. MIT Press, 2001.

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

[Son *et al.*, 2002] Tran Cao Son, Chitta Baral, Nam Tran, and Sheila Meilraith. Domain-dependent knowledge in answer set planning. CS 007, New Mexico State University, 2002.

[Stein and Morgenstern, 1994] Lynn Andrea Stein and Leora Morgenstern. Motivated action theory: A formal theory of causal reasoning. *Artificial Intelligence*, 71:1–42, 1994.