Planning under Uncertainty in Linear Time Logic*

Marta Cialdea Mayer, Carla Limongelli, Andrea Orlandini, Valentina Poggioni

Università di Roma Tre – Dipartimento di Informatica e Automazione
Via della Vasca Navale, 79 - 00146 Rome (Italy)
{cialdea,limongel,orlandin,poggioni}@dia.uniroma3.it

Abstract. The "planning as satisfiability" approach for classical planning establishes a correspondence between planning problems and logical theories, and, consequently, between plans and models. This work proposes a similar framework for contingency planning: considering contingent planning problems where the sources of indeterminism are incomplete knowledge about the initial state, non-inertial fluents and non-deterministic actions, it shows how to encode such problems into Linear Time Logic. Exploiting the semantics of the logic, and the notion of *conditioned model* introduced in this work, a formal characterization is given of the notion of contingent plan (a plan together with the set of conditions that ensure its executability).

1 Introduction

Planning under uncertainty and incomplete information is felt as an important issue in the planning community. In fact, classical planning, where the whole plan is synthesized off-line, is based on a number of simplifying assumptions: the world is completely known and predictable and no sensor information can be gathered at execution time. Such hypotheses however are too restrictive to model a realistic environment, where the description of the initial state may be incomplete, actions may have non-deterministic effects, and the agent interacts with the environment and gathers information from sensors.

In order to handle such sources of uncertainty, partial-order planning techniques have been extended to planning with incomplete information [15, 18]. Also logic-based approaches to planning have been investigated: for instance, [13] proposes a model based on deduction; [20] reduces planning to truth evaluation of quantified boolean formula; [14] extends the "planning as satisfiability" approach in propositional logic, originally introduced in [12]; [16] uses a logic of knowledge; [3] presents an approach based on model checking.

When aiming at solving problems defined in more complex and realistic domains, the need for more expressive planning languages is even stronger than in classical planning. Such a problem has been addressed, for instance, in [10, 21, 7]. Considering that, in this respect, logical approaches to planning are more

* This work has been partially supported by ASI - Agenzia Spaziale Italiana.

flexible and expressive than others, this work proposes a logical framework for planning with uncertainty in Linear Time Logic (LTL), following the "planning as satisfiability" approach. This work extends [9], where classical planning, modeled in LTL, is reduced to model search. The model of time underlying LTL is a countably infinite sequence of states (or *time points*), that can be identified with $\mathbb{N}$. A temporal interpretation describes how the world evolves in time, hence a set of formulae constrains the behaviour of the world and can be taken as the specification of a planning domain. The advantages of using LTL instead of classical propositional logic, like in [12, 14], come from its greater expressive power. In fact, there is no need to represent time points explicitly, neither to fix a bound to the plan length in advance, and iteratively deepen such a bound in order to obtain a complete search procedure. The expressive power of LTL allows one also to represent domain restrictions, intermediate tasks, domain and control knowledge (useful to guide the search), as well as temporally extended goals [1, 2, 17].

This paper defines how to encode a planning problem, represented in a general planning language allowing for different forms of non-determinism (incomplete knowledge about the initial state, non-deterministic operators and non-determinism in the environment) and extended goals, into a set of LTL formulae (Section 2), thus extending the analogous translation of classical planning problems defined in [6] and used by the system presented in [9].

In order to deal with uncertain planning domains, the "planning as satisfiability" paradigm has to be adjusted, since a model always conveys complete information on what is true and what is false. This issue is addressed in the present work considering *conditioned models*, i.e. suitably structured partial descriptions of temporal interpretations, from which *contingent plans* can be extracted. A contingent plan is a a pair $\langle$*plan, conditions*$\rangle$, i.e. a plan together with the set of conditions that ensure its feasibility. Such a correspondence provides a formal characterization of the notion of contingent plan, through the semantics of LTL. In Section 3, the notion of conditioned model – already introduced in [8] – is revisited and extended in such a way that any contingent plan extracted from a conditioned model of a set S, representing a planning problem P, is a solution for P (the corresponding version in [8] was in fact not adequate to the aims of planning, since it misses such a property). Finally, the notion of contingent plan is extended so that observations gathered during execution are taken into account. Such a characterization can be used when re-planning is needed.

2 A Planning Language for Non-Deterministic Domains and its Encoding in LTL

In this section, we define the kind of planning problems we are dealing with and their encoding in LTL. The problems are presented by means of a PDDL-like syntax. The language we use can be seen as an extension of the planning language used by the system presented in [9], that allows also for the specification of different forms of domain-specific knowledge, such as heuristic knowledge and information on domain invariants. Such features are not considered in this work, since we focus here on the encoding of general planning problems. The language allows also for universal and existential quantification (over finite and fixed domains) – that are however treated as conjunctions or disjunction of propositional formulae –, but for simplicity we restrict here the language to be propositional.

The language we consider is built over an alphabet $\mathbf{P} = \mathcal{A} \cup \mathcal{F}$ of propositional letters, partitioned into the two sets $\mathcal{A}$ of action names and $\mathcal{F}$ of fluents. In turn, $\mathcal{F}$ is partitioned into the two sets $\mathcal{I}$ of inertial fluents and $\mathcal{F} - \mathcal{I}$ of non-inertial ones. Inertial fluents [11] are subject to the "common sense law of inertia": if no action affects them, their value stays unchanged. The value of a non-inertial fluent, on the contrary, cannot be always predicted from its previous value and the agent's actions, but it may be subject to exogenous events. It can also be affected by the agent's actions, but the effect of actions on non-inertial fluents is guaranteed for the next state only.

We consider the language of LTL over $\mathbf{P}$ built by means of the connectives $\neg$, $\wedge$, $\vee$ and the future time operators $\Box$ (always), $\Diamond$ (eventually), $\bigcirc$ (next). Implication, double implication and exclusive or ($\oplus$) are defined symbols.

To our aims, a non standard way of describing the semantics of LTL is more convenient [8]. It makes use of *labelled formulae*, i.e. expressions of the form $n : A$ where $n \in \mathbb{N}$ and A is a formula. Intuitively, the label identifies the state where the formula is true. If ℓ is a literal, then $n : \ell$ is a labelled literal. A set S of labelled literals is complete if for all $n \in \mathbb{N}$ and all $p \in \mathbf{P}$, either $n : p \in S$ or $n : \neg p \in S$. It is consistent if it does not contain both $n : p$ and $n : \neg p$ for any $n \in \mathbb{N}$ and $p \in \mathbf{P}$.

Definition 1. *A temporal interpretation $\mathcal{M}$ is a consistent and complete set of labelled literals.*

Intuitively, if $n : p \in \mathcal{M}$ then p is true at state n, and if $n : \neg p \in \mathcal{M}$ then p is false at state n.

If $\mathcal{M}$ is an interpretation of $\mathbf{P}$, $n \in \mathbb{N}$ and A is a formula, the relation $\mathcal{M} \models n : A$ corresponds to the usual satisfiability relation $\mathcal{M}_n \models A$ (the n-th state of $\mathcal{M}$ satisfies A) and is recursively defined as follows:

$\mathcal{M} \models n : p$ iff $n : p \in \mathcal{M}$, when $p \in \mathbf{P}$;
$\mathcal{M} \models n : \neg A$ iff $\mathcal{M} \not\models n : A$;
$\mathcal{M} \models n : A \wedge B$ iff $\mathcal{M} \models n : A$ and $\mathcal{M} \models n : B$;
$\mathcal{M} \models n : A \vee B$ iff either $\mathcal{M} \models n : A$ or $\mathcal{M} \models n : B$;
$\mathcal{M} \models n : \bigcirc A$ iff $\mathcal{M} \models n + 1 : A$;
$\mathcal{M} \models n : \Box A$ iff for all $k \geq n$, $\mathcal{M} \models k : A$;
$\mathcal{M} \models n : \Diamond A$ iff for some $k \geq n$, $\mathcal{M} \models k : A$.

A formula A is true in $\mathcal{M}$ (and $\mathcal{M}$ is a model of A) iff $\mathcal{M} \models 0 : A$. If S is a set of formulae, $\mathcal{M} \models S$ iff $\mathcal{M} \models A$ for all $A \in S$. From now on, a set S of LTL-formulae is considered the same as $\{0 : A \mid A \in S\}$.

The specification of a planning problem is divided into different sections. The declaration of fluents, partitioned into inertial and non-inertial, is contained in a specific section `fluents`.

The `init` section contains a set of classical formulae describing the initial state. Such a section may contain disjunctive information and is not required to be complete: all the unspecified information are supposed to be unknown, since the agent's knowledge of the initial state may be incomplete.

The goal of the problem is described in a corresponding section `goal`. The formula in this section may be a generic temporal formula, similarly to [17] (based on CTL), so that reachability goals as well as temporally extended goals can be encoded. For example, if in is a fluent, then $\Diamond in$ is a reachability goal. The example 1 below provides an instance of a temporally extended goal: $\Diamond(in \wedge \Diamond\neg in)$, i.e. the agent has to enter the room and exit again. Goals can also have the form $\bigcirc\bigcirc\bigcirc in$ or $\Box\neg in$.

Each action is described in an `action` section, containing the operator's name, its preconditions and effects. Preconditions are classical formulae of any form. Conditional effects (`(when <condition> <effect>)`) are allowed, as well as non-deterministic ones, expressed as disjunctions or exclusive disjunctions of literals (atoms or negated atoms).

The following example shows how to specify non-inertial fluents and incomplete knowledge in the initial state:

Example 1. The agent has to enter a room, passing through a door, and exit again, without knowing whether the door is initially open or not. He can push the door open (from both sides), but the status of the door is not under the agent's complete control, for instance because other agents may pass by and open or close the door.

```
(:fluents (:non-inertial open)
          (:inertial in))
(:action pass
     :preconditions open
     :effects (and (when ¬in in)
                   (when in ¬in)))
(:action push
     :preconditions ¬open
     :effects open)
(:init ¬in)
(:goal ◇(in ∧ ◇¬in))
```

Here the execution of *pass* causes one to be inside the room if coming from outside, and to be out otherwise.

An example of non-deterministic action is the following:

Example 2. The agent has to get a file and print it. He does not know whether a postscript or pdf version of the document is available. Depending on this, he will use either Acrobat Reader or GhostView to print it.

```
(:fluents (:inertial (printed,have_ps,have_pdf)))
(:action gv
     :preconditions have_ps
     :effects printed)
(:action acroread
     :preconditions have_pdf
     :effects printed)
(:action take_file
     :preconditions ¬have_ps ∧ ¬have_pdf
     :effects have_ps ⊕ have_pdf)
(:init ¬have_ps, ¬have_pdf, ¬printed)
(:goal ◇printed)
```

Here *take_file* non-deterministically causes one to obtain a pdf or postscript file (and not both). The symbol $\oplus$ is used for exclusive OR.

In the rest of this section we show how to encode a planning problem into a set of LTL formulae. The encoding is an extension of the encoding of classical planning problems proposed in [6] and used in [9]. The specification of a planning problem in the language described above is encoded by the set S_0 of LTL formulae obtained as follows.

Initial State Axioms: All the formulae describing the initial state (in the `init` section) are included in S_0. This results in a (possibly incomplete) description of the initial state.

Goal Axioms: The encoding of the goal G (stated in the `goal` section) is G itself.

Action Precondition Axioms: For each operator a having preconditions $A_1, ..., A_n$, S_0 contains the formula: $\Box(a \rightarrow A_1 \wedge ... \wedge A_n)$.

Non-deterministic Effect Axioms: For each action a, having disjunctive effects $\ell_1 \star ... \star \ell_k$, where $\star$ is either $\vee$ or $\oplus$, under conditions C, (`when` C $(\ell_1 \star ... \star \ell_k)$) S_0 contains: $\Box(a \wedge C \rightarrow \bigcirc\ell_1 \star ... \star \bigcirc\ell_k)$ (if the effect is unconditioned, C is omitted).

Effect Axioms: If f is a non-inertial fluent or a fluent that can be *non deterministically* affected by some action, let G_f^+ be the disjunction of all the $a_i \wedge C_i$ that have f as a deterministic effect (G_f^+ specifies all the conditions that *deterministically* lead to change the truth value of f from false to true), and G_f^- the disjunction of all the $a_i \wedge C_i$ that have $\neg f$ as a deterministic effect (G_f^- specifies all the conditions that *deterministically* lead to change the truth value of f from true to false). Then S_0 contains the following effect axioms for each of such fluents:

$$\Box(G_f^+ \rightarrow \bigcirc f), \Box(G_f^- \rightarrow \bigcirc\neg f) \tag{1}$$

Inertia Axioms: For each inertial fluent f, such that:

(i) $\neg f$ is a non-deterministic effect of some actions $a_1, .., a_k$, under conditions, respectively, $C_1, ..., C_k$;

(ii) f is a non-deterministic effect of some actions $b_1, .., b_n$, under conditions, respectively, $D_1, ..., D_n$,

S_0 contains the following axioms:

$$\begin{gathered}\Box((f \wedge \neg G_f^- \wedge \neg(a_1 \wedge C_1) \wedge ... \neg(a_k \wedge C_k)) \rightarrow \bigcirc f) \\ \Box(\bigcirc f \rightarrow (f \vee G_f^+ \vee (b_1 \wedge D_1) \vee ... \vee (b_n \wedge D_n)))\end{gathered} \tag{2}$$

where G_f^+ and G_f^- are as specified in the previous item and they refer to the action that have f or $\neg f$ as a deterministic effect.

Successor State Axioms: For each inertial fluent f, that is never a non-deterministic effect of any action, S_0 contains the following paraphrases of Reiter's Successor State Axiom [19]:

$$\Box(\bigcirc f \equiv G_f^+ \vee (f \wedge \neg G_f^-))$$

Incompatibility Axioms: S_0 contains also a set of axioms describing incompatibility relations between actions, of the form $\Box(\neg a \vee \neg b)$. What has to be explicitly encoded, however, are the incompatibilities of actions a and b such that a deletes a precondition of b, or vice-versa. In fact, if two actions a and b have conflicting preconditions or effects, such an incompatibility is taken into account by the logic itself, since a and b cannot be true in the same state. For convenience, we also declare two actions a and b to be incompatible whenever a has a non-deterministic effect f and b a (deterministic or non-deterministic) effect $\neg f$ (or vice-versa).

The formulae (1) e (2) correspond to the Successor State Axioms for non-inertial fluents or fluents non-deterministically affected by some action.

The adequacy of the analogous encoding for classical planning problems has been shown in [6]. In the new setting, the encoding can be recognized correct when we consider that a successor state axiom is equivalent to the conjunction of the following formulae:

$$\begin{array}{ll} (i)\ \ \Box(G_f^+ \rightarrow \bigcirc f) & (iii)\ \Box(f \wedge \neg G_f^- \rightarrow \bigcirc f) \\ (ii)\ \Box(G_f^- \rightarrow \bigcirc \neg f) & (iv)\ \ \Box(\neg f \wedge \neg G_f^+ \rightarrow \bigcirc \neg f) \end{array}$$

Non-inertial fluents clearly do not satisfy (iii) and (iv), while (i) and (ii) are present in S_0 as effect axioms. If f is a non-deterministic effect of some action, (iii) and (iv) are in S_0 as inertia axioms, (i) and (ii) are replaced by the effect and non-deterministic effect axioms.

Note that, differently from [5], an action name is true in the state where the action is performed, and not in the resulting state. In this way, there is no need to characterize at the meta-level the notion of "legal plan".

Figure 1 shows the complete encoding of the two examples presented before, filtered by the application of some simple logical simplifications.

LTL encoding of Example 1	***LTL encoding of Example 2***
Initial state axioms: $\neg in$ **Goal axioms:** $\Diamond(in \wedge \Diamond\neg in)$ **Action preconditions axioms:** $\Box(pass \rightarrow open)$, $\Box(push \rightarrow \neg open)$ **Fluents behaviour:** $\Box(push \rightarrow \bigcirc open)$, $\Box(\bigcirc in \equiv$ $(in \wedge \neg pass) \vee (\neg in \wedge pass))$	**Initial state axioms:** $\neg have\_ps,\ \neg have\_pdf,\ \neg printed$ **Goal axioms:** $\Diamond printed$ **Action preconditions axioms:** $\Box(take\_file \rightarrow \neg have\_ps \wedge \neg have\_pdf)$ $\Box(acroread \rightarrow have\_pdf)$ $\Box(gv \rightarrow have\_ps)$ **Fluents behaviour:** $\Box(take\_file \rightarrow (\bigcirc have\_ps \wedge \bigcirc\neg have\_pdf)$ $\vee(\bigcirc have\_pdf \wedge \bigcirc\neg have\_ps)))$ $\Box(have\_ps \rightarrow \bigcirc have\_ps)$ $\Box(have\_pdf \rightarrow \bigcirc have\_pdf)$ $\Box(\bigcirc have\_ps \rightarrow have\_ps \vee take\_file)$ $\Box(\bigcirc have\_pdf \rightarrow have\_pdf \vee take\_file)$ $\Box(\bigcirc printed \equiv (gv \vee acroread \vee printed))$

Fig. 1. The complete encodings of examples 1 and 2

3 Contingent Plans

In this section, we identify a planning problem with its temporal logic encoding and formalize the notion of contingent plan in terms of temporal interpretations.

Instead of producing the whole conditional plan off-line, and then executing it, we can consider the possibility that planning and execution (with sensing actions) are interleaved. In that case, a single contingent plan can be generated at first. During its execution, its conditions are checked by means of sensing actions. When a condition turns out to be false, a different contingent plan is generated and considered. Obviously, such a procedure is complete only if all actions are reversible. Such a restricion is discussed in Section 4.

Let S_0 be a set of formulae encoding a planning problem, as illustrated before. From a model $\mathcal{M}$ of S_0 we want to extract a contingent plan solving the problem represented by S_0, i.e. a significant subset of $\mathcal{M}$, describing a plan, together with a set of conditions ensuring the feasibility of the plan. In general, in fact, it is not the whole model $\mathcal{M}$ that is of interest and has to be explained, but only a finite fragment of the model, up to the achievement of the goals. Even the description of $\mathcal{M}$ up to such a "final" state need not be complete. However, it must be rich enough: it must contain enough information on how to reach the goal and which conditions ensure its reachability. The following definition formalizes this concept. For simplicity we assume here that S_0 contains formulae in negation normal form. If S is a set of labelled formulae, then $lits(S)$ is the set of labelled literals in S.

Definition 2. *Let S_0 be a set of labelled formulae, $S \supseteq S_0$ and $m \in \mathbb{N}$. Then S is a* saturated extension *of S_0 up to m if for every $n : A \in S$ such that $n \leq m$:*

1. *if $A = A_0 \wedge A_1$, then both $n : A_0 \in S$ and $n : A_1 \in S$;*
2. *if $A = A_0 \vee A_1$, then either $n : A_0 \in S$ or $n : A_1 \in S$;*
3. *if $A = \Box A_0$ then for all k such that $n \leq k \leq m$, $k : A_0 \in S$;*
4. *if $A = \Diamond A_0$ then for some k such that $n \leq k \leq m$, $k : A_0 \in S$;*
5. *if $A = \bigcirc A_0$ and $n < m$ then $n + 1 : A_0 \in S$.*

If $\mathcal{M}$ is a model of S_0 and S a saturated extension of S_0 up to m, such that $lits(S) \subset \mathcal{M}$, then $lits(S)$ is called a significant description of $\mathcal{M}$ as a model of *S_0.*

Note that, mainly because of clause 4 above, any significant description of a model of the encoding of a planning problem implies that the goals are achieved.

It can easily be proved, by use of the same techniques used to prove soundness of tableau methods, that if $\mathcal{M}$ is a model of S_0 then there exists a significant description of $\mathcal{M}$ as a model of S_0.

The following definitions are introduced to the aim of characterizing an *adequate* set of conditions ensuring the feasibility of a plan.

Definition 3. *If S is a set of labelled literals the* history *of S up to n, $h(S, n)$, is*

$$\{k : \ell \mid k : \ell \in S \text{ and } k \leq n\}$$

Let S_0 be a set of labelled formulae, and K and U sets of labelled literals. Then U is a set of conditions *explaining K in the context of S_0 (briefly, U explains K in S_0) iff U is a minimal set (w.r.t. set inclusion) such that:*

for all $n:\ell \in K$: $S_0, h(U, n-1) \models n:\ell$.

The condition above amounts to saying that, for all $n:\ell \in K$ and for every model $\mathcal{M}'$ of S_0, if $\mathcal{M}'$ is also a model of the subset $h(U, n-1)$ of the history of U up to $n-1$, then $\mathcal{M}' \models n:\ell$, too. In informal terms, $h(U, n-1)$ "explains", in the context of S_0, all that happens, according to K, up to the n-th state. Moreover, no subset of U has such a property. If K implies, in the context of S_0, that the goals are achieved within the n-th step, then $h(U, n-1)$ actually guarantees the achievement of the goals.

Definition 4. *Let S_0 be a set of labelled formulae representing a planning problem. A* conditioned model *of S_0 is a pair $\langle K, U\rangle$, where $K \cup U$ is a significant description of some model $\mathcal{M}$ of S_0 and U explains K in S_0.*

If $\langle K, U\rangle$ is a conditioned model of S_0 then the contingent plan corresponding to $\langle K, U\rangle$ solving the problem represented by S_0 is the pair $\langle \mathcal{P}, \mathcal{C}\rangle$, where:

$$\begin{aligned}
\mathcal{P} &= \{n:a \mid n:a \in U \text{ and } a \in \mathcal{A}\}\\
\mathcal{C} &= \{n:p \mid n:p \in U \text{ and } p \in \mathcal{F}\} \cup \{n:\neg p \mid n:\neg p \in U \text{ and } p \in \mathcal{F}\}
\end{aligned}$$

In simpler words, a contingent plan solving the problem encoded by S_0 can be extracted from the set U of conditions of a conditioned model of S_0, by dropping negative action literals, i.e. labelled literals of the form $n:\neg a$ for $a \in \mathcal{A}$, and splitting the rest of U into the set of action names and the fluents (contingencies guaranteeing the feasibility of the plan).

For instance, let S_1 be the encoding of the problem presented in Example 1, and S the following subset of one of its models:

$$\begin{aligned}
S = \{\ & 0:open,\ 0:pass,\ 0:\neg push,\ 0:\neg in,\\
& 1:\neg open,\ 1:\neg pass,\ 1:push,\ 1:in,\\
& 2:open,\ 2:pass,\ 2:\neg push,\ 2:in,\\
& 3:open,\ 3:\neg pass,\ 3:\neg push,\ 3:\neg in\}
\end{aligned}$$

S is a significant description of the model up to the third state; in particular it implies that the goals are achieved at the third state. Let us partition S into:

$$\begin{aligned}
U = \{\ & 0:open,\ 0:pass,\ 0:\neg push, 1:\neg open,\ 1:\neg pass,\ 1:push,\\
& 2:pass,\ 2:\neg push,\ 3:open,\ 3:\neg pass,\ 3:\neg push\}\\
K = \{\ & 0:\neg in,\ 1:in,\ 2:open,\ 2:in,\ 3:\neg in\}
\end{aligned}$$

Then U explains K in S_1. In fact:

$$\begin{aligned}
& h(U,-1) = \emptyset\\
& h(U,0) = \{0:open,\ 0:pass,\ 0:\neg push\}\\
& h(U,1) = h(U,0) \cup \{1:\neg open,\ 1:\neg pass,\ 1:push\}\\
& h(U,2) = h(U,1) \cup \{2:pass,\ 2:\neg push\}
\end{aligned}$$

and

$$\begin{array}{ll} S_1 \models 0 : \neg in & S_1,\ h(U,1)\ \models 2 : in \\ S_1,\ h(U,0)\ \models 1 : in & S_1,\ h(U,2)\ \models 3 : \neg in \\ S_1,\ h(U,1)\ \models 2 : open & \end{array}$$

Omitting the negative "action" literals, U can be read as saying:

IF $0 : open$, $0 : pass$, $1 : \neg open$, $1 : push$, and $2 : pass$,
THEN the goal is achieved.

The contingent plan that is extracted from U is then the pair:

$$\langle\{0 : pass,\ 1 : push,\ 2 : pass\},\{0 : open,\ 1 : \neg open\}\rangle$$

Note that, according to Definition 4, the success of the plan could also be conditioned by contingencies regarding the same state where the goal is achieved. For instance, let us consider the following simplification of the problem presented as Example 2:

```
(:fluents                                   (:action take_file
   (:inertial have_ps,have_pdf))               :preconditions
(:init ¬have_ps, ¬have_pdf)                           ¬have_ps ∧ ¬have_pdf
(:goal ◇have_ps)                               :effects have_ps ⊕ have_pdf)
```

A conditioned model of its encoding is:

$$\begin{aligned} K &= \{0 : \neg have\_ps,\ 0 : \neg have\_pdf\} \\ U &= \{0 : take\_file,\ 1 : \neg have\_pdf,\ 1 : have\_ps\} \end{aligned}$$

The goal *have_ps* is achieved at state 1, but it is conditioned by contingencies related to state 1 itself. In this case, although for $n = 1$, $S_0, U \models n : have\_ps$, the achievement of the goal is not guaranteed only by $S_0 \cup h(U, n-1)$, but the whole set U has to be taken into account.

Note that both $1 : have\_ps$ and $1 : \neg have\_pdf$ occur as conditions in U, although they are inter-derivable in the context of S_0; in fact, none of the two literals is derivable from S_0 and $h(U,0)$, since, at that step, the effect of $0 : take\_file$ cannot be predicted. If this situation is to be excluded, then we must also require for a pair $\langle K, U\rangle$ to be a conditioned model of the encoding S_0 of a problem, that $S_0, K \models 0 : G$, where G is the representation of the goal.

Let $\langle K, U\rangle$ be a conditioned model of the encoding S_0 of a planning problem and let us assume that, during the execution of the contingent plan extracted from $\langle K, U\rangle$, the set $\mathcal{O}$ of observations has been gathered. Let us assume, moreover, that $n : \ell$ is the result of the last sensing action and the only observation that is inconsistent with U. Then we want to synthesize a plan that is "complementary" to the previous one with respect to $n : \ell$. The new plan must take $\mathcal{O}$ into account, i.e. the observations, and whatever is derivable from them and S_0, should not appear as conditions in the new plan but they will be considered as known literals in that step.

Definition 5. *Let S_0 be the encoding of a planning problem, and $\mathcal{O}$ a set of observations, i.e. a set of labelled literals that is consistent with S_0. A conditioned model of S_0 in the presence of $\mathcal{O}$ is a conditioned model of $S_0 \cup \mathcal{O}$, i.e. a pair $\langle K, U\rangle$ such that $K \cup U$ is a significant description of $S_0 \cup \mathcal{O}$ and for all $n : \ell \in K$: $S_0, \mathcal{O}, h(U, n-1) \models n : \ell$.*

Let $\langle K, U\rangle$ be a conditioned model of S_0, $\mathcal{O}$ a set of observations, and n the maximum label of the literals in $\mathcal{O}$. Let us assume, moreover, that $n : \ell$ is the only labelled literal that is inconsistent with U. Then a pair $\langle K', U'\rangle$ is complementary to $\langle K, U\rangle$ in the presence of $\mathcal{O}$ if it is a conditioned model of S_0 in the presence of $\mathcal{O}$ and $K \cup U$ and $K' \cup U'$ describe the same models up to step $n-1$, i.e. for all $k < n$: $\{k : \ell \mid k : \ell \in K \cup U\} = \{k : \ell \mid k : \ell \in K' \cup U'\}$.

For instance, let S_2 be the encoding of the problem presented as Example 2, where the *take_file* action has a non-deterministic effect, and let $\langle K, U\rangle$ be the conditioned model of S_2 where:

$$\begin{aligned}
K = \{ & \ 0 : \neg have_ps,\ 0 : \neg have_pdf,\ 0 : \neg printed,\ 0 : \neg gv,\ 0 : \neg acroread, \\
& \ 1 : \neg printed,\ 1 : \neg take_file, \\
& \ 2 : printed,\ 2 : have_ps,\ 2 : \neg have_pdf\ ,\ 2 : \neg acroread,\ 2 : \neg take_file\} \\
U = \{ & \ 0 : take_file,\ 1 : have_ps,\ 1 : \neg have_pdf,\ 1 : gv,\ 1 : \neg acroread,\ 2 : \neg gv\}
\end{aligned}$$

Let us assume that, during the execution of this plan, after performing the *take_file* action at step 0, *have_ps* turns out to be false at step 1, so the observation $\mathcal{O} = \{1 : \neg have\_ps\}$ must be taken into account when synthesizing the new plan. A suitable new plan is unconditioned:

$$\langle \{0 : take\_file,\ 1 : acroread\}, \emptyset \rangle.$$

In fact, this is the plan corresponding to the conditioned model $\langle K', U'\rangle$ of $S_2 \cup \mathcal{O}$, where:

$$\begin{aligned}
K' = \{ & \ 0 : \neg have_ps,\ 0 : \neg have_pdf,\ 0 : \neg printed, 0 : \neg gv,\ 0 : \neg acroread, \\
& \ 1 : \neg printed,\ 1 : \neg have_ps,\ 1 : have_pdf, 1 : \neg gv,\ 1 : \neg take_file, \\
& \ 2 : printed,\ 2 : \neg have_ps,\ 2 : have_pdf, 2 : \neg gv,\ 2 : \neg take_file\ \} \\
U' = \{ & \ 0 : take_file,\ 1 : acroread,\ 2 : \neg acroread\ \}
\end{aligned}$$

In fact, $S_2, \mathcal{O}, 0 : take\_file \models 1 : have\_pdf$.

4 Concluding remarks

The "planning as satisfiability" approach for classical planning establishes a correspondence between planning problems and logical theories, and, consequently, between plans and models. This work proposes a similar framework for contingency planning: considering contingent planning problems where the sources of indeterminism are incomplete knowledge about the initial state, non-inertial fluents and non-deterministic actions, it shows how to encode such problems into LTL. Exploiting the semantics of the logic, and the notion of conditioned model, we give a formal characterization of the notion of contingent plan.

A prototype system is under development, that simulates the interleaving of planning, execution and monitoring and it is based on the tableau calculus presented in [8]. We are presently testing the system and gathering experimental data.

As already remarked, the procedure based on a planning - executing - sensing - re-planning loop is complete (the goal is reached whenever possible) only under the assumption that actions are reversible. Such a limit, however, can be overcome in a context where control knowledge can be easily encoded. In fact, domain specific heuristic information can prevent dead-ends and guarantee the achievement of the goal, exploiting a good (domain-dependent) policy in the choice of actions.

Much work still remains to be done: from a practical point of view the prototype must be improved, in order to prove its effectiveness. The performances of the classical planning system presented in [9], that is based on the "planning as satisfiability (in LTL)" approach, have been shown to be comparable with other systems. This is mainly due to an efficient implementation of propositional subformulae, exploiting ROBDD [4]. The prototype, that at present uses a naive representation of formulae, has to be re-implemented in order to benefit from that technique. Once this is done, the system has to be completed with the other distinguishing features of this approach, i.e. the possibility to include heuristic knowledge and information about domain invariants, as well as the equipment with utility tools, such as consistency and redundancy checks, that can easily be done when the problem representation is entirely logical.

From the theoretical standpoint, an important issue to be addressed concerns the possibility to synthesize looping plans on the same basis. In fact, when linear plans are looked for, plan search stops as soon as a state s_1 is reached whose temporal description is identical to an already existing one, s_0. On the contrary, if loops are allowed, such situations have to be handled differently: the sequence of steps leading from s_0 to s_1 is recorded as a loop, and the search proceeds in a different direction. Some non-trivial work has however to be done in order to identify the correct conditions to exit the loop, and include them among the conditions of the contingent plan.

References

1. F. Bacchus and F. Kabanza. Using temporal logic to control search in a forward chaining planner. In *Proc. of the TIME-95 International Workshop on Temporal Representation and Reasoning*, Melbourne, Florida, April 1995.
2. Fahiem Bacchus and Froduald Kabanza. Planning for temporally extended goals. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, pages 1215–1222. AAAI Press / The MIT Press, 1996.
3. P. Bertoli, A. Cimatti, M. Roveri, and P. Traverso. Planning in non deterministic domains under partial observability via symbolic model checking. In *Proc. of the 17th Int. Joint Conf. on Artificial Intelligence IJCAI 2001*, pages 473–478, 2001.
4. Randal E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, August 1986.

5. D. Calvanese, G. De Giacomo, and M.Y. Vardi. Reasoning about actions and planning in LTL action theories. In *Proc. of the 8th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR'02)*, pages 593–602, 2002.
6. S. Cerrito and M. Cialdea Mayer. Using linear temporal logic to model and solve planning problems. In F. Giunghiglia, editor, *Proceedings of the 8th International Conference on Artificial Intelligence: Methodology, Systems, Applications (AIMSA'98)*, pages 141–152. Springer, 1998.
7. A. Cesta and A. Oddi. DDL.1: a formal description of a constraint representaton language for physical domains. In M. Ghallab and A. Milani, editors, *New Direction in AI Planning*, pages 341–352. IOS Press, 1996.
8. M. Cialdea Mayer and C. Limongelli. Linear time logic, conditioned models and planning with incomplete knowledge. In C. Fermüller and U. Egly, editors, *Proc. of the Int. Conf. on Automated Reasoning with Analytic Tableaux and Related Methods*, volume 2381 of *LNAI*, pages 70–84. Springer, 2002.
9. M. Cialdea Mayer, A. Orlandini, G. Balestreri, and C. Limongelli. A planner fully based on linear time logic. In S. Chien, S. Kambhampati, and C.A. Knoblock, editors, *Proc. of the 5th Int. Conf. on Artificial Intelligence Planning and Scheduling (AIPS-2000)*, pages 347–354. AAAI Press, 2000.
10. K. Currie and A. Tate. O-Plan: the open planning architecture. *Journal of Artificial Intelligence*, 52:49–86, 1991.
11. E. Giunchiglia, G. N. Kartha, and V. Lifschitz. Representing action: Indeterminacy and ramifications. *Artificial Intelligence*, 95(2):409–438, 1997.
12. H. Kautz and B. Selman. Planning as satisfiability. In B. Neumann, editor, *10th European Conf. on Artificial Intelligence*, pages 360–363. Wiley & Sons, 1992.
13. H. Levesque. What is planning in the presence of sensing? In *Proc. of the 13th National Conference on Artificial Intelligence, AAAI-96*, pages 1139–1146. AAAI Press, 1996.
14. E. Giunchiglia P. Ferraris. Planning as satisfiability in nondeterministic domains. In *Proc. of the Nat. Conf. on Artificial Intelligence (AAAI'00)*, 2000.
15. M. A. Peot and D. E. Smith. Conditional nonlinear planning. In J. Hendler, editor, *Proc. of the First International Conference on Artificial Intelligence Planning Systems*, pages 189–197. Morgan Kaufmann, 1992.
16. R. P. A. Petrick and F. Bacchus. A knowledge-based approach to planning with incomplete information and sensing. In *Proc. of the 6th Int. Conf. on Artificial Intelligence Planning and Scheduling (AIPS-2002)*, 2002.
17. M. Pistore and P. Traverso. Planning as model checking for extended goals in non-deterministic domains. In *Proc. IJCAI'01*. AAAI Press, 2001.
18. L. Pryor and G. Collins. Planning for contingencies: a decision-based approach. *Journal of Artificial Intelligence Research*, 4:287–339, 1996.
19. R. Reiter. The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. In V. Lifschitz, editor, *Artificial Intelligence and mathematical theory of computation: Papers in honor of John McCarthy*, pages 359–380. Academic Press, 1991.
20. J. Rintanen. Constructing conditional plans by a theorem-prover. *Journal of Artificial Intellingence Research*, 10:323–352, 1999.
21. B. Stephan and S. Biundo. Deduction based refinement planning. In B. Drabble, editor, *Proceedings of the 3rd International Conference on Artificial Intelligence Planning Systems (AIPS-96)*, pages 213–220. AAAI Press, 1996.