Conditional progressive planning: a preliminary report

Lars Karlsson
Department of Technology
Örebro University
SE-701 82 Örebro, Sweden
Email: lars.karlsson@tech.oru.se

Abstract. In this article, we describe a conditional planner called ETLplan, which is an extension of the sequential planner TLplan. ETLplan is a progressive planner, and it uses strategic knowledge encoded in an epistemic-temporal logic to reduce its search space. It utilizes an advanced representation of knowledge and action, with a semantics based on the notion of an epistemic situation. Besides presenting the planner itself — its representation of actions and plans, and its algorithm — we also provide some promising data from performance tests.

1 Introduction

Planning is concerned with methods for finding courses of action that an agent can use to achieve its goals. In the earliest planning systems such as STRIPS [9], the agent was supposed to have complete information about the state of its environment and about its own state, and hence the plans they generated were simply sequences of actions. In conditional planning, sometimes also called contingency planning, one assumes that the agent may not always have complete information at planning time. Consequently, finding a plan that is guaranteed to lead to a specific goal is more difficult. However, the agent may also have some means to improve its situation by sensing its environment. Depending on the outcome of that sensing, it may decide to continue its plan in different ways. Thus, a conditional plan can contain sensing actions and conditional branches relating to the different outcomes of those sensing actions. One of the earliest conditional planners was CNLP [14] (Conditional Non-Linear Planner), which however suffered from two problems: it had a very simple — one might say oversimplified — model of sensing, and it was far too slow for practical applications. These two problems — representing sensing, and performance — are key issues in the design of conditional planning systems.

In this article, we describe a conditional planner called ETLplan. It is based on a sequential (non-conditional) planner called TLplan [1, 2]. Two of the features that makes TLplan interesting are its fairly expressive representation and its good performance. The latter is indicated by its outperforming most other comparable planners in empirical tests, as has been documented in [2]. In addition, another sequential planner [6] based on the same principles as TLplan won its track in the AIPS-2000 planning competition.

TLplan and ETLplan are progressive planners; they start from an initial situation and apply actions to that and subsequent resulting situations, until a situation where the goal is satisfied is reached. The fact that the planners are progressive implies

that when it comes to actions, one can reason from causes to effects, and one always has a completely specified situation. This simplifies causal reasoning significantly, in particular compared to regressive planners (e.g. CNLP). The latter have to reason in the other direction and without a completely specified situation. The disadvantage of progressive planning, on the other hand, is that the search space is potentially very large even for small problems. This problem is solved in TLplan and ETLplan by the use of strategic knowledge, which helps pruning away unpromising plan prefixes.

ETLplan adds two features to TLplan. First, the representation of actions and situations used in ETLplan permits actions that have sensing effects, that is the agent may observe certain fluents (state variables). Second, based on these observations, the planning algorithm can generate conditional plans. These two extensions are the major contributions of this article. In the rest of the article, we describe the plan representation of ETLplan, the use of strategic knowledge and the planning algorithm. We also briefly provide some performance data.

2 Representation

2.1 Syntax: Fluents

The state of the world is described in terms of fluents (state variables), which are atomic formulae in a standard first-order language. Examples of fluents are `tails(coin1)` and `metal(package1)`. A fluent formula is a logical combination of fluents using the standard connectives and quantifiers.

2.2 Syntax: Actions

In the original TLplan, the conditions and effects of actions were encoded as STRIPS or ADL operators [2]. Such operators cannot express non-deterministic (chance) effects of actions or — in particular — actions that involve sensing. Therefore, we introduce a new type of operator for ETLplan. An operator consists of a pair $\langle P, R\rangle$ where P is a precondition (a fluent formula) and R is a set of result descriptions. Each result description $r \in R$ is a tuple $\langle C, E, O\rangle$ where

- C is a context condition (a fluent formula) that determines when (in what states) the result is applicable.
- E is a set of literals (positive or negative fluents) that specifies the effects of the result. We let E^+ denote the positive fluents in E and let E^- denote the negative ones.
- O is a set of literals that specifies the observations of the result. These observations are assumed to be made by the agent executing the plan.

Example. The following is an ETLplan operator for flipping a coin, where the side that comes up is observed. The single argument x stands for the coin that is flipped.

operator:	`FlipCoin(`x`)`		
precond:	`hasCoin(`x`),`		
	context	*effects*	*observations*
result 1:	`(true, { tails(`x`), ¬heads(`x`) }, { tails(`x`) }),`		
result 2:	`(true, { heads(`x`), ¬tails(`x`) }, { heads(`x`) })`		

When occurring in a plan, an instantiated operator is referred to by using its name and its arguments: *action* `::=` *op-name*`(`*args...*`)`. For instance, the operator above instantiated with $x =$ `coin1` could be referred to as `FlipCoin(coin1)`.

2.3 Semantics: Situations and epistemic situations

We use a model of knowledge and action which is based on the concepts of a situation and an epistemic situation. In short, a situation describes one possible state of the world at a given point in time, and an epistemic situation, which is essentially a set (equivalence class) of situations, describes the agent's knowledge at a point in time. A transition relation over situations provides the temporal dimension; the transitions are due to applications of actions. Our model is along the lines of Moore [13], where a modal knowledge relation over situations/states defines the epistemic state. In our case, though, the knowledge relation is simplified to be an equivalence relation.

A situation s is an object which has the following properties and relations:

- a state $state(s)$, which is the set of atomic statements that are true in the situation (all others are false);
- an observation set $obs(s)$, which is a set of literals;
- a transition relation $res(s, s')$ which specifies what situations s' have resulted from executing some specific action in s. The situation s can have several resulting s'; this makes it possible to represent nondeterministic effects of an action. On the other side, each situation has a unique history: for each s' there can only be one s such that $res(s, s')$.

An epistemic situation $\overline{s}$ is an equivalence class of situations, and represents the agent's knowledge at a point in time. Situations in the same epistemic situation cannot be distinguished by the planning agent, given his current knowledge. The more situations, the less the agent knows. We denote the equivalence relation defining the equivalence classes over the space of situations by K.

2.4 Semantics: Effects of actions

Applying an operator to an epistemic situation results in one or more new epistemic situations, provided the preconditions are true. These new epistemic situations are obtained by first applying the operator to the individual situations in the original epistemic situation, and then partitioning the resulting situations into epistemic situations according to their observation sets. The results of the `FlipCoin` operator in a specific epistemic situation is shown in figure 1.

Technically, the results of an operator/action A in a situation is defined as follows: if the precondition P is true in s then for each result description $r_i \in R$ such that:

1. the context condition C_i is true in s

there is a situation s' such that:

2. $res(s, s')$ — s' has resulted from s;
3. $state(s') = (state(s) \cup E_i^+) \setminus E_i^-$ — the effects of the result take place; and

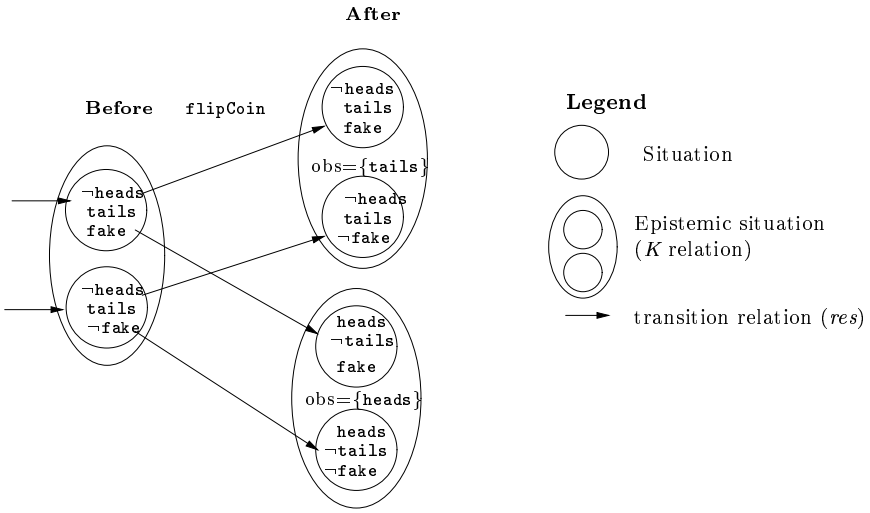

Figure 1: The results of flipping a coin (`FlipCoin` above). Notice the two resulting epistemic situations; the upper one is due to *result 1* in `FlipCoin`, and the lower one is due to *result 2*. In each one the agent knows whether heads or tails are up, but remains ignorant on whether the coin is a fake one. The `has-coin` fluent has been omitted for space reasons, as have the argument `coin1` of the operator and fluents.

4. $obs(s') = O_i$ — the observations of the result are made.

In addition, these are the only results: all situations satisfying condition 2 above — $res(s, s')$ — must also satisfy conditions 1, 3 and 4 for some result r_i.

The effects of an action/operator on an epistemic situation is defined as follows. For two situations s'_1 and s'_2, $K(s'_1, s'_2)$ holds, that is they are in the same epistemic situation $\overline{s}'$, if and only if:

1. $obs(s'_1) = obs(s'_2)$, that is the same observations are made in the two situations; and

2. there are s_1 and s_2 such that $res(s_1, s'_1)$, $res(s_2, s'_2)$, and $K(s_1, s_2)$, that is they (s'_1 and s'_2) have resulted from two situations belonging to the same epistemic state $\overline{s}$.

In summary, the application of an action to an epistemic situation $\overline{s}$ yields a set of new epistemic situations $\{\overline{s}'_1, \overline{s}'_2, \ldots\}$ which are defined by the equivalence relation K. If the results of the action include observations, then there might be more than one new epistemic situation; if there are no observations, there is only one.

2.5 Syntax: Conditional plans

Plans in ETLplan are conditional. This means that there can be points in the plans where the agent can choose between different ways to continue the plan depending on some explicit condition. Therefore, in addition to the sequencing plan operator (`;`), we introduce a conditional operator (`cond`). The syntax of a conditional plan is as follows:

plan ::= `success` | *action* `;` *plan* | `cond` *branch*`*`
branch ::= `(`*cond* `:` *plan*`)`

A condition *cond* is a conjunction of fluent literals. The conditions for a branch should be exclusive and exhaustive relative to the potential epistemic situations at that point in the plan. `Success` denotes predicted plan success.

Example. Figure 2 shows a plan for the "bomb in toilet" scenario. There is a bomb in one of five packages (`p1`,...,`p5`), and the bomb can be disarmed by dunking the package that contains it into a toilet (`t1`). Only one package can be dunked into the toilet; after that, the toilet is clogged. In the plan, the agent uses a metal detector (`detect-metal(p`n`)`) to discover in which package the bomb is (indicated by `metal(p`n`)`). When detected, the package with the bomb is immediately dunked into a toilet (`dunk(p`n`,t1)`). This scenario is further discussed in section 5.

```
detect-metal(p1) ;
cond (metal(p1) : dunk(p1,t1) ; success)
     (¬metal(p1) : detect-metal(p2);
      cond (metal(p2) : dunk(p2,t1) ; success)
           (¬metal(p2) : detect-metal(p3);
            cond (metal(p3) : dunk(p3,t1) ; success)
                 (¬metal(p3) : detect-metal(p5);
                  cond (metal(p5) : dunk(p5,t1) ; success)
                       (¬metal(p5) : dunk(p4,t1) ; success))))
```

Figure 2: A conditional plan for the "bomb in toilet" scenario.

2.6 Semantics: Application of conditional plans

The application of a plan to an epistemic situation results in a set of new epistemic situations. First, the application of an action a to an epistemic situation $\overline{s}_i$ is defined as:

$$Apply(a, \overline{s}_i) = \{\overline{s}'_j\}_j \text{ where the set } \{\overline{s}'_j\}_j \text{ is obtained as in section 2.4.} \quad (1)$$

Next, the application of a sequence is defined as follows:

$$Apply(a\,;p, \overline{s}_i) = \bigcup_j Apply(p, \overline{s}'_j) \text{ where each } \overline{s}'_j \in Apply(a, \overline{s}_i) \quad (2)$$

The application of a conditional plan element is defined as an application of the branch whose context condition holds in the current epistemic state (there should only be one such branch):

$$\begin{array}{l} Apply(\texttt{cond } (c_1\texttt{:}p_1)\ldots(c_n\texttt{:}p_n), \overline{s}_i) = \\ \quad Apply(p_j, \overline{s}_i) \text{ for the } j \text{ such that } (\overline{s}_i) \models c_j \end{array} \quad (3)$$

The expression $(\overline{s}_i) \models c_j$ denotes that c_j is entailed by all states of the situations in $\overline{s}_i$.

This concludes how actions and plans are represented in ETLplan. Next, we proceed to investigate how strategic knowledge can be represented and used.

3 Strategic knowledge

The key feature of TLplan is the possibility to encode strategic knowledge. TLplan is a progressive total-order planner, applying actions in a sequence starting from a completely specified initial situation. Hence it is always working on a completely specified situation, and strategic knowledge are applied to such situations. Strategic knowledge is encoded as expressions (search control formulae) in first-order linear temporal logic (LTL) [8] and is used to determine when a situation should not be explored further. One example could be the condition "never pick up an object and then immediately drop it again". If this condition is violated, that is evaluates to `false` in some situation, the plan prefix leading there is not explored further and all its potential continuations are cut away from the search tree. A great advantage of this approach is that one can write search control formulae without any detailed knowledge about how the planner itself works; it is sufficient to have a good understanding about the problem domain.

We will briefly present LTL here, and then proceed to the extensions made to include epistemic situations. LTL is based on a standard first-order language consisting of predicate symbols, constants and function symbols and the usual connectives and quantifiers. In addition, there are four temporal modalities: $\mathcal{U}$ (until), $\Box$ (always), $\Diamond$ (eventually), and $\bigcirc$ (next). Interpreted over a (possibly infinite) sequence of situations $B = \langle s_1, s_2, \ldots \rangle$ and a situation s_i in that sequence, the expression $\phi_1 \mathcal{U} \phi_2$ means that ϕ_2 holds in the current (i.e. s_i) or some future situation, and until that situation ϕ_1 holds; $\Box\phi$ means that ϕ holds in this and all subsequent situations; $\Diamond\phi$ means that ϕ holds in this or some subsequent situation; and $\bigcirc\phi$ means that ϕ holds in the next situation s_{i+1}. For details, see e.g. [8].

For the purpose of ETLplan, we can interpret the temporal modalities over a sequence of epistemic situations $B = \langle \overline{s}_1, \overline{s}_2, \ldots \rangle$ and a current epistemic situation $\overline{s}_i$ in that sequence. As indicated by figure 1, the application of a plan actually generates a tree structure of epistemic situations and not a straight sequence. But the different branches in this tree are effectively sequences, and the ETL search control formulas will be applied to individual branches.[1]

In addition to the temporal modal operators, TLplan and ETLplan also include a goal operator $\mathcal{G}$ (this is not part of LTL), which is useful for referring to the goal in search control formulae. We let $\mathcal{G}\varphi$ denote that it is among the agent's goals to achieve the fluent formula φ. Semantically, this modality will be interpreted relative to a set of goal states G (i.e. the set of states that satisfy the goal.) For ETL, we also introduce one new modal operator $\mathcal{K}$ (knows), where $\mathcal{K}\varphi$ means that the fluent formula φ is known to be true in the current epistemic situation. In ETL, we restrict fluent formulae to appear only inside the $\mathcal{K}$ and $\mathcal{G}$ operators.

We can now define the semantics of these modal operators relative to a branch B of epistemic situations, a current epistemic situation $\overline{s}_i$, a variable assignment V, and a set of goal states G, as follows (φ is a fluent formula):

- $(B, \overline{s}_i, V, G) \models \phi_1 \mathcal{U} \phi_2$ iff there exists a $j \geq i$ such that $(B, \overline{s}_j, V, G) \models \phi_2$ and for all k such that $i \leq k \leq j$, $(B, \overline{s}_k, V, G) \models \phi_1$.
- $(B, \overline{s}_i, V, G) \models \Box\phi$ iff for all $j \geq i$, $(B, \overline{s}_j, V, G) \models \phi$.

[1] Temporal formulas are applied to individual branches in for instance the computation tree logic CTL* [5]. It would be possible introduce the all-branches and some-branch modal operators of CTL* into ETL, but they are not needed in the planner presented here.

Algorithm *Progress(f,$\overline{s}$)*
Case:

$$
\begin{array}{lll}
1. & f = \mathcal{K}f_1: & f^+ := \texttt{true} \text{ if } s \models f_1 \text{ for all } s \in \overline{s},\ \texttt{false} \text{ otherwise} \\
2. & f = \mathcal{G}f_1: & f^+ := \texttt{true} \text{ if } s \models f_1 \text{ for all } s \in G,\ \texttt{false} \text{ otherwise} \\
3. & f = f_1 \wedge f_2: & f^+ := Progress(f_1, \overline{s}) \wedge Progress(f_2, \overline{s}) \\
4. & f = \neg f_1: & f^+ := \neg Progress(f_1, \overline{s}) \\
5. & f = \bigcirc f_1: & f^+ := f_1 \\
6. & f = f_1 \mathcal{U} f_2: & f^+ := Progress(f_2, \overline{s}) \vee (Progress(f_1, \overline{s}) \wedge f) \\
7. & f = \Diamond f_1: & f^+ := Progress(f_1, \overline{s}) \vee f \\
8. & f = \Box f_1: & f^+ := Progress(f_1, \overline{s}) \wedge f \\
9. & f = \forall x[f_1]: & f^+ := \bigwedge_{c \in U} Progress(f_1(x/c), \overline{s}) \\
10. & f = \exists x[f_1]: & f^+ := \bigvee_{c \in U} Progress(f_1(x/c), \overline{s})
\end{array}
\tag{4}
$$

Return f^+

Figure 3: The ETLplan progression algorithm.

- $(B, \overline{s}_i, V, G) \models \Diamond\phi$ iff there exists a $j \geq i$ such that $(B, \overline{s}_j, V, G) \models \phi$.
- $(B, \overline{s}_i, V, G) \models \bigcirc\phi$ iff $(B, \overline{s}_{i+1}, V, G) \models \phi$.
- $(B, \overline{s}_i, V, G) \models \mathcal{G}\varphi$ iff for all $s \in G$, $(s, V) \models \varphi$.
- $(B, \overline{s}_i, V, G) \models \mathcal{K}\varphi$ iff for all $s \in \overline{s}_i$, $(s, V) \models \varphi$.

In order to efficiently evaluate the LTL formulas, TLplan incorporates a progression algorithm that takes as input an LTL formula f and a situation and returns a formula f^+ that is "one step ahead", i.e. corresponds to what remains to evaluate of f in subsequent situations. For ETLplan, we modify the progression algorithm to apply to epistemic situations $\overline{s}$ (the goal states G in step 2 are fixed for a given planning problem and need not be passed along). It is shown in figure 3. Note that the algorithm assumes that all quantifiers range over a finite universe U.

Example. The following is a control formula stating that a robot should never pick an object up and then drop it immediately again.

$$
\begin{array}{l}
\Box\neg(\mathcal{K}(\neg\exists x[\texttt{robot-holds}(x)])\wedge \\
\quad \bigcirc(\mathcal{K}(\exists x[\texttt{robot-holds}(x)]) \wedge \bigcirc\mathcal{K}(\neg\exists x[\texttt{robot-holds}(x)])))
\end{array}
\tag{5}
$$

4 The planning algorithm

ETLplan is a progressive planner, which means that it starts from an initial epistemic situation and then tries to sequentially apply actions until an epistemic situation where the goal is satisfied is reached. The algorithm is shown in figure 4. It takes as input an epistemic situation $\overline{s}$, an ETL search control formula f, a goal formula g and a set of actions A. It is initially called with the given initial epistemic situation and an un-progressed search control formula. Step 1 checks if the goal is satisfied. Step 2 progresses the search control formula; if it evaluates to `false`, the epistemic situation is considered "bad" and the planner does not explore this branch further. In step 3, actions are chosen; when there are no more actions to try, we stop (step 4).

Algorithm ETLplan$(\overline{s}, f, g, A)$
1. If $\overline{s} \models g$ then return `success`.
2. Let $f^+ := \mathit{Progress}(f, \overline{s})$;
   if $f^+ =$ `false` then return failure.
3. Choose an action $a \in A$ whose precondition is satisfied in all $s \in \overline{s}$
   (backtrack point).
4. If no such action a exists, return failure.
5. Let $S^+ = \mathit{Apply}(a, \overline{s})$.
6. For each $\overline{s}_i^+ \in S^+$, let $P_i :=$ ETLplan$(\overline{s}_i^+, f, g, A)$.
   If there is any $P_i =$ failure, then return failure.
7. (a) If $|S^+| = 1$, return a ; P_1.
   (b) Otherwise return a ; `(cond` $(c_1 : P_1) \ldots (c_n : P_n)$`)`
   where each $c_i = \mathit{obs}(s)$ for $s \in \overline{s}_i^+$.

Figure 4: The ETLplan planning algorithm.

Steps 1–4 are quite similar to corresponding steps in the original TLplan algorithm. However, steps 5–7 are significantly different. In step 5, a set of new epistemic situations are obtained. In original TLplan, the application of an action to a situation always resulted in a single new situation. The different outcome of step 5 also implies that the subsequent steps in ETLplan are different from TLplan. In step 6, we recursively continue planning from each resulting epistemic situation $\overline{s}_i^+$, which yields a continued plan P_i for each $\overline{s}_i^+$. If we encounter a failure plan, we consider the entire current plan a failure. Note that we plan for each resulting epistemic situation (i.e. contingency) separately. Finally, in step 7, the different continued plans obtained in step 6 are used as branches in a conditional plan element, which is added to the action selected in step 3. The conditions for each branch are obtained from the observations in the corresponding epistemic situations (case (b)). If there is only one branch, a simple sequence is sufficient (case (a)).

5 Implementation and experimental results

In this section, we present some preliminary experimental results which are intended to give an indication of the performance of the ETLplan system. The system is implemented in Allegro Common-Lisp. All experiments described here were performed on a 300 MHz Pentium II running the Linux operative system.

The skiing vacation scenario. This scenario is from [14] (CNLP), and involves an agent who is going on a skiing vacation in the mountains. There are two possible ski resorts, and the road to any or both of them might be blocked. ETLplan solved the problem in 300 msec, using two very simple search control formulas to avoid suboptimal movements and to avoid leaving one's skis behind. This can be compared to the conditional planner Sensing Graph Plan (SGP), one of the fastest conditional planners today, which is reported to have required 2152 msec under equivalent conditions [17].

The bomb in toilet scenario. This scenario with the five packages, the bomb and the toilet was discussed in section 2.5. The ETL encoding of this scenario utilized two search control formulas to avoid unnecessary clogging and delays in dunking the bomb,

and the execution time was 360 msec. For comparison, it took SGP 400 msec to solve this problem [17].

The cold-room scenario. This is a more realistic scenario from [10] involving a poorly illuminated table with one green and one red test tube. A robot should move the red tube to a second table in a cold-room. In order to accomplish this, the robot has to pass via a third table with better illumination to verify that it actually picked the correct tube from the first table. In addition, the door to the cold-room should not be left open. A conjunction of 8 search control formulae were used in this scenario, and one of them was shown in the example at the end of section 3. The problem was solved by ETLplan in 2780 msec, to be compared to 11780 msec by a possibilistic conditional Graph Plan derivative [10] implemented in C++ and running on a Sparc Ultra-2.

6 Related work

The very first conditional planner was Warplan-C [16], which could add conditional branches to a sequential plan. CNLP [14], which was discussed in section 1, is the first partial-order conditional planner. Cassandra [15], also a partial-order planner, introduced more realistic models of sensing, and C-Buridan [7] is a probabilistic partial-order planner that can compute a probability of success for a conditional plan. None of these planners were enough efficient for moderately large planning problems. Sensing Graph Plan (SGP) [17], an extension of the fast planner Graph Plan [4], is arguably the first conditional planner to have a more practical level of efficiency. It utilizes a model of knowledge similar to the one of ETLplan. In at least one respect, the plan representation of SGP is weaker than the one of ETLplan: in SGP, sensing actions cannot have preconditions and what fluents are sensed may not be context dependent. Another conditional Graph Plan derivative is Guéré's and Rachid's possibilistic planner [10], which also has yielded some impressive performance results. The conditional probabilistic planners C-MAXPLAN and Zander [12] are based on transforming propositional planning problems to stochastic satisfiability problems, which then can be solved in a highly efficient manner. Finally, we should mention the work done on partially observable Markov decision processes, or POMDPs; see for instance [11]. Compared to conditional planning, this work is on the more general problem of generating policies (mappings from belief states to actions, where a belief state is a set of states with associated probabilities.) POMDPs are based on explicit enumerations of states or propositional state representations.

7 Conclusions and future work

In this paper, we have presented work on the planner ETLplan. It is a progressive planner which utilizes strategic knowledge to reduce its search space. We have described the fairly rich representation that underlies the planner: actions with context-dependent and non-deterministic effects and observations, and plans with conditional branches. The semantics of this representation is based on the notion of an epistemic situation that describes the agent's knowledge at a point in time. We have also described the planning algorithm and how we utilize an epistemic-temporal logic to encode search control knowledge that can be used to prune unpromising branches in the search tree.

ETLplan is based on TLplan [1, 2]. The novel contributions of ETLplan are the possibilities to represent observations and sensing, to refer to the agent's knowledge in

the search control formulae, and to use observations to generate conditional plans. In addition, nondeterministic effects can be represented.

In this paper, we have also given some preliminary and quite promising performance data on ETLplan. Further experiments are necessary to make a more thorough evaluation of the planner, and plenty can also be done to make the implementation more efficient. For instance, techniques that has been used to speed up TLplan by statically or dynamically eliminating irrelevant actions [3] could be adapted for ETLplan. Finally, an interesting line of future work would be to develop novel means to make the planner more efficient, such as use of subgoal information.

Acknowledgements

Thanks to Silvia Coradeschi, Alessandro Saffiotti and the anonymous referees for comments on earlier versions of this paper. This work was supported by the Swedish KK foundation.

References

[1] F. Bacchus and F. Kabanza. Using temporal logic to control search in a forward chaining planner. In M. Ghallab and A. Milani, editors, *New Directions in Planning*, pages 141–153. IOS Press, Amsterdam, 1996.

[2] F. Bacchus and F. Kabanza. Using temporal logics to express search control knowledge for planning. *Artificial Intelligence*, 116:123–191, 2000.

[3] Fahiem Bacchus and Yee Wye Teh. Making forward chaining relevant. In *Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems*, pages 54–61. AAAI Press, Menlo Park, Calif., 1998.

[4] A. Blum and M.L. Furst. Fast planning through planning graph analysis. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, Montreal, 1995. Morgan Kaufmann.

[5] E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal-logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.

[6] Patrick Doherty and Jonas Kvarnström. TALplanner: a narrative temporal logic-based forward-chaining planner. In *TIME'99*, 1999.

[7] Denise Draper, Steve Hanks, and Daniel Weld. Probabilistic planning with information gathering and contingent execution. In *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems*. AAAI Press, Menlo Park, Calif., 1994.

[8] E.A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B*, chapter 16, pages 997–1072. MIT Press, 1990.

[9] Richard E. Fikes and Nils J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[10] Emmanuel Guéré and Rachid Alami. A possibilistic planner that deals with nondeterminism and contingency. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, Stockholm, 1999. Morgan Kaufmann.

[11] L.P. Kaebling, M.L. Littman, and A.R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1–2):99–134, 1998.

[12] Stephen M. Majercik and Michael L. Littman. Contingent planning under uncertainty via stochastic satisfiability. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence*, pages 549–556, Orlando, Florida, 1999. AAAI Press, Menlo Park, California.

[13] Robert C. Moore. A formal theory of knowledge and action. In J.R. Hobbs and R.C. Moore, editors, *Formal Theories of the Commonsense World*, chapter 9. Ablex, Norwood, New Jersey, 1985.

[14] M. Peot and D. Smith. Conditional nonlinear planning. In *Proceedings of the First International Conference on Artificial Intelligence Planning Systems*. Morgan Kaufmann, 1992.

[15] Louise Pryor and Gregg Collins. Planning for contingencies: A decision-based approach. *Journal of Artificial Intelligence Research*, 4:287–339, 1996.

[16] D. H. D. Warren. Generating conditional plans and programs. In *Proceedings of the Summer Conference on AI and Simulating Behavior*, Edingburgh, 1976.

[17] D.S. Weld, C.R. Anderson, and D.E. Smith. Extending graphplan to handle uncertainty and sensing actions. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, Madison, Wisconsin, 1998. AAAI Press, Menlo Park, California.