Proceeding of the 6th International Symposium on Artificial Intelligence and Robotics & Automation in Space: i-SAIRAS 2001, Canadian Space Agency, St-Hubert, Quebec, Canada, June 18-22, 2001.

Merging Planning and Verification Techniques for "Safe Planning" in Space Robotics

Luigia Carlucci Aiello[1] Amedeo Cesta[2] Enrico Giunchiglia[3] Paolo Traverso[4]

[1]DIS - Univ. di Roma, aiello@dis.uniroma1.it
[2]IP-CNR, Roma, cesta@ip.rm.cnr.it
[3]DIST - Univ. di Genova, enrico@dist.unige.it
[4]ITC-IRST, Trento, traverso@itc.it

Keywords: Planning, Formal Verification, Model Checking, Satisfiability.

Abstract

Space applications require software systems that are both autonomous and safe. Planning techniques can automatically generate plans that achieve given goals. Formal verification techniques, like model checking, can guarantee a higher degree of safety. In this paper, we combine planning and formal verification techniques in order to address the problem of "Safe Planning". By "Safe Planning", we mean the task of generating/validating plans that not only achieve the goal, but verify also a set of other user defined properties, e.g., safety properties.

This work has been carried out in the context of "SACSO" (SAfety Critical SOftware for planning in space robotics), a national project funded by the Italian Space Agency (ASI).

1 Introduction

The increasing complexity of the services requested to robotic devices in space applications results in a need for more and more sophisticated and autonomous systems. A compelling requirement for space autonomy led to the development of systems that perform complex, time consuming and critical tasks, possibly without the need of human intervention [1]. Planning is a research area in artificial intelligence aiming at the construction of systems — called planners — that enable a robot to autonomously synthesize a course of actions that will achieve its goals. Planning has been studied since the early days of Artificial Intelligence (AI); recently the interest has been renewed and the results abound, with systems able to automatically generate plans with more than 100 actions. See, e.g., [1], for an application of planning techniques to space robotics.

On the other hand, the same increasing complexity of the requested services, causes an analogous increase in the complexity of the specifications and of the programs controlling the robotic devices. Concerns arise about the correctness of the program, about the correctness of the specifications of the program and/or of the environment in which the robots are supposed to operate (see, e.g., [2]). Model checking (see, e.g., [3]) is a research area in computer science devoted to the definition of procedures for the automatic verification of programs and/or specifications. Current model checkers, are able to efficiently verify systems with more than 10^{120} states [4]. The characteristic that makes model checking particularly attractive is that it is a "push button" technology: it is completely automatic.

There are two striking similarities between plan-

ning and model checking: Both perform state-space exploration, and both are fully automatic. However, the planning and formal verification fields have until recently evolved separately, each adopting and developing its own techniques and tools. In the last few years, we have seen a cross-fertilization between these two areas in which ideas originally thought in formal verification have been successfully employed in planning (see, e.g., [5, 6]), and vice versa (see, e.g., [7]). More in details, in [5], the authors show how it is possible to use Ordered Binary Decision Diagrams (BDDs) —traditionally used in formal verification [8] — for planning. In [7], the authors show how it is possible to adopt the Davis, Logemann, Loveland procedure [9] for state exploration —an idea at the basis of the "planning as satisfiability" framework proposed in [10]. These works show that it is possible to adopt model checking techniques for generating plans that achieve a given goal, e.g. lead the system to a desired state.

In this paper, we address the problem of "safe planning". By "safe planning", we mean the task of generating/validating plans that not only achieve the goal, but verify also a set of other user defined properties, e.g., safety properties. This is particularly important in the context of space applications, where autonomy and safety are indeed two crucial properties any system has to guarantee. More in details, we show how it is possible to further exploit the similarities between planning and model checking, and define procedures

- for the validation of user defined plans with respect to given (safety) properties, and
- for the generation of plans satisfying the desired properties.

In this paper we focus on one of the possible techniques for safe planning, the planning as satisfiability approach introduced in [10] in the classical setting, and extended in [11] for dealing with nondeterministic domains. Other approaches —and in particular approaches BDD-based approaches— are also possible and briefly discussed in the conclusions.

The current work is part of "SACSO" (SAfety Critical SOftware for planning in space robotics), a national project funded by ASI. In this project, our goal is to incorporate the techniques here described in JERRY [11], a system that supports the interactive design, planning, control and supervision of a autonomous systems operating in a space environment.

The paper is structured as follows. In Section 2 we briefly introduce the basic planning terminology and definitions used in the rest of the paper. Section 3 is devoted to the introduction of the temporal logic we use to formally define the properties that safe plans have to meet. In Section 4 we show how it is possible to check whether a valid plan (either user defined, or automatically generated by a planner) is also safe. The automatic generation of safe plans is the object of Section 5. In Section 6, we draw some conclusions, discuss some related work, and discuss how different techniques based on BDDs can be applied to safe planning.

2 Possible and Valid Plans

We start with a set of atoms partitioned into a set of *fluent symbols* and a set of *action symbols*. Intuitively, a fluent represents a property of the world that changes over time, because of the execution of actions. A *formula* is a propositional combination of atoms. An *action* is an interpretation of the action symbols. A *state* is an interpretation of the fluent signature. A *transition* is a triple $\langle s, a, s' \rangle$ where s, s' are states and a is an action: Intuitively s is the initial state of the transition, and s' is its resulting state. Intuitively, to execute an action a means to execute concurrently the "elementary actions" represented by the action symbols satisfied by a.

An *action description* D is a finite set of expressions describing how actions change the state of the world, i.e., the set of possible transitions. We do not make any assumption on D, except the following:

1. the effects of actions depend on the state of the world in which they are executed.

This restriction does not allow for "non-markovian" action descriptions like the one definable in

$\mathcal{ARD}$ [12]. Under the assumption that D is Markovian, it is possible to associate a transition diagram with each action description D. The *transition diagram* represented by D is the directed graph which has the states of D as vertices, and which includes an edge from s to s' labeled a for every transition $\langle s, a, s' \rangle$ that is possible according to D. Notice that D may be deterministic or non-deterministic.

Besides the above assumption, we also assume that

2. it is possible to compute a propositional formula tr_i^D whose satisfying assignments correspond to the possible transitions caused by the execution of an action in D.

More in detail, we assume that in tr_i^D there is a propositional variable A_i for each ground action symbol A, and two propositional variables F_i and F_{i+1} for each ground fluent F in D: Intuitively, F_i represents the value of F in the initial state of the transition, and F_{i+1} represents the value of F in the resulting state, after having performed the action.

As in the standard planning literature, a *planning problem* for D is characterized by two formulas I and G in the fluent signature, i.e., is a triple $\langle I, D, G \rangle$, where I and G encode the initial and goal state(s) respectively. A *plan* (of *length* $n \geq 0$) is a finite sequence $a^1; \ldots; a^n$ of actions.

Consider a planning problem $\pi = \langle I, D, G \rangle$. Intuitively, to ensure that a plan $a^1; \ldots; a^n$ is valid, we have to check

- that the plan is "always executable" in any initial state, i.e., executable for any initial state and any possible outcome of the actions in the plan, and
- that any "possible result" of executing the plan in any initial state is a goal state.

In order to make the above definition precise we have to give the following definitions.

A *history* for an action description D is a path in the corresponding transition diagram, that is, a finite sequence

$$s^0, a^1, s^1, \ldots, a^n, s^n \tag{1}$$

($n \geq 0$) such that $s^0, s^1, \ldots, s^n$ are states, $a^1, \ldots, a^n$ are actions, and

$$\langle s^{i-1}, a^i, s^i \rangle \qquad (1 \leq i \leq n)$$

are transitions which are possible according to D. n is the *length* of the history (1).

A plan $\vec{a} = a^1; \ldots; a^n$ is *possible* for π if there exists a history (1) for D such that

- s^0 is an initial state, and
- s^n is a goal state.

An action a is *executable* in a state s if for some state s', $\langle s, a, s' \rangle$ is a possible transition according to D. Let s^0 be a state. The plan $a^1; \ldots; a^n$ is *always executable in* s^0 if for any history

$$s^0, a^1, s^1, \ldots, a^k, s^k$$

with $k < n$, a^{k+1} is executable in s^k. Assume that $\vec{a}$ is a plan which is always executable in a state s^0. A state s^n is a *possible result of executing* $\vec{a}$ *in* s^0 if there exists a history (1) for D.

A plan $\vec{a} = a^1; \ldots; a^n$ is *valid* for π if for any initial state s^0,

- $\vec{a}$ is always executable in s^0, and
- any possible result of executing $\vec{a}$ in s^0 is a goal state.

Indeed, if D is deterministic, any possible plan is also valid. However, this is not the case if D is nondeterministic. A possible plan *may* lead to the goal; while valid plans are ensured to reach a goal state despite the potential multiple initial states and nondeterminism in the action description. In the planning literature, valid plans are also called "conformant" [13].

3 Safe Plans

Consider a planning problem $\pi = \langle I, D, G \rangle$. In the previous section we have formally defined the notion of valid plan as a sequence of actions which is guaranteed to reach a goal state. Depending on the characteristics of π (e.g., depending on the language used

to describe D) different planning procedures can be used. However, as we have already argued in the introduction, we do not only want that a plan be "valid", but also "safe" with respect to a set of user-defined properties. For example, we do not want a plan which can go through a state which is potentially dangerous for humans.

In the formal verification literature, these (safety) properties are nicely formalized by means of temporal logics. Here we consider a simple temporal logic, whose syntax and semantics resembles Pnueli's Linear Temporal Logic (LTL) [14]. The definition of temporal formula (used for the specification of the properties) is the following:

- A fluent symbol or the negation of a fluent symbol is a *temporal formula*, and
- If α and β are temporal formulas, also

$$\begin{array}{cccc} (\alpha \wedge \beta), & (\alpha \vee \beta), & \mathcal{X}_s\alpha, & \mathcal{X}_w\alpha, \\ \mathcal{F}\alpha, & \mathcal{G}\alpha, & (\alpha\mathcal{U}\beta), & (\alpha\mathcal{R}\beta), \end{array}$$

  are temporal formulas.

Notice that we have assumed temporal formulas to be in negation normal form, i.e., that negations occur only in front of fluent symbols. This is not a limitation given that —according to the semantics defined below— (assuming we relax for a moment the assumption to be in negation normal form)

$$\begin{array}{c} \neg\mathcal{X}_s\alpha \equiv \mathcal{X}_w\neg\alpha, \\ \neg\mathcal{F}\alpha \equiv \mathcal{G}\neg\alpha, \\ \neg(\alpha\mathcal{U}\beta) \equiv (\neg\alpha\mathcal{R}\neg\beta). \end{array} \quad (2)$$

Some of the above equivalences hold in standard LTL. However, the meaning of the temporal connectives $\mathcal{X}_w, \mathcal{F}, \mathcal{G}, \mathcal{R}$ have to be adjusted in order to accommodate the fact that we are dealing with a finite time line (see [15], pag. 1006). Intuitively, if n represent our horizon,

- $\mathcal{X}_s\alpha$ reads "there exists a successor moment and α holds there",
- $\mathcal{X}_w\alpha$ reads "if there exists a successor moment then α holds there",
- $\mathcal{F}\alpha$ reads "for some subsequent time $\leq n$ α holds",
- $\mathcal{G}\alpha$ reads "for all subsequent times $\leq n$ α holds",
- $\alpha\mathcal{U}\beta$ reads "for some subsequent time $\leq n$ β holds, and α holds until then",
- $\alpha\mathcal{R}\beta$ reads "if for some subsequent time $\leq n$ β does not hold, then α holds in a state before then".

According to the above intuitive meanings, the following equivalences hold:

$$\mathcal{F}\alpha \equiv \top\mathcal{U}\alpha, \qquad \mathcal{G}\alpha \equiv \bot\mathcal{R}\alpha, \quad (3)$$

where $\top, \bot$ are the symbols for truth and falsity respectively.

Let h be a history (1), and α a temporal formula. We say that that h *satisfies* α ($h \models^n \alpha$) iff $h \models_0^n \alpha$, where, for any $i \leq n$, the definition of $h \models_i^n \alpha$ is given inductively on the structure of the formulas:

1. $h \models_i^n \alpha$ if $s^i \models \alpha$ and α is a fluent literal,
2. $h \models_i^n (\alpha \wedge \beta)$ if $h \models_i^n \alpha$ and $h \models_i^n \beta$,
3. $h \models_i^n (\alpha \vee \beta)$ if $h \models_i^n \alpha$ or $h \models_i^n \beta$,
4. $h \models_i^n \mathcal{X}_s\alpha$ if $i < n$ and $h \models_{i+1}^n \alpha$,
5. $h \models_i^n \mathcal{X}_w\alpha$ if $i = n$ or $h \models_{i+1}^n \alpha$,
6. $h \models_i^n \mathcal{F}\alpha$ if $\exists j$: $i \leq j \leq n$, $h \models_j^n \alpha$,
7. $h \models_i^n \mathcal{G}\alpha$ if $\forall j$: $i \leq j \leq n$, $h \models_j^n \alpha$,
8. $h \models_i^n \alpha\mathcal{U}\beta$ if $\exists j$: $i \leq j \leq n$, $h \models_j^n \beta$ and $\forall k$: $i \leq k < j$, $h \models_k^n \alpha$,
9. $h \models_i^n \alpha\mathcal{R}\beta$ if $\forall j$: $i \leq j \leq n$, $h \models_j^n \beta$ or $\exists k$: $i \leq k < j$, $h \models_k^n \alpha$.

Given the above semantics, it is easy to see that both (2) and (3) hold.

Temporal logic allows us to specify different interesting requirements on plans. The user can specify a safety property with the formula $\mathcal{G}\alpha$, intuitively standing for "the property α should be maintained", or $\mathcal{G}\neg\beta$, meaning for instance that "a dangerous or

undesired situation β should be avoided". The formula $\mathcal{GF}\alpha$ can be used to specify that "always a state where α holds should be eventually reached", while $\mathcal{FG}\beta$ states that "the system should get to the point where property β can be maintained". As further examples, formulas can be composed in order to specify a sequential ordering in which properties should be verified: $\mathcal{F}(\alpha \wedge \mathcal{F}\beta)$ states that "α should be eventually true before β", while $\mathcal{F}(\alpha \wedge \mathcal{FG}\beta)$ can be used to specify that "α should be reached first, and then β should be reached and maintained".

4 Safe Planning: Plan Validation

We now consider the problem of plan validation with respect to a finite set of properties. Let α be the conjunction of the properties, let $\pi = \langle I, D, G\rangle$ a planning problem, and let $\vec{a} = a^1; \ldots; a^n$ be a valid plan.

We say that $\vec{a}$ is *safe* with respect to α if each history (1) satisfies α. The problem of validating $\vec{a}$ with respect to α can be re-cast as a satisfiability problem. The basic idea is to check whether the execution of the actions in $\vec{a}$ "entails" the property α, reformulated as a propositional formula. In the following, for any number i and formula H, H_i is the expression obtained from H by substituting each atom B with B_i. Intuitively, the subscript i represents time:

- If F is a fluent symbol, the atom F_i expresses that F holds at time i.
- If A is an action symbol, the atom A_i expresses that A is among the elementary actions executed at time i.

The reformulation $[\alpha]^n$ of α as a propositional formula is defined as $[\alpha]^n_0$, and $[\alpha]^n_i$ is:

1. $[\alpha]^n_i$ is α_i, if α is a fluent literal,
2. $[\alpha \wedge \beta]^n_i$ is $[\alpha]^n_i \wedge [\beta]^n_i$,
3. $[\alpha \vee \beta]^n_i$ is $[\alpha]^n_i \vee [\beta]^n_i$,
4. $[\mathcal{X}_s\alpha]^n_i$ is $\bot$ if $i = n$, and is $[\alpha]^n_{i+1}$ otherwise,
5. $[\mathcal{X}_w\alpha]^n_i$ is $\top$ if $i = n$, and is $[\alpha]^n_{i+1}$ otherwise,
6. $[\mathcal{F}\alpha]^n_i$ is $\vee^n_{j=i}[\alpha]^n_j$,
7. $[\mathcal{G}\alpha]^n_i$ is $\wedge^n_{j=i}[\alpha]^n_j$,
8. $[\alpha\mathcal{U}\beta]^n_i$ is $\vee^n_{j=i}([\beta]^n_j \wedge^{j-1}_{k=i} [\alpha]^n_k)$,
9. $[\alpha\mathcal{R}\beta]^n_i$ is $\wedge^n_{j=i}([\beta]^n_j \vee^{j-1}_{k=i} [\alpha]^n_k)$.

For any formula, the correspondence between its semantics and its translation as a propositional formula is immediate. Given the above mapping, and under the assumption that $\vec{a}$ is valid for π, the check that $\vec{a}$ is safe with respect to α amounts to check that

$$I_0 \wedge \wedge_{i=0}^{n-1} a_i^{i+1} \wedge \wedge_{i=0}^{n-1} tr_i^D \models [\alpha]^n. \qquad (4)$$

For example, if α is a safety property $\mathcal{G}\beta$, (4) holds if for any history (1) in which s^0 is an initial state, s^i satisfies β for any $i : 0 \leq i \leq n$.

5 Safe Planning: Plan Generation

Consider a finite set of properties, Let α be their conjunction, and let $\pi = \langle I, D, G\rangle$ a planning problem. Safe planning is the task of finding a valid plan which is also safe with respect to α. As before, we do not make any assumption about D: it can be deterministic or nondeterministic, it can allow for the concurrent execution of actions or not. However, for simplicity we assume that I is satisfied by at most one state.

The simplest approach for the generation of safe plans, is to generate (possible) plans, and test whether each generated plan is valid and safe. In this way, we obtain

1. a correct but possibly incomplete safe procedure in general, and
2. a correct and complete safe procedure if we generate and test all the possible plans.

Given a planning problem π and a natural number n, we say that a procedure for safe planning is

- *correct* (for π, n) if any returned plan $\alpha_1; \ldots; \alpha_n$ is valid and safe for π, and
- *complete* (for π, n) if it returns *False* when there is no valid and safe plan $\alpha_1; \ldots; \alpha_n$ for π.

In the planning as satisfiability framework, all the possible plans can be generated by enumerating the assignments satisfying

$$I_0 \wedge \wedge_{i=0}^{n-1} tr_i^D \wedge G_n. \quad (5)$$

The check whether a plan $\vec{a} = a^1, \ldots, a^n$ is safe — assuming we already know is valid— can be done as described in Section 4. Indeed, if I is satisfied by only one state (as we assumed at the beginning of the Section) and D is deterministic, any possible plan is also valid, and thus we are done: we can easily devise a procedure that generates valid plans till one which is also safe is found.

However, as described in [16], the check whether $\vec{a}$ is valid is complicated in the case that D is non deterministic. In fact, the check that a possible plan is also valid amounts to perform an entailment check similar to the one in (4). More in detail, a plan $a^1; \ldots; a^n$ is valid iff

$$I_0 \wedge \neg Z_0 \wedge \wedge_{i=0}^{n-1} a_i^{i+1} \wedge \wedge_{i=0}^{n-1} trt_i^D \models G_n \wedge \neg Z_n, \quad (6)$$

where trt_i^D is defined on the basis of tr_i^D, and Z is a newly introduced fluent symbol, see [16]. Thus, given what we said so far, in order to have a correct and complete procedure for safe planning we should:

- generate (all) the possible plans by satisfying (5),
- test if each generated plan $a^1; \ldots; a^n$ is valid by checking whether (6) holds, and
- if $a^1; \ldots; a^n$ is valid, check whether it is also safe with respect to a property α, by checking whether (4) holds.

This is not necessary. Indeed, the last two checks can be combined into one: A (possible) plan $a^1; \ldots; a^n$ is both valid for π and safe with respect to α iff

$$I_0 \wedge \neg Z_0 \wedge \wedge_{i=0}^{n-1} a_i^{i+1} \wedge \wedge_{i=0}^{n-1} trt_i^D \models G_n \wedge \neg Z_n \wedge [\alpha]^n.$$

Furthermore, as an optimization, if in the generation phase we consider only the plans corresponding to the assignments satisfying

$$I_0 \wedge \wedge_{i=0}^{n-1} tr_i^D \wedge G_n \wedge [\alpha]^n \quad (7)$$

we still get a correct and complete procedure for safe planning: The assignment satisfying (7) and not (5) for sure do not correspond to safe plans.

6 Conclusions and Related Work

In this paper we have extended the planning as satisfiability approach to the case of "safe planning", i.e. to the generation and validation of plans that have to guarantee both the reachability of a goal state and the fact that user defined temporal properties hold. We allow domains to be non-deterministic, as several space applications require. In non-deterministic domains, a plan may result in several different possible executions. Safe planning provides the ability of generating plans and validating them against all the possibly different executions. This work is a starting point in the SACSO project, a project sponsored by ASI. The aim of SACSO is the development of techniques and tools for the support of autonomy and safety in space applications.

Within the SACSO project, we are also addressing the problem of safe planning by applying symbolic model checking techniques [17] based on BDDs [8], an alternative approach to planning as satisfiability. The approach is based on the idea that plans can be generated and validated by searching through sets of states compactly represented by BDDs. This idea has been first introduced in [5] (see also [6] for an introduction), and then extended to generate valid plans for reachability goals in non-deterministic domains, see, e.g., [18, 19]. In SACSO, we plan to exploit the work presented in [20], and then extended in [21] to exploit symbolic techniques. In [20, 21] safe plans can be specified with formulas in the CTL temporal logic [22]. CTL allows the user to distinguish between requirements that should hold on all the possible non-deterministic plan executions, and

others that may hold only on some executions. The plans that are generated and validated can include conditionals and iterations. The work has been implemented in the MBP planner [23].

Within the planning literature, most of the works on planning for "temporally extended goals" restricts to deterministic domains, see for instance [24, 25]. Deductive planning can be considered as the first attempt to merge planning and formal verification techniques. In this case, theorem proving can be used to generate (and validate) plans that satisfy temporal goals, and planning domains can be non-deterministic, see, e.g., [26]. However, the practical automatic generation of plans by deductive planning is still an open problem. SimPlan [27] generates plans for LTL-like goals in non-deterministic domains by searching through an explicit representation of the state-space. The use of symbolic model checking techniques, based either on SAT or on BDDs, opens up the possibility to deal in practice with large state spaces that are very hard to be tackled with explicit state techniques.

References

[1] N. Muscettola, P. Nayak, B. Pell, and B. Williams. Remote Agent: To Boldly Go Where No AI System Has Gone Before. *Artificial Intelligence*, 103:5–48, 1998.

[2] B. Smith, M.S. Feather, and N. Muscettola. Challenges and methods in testing the remote agent planner. In *Proc. 5th Int.nl Conf. o Artificial Intelligence Planning and Scheduling (AIPS 2000)*, 2000.

[3] E. Clarke, O. Grumberg, and D. Long. Model checking. *Proc. Int'l Summer School on Deductive Program Design*, 1994.

[4] J. R. Burch, E.M. Clarke, D.E. Long, K.E. McMillan, and D.L. Dill. Symbolic model checking for sequential circuit verification. *IEEE Transactions on Computer-Aided Design of Integrated Circuits*, 13(4):401–424, April 1994.

[5] A. Cimatti, E. Giunchiglia, F. Giunchiglia, and P. Traverso. Planning via model checking: a decision procedure for AR. In *Lecture Notes in Computer Science*, volume 1348, 1997.

[6] Fausto Giunchiglia and Paolo Traverso. Planning as model checking. In *Lecture Notes in Computer Science, Proc. of the 5th European Conference on Planning (ECP-99)*. Springer, September 1999.

[7] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *Proceedings of the Fifth International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS '99)*, 1999.

[8] Randal E. Bryant. Symbolic Boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys*, 24(3):293–318, September 1992.

[9] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *Journal of the ACM*, 5(7), 1962.

[10] Henry Kautz and Bart Selman. Planning as satisfiability. In *Proc. ECAI-92*, pages 359–363, 1992.

[11] P. Traverso A. Cesta, E. Giunchiglia. Interactive autonomy for space applications. In *Proc. Intl. Workshop on Planning and Scheduling for Space*, March 2000.

[12] Enrico Giunchiglia and Vladimir Lifschitz. Dependent fluents. In *Proc. IJCAI-95*, pages 1964–1969, 1995.

[13] David Smith and Daniel Weld. Conformant graphplan. In *Proc. AAAI-98*, pages 889–896, 1998.

[14] Amir Pnueli. The temporal logic of programs. In *Proc. 18th Ann. IEEE Symposium on Foundations of Computer Science*, pages 46–57, 1977.

[15] Allen Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *The Handbook of Theoretical Computer Science*, pages 997–1072. Elsevier Science Publishers, 1990.

[16] Enrico Giunchiglia. Planning as satisfiability with expressive action languages: Concurrency, constraints and nondeterminism. In *Seventh International Conference on Principles of Knowledge Representation and Reasoning (KR'00)*, 2000.

[17] K.L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publ., 1993.

[18] A. Cimatti, M. Roveri, and P. Traverso. Automatic OBDD-based Generation of Universal Plans in Non-Deterministic Domains. In *Proceeding of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, Madison, Wisconsin, 1998. AAAI-Press. Also IRST-Technical Report 9801-10, Trento, Italy.

[19] P. Bertoli, A. Cimatti, M. Roveri, and P. Traverso. Planning in Nondeterministic Domains under Partial Observability via Symbolic Model Checking. In *Proc.* 7^{th} *International Joint Conference on Artificial Intelligence (IJCAI-01)*. AAAI Press, August 2001.

[20] M. Pistore and P. Traverso. Planning as Model Checking for Extended Goals in Nondeterministic Domains. In *Proc.* 7^{th} *International Joint Conference on Artificial Intelligence (IJCAI-01)*. AAAI Press, August 2001.

[21] M. Pistore, R. Bettin, and P. Traverso. Symbolic Techniques for Extended Goals in Nondeterministic Domains. Technical Report 01-1328, IRST, 2001.

[22] E. A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*, chapter 16, pages 995–1072. Elsevier, 1990.

[23] P. Bertoli, A. Cimatti, M. Pistore, M. Roveri, and P. Traverso. MBP: A Model Based Planner. Technical Report 01-1101, IRST, 2001.

[24] G. de Giacomo and M.Y. Vardi. Automata-theoretic approach to planning with temporally extended goals. In *Proc. of ECP99*, 1999.

[25] F. Bacchus and F. Kabanza. Using temporal logic to express search control knowledge for planning. *J. of Artificial Intelligence*, 1998. Submitted for pubblication.

[26] W. Stephan and S. Biundo. A New Logical Framework for Deductive Planning. In *Proc. of IJCAI93*, pages 32–38, 1993.

[27] F. Kabanza, M. Barbeau, and R. St-Denis. Planning control rules for reactive agents. *Artificial Intelligence*, 95(1):67–113, 1997.