Solving High-Level Planning Programs (Extended Abstract)

Giuseppe De Giacomo and Fabio Patrizi
Dipartimento di Informatica e Sistemistica
Sapienza Università di Roma
Rome, Italy.
{degiacomo,patrizi}@dis.uniroma1.it

Sebastian Sardina
School of CS and IT
RMIT University
Melbourne, Australia
sebastian.sardina@rmit.edu.au

Introduction

In this work, we consider a middle ground between automated planning (Ghallab, Nau, and Traverso 2004; Nau 2007; Green 1969; Weld 1999) and agent-oriented high-level programming (Shoham 1993; Lespérance et al. 1995; Levesque and Reiter 1998; Rao 1996). Specifically, we propose a framework for high-level programming of autonomous intelligent agents using pure declarative goals.

Automated planning allows the specification of behavior in a *declarative* manner, thus providing an abstract, flexible, and powerful mechanism that caters for *flexible* behavior: any conceivable way of achieving the desired outcome may be constructed (from first-principle). On the other hand, agent-oriented programming accommodates useful "*know-how*" domain knowledge encoding the typical operations of the domain of concern since agent systems generally "act as they go".

Interestingly, the advantages of each of the two approaches are the weaknesses of the other. Lookahead planning is intrinsically difficult computationally since plans are built from first-principle, and it is not tailored for long-term behavior in changing domains, where the actual behavior depends on contingencies. Similarly, agent-oriented approaches typically rely entirely on procedural knowledge that ought to be crafted at design time and upon which the ultimate behavior of the system shall depend entirely: no "new" plans can be generated.

In this paper, we propose a novel account that mixes programming with planning, thus leveraging on the advantages of both approaches. In concrete, we assume a typical planning domain $\mathcal{D}$ describing the dynamics of the world and a so-called *planning target program* $\mathcal{T}$ to be realized in $\mathcal{D}$. Target $\mathcal{T}$ is a basically a high-level program composed of goals, both achievement and maintenance ones. Technically, the target is a transition system in which states specify *choice points* and transitions specify a pair of *maintenance* and *achievement* goals to be chosen by the agent and realized next. At any point in time, the external world and the target agent are in some of their respective states, and the agent decides, autonomously, which goal to achieve next in order to be in its following desire state (e.g., be at the airport). At the same time, it specifies a goal that has to be maintained while achieving the latter (e.g., always keep the ticket and enough money).

A course of actions is then synthesized for that end, while respecting the corresponding maintenance. Once its request is fulfilled, the agent evolves to its next state, where he can choose other pairs of goals to realize (e.g., do the check-in while keeping the cell phone handy). Observe that while the target itself needs to be programmed from declarative goals representing the different stages in the life of the agent, each step within the target is indeed a planning task. The problem then is: *can the target planning program be realized fully in the dynamic domain?* That is, *can each of the possible options that the target agent may autonomously choose be guaranteed always in the environment of concern?*

In the rest of the paper, we start by formally defining the problem described above, by relying on transition systems and the formal notion of simulation. We then briefly review a recent LTL-based synthesis technique developed in the literature on Verification (Pettersson 2005). Finally, we develop a sound and complete technique to synthesize a solution to realizing the target planning program, by reducing the planning problem into that of of LTL realizability.

The Framework

Our framework consists of two main ingredients: *(i)* a *dynamic domain*, formalizing the environment that the agent acts in, and *(ii)* a *target planning program*, providing a high-level representation of the desired domain evolutions.

The *dynamic domain* is a usual *deterministic* planning domain, defined as a tuple $\mathcal{D} = \langle P, 2^P, A, S_0, \rho \rangle$, where:[1]

- $P = \{p_1, \ldots, p_n\}$ is a finite set of *domain propositions*;
- 2^P is the set of *domain states*;
- $A = \{a_1, \ldots, a_r\}$ is the finite set of *domain actions*;
- $S_0 \in 2^P$ is the *initial state*;
- $\rho : 2^P \times A \rightarrow 2^P$ is the (partial) *transition function*. We freely interchange notations *(i)* $\rho(S, a) = S'$; *(ii)* $\langle S, a, S' \rangle \in \rho$; and *(iii)* $S \xrightarrow{a} S'$.

The dynamic domain is also referred to as *environment*.

[1] We do not deal with non-deterministic domains for simplicity of exposition, but all the results extend directly to domains with non-deterministic actions.

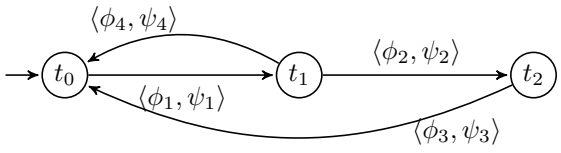

Figure 1: An abstract target planning program. A label $\langle\phi,\psi\rangle$ in an edge represents the achievement goal ϕ and the maintenance goal ψ for the corresponding transition.

Given $\mathcal{D}$, a *plan* $\eta = a^1 \cdots a^v$ is a sequence of actions from A. Given a state $S \in 2^P$, a plan $\eta = a^1 \cdots a^v$ is *executable* in S if there are states $S^0, \ldots, S^v$ such that $S^0 = S$ and $S^i \xrightarrow{a^{i+1}} S^{i+1}$ in $\mathcal{D}$ for $i = 0, \ldots, v-1$. A plan $\eta = a^1 \cdots a^v$ executable in S generates an *induced ($\mathcal{D}$-)history* $h_{\eta,S} = S^0 \xrightarrow{a^1} S^1 \cdots S^{v-1} \xrightarrow{a^v} S^v$, representing the domain evolution that η yields on $\mathcal{D}$, when executed from S.

A *target planning program* is a high-level specification of the behavior desired for the dynamic domain, described by formulae that domain executions are required to satisfy.

Definition 1 (Target Planning Program). A *target planning program*, or simply a *target program*, for a dynamic domain $\mathcal{D}$ is a tuple $\mathcal{T} = \langle T, \Psi, \Phi, t_0, \delta\rangle$, where:

- $T = \{t_0, \ldots, t_q\}$ is the finite set of *program states*;
- $\Psi = \{\psi_1, \ldots, \psi_m\}$ is a set of *maintenance* goal propositional formulae over P;
- $\Phi = \{\phi_1, \ldots, \phi_t\}$ is the set of *achievement* goal propositional formulae over P;
- $t_0 \in T$ is the *program initial state*;
- $\delta : T \times \Psi \times \Phi \to T$ is the *transition function* such that for each $t \in T$ there exists at least one $t' \in T$, $\psi \in \Psi$ and $\phi \in \Phi$ such that $\delta(t, \psi, \phi) = t'$. We also use notations $\langle t, \psi, \phi, t'\rangle \in \delta$ and $t \xrightarrow{\psi/\phi} t'$.

Target programs (whose transitions are *deterministic* for given t, ψ, ϕ) provide a description of the high-level requests, i.e., maintenance and achievement goals over P, that a client might request for execution on the dynamic domain, at each point in time. The client can, in fact, be the agent executing the target program itself, which generates requests, according to its *intentions*, following the target program. An abstract target program is depicted in Figure 1.

Intuitively, when the target program is realized, a typical session is as follows: initially, $\mathcal{D}$ and $\mathcal{T}$ are in their respective initial states; from current state t, the client requests a transition $t \xrightarrow{\psi/\phi} t'$; as a result, a plan from S is executed which *(i)* eventually leads the domain to a state S_g s.t. $S_g \models \phi$, while *(ii)* constraining $\mathcal{D}$ to traverse only states satisfying ψ; upon plan completion, the program moves to t'; the client selects a new transition $t' \xrightarrow{\psi'/\phi'} t''$ in $\mathcal{T}$ and a new iteration takes place. Notice that at any point in time all choices available to the client in $\mathcal{T}$ must be guaranteed. Next, we formalize target planning program semantics.

Definition 2 (Plan-based simulation relation). Let $\mathcal{D}$ be a dynamic domain and $\mathcal{T}$ a target planning program. A *plan-based simulation relation*, or *PLAN-simulation relation*, is a relation $\preceq_{PLAN} \subseteq T \times 2^P$ such that if $\langle t, S\rangle \in \preceq_{PLAN}$ (written $t \preceq_{PLAN} S$) then, for each transition $t \xrightarrow{\psi/\phi} t'$ in $\mathcal{T}$ there exists a plan $a^1 \cdots a^v$ from S, inducing a $\mathcal{D}$-history $h : S^0 \xrightarrow{a^1} \cdots \xrightarrow{a^v} S^v$, with $S^0 = S$, such that: *(i)* $S^v \models \phi$; *(ii)* $S^i \models \psi$ for $i = 1, \ldots, v$; and *(iii)* $t' \preceq_{PLAN} S^v$. ■

Observe the strong similarity of this definition with the formal notion of simulation relation (Milner 1971): PLAN-simulation relations can be seen as kinds of high-level simulation relations.

A target program state $t \in T$ is *plan-simulated* by a $\mathcal{D}$ state $S \in 2^P$, denoted $t \preceq_{PLAN} S$, if there exists a PLAN-simulation relation R s.t. $\langle t, S\rangle \in R$. Clearly, $\preceq_{PLAN}$ is a PLAN-simulation relation itself and, in particular, the largest one.

A target program $\mathcal{T}$ is *realizable* by a dynamic domain $\mathcal{D}$ if $t_0 \preceq_{PLAN} S_0$. When this happens, we want to build a *plan generator*, defined as a function $g : T \times 2^P \times \Psi \times \Phi \to A^*$ which, given *(i)* a current domain state $S \in 2^P$, *(ii)* a current target program state $t \in T$, *(iii)* a requested achievement goal $\phi \in \Phi$ and *(iv)* a requested maintenance goal $\psi \in \Psi$, returns a plan $\eta = g(t, S, \psi, \phi) = a^1 \cdots a^v$ executable in S, which is able to lead $\mathcal{D}$ to a state S_g such that $S_g \models \phi$ while maintaining ψ satisfied all along induced history $h_{\eta,S}$ and such that all possible requests issued next can be fulfilled. The problem we face here is: *how such function can be built?*

Reactive Synthesis in LTL

Linear Temporal Logic (LTL) is a well-known logic used to specify dynamic or temporal properties of programs, see e.g., (Vardi 1996). *Formulas* of LTL are built from a set $\mathcal{P}$ of atomic propositions and are closed under the boolean operators, the unary temporal operators $\bigcirc$ (*next*), $\Diamond$ (*eventually*), and $\Box$ (*always*, from now on), and the binary temporal operator *until* (which in fact can be used to express both $\bigcirc$ and $\Box$, though it will not be used here). LTL formulas are interpreted over infinite sequences π of propositional interpretations for $\mathcal{P}$, i.e., $\pi \in (2^{\mathcal{P}})^\omega$. If π is an interpretation, i a natural number, and μ a propositional formula, we denote by $\pi, i \models \mu$ the fact that μ is true in the i-th propositional interpretation of π. Such interpretation is extended to the temporal operators as follows (we omit *until* for brevity).

$\pi, i \models \bigcirc\mu$ iff $\pi, i{+}1 \models \mu$;
$\pi, i \models \Diamond\mu$ iff for some $j \geq i$, we have that $\pi, j \models \mu$;
$\pi, i \models \Box\mu$ iff for all $j \geq i$, we have that $\pi, j \models \mu$.

An interpretation π satisfies μ, written $\pi \models \mu$, if $\pi, 0 \models \mu$. Standard logical tasks such as satisfiability or validity are defined as usual, e.g., a formula μ is *satisfiable* if there exists an interpretation that satisfies it. Checking satisfiability or validity for LTL is PSPACE-complete.

Here we are interested in a different kind of logical task, which is called *realizability*, or *Church problem*, or simply *synthesis* (Vardi 1996; Pnueli and Rosner 1989). Namely, we partition $\mathcal{P}$ into two disjoint sets $\mathcal{X}$ and $\mathcal{Y}$. We assume to have *no control* on the truth value of the propositions in $\mathcal{X}$, while we can control those in $\mathcal{Y}$. The problem then is: *can we control the values of $\mathcal{Y}$ such that for*

all possible values of $\mathcal{X}$ a certain LTL formula remains true? More precisely, interpretations now assume the form $\pi = (X_0, Y_0)(X_1, Y_1)(X_2, Y_2)\cdots$, where (X_i, Y_i) is the propositional interpretation at the i-th position in π, now partitioned in the propositional interpretation X_i for $\mathcal{X}$ and Y_i for $\mathcal{Y}$. Let us denote by $\pi_{\mathcal{X}}|_i$ the interpretation π projected only on $\mathcal{X}$ and truncated at the i-th element (included), i.e., $\pi_{\mathcal{X}}|_i = X_0 X_1 \cdots X_i$. The *realizability problem* checks the existence of a function $f : (2^{\mathcal{X}})^* \rightarrow 2^{\mathcal{Y}}$ such that for all π with $Y_i = f(\pi_{\mathcal{X}}|_i)$ we have that π satisfies the formula μ. The *synthesis problem* consists in actually computing such a function. Observe that in realizability/synthesis we have no way of constraining the value assumed by the propositions in $\mathcal{X}$: the function we are looking for only acts on propositions in $\mathcal{Y}$. This means that the most interesting formulas for the synthesis have the form $\varphi_a \rightarrow \varphi_r$, where φ_a captures the "relevant" assignments of the propositions in $\mathcal{X}$ (and $\mathcal{Y}$) and φ_r specifies the property we want to assure for such relevant assignments. The realizability (and actual synthesis) are 2EXPTIME-complete for arbitrary LTL formulas (Pnueli and Rosner 1989). However, recently, several well-behaved patterns of LTL formulas have been identified, for which efficient procedures based on model checking technologies applied to game structures can be devised. Here, we shall focus on one of the most general well-behaved patterns, called "*Generalized Reactivity (1)*" or *GR(1)* (Piterman, Pnueli, and Sa'ar 2006). Such formulas have the form $\varphi_a \rightarrow \varphi_r$, with φ and ψ of the following shape

$$\varphi_a\text{: } \mu[\mathcal{X}, \mathcal{Y}] \wedge \bigwedge_j \Box\mu_j[\mathcal{X}, \mathcal{Y}, \bigcirc\mu[\mathcal{X}]] \wedge \bigwedge_k \Box\Diamond\mu_k[\mathcal{X}, \mathcal{Y}],$$
$$\varphi_r\text{: } \mu[\mathcal{X}, \mathcal{Y}] \wedge \bigwedge_j \Box\mu_j[\mathcal{X}, \mathcal{Y}, \bigcirc\mu[\mathcal{X}, \mathcal{Y}]] \wedge \bigwedge_k \Box\Diamond\mu_k[\mathcal{X}, \mathcal{Y}],$$

where $\mu[\mathcal{Z}]$ stands for any boolean combination of symbols from $\mathcal{Z}$. Notice that: *(i)* with the first conjunct we can express initial conditions; *(ii)* with the second (big) conjunct we can express transitions —and we have the further constraint that in doing so within φ_a we cannot talk about the next value of the propositions in $\mathcal{Y}$; and *(iii)* with the third (big) conjunct we can express *fairness* conditions of the form "it is always true that eventually something holds." For such formulas we have the following result.

Theorem 1 (Piterman, Pnueli, and Sa'ar 2006). *Realizability (and synthesis) of GR(1) LTL formulas $\varphi_a \rightarrow \varphi_r$ can be determined in time $O((p * q * w)^3)$, where p and q are the number of conjuncts of the form $\Box\Diamond\mu$ in φ_a and φ_r, respectively,[2] and w is the number of possible value assignments of $\mathcal{X}$ and $\mathcal{Y}$ under the conditions of $\varphi_a \rightarrow \varphi_r$.*

Solving Planning Programs

We now show how computing a plan generator (PG) can be reduced to realizability (and synthesis) of a GR(1) LTL formula Υ. The reader should keep in mind that, although the reduction can be informally understood as a set of constraints on the strategy to get the solution, its formal justification is simply Theorem 2, stating its soundness and completeness.

The intuition behind the reduction is as follows. At some point in time, the target program $\mathcal{T}$ and environment $\mathcal{D}$ are in one of their states, say t and S, respectively. $\mathcal{T}$ requests a transition $t \xrightarrow{\psi/\phi} t'$ to be realized. The PG then builds a plan η executable in S. In selecting each action, the PG is to satisfy two constraints. First, when the plan is executed in S, maintenance goal ψ may not be violated. Second, upon execution completion, $\mathcal{T}$ moves to state t' and $\mathcal{D}$ must be in a state S' such that: *(i)* ϕ holds and *(ii)* for all transitions outgoing from t' (i.e., all possible $\mathcal{T}$ next requests), a new plan exists which satisfies above constraints.

We start building the GR(1) LTL formula $\Upsilon = \varphi_a \rightarrow \varphi_r$ by specifying the sets of uncontrolled and controlled propositions $\mathcal{X}$ and $\mathcal{Y}$, and then build assumption formula φ_a and requirement formula φ_r.

Uncontrolled and controlled propositions The set of *uncontrolled* propositions $\mathcal{X}$ is the union of sets: *(i)* $P = \{p_1, \ldots, p_n\}$ ($\mathcal{D}$ propositions); *(ii)* $P_T = \{pt_0, \ldots, pt_q\}$ (one proposition for each target's state, where pt_i denotes that $\mathcal{T}$ is in state t_i); *(iii)* $P_\delta = \{p\delta_{\psi,\phi} \mid \langle t, \psi, \phi, t'\rangle \in \delta\}$ (one proposition for each target's transition, where $p\delta_{\psi,\phi}$ states that $\mathcal{T}$ is asking for the achievement of goal ϕ while maintaining goal ψ).

The set of *controlled* propositions $\mathcal{Y}$ contains set $P_A = \{pa_1, \ldots, pa_r\}$ (one proposition for to actions a_i in A, where pa states that action a is to be executed next) plus proposition $last$ (stating that last action of current plan is to be executed next).

Assumption formula Next, we build a formula $\varphi_a = \varphi^a_{init} \wedge \varphi^a_{trans}$ capturing the *assumptions* on the overall framework the plan generator is acting on. For legibility, we define some syntactic shortcuts:

- for each $\mathcal{D}$ state $S \in 2^P$ we define a propositional formula $\varphi_S = \bigwedge_{i=1}^{n} l_i$, where $l_i = p_i$ if $p_i \in S$; and $l_i = \neg p_i$ otherwise;
- for each target state $t \in T$, we define a propositional formula $req_t = \bigvee_{\langle t,\psi,\phi,t'\rangle \in \delta} p\delta_{\psi,\phi}$, representing the fact that the target is requesting at least one pair of maintenance and achievement goals available in state t.

The assumption formula is meant to encode how the overall system is *expected* to behave; technically, it encodes the synchronous execution of dynamic domain $\mathcal{D}$ and target specification $\mathcal{T}$.

Propositional formula $\varphi^a_{init} = \varphi_{S_0} \wedge pt_0$ characterizes the (legal) initial states of the overall system, by requiring $\mathcal{D}$ and $\mathcal{T}$ to start in their respective initial states. Note no constraint on proposition $last$ nor on any proposition in P_δ are imposed here.

LTL formula $\varphi^a_{trans} = \Box trans_{\mathcal{D}} \wedge \Box trans_{\mathcal{T}}$ characterizes the assumptions on the overall system evolution. Specifically, $trans_{\mathcal{D}}$ defines the "rules" for the domain and $trans_{\mathcal{T}}$ defines those for the target. The former is defined as follows:

$$trans_{\mathcal{D}} = \bigwedge_{\langle S,a,S'\rangle \in \rho} [\varphi_S \wedge pa \rightarrow \bigcirc\varphi_{S'}]$$

where each conjunct states that if the world is in state S and action a is to be executed, then S' is the *next* state of the world.

Formula $trans_{\mathcal{T}}$, in turn, is built as conjunction of the following formulae:

[2] We assume that both φ_a and φ_r contain at least one conjunct of such a form, if not, we vacuously add the trivial one $\Box\Diamond\top$.

- $\bigvee_{pt \in P_T}[pt \wedge \bigwedge_{pt' \in P_T \setminus \{pt\}} \neg pt']$, that is, the target is in exactly one of its states.
- $\bigwedge_{t_i \in T}[pt_i \rightarrow req_{t_i}]$, that is, in each state, the target ought to be requesting some of the possible goals available in current state;
- $\bigwedge_{p\delta_{\psi,\phi}, p\delta_{\psi',\phi'} \in P_\delta, p\delta_{\psi,\phi} \neq p\delta_{\psi',\phi'}}[p\delta_{\psi,\phi} \rightarrow \neg p\delta_{\psi',\phi'}]$, that is, at most one goal request is allowed;
- $\bigwedge_{\langle t_i,\psi,\phi,t_j\rangle \in \delta}[pt_i \wedge p\delta_{\psi,\phi} \wedge last \rightarrow \bigcirc pt_j]$, that is, if transition $t_i \xrightarrow{\psi,\phi} t_j$ is currently being requested and last action of current plan is to be executed next, then the target shall move next to successor state t';
- $\bigwedge_{t_i \in T, \langle t_i,\psi,\phi,t_j\rangle \in \delta}[(pt_i \wedge p\delta_{\psi,\phi} \wedge \neg last) \rightarrow \bigcirc pt_i]$, that is, the target remains still if the current plan has not yet been completed.
- $\bigwedge_{t_i \in T, \langle t_i,\psi,\phi,t_j\rangle \in \delta}[(pt_i \wedge p\delta_{\psi,\phi} \wedge \neg last) \rightarrow \bigcirc p\delta_{\psi,\phi}]$, that is, the target remains requesting the same transition if the current plan has not yet ben completed.

Requirement Formula Let us now build formula $\varphi_r = \varphi^r_{trans} \wedge \varphi^r_{goal}$, which captures the *requirements* for the module to be synthesized, i.e., the plan generator: an automaton which, at each step, selects an action for execution.

LTL formula $\varphi^r_{trans} = \Box(\varphi^{act}_{trans} \wedge \varphi^{last}_{trans} \wedge \varphi^{maint}_{trans})$ encodes constraints on action executions and how target agents are "fulfilled". Namely:

- $\varphi^{act}_{trans} = \bigvee_{pa \in P_A}[pa \wedge \bigwedge_{pa' \in P_A, pa' \neq pa} \neg pa']$, i.e., one and only one domain action is expected to be executed at each step;
- $\varphi^{last}_{trans} = \bigwedge_{p\delta_{\psi,\phi} \in P_\delta}[p\delta_{\psi,\phi} \wedge last \rightarrow \bigcirc \phi]$, i.e., upon plan completion, requested goal ϕ is indeed achieved.
- $\varphi^{maint}_{trans} = \Box \bigwedge_{p\delta_{\psi,\phi} \in P_\delta}[p\delta_{\psi,\phi} \rightarrow \psi]$, i.e., maintenance goals are respected along plans' executions.

Finally, by using simply one *fairness* conjunct, we are able to encode the synthesis objective, i.e., the realization of the achievement goals and preservation of maintenance ones. Formally, we have:

$$\varphi^r_{goal} = \Box \diamond last$$

That is, we require that each plan is *always eventually completed*. This implies, in turn, that all requested goals are (always eventually) fulfilled.

It is not hard to check that the LTL formula Υ obtained is indeed in GR(1) format. Hence, the results from (Piterman, Pnueli, and Sa'ar 2006) are directly available and we are able to prove our main result:

Theorem 2 (Soundness & Completeness). *There exists a solution to the target planning program $\mathcal{T}$ in the dynamic domain $\mathcal{D}$ iff the LTL formula Υ, constructed as above, is realizable.*

That is, checking the realizability of Υ is a sound and complete technique for solving the target planning program in the domain of concern. We stress that by solving realizability with the techniques in (Piterman, Pnueli, and Sa'ar 2006) we do get an actual solution for the realization of the planning program, not merely verify its existence.

Analyzing the structure of Υ, we get that: *(i)* φ_a contains no subformulas of the form $\Box\Diamond\mu$; *(ii)* φ_r contains just one such subformulas; *(iii)* the number of possible value assignments of $\mathcal{X}$ and $\mathcal{Y}$ under the conditions of $\varphi_a \rightarrow \varphi_r$ is $O(|2^P| * |T| * |\Psi| * |\Phi|)$ (observe that variables that represent states in a TS are pairwise disjoint). Consequently, from Theorem 1, we get:

Theorem 3 (Complexity upperbound). *Checking the existence of a solution for a target planning program in a dynamic domain can be done in $O((|2^P| * |T| * |\Psi| * |\Phi|)^3)$.*

Such bound can be refined by replacing 2^P with the number of environment's states that are actually reachable from the initial state.

Conclusion

This work combines automated planning and high-level agent-oriented programming into the novel problem of synthesizing a high-level planning program for execution on a planning domain. We provided a solution by resorting on synthesis for a well-behaved class of LTL formulas, which constitutes one of the most general classes in which LTL synthesis can be reduced to model-checking of game structures. This allows us to leverage on existing results and tools (including TLV[3] and Anzu[4]) and leaves us room for interesting extensions (e.g., explicit fairness assumptions on the dynamic domain).

References

Ghallab, M.; Nau, D. S.; and Traverso, P. 2004. *Automated Planning: Theory and Practice*. Morgan Kaufmann.

Green, C. C. 1969. Theorem proving by resolution as a basis for question-answering systems. *Machine Intelligence* 4:183–205.

Lespérance, Y.; Levesque, H. J.; Lin, F.; Marcu, D.; Reiter, R.; and Scherl, R. B. 1995. Foundations of a logical approach to agent programming. In *Proc. of ATAL'95*.

Levesque, H. J., and Reiter, R. 1998. High-level robotic control: Beyond planning. A position paper. In *AIII 1998 Spring Symposium: Integrating Robotics Research: Taking the Next Big Leap*.

Milner, R. 1971. An algebraic definition of simulation between programs. In *Proc. of IJCAI 1971*, 481–489.

Nau, D. S. 2007. Current trends in automated planning. *AI Magazine* 28(4):43–58.

Pettersson, O. 2005. Execution monitoring in robotics: A survey. *Robotics and Autonomous Systems* 53(2):73–88.

Piterman, N.; Pnueli, A.; and Sa'ar, Y. 2006. Synthesis of Reactive(1) Designs. In *VMCAI*, 364–380.

Pnueli, A., and Rosner, R. 1989. On the Synthesis of a Reactive Module. In *Proc. of POPL 1989*, 179–190.

Rao, A. S. 1996. Agentspeak(L): BDI agents speak out in a logical computable language. In *Proc. of MAAMAW'96*, 42–55.

Shoham, Y. 1993. Agent-oriented programming. *Artificial Intelligence Journal* 60:51–92.

Vardi, M. Y. 1996. An automata-theoretic approach to linear temporal logic. In *Logics for Concurrency: Structure versus Automata*, vol. 1043 of *LNCS*. Springer. 238–266.

Weld, D. S. 1999. Recent advances in AI planning. *AI Magazine* 20(2):93–123.

[3] www.cs.nyu.edu/acsys/tlv/

[4] www.ist.tugraz.at/staff/jobstmann/anzu/