An Effective Approach to Realizing Planning Programs*

Alfonso E. Gerevini† and **Fabio Patrizi**‡ and **Alessandro Saetti**†

†Dipartimento di Ingegneria dell'Informazione, Università degli Studi di Brescia, Brescia, Italy
{gerevini,saetti}@ing.unibs.it
‡Department of Computing, Imperial College London, UK
fpatrizi@imperial.ac.uk

Abstract

Planning programs are loose, high-level, declarative representations of the behavior of agents acting in a domain and following a path of goals to achieve. Such programs are specified through transition systems that can include cycles and decisions to make at certain points. We investigate a new effective approach for solving the problem of realizing a planning program, i.e., informally, for finding and combining a collection of plans that guarantee the planning program executability. We focus on deterministic domains and propose a general algorithm that solves the problem exploiting a planning technique handling goal constraints and preferences. A preliminary experimental analysis indicates that our approach dramatically outperforms the existing method based on formal verification and synthesis techniques.

Introduction

Planning programs (*p-program*s, for short) are loose, high-level, declarative representations of the behavior of agents acting in a domain (De Giacomo, Patrizi, & Sardina 2010). Technically, they are transition systems, with transitions labelled by goals over the domain, and states representing decision points (the executor chooses the transition to execute).

Informally, in order for a *p*-program to be executable by the involved agent(s), each labeling goal requires a plan achieving it. These plans must be synchronized so that the final world state generated by a plan is a suitable initial state for the plans associated with the next possible goals to achieve. When this is the case, the *p*-program is *realized*. In general, however, computing a realization does not just amount to associating transition goals with appropriate plans. Indeed, as plans are executed, both *p*-program's and underlying domain's states progress, so, in general, the *p*-program can reach a state v many times, possibly with different domain states, say $s' \neq s''$. Clearly, there is no guarantee for the same plan to be executable in both s' and s'', and so in particular for the plan labeling a same transition outgoing from v. Therefore, a *p*-program is realized by a function, called *realization*, taking in input a *p*-program transition *and* a domain state, and returning an appropriate plan to execute.

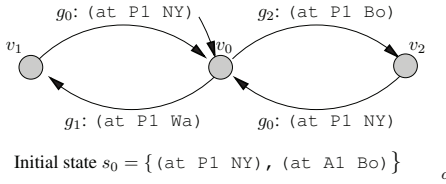
*Work partially supported by EU Programme FP7/2007-2013, under grant agreement 257593 (ACSI).

A solution technique to compute realizations was proposed by De Giacomo, Patrizi, & Sardina (2010), based on using TLV (Pnueli & Shahar 1996), a tool for synthesis of LTL specifications (Pnueli & Rosner 1989). However, our experiments show that this approach performs poorly in practice.

Here, we address the problem of *effectively* constructing *p*-program realizations with a new approach using (classical) planning techniques. The main motivation is that realizing a *p*-program involves constructing and synchronizing a set of plans. Even more importantly, plan computation is a computationally hard problem that is crucial to address efficiently in order to solve the realization problem effectively. Therefore, efficient plan computation is a key issue for practical effectiveness, and the use of automated planning systems is a natural approach to address it. Preliminary experiments indicate that our approach significantly outperforms the one considered by De Giacomo, Patrizi, & Sardina (2010).

In this paper, we *(i)* propose the first, to the best of our knowledge, algorithm for constructing *p*-program realizations, based on planning technology, using goal constraints and preferences; *(ii)* provide experimental evidence of its effectiveness; and *(iii)* show the usefulness of using goal preferences to improve performance.

P-programs can also be considered as complex routines, typically including conditions and cycles, to carry on in the domain. The idea of constructing routines on a domain is not new to planning: Baier & McIlraith (2006) consider finite plans with trajectory constraints, that can be seen as (finite) sequential routines; while previous work on planning with temporally extended goals addresses the problem of building cyclic plans to satisfy LTL formulas, modeling desired domain evolutions (Kabanza & Thiébaux 2005). A relevant question concerns the possibility of adapting such techniques to *p*-program realizations. Precisely, the point is whether one can encode the *p*-program requirements into a temporal formula φ such that a plan satisfying φ also corresponds to a realization. To the best of our knowledge, none of them applies in general: Baier & McIlraith (2006) consider only finite plans, while the presence of cycles in *p*-programs requires the ability to build potentially infinite ones; Kabanza & Thiébaux (2005) only consider linear, though possibly cyclic, plans, which cannot cope with executor decisions, that can introduce a sort of nondeterminism and is a distinguishing feature in our problem.



Domain state	Transition	Plan
$s_0 = \{$(at P1 NY), (at A1 Bo)$\}$	$\langle v_0, g_2, v_2\rangle$	$\langle a_1, a_2, a_3, a_4\rangle$
$s_0 = \{$(at P1 NY), (at A1 Bo)$\}$	$\langle v_0, g_1, v_1\rangle$	$\langle a_1, a_2, a_5, a_6\rangle$
$s_1 = \{$(at P1 Bo), (at A1 Bo)$\}$	$\langle v_2, g_0, v_0\rangle$	$\langle a_7, a_1, a_8\rangle$
$s_2 = \{$(at P1 Wa), (at A1 Wa)$\}$	$\langle v_1, g_0, v_0\rangle$	$\langle a_9, a_{10}, a_8\rangle$
$s_3 = \{$(at P1 NY), (at A1 NY)$\}$	$\langle v_0, g_2, v_2\rangle$	$\langle a_2, a_3, a_4\rangle$
$s_3 = \{$(at P1 NY), (at A1 NY)$\}$	$\langle v_0, g_1, v_1\rangle$	$\langle a_2, a_5, a_6\rangle$

Figure 1: Graphical representation of a planning program for ZenoTravel and a respective realization. Nodes are program states; labelled edges are program transitions.

Planning Program Realization

We deal with a specialization of the planning program realization problem (De Giacomo, Patrizi, & Sardina 2010) by assuming a *deterministic* underlying planning domain $\mathcal{D}$, and all maintenance goals satisfied in every $\mathcal{D}$'s state. A *deterministic planning domain* is a tuple $\mathcal{D} = \langle P, A, \tau\rangle$, where: P is the finite set of *propositions*; A is the finite set of *actions*; and $\tau : 2^P \times A \to 2^P$ is the domain *transition function*. Notations $S \subseteq 2^P$, $G \subseteq P$, Π and $Last(\pi(s))$ respectively refer to: the set of domain (or $\mathcal{D}$-) states; a set of goals (also called *goal situation*); the set of all executable plans for $\mathcal{D}$ from some $s \in S$; and the final $\mathcal{D}$-state obtained upon executing plan π from s (written $\pi(s)$).

Formally, a *planning program* for a (deterministic) planning domain $\mathcal{D}$ is a tuple $\mathcal{P} = \langle V, v_0, \Gamma, \delta\rangle$ where: V is the finite set of *program* (or $\mathcal{P}$-) *states*; $v_0 \in V$ is the *initial* $\mathcal{P}$-state; $\Gamma \subseteq 2^P$ is a set of goal situations; and $\delta \subseteq V \times \Gamma \times V$ is the program *transition relation*. To intuitively understand how *p*-programs work, assume a program $\mathcal{P}$ and a domain $\mathcal{D}$ in state v and s, respectively. The executor behaves as follows: firstly, it selects an arbitrary transition $\langle v, G, v'\rangle \in \delta$ and executes a plan π such that $\pi(s)$ achieves G (i.e., $G \subseteq Last(\pi(s))$), thus leading $\mathcal{D}$ to state $s' = Last(\pi(s))$; secondly, it progresses the *p*-program to state v'; and, finally, it starts a new iteration, if some edge outgoing from v' exists, while terminating otherwise.

Example 1 *The sale representative* P1 *of a company, who is responsible for Boston (*Bo*) and Washington (*Wa*), lives in New York (*NY*), and needs to move by plane (*A1*) to/from* Bo *and* Wa*, according to customer requests, which are known only at execution time. Assuming for simplicity that once out of* NY*, before moving to another city, the company requires* P1 *to go back to* NY*. A graphical representation of this behavior is provided in Fig. 1, where the transition system represents a planning program for* P1*, defined on a simplified version of domain* ZenoTravel *(Long & Fox 2003), where only actions* flight*,* board *and* debark *are defined (with obvious parameters and semantics). Transition selection takes place at execution time, based on customer and company requests. From initial state v_0,* P1 *can accept requests for either* Bo *or* Wa*. For instance, if a request for* Bo *arrives, transition $\langle v_0, g_2, v_2\rangle$ is selected. When in* Bo *(resp.* Wa*), however,* P1 *can accept only company's request for going back to* NY*, i.e., transition $\langle v_2, g_0, v_0\rangle$ (resp. $\langle v_1, g_0, v_0\rangle$).*

Executing a *p*-program $\mathcal{P}$ corresponds to bringing about a plan π every time a new transition is selected, according to $\mathcal{P}$. The challenge is constructing an appropriate plan at each step while guaranteeing realizability of all possible future $\mathcal{P}$-transitions, as formalized next by specializing the notions of *plan-based simulation relation* and *planning program realization* (De Giacomo, Patrizi, & Sardina 2010).

Given a planning domain $\mathcal{D}$ and a *p*-program $\mathcal{P}$ as above, a *plan-based simulation relation* is a relation $R \subseteq V \times S$, such that $\langle v, s\rangle \in R$ implies that for every $\mathcal{P}$-transition $\langle v, G, v'\rangle \in \delta$ there exists a plan π over $\mathcal{D}$ such that: *(i)* $\pi(s)$ achieves G; and *(ii)* $\langle v', Last(\pi(s))\rangle \in R$.

A $\mathcal{P}$-state v is *plan-simulated* by a $\mathcal{D}$-state s, written $v \preceq_P s$, if there exists a plan-based simulation relation R such that $\langle v, s\rangle \in R$. Given $\mathcal{D}$, $\mathcal{P}$ and a domain initial state $s_0 \in S$, we say that $\mathcal{P}$ *is realizable in* $\mathcal{D}$ *from* s_0 if $v_0 \preceq_P s_0$. When this is the case, a *p*-program *realization* can be defined. Formally, a *realization* of $\mathcal{P}$ (in $\mathcal{D}$ from s_0) is a partial function $\rho : S \times \delta \to \Pi$ such that for every $\langle v, s\rangle \in \preceq_P$ and every transition $d = \langle v, G, v'\rangle \in \delta$: *(i)* $\rho(s, d)$ achieves G from s; and *(ii)* $v' \preceq_P Last(\pi(s))$. Observe that ρ returns only plans that *preserve* the plan-based simulation relation.

The *Planning Program Realization* problem requires to build a realization of a *p*-program $\mathcal{P}$ for a planning domain $\mathcal{D}$, given an initial $\mathcal{D}$-state $s_0 \in S$.

Example 2 (Ex. 1 cont.) *Incoming requests are fulfilled by* P1 *by selecting the corresponding transition and then executing an appropriate plan for the current domain state. A realization function for the planning program of Ex. 1 is shown in the table of Fig. 1.*

A Planning-based Algorithm

Fig. 2 shows RealizePlanProg, an algorithm for building *p*-program realizations. Starting from an *open (realization) pair* $\langle s, v\rangle$ (initially $\langle s_0, v_0\rangle$), for each transition d outgoing from v, RealizePlanProg constructs a plan π realizing d, and then progresses the states of $\mathcal{D}$ and $\mathcal{P}$ (according to $\pi(s)$ and d, respectively), possibly generating a new open pair $\langle s', v'\rangle$ to process similarly. Each plan π is associated with the respective pair $\langle s, v\rangle$ by the realization function $\rho(s, d)$. If the algorithm generates an open pair $\langle s, v\rangle$ s.t. for some transition outgoing from v no realizing plan can be computed from s, backtracking is required, i.e., the plans generating $\langle s, v\rangle$ need to be replaced in ρ. The algorithm terminates when no more open pairs are left, or it becomes clear that no realization can be found.

Function ρ implicitly defines the set of open pairs, also called the *(realization) frontier*: it is obtained by simulating all possible planning program executions, starting from $\langle s_0, v_0\rangle$, using ρ to realize the transitions, and picking up all pairs $\langle s, v\rangle$ such that for some transition d from v, $\rho(s, d)$ is currently undefined. This essentially corresponds to a straightforward graph visit, and will be summarized by function $Frontier(\rho, s_0, v_0)$, returning the set of open pairs for ρ.

RealizePlanProg maintains three auxiliary functions $State : V \to 2^S$, $Tabu : V \to 2^S$ and $Source : S \times V \to$

Algorithm: RealizePlanProg($\mathcal{P}, \mathcal{D}, s_0$)

Input: a planning program $\mathcal{P} = \langle V, v_0, \Gamma, \delta \rangle$, a planning domain $\mathcal{D} = \langle P, A, \tau \rangle$, and an initial $\mathcal{D}$-state s_0;

Output: a realization of $\mathcal{P}$ in $\mathcal{D}$ from s_0, or `failure`.

1. $\forall s, d \cdot \rho(s, d) \leftarrow$ `noPlan`;
2. $State(v_0) \leftarrow \{s_0\}$; $\forall v \neq v_0 \cdot State(v) \leftarrow \emptyset$;
3. $\forall v \cdot Tabu(v) \leftarrow \emptyset$;
4. $Open \leftarrow \{\langle s_0, v_0 \rangle\}$;
5. **while** $Open$ is *not* empty **do**
6. **extract** an open pair $\langle s, v \rangle \in Open$;
7. $\pi \leftarrow$ `noPlan`;
8. **foreach** transition $d = \langle v, G, v' \rangle \in \delta$ **do**
9. **if** $\rho(s, d) =$ `noPlan` **then**
10. $\pi \leftarrow \mathsf{Plan}(s,\ A,\ G, State(v'), Tabu(v'))$;
11. **if** π is `failure` **then break**;
12. **else**
13. $\rho(s, d) \leftarrow \pi$;
14. **if** $Last(\pi(s)) \notin State(v')$ **then**
15. **add** $\langle Last(\pi(s)), v' \rangle$ **to** $Open$;
16. **add** $Last(\pi(s))$ **to** $State(v')$;
17. **add** $\langle s, d \rangle$ **to** $Source(Last(\pi(s)), v')$;
18. **if** π is `failure` **then**
19. **if** $\langle s, v \rangle = \langle s_0, v_0 \rangle$ **then return** `failure`;
20. **else**
21. **add** s **to** $Tabu(v)$;
22. **remove** s **from** $State(v)$;
23. **foreach** $\langle s'', d = \langle v'', G, v \rangle \rangle \in Source(s, v)$ **do**
24. $\rho(s'', d) \leftarrow$ `noPlan`;
25. $Open = Frontier(\rho, s_0, v_0)$;
26. **return** ρ.

Figure 2: The algorithm RealizePlanProg.

$2^{S \times \delta}$. Intuitively, $State(v)$ records all $\mathcal{D}$-states reached when $\mathcal{P}$ is in v, for some $\mathcal{P}$ execution, according to current (partial) ρ, $Tabu(v)$ indicates the states of $\mathcal{D}$ that are forbidden when v is reached, and $Source$ associates each open pair $\langle s', v' \rangle$ to those pairs $\langle s, d \rangle$ ($d = \langle v, G, v' \rangle$) such that, for $\pi = \rho(s, d)$, $\pi(s)$ ends in s'. Essentially, $Source(f)$ says why an open pair was generated by (current) ρ.

Initially, ρ is undefined (special value `noPlan`), $State$ returns the empty set excepting for $State(v_0) = \{s_0\}$, $Tabu$ returns the empty set, and the set $Open$ contains only $\langle s_0, v_0 \rangle$ (lines 1–4). At each iteration of the external loop (lines 5–25), an arbitrary open pair $\langle s, v \rangle$ is extracted from $Open$, and processed. Open pair processing involves: *(i)* for each transition $d = \langle v, G, v' \rangle$ outgoing from v and not processed yet, computing a plan π that achieves G from s (lines 8–10); *(ii)* updating ρ, $Open$, and the auxiliary functions (lines 11–25). The external loop terminates if $Open$ is empty, in which case ρ is returned (line 26).

Task *(i)* is actually performed by executing Plan, a procedure that constructs a plan π, with end state s' s.t. $G \subseteq s'$, $s' \notin Tabu(v')$, and preferably $s' \in State(v')$. That is, π achieves G, its end state is not in $Tabu(v')$, and end states in $State(v')$ are preferred. Intuitively, states in $State(v')$ are used as *preferences* to minimize the number of generated open pairs, while states in $Tabu(v')$ are used to prevent next iterations from generating unrealizable open pairs.

For task *(ii)*, if a plan π is found, function ρ is updated with π; if $s' = Last(\pi(s))$ is not already in $State(v')$, the set of open pairs is extended with $\langle s', v' \rangle$, i.e., the open pair obtained by realizing d with π from s, and progressing $\mathcal{P}$ from v according to d; $State(v')$ is updated by adding s'; and $\langle s, d \rangle$ is added to $Source(s', v')$ (lines 13–17). If plan computation is unsuccessful, i.e., for some $\mathcal{P}$-transition outgoing from v, Plan is unable to find an appropriate plan (from s), then open pair $f = \langle s, v \rangle$ cannot be realized. In the special case that $f = \langle s_0, v_0 \rangle$, no $\mathcal{P}$-realization can be built, and hence the procedure terminates, returning failure (lines 18–19). Otherwise, backtracking is performed on ρ (lines 21–25): s is added to $Tabu(v)$; s is removed from $State(v)$, as clearly no longer preferred, being tabu; ρ is set undefined on all f sources, as the corresponding plans need to be recomputed in order to avoid generating f; and, finally, $Frontier(\rho, s_0, v_0)$ defines the new set $Open$ of open pairs.

Termination of RealizePlanProg is guaranteed because at every iteration of the external loop (lines 5–25), an open pair $\langle s, v \rangle$ is extracted from $Open$ (line 6), and every time a state is removed from $State(v)$, it is added to $Tabu(v)$ (lines 21–22), where it remains until the execution terminates; therefore, a same open pair $\langle s, v \rangle$ is never added to $Open$ more than once, either because $s \in State(v)$ (lines 14-15) or because s cannot be achieved by the plan returned by Plan, since $s \in Tabu(v)$.

As for soundness, if a pair $\langle s, v \rangle$ is not in $Open$, either it is not reached when executing $\mathcal{P}$ according to ρ, or all transitions d outgoing from v are correctly realized by $\rho(s, d)$; thereby, when $Open = \emptyset$, for each pair $\langle s, v \rangle$ s.t. v is reached with $\mathcal{D}$-state s, the latter holds, and ρ is indeed a realization.

Interestingly, our algorithm is parametric w.r.t. the specific planning procedure adopted, thus allowing us to generate algorithms based on different planning approaches and heuristics, by simply replacing procedure Plan. W.r.t. this, notice that RealizePlanProg is complete iff Plan does, since all possible plans that realize a given transition are explored only in such a case.

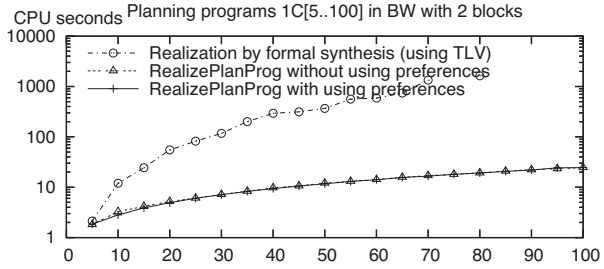


Figure 3: CPU time of RealizePlanProg and TLV for planning programs with structures `1C`[5..100] in domain `Blocksworld` with 2 blocks. The x-axis refers to the number of program states.

Experimental Results

We carried out preliminary experiments to evaluate: *(i)* the effectiveness of our approach, and *(ii)* the impact of goal preferences in minimizing the number of generated open pairs. We executed two classes of experiments, both taking CPU time as evaluation metric: one is aimed at comparing the performance of our algorithm against TLV (Pnueli & Shahar 1996) – an engine for temporal specification synthesis (Pnueli & Rosner 1989), used as suggested by De Gia-

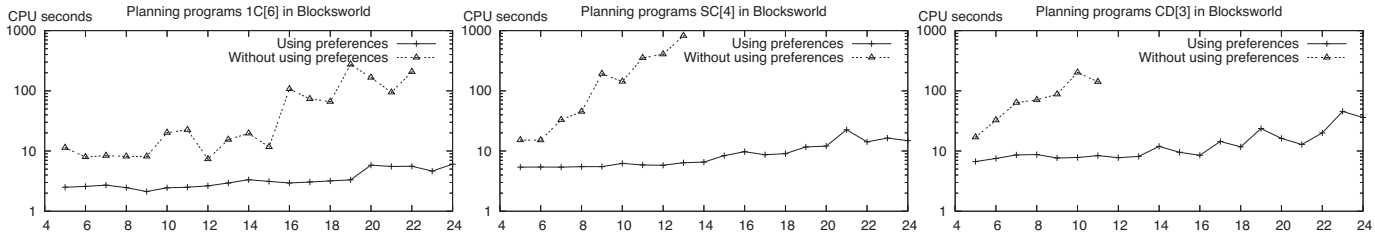


Figure 4: CPU time of RealizePlanProg with and without using goal preferences for planning programs with structures 1C[6], SC[4] and CD[3] (s.t. $|\delta| = 6$) in domain Blocksworld. The x-axis refers to the number of blocks.

como, Patrizi, & Sardina (2010); while the other is meant to evaluate the impact of preferences on RealizePlanProg.

Our implementation of RealizePlanProg encodes preferred and tabu goal states by introducing dummy propositions, numerical fluents and actions, using a compilation scheme inspired by Gerevini *et al.* (2009); procedure Plan is replaced with planner LPG (Gerevini, Saetti, & Serina 2008), though, in principle, any PDDL2.1 planner handling numerical fluents can be incorporated.

We considered, for domains Blocksworld, Storage, and ZenoTravel, *p*-programs with 3 structures of δ, forming a single cycle (1C), multiple binary cycles in sequence (SC), and a complete directed graph (CD):

1C[n]: δ=$\{\langle v_i, G_i, v_{((i+1) \bmod n)}\rangle \mid v_i \in V, 1 \leq i < n\}$,

SC[n]: δ=$\{\langle v_i, G_i, v_{i+1}\rangle, \langle v_{i+1}, G_{i+n-1}, v_i\rangle | v_i \in V, 1 \leq i < n\}$,

CD[n]: δ=$\{\langle v_i, G_{i \cdot n+j}, v_j\rangle, \langle v_j, G_{j \cdot n+i}, v_i\rangle | v_i, v_j \in V, 1 \leq i \leq n, 1 \leq j \leq n, i \neq j\}$, with $n = |V|$ and G_x the x-th set of goals.

We generated 20 planning programs with structures: 1C[6], SC[4], CD[3], 1C[5..100], SC[26], and CD[8], with both the initial state and the $|\delta|$ goal sets randomly chosen.

RealizePlanProg and TLV were compared on *p*-programs with structure 1C[6], defined on a 2-block Blocksworld domain. The corresponding performance gap is shown in Fig. 3, which indicates that even on a toy problem RealizePlanProg is up to two orders of magnitude faster. Also, TLV was not able to realize, within a 30-minute CPU-time threshold, any other *p*-program we considered on Blocksworld, with more than 2 blocks. Though further experiments are required, we expect a possibly increasing gap on harder domains, due to the lack of heuristic-based search (for plan construction) in TLV.

Fig. 4 and Tab. 1 show the performance gap of RealizePlanProg with and without using preferences, on several structures of δ, in domains Blocksworld, Storage and ZenoTravel, in terms of: IPC6 speed score,[1] number of solved problems, and number of generated open pairs. Procedure RealizePlanProg using goal preferences realizes all the considered programs and is up to two orders of magnitude faster than without using preferences. This is essentially due to the significantly reduced number of open pairs generated (and processed) when using preferences.

Planning program		IPC6 score (#solved)		Average #open pairs	
Structure	$\lvert\delta\rvert$	+pref.	–pref.	+pref.	–pref.
Storage					
1C[50]	50	20 (20)	6.13 (20)	51.4	107
SC[26]	50	20 (20)	0.17 (2)	81.7	2934
CD[8]	56	20 (20)	0.80 (4)	228.5	3081
ZenoTravel					
1C[50]	50	20 (20)	6.81 (20)	51.3	63.7
SC[26]	50	20 (20)	1.31 (7)	88.5	2454
CD[8]	56	20 (20)	0.0 (0)	281	3040

Table 1: IPC score, number of solved problems (in parenthesis) and average number of generated open pairs of RealizePlanProg with and without using goal preferences for programs with structures 1C[50], SC[26] and CD[8] in Storage and ZenoTravel.

Conclusions and Future work

In this paper, we addressed the problem of effectively constructing planning program realizations. We proposed a planning-based algorithm, using goal constraints and preferences, and provided (preliminary) experimental evidence of its effectiveness, by comparison with a previous approach that uses techniques for synthesis of LTL specification.

Several research directions remain open. From a theoretical perspective, future work includes *(i)* the study of the problem complexity for deterministic domains, while, from a practical viewpoint, *(ii)* a larger experimental study using other planners in place of LPG and considering more complex domains and structures of program transition relation.

[1] Higher values indicate better performance. For details on IPC6 score, see http://ipc.informatik.uni-freiburg.de.

References

Baier, J. A., and McIlraith, S. 2006. Planning with temporally extended goals using heuristic search. In *Proc. of ICAPS-06*.

De Giacomo, G.; Patrizi, F.; and Sardina, S. 2010. Agent programming via planning programs. In *Proc. of AAMAS-10*.

Gerevini, A., E.; Haslum, P.; Long, D.; Saetti, A.; and Dimopoulos, Y. 2009. Deterministic planning in the fifth international planning competition: PDDL3 and experimental evaluation of the planners. *Artificial Intelligence* 173(5-6):619–668.

Gerevini, A.; Saetti, A.; and Serina, I. 2008. An approach to efficient planning with numerical fluents and multi-criteria plan quality. *Artificial Intelligence* 172(8-9):899–944.

Kabanza, F., and Thiébaux, S. 2005. Search control in planning for temporally extended goals. In *Proc. of ICAPS-05*.

Long, D., and Fox, M. 2003. The 3rd international planning competition: Results and analysis. *JAIR* 20:1–59.

Pnueli, A., and Rosner, R. 1989. On the Synthesis of a Reactive Module. In *Proc. of POPL-89*.

Pnueli, A., and Shahar, E. 1996. A platform for combining deductive with algorithmic verification. In *Proc. of CAV-96*.