

A Systematic Analysis of Levels of Integration between High-Level Task Planning and Low-Level Feasibility Checks

Esra Erdem¹, Volkan Patoglu¹, and Peter Schüller²

¹ Faculty of Engineering and Natural Sciences, Sabancı University, İstanbul, Turkey

² Department of Computer Engineering, Marmara University, İstanbul, Turkey

Abstract. We provide a systematic analysis of levels of integration between discrete high-level reasoning and continuous low-level feasibility checks to address hybrid planning problems in robotic applications. We identify four distinct strategies for such an integration: (i) low-level checks are done for all possible cases in advance and then this information is used during plan generation; (ii) low-level checks are done exactly when they are needed during the search for a plan; (iii) low-level checks are done after a plan is computed, and then a new plan is computed if the plan is found infeasible; (iv) similar to the previous strategy of replanning but a new plan is computed subject to the constraints obtained from previous low-level checks. We perform experiments on hybrid planning problems in housekeeping domain considering these four methods of integration, as well as some of their combinations. We analyze the usefulness of different levels of integration in this domain, both from the point of view of computational efficiency (in time and space) and from the point of view of plan quality relative to its feasibility. We discuss advantages and disadvantages of each strategy in the light of experimental results.

1 Introduction

Successful deployment of robotic assistants in our society requires these systems to deal with high complexity and wide variability of their surroundings to perform typical everyday tasks robustly and without sacrificing safety. Consequently, there exists a pressing need to furnish these robotic systems not only with discrete high-level reasoning (e.g., task planning, diagnostic reasoning) and continuous low-level feasibility checks (e.g., trajectory planning, deadline and stability enforcement), but also their tight integration resulting in hybrid planning.

Motivated by the importance of hybrid planning, recently there have been some studies on integrating discrete task planning and continuous motion planning, presented at various special sessions and workshops organized at major conferences at AAAI 2010, ICAPS 2012, AAAI 2013, RSS 2013, ICRA 2013, ICLP 2013,

Proceedings of the 21st RCRA workshop on *Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion* (RCRA 2014).
Vienna, Austria, July 17–18, 2014.

IROS 2013. These studies can be discussed in two groups, where integration is done at the search level or at the representation level.

For instance, at the search level, [17, 18, 21, 31, 32, 37] take advantage of a search algorithm to incrementally build a task plan, while checking its kinematic/geometric feasibility at each step by a motion planner; all these approaches use different methods to utilize the information from the task-level to guide and narrow the search in the configuration space. In this way, the task planner helps the search process during motion planning. Each one of these approaches presents a specialized combination of task and motion planning at the search level, and does not consider a general interface between task and motion planning.

On the other hand, at the representation level, [1, 3, 6, 7, 19] integrate task and motion planning by considering a general interface between them, using “external predicates/functions”, which are predicates/functions that are computed by an external mechanism, e.g., by a C++ program. The idea is to use external predicates/functions in the representation of actions, e.g., for checking the feasibility of a primitive action by a motion planner. So, instead of guiding the task planner at the search level by manipulating its search algorithm directly, the motion planner guides the task planner at the representation level by means of external predicates/functions. [3, 7] apply this approach in the action description language $\mathcal{C}+$ [16] using the causal reasoner CCALC [27]; [1, 6] apply it in Answer Set Programming (ASP) [2, 25] using the ASP solver CLASP [13]; [19] extends the planning domain description language PDDL [12] to support external predicates/functions (called semantic attachments) and modifies the planner FF [20] accordingly.

In these approaches, integration of task and motion planning is achieved at various levels. For instance, [6, 7] do not delegate all sorts of feasibility checks to external predicates as in [3, 19], but implement only some of the feasibility checks (e.g., checking collisions of robots with each other and with other objects, but not collisions of objects with each other) as external predicates and use these external predicates in action descriptions to guide task planning. For a tighter integration, feasibility of task plans is checked by a dynamic simulator; in case of infeasible plans, the planning problem is modified with respect to the causes of infeasibilities, and the task planner is asked to find another plan.

In this paper, our goal is to better understand how much of integration between high-level reasoning and continuous low-level reasoning is useful for service robotic applications. For that, we consider integration at the representation level, since this approach allows a modular integration via an interface, external predicates/functions, which provides some flexibility of embedding continuous low-level reasoning into high-level reasoning at various levels. Such a flexible framework allowing a modular integration is important for a systematic analysis of levels of integration.

We identify four distinct strategies to integrate a set of continuous feasibility checks into high-level reasoning, grouped into two: *directly integrating* low-level checks into high-level reasoning while a feasible plan is being generated, and generating plans and then *post-checking* the feasibility of these plans with respect

to the low-level checks. For direct integration we investigate two methods of integration:

- (i) low-level checks are done for all possible cases in advance and then this information is used during plan generation,
- (ii) low-level checks are done when they are needed during the search for a plan.

For post-checking we look at two methods of integration:

- (iii) low-level checks are done after a plan is computed, and then a new plan is computed if the plan is found infeasible;
- (iv) similar to the previous strategy of replanning but a new plan is computed subject to the constraints obtained from previous low-level checks.

We consider these four methods of integration, as well as some of their combinations. For instance, some geometric reasoning can be integrated within search as needed, whereas some temporal reasoning is utilized only after a plan is computed in a replanning loop. Considering each method and some of their combinations provide us different levels of integration.

To investigate the usefulness of these levels of integration at representation level, we consider the expressive representation formalisms and state-of-the-art efficient solvers of Answer Set Programming (ASP) [2, 24–26, 28]. In particular, we use HEX programs [5] to describe actions and change, and the efficient ASP solver *dlvhex* [4] to compute plans. Unlike the formalisms and solvers used in other approaches [1, 3, 6, 7, 19], that study integration at representation level, HEX and *dlvhex* [4] allow external predicates/functions to take relations (e.g., a fluent describing locations of all objects) as input without having to explicitly enumerate the objects in the domain. Other formalisms and solvers allow external predicates/functions to take a limited number of objects and/or object variables as input only, and thus they do not allow embedding all continuous feasibility checks in the action descriptions. In that sense, the use of HEX programs with *dlvhex* extends the scope of our experiments with different levels of integration.

We perform experiments on planning problems in a housekeeping domain (like in [1, 6]). The housekeeping domain involves two sorts of feasibility checks: geometric reasoning (e.g., collision checks with static objects, and with movable objects) and temporal reasoning (e.g., restrictions on the total duration of plans). We analyze the usefulness of different levels of integration in this domain, both from the point of view of computational efficiency (in terms of computation time and space) and from the point of view of plan quality relative to its feasibility (in terms of the number of feasible plans, infeasible plans, and low-level feasibility checks).

Our studies on analysis of different methods/levels of integration at representation level is complementary to studies of Lagriffoul et al. [23] to better understand the usefulness of integration of task planning with geometric reasoning at the search level.

Results of our initial studies on systematic analysis of different methods of integration are presented at ICRA 2013 Workshop on Combining Task

and Motion Planning [11] and AAAI 2013 Workshop on Intelligent Robotic Systems [10]. The results presented in this manuscript are more comprehensive.

2 Answer Set Programming

Answer Set Programming (ASP) [2, 24–26, 28] is a form of knowledge representation and reasoning paradigm oriented towards solving combinatorial search problems as well as knowledge-intensive problems. The idea is to represent a problem as a “program” whose models (called “answer sets” [14, 15]) correspond to the solutions. The answer sets for the given program can then be computed by special implemented systems called answer set solvers. ASP has a high-level representation language that allows recursive definitions, aggregates, weight constraints, optimization statements, and default negation. ASP also provides efficient solvers, such as CLASP [13], which has recently won first places at various automated reasoning competitions, such as SAT competitions and ASP competitions.

Due to the continuous improvement of the ASP solvers and highly expressive representation language of ASP which is supported by a strong theoretical background that results from a years of intensive research, ASP has been applied fruitfully to a wide range of areas. Here are, for instance, three applications of ASP used in industry: decision support systems [29] (used by United Space Alliance), automated product configuration [36] (used by Variantum Oy), and workforce management [33] (used by Gioia Tauro seaport).

ASP has also been used for various robotic applications, such as housekeeping robotics [1, 6], cognitive factories [9], and multi-path finding [8].

In our studies, we use HEX programs to describe actions and change. As defined in [5], a HEX program consists of rules of the form:

$$\alpha_1 \vee \dots \vee \alpha_k \leftarrow \beta_1, \dots, \beta_n, \mathbf{not} \beta_{n+1}, \dots, \beta_m$$

where $m, k \geq 0$, each α_i is an atom, and each β_i is an atom or an external atom. An external atom is an expression of the form

$$\&g[Y_1, \dots, Y_n](X_1, \dots, X_m)$$

where $Y_1, \dots, Y_n$ and $X_1, \dots, X_m$ are two lists of terms (called input and output lists, respectively), and $\&g$ is an external predicate name. Intuitively, an external atom provides a way for deciding the truth value of an output tuple depending on the extension of a set of input predicates. External atoms allow us to embed results of external computations into ASP programs. Therefore, external predicates are usually implemented in a programming language of the user’s choice.

For instance, to describe a precondition of a robot moving from one location to another, we define external atoms of the form $\&path_exists_static[x_1, y_1, x_2, y_2]()$ in such a way that $\&path_exists_static[x_1, y_1, x_2, y_2]()$ returns true if and only if there is a continuous collision-free trajectory between two locations (x_1, y_1) and (x_2, y_2) . This external predicate can be implemented in C++, utilizing the bidirectional RRT (Rapidly Exploring Random Trees) [22] as implemented in

the OMPL [35] library. Once the external predicate is defined and implemented, we can use it to represent a precondition of a robot r_1 moving from one location (x_1, y_1) to another (x_2, y_2) at time step t as follows:

$$\perp \leftarrow at(r_1, x_1, y_1, t), goto(r_1, x_2, y_2, t), \\ not \ \&path_exists_static[x_1, y_1, x_2, y_2]()$$

However, these collision checks are performed considering only the static objects in the environment: the external computation is performed considering only the two locations. To be able to check for collisions of movable objects as well, external computations should also take into account locations of other robots as well as their movements. For that, we can define external atoms of the form $\&path_exists_dynamic[goto, at]()$. This external predicate gets as input the set of atoms of the form $goto(r, x, y, t)$ (describing which robot is moving where at time step t) and of the form $at(r, x, y, t)$ (describing the locations of all robots at time step t). It returns true if and only if for each time step and each robot there is a collision-free motion plan from the location given by at at step t to the location given by $goto$ at step t . Therefore, we can utilize this external predicate to ensure collision-free moves of robots between locations as follows:

$$\leftarrow \text{not } \&path_exists_dynamic[goto, at]()$$

The ASP solver dlhex supports HEX programs. In particular, given an action domain and a planning problem as HEX programs, dlhex computes an answer set for these two programs that characterizes a plan, if there exists one.

It is important to note here that dlhex performs external checks as needed. This allows elimination of infeasible plans while they are being computed. To put in other words, a plan does not need to be completely computed if its infeasibility is detected by the externally defined feasibility checks.

3 Levels of Integration

An action domain description can be viewed as a transition system, i.e., a directed graph whose nodes correspond to world states and edges to transitions between states via execution of actions. Transitions in such a domain can be formalized in ASP over time steps $0, \dots, t$, whereas a planning problem instance can be formalized in ASP by an initial state S_0 (as facts) and goal conditions (as constraints) in such a domain. Then an answer set for the union of these two ASP programs, if there exists one, characterizes a history for a plan $\langle A_0, A_1, \dots, A_{n-1} \rangle$, i.e., a path $\langle S_0, A_0, S_1, A_1, \dots, S_{n-1}, A_{n-1}, S_n \rangle$ in the transition system described by the action domain where goal conditions are satisfied at S_n .

A *low-level continuous reasoning module* gets as input, a part of a plan history and returns whether this part of the plan history is feasible or not with respect to some geometric, dynamic or temporal reasoning. For example, the feasibility of a robot's action of moving from a location (x, y) to another location (x', y') characterized by a part of the history of a plan, can be checked by a low-level

continuous reasoning module that takes these locations as input and utilizes a motion planner to do collision checks.

Let L denote a low-level reasoning module that can be used for the feasibility checks of plans for a planning problem instance H . We consider four different methods of utilizing L for computing feasible plans for H , grouped into two classes: *directly integrating* reasoning L into H , and *post-checking* plans of H using L .

For *directly integrating* low-level reasoning into plan generation, we propose the following two methods of integration:

- PRE – *Precomputation* We perform all possible feasibility checks of L that can be required by H , in advance. For each failed check, we identify the actions that cause the failure, and then add a constraint to the action descriptions in H ensuring that these actions do not occur in a plan computed for H . We then try to find a plan for the augmented planning problem instance H^{pre} . Clearly, every plan obtained with this method satisfies all low-level feasibility checks.
- INT – *Interleaved Computation* We do not precompute feasibility checks, but we interleave these low-level checks with high-level reasoning in the search of a plan through use of external predicates: for each action considered during the search, the necessary low-level checks are immediately performed to find out whether including this action will lead to an infeasible plan. An action is included in the plan only if it is feasible. The results of feasibility checks of actions can be stored not to consider infeasible actions repeatedly in the search of a plan. Plans generated by interleaved computation satisfy all low-level checks.

Alternatively, we can integrate low-level checks L with H , by means of *post-checking* plans of H relative to L . We propose the following two methods of integration to perform post-checks on solutions:

- REPL – *Replanning*: We generate a plan for H . For each low-level check in L , we check feasibility of this plan. We discard the plan if it is infeasible and generate a new plan for H until a feasible plan is obtained or all possible candidates are discarded, since a feasible plan does not exist.
- GREPL – *Guided Replanning*: We generate a plan for H . For each low-level check in L , we check feasibility of the plan. Whenever a low-level feasibility check fails, we identify the conditions that cause the failure, and then add a constraint to H ensuring that these conditions do not occur in a plan computed for H . Then, we generate a new plan for the updated planning problem instance H^+ and perform the feasibility checks. We continue with generation of plans and low-level checks until we find a feasible plan, or find out that such a feasible plan does not exist.

A comparison of these methods, in terms of their applicability with existing planners and reasoners and in terms of bilateral guidance between task planning and feasibility checks, is presented in Table 1. Since INT requires special constructs

Table 1. A comparison of methods of integration: Benefits (+) and drawbacks (−)

	PRE	INT	REPL	GREPL
Applicability with existing tools	+	−	+	+ / −
Low-level guides task planning	+	+	−	+
Task planning triggers low-level	−	+	+	+

like external predicates/functions, it is applicable with planners and reasoners that support these constructs. Since GREPL requires representation of planning problems with complex goals and temporal constraints, it is applicable with planners and reasoners that support representation of such problems. The methods PRE and REPL do not need such special constructs, so they are applicable with a wide range of planners and reasoners. The methods PRE and INT utilize all relevant information conveyed from low-level feasibility checks for planning. GREPL utilizes such information for replanning, only after infeasibility of plans are detected. REPL does replan with no such guidance, so low level feasibility checks do not trigger task planning. Low-level feasibility checks are triggered by task planning in all the methods except PRE, where they are all performed before planning.

Let us denote by L_{PRE} and L_{INT} the low-level checks directly integrated into plan generation, with respect to PRE and INT, respectively. Let us denote by L_{POST} the low-level feasibility checks performed after plan generation, with respect to REPL or GREPL. In our systematic analysis of levels of integration, we do consider the hybrid framework depicted in Fig. 1: by selectively enabling some of the feasibility checks, we consider combinations of feasibility checks being performed by different types of integration. For instance, to analyze the usefulness of PRE, we disable the other integrations (i.e., $L_{\text{INT}} = L_{\text{POST}} = \emptyset$); to analyze the usefulness of a combination of PRE and REPL, we disable other integrations (i.e., $L_{\text{INT}} = \emptyset$).

4 Evaluation Criteria

We consider two quantitative metrics to evaluate different types and levels of integration: solution quality and computation time.

We quantify solution quality by addressing the following two questions:

- How many feasible plans are found in a given time?
- How many infeasible plans are discarded meanwhile?

Note that, with the methods PRE and INT, since all relevant information from feasibility checks are utilized for task planning, all computed plans are feasible. With the other methods, REPL and GREPL, some infeasible plans may be generated and discarded until a feasible plan is computed. Since such infeasible plans guide replanning in GREPL by means adding constraints to planning problems, solution quality of GREPL is expected to be higher than REPL.

We quantify computational efficiency by means of

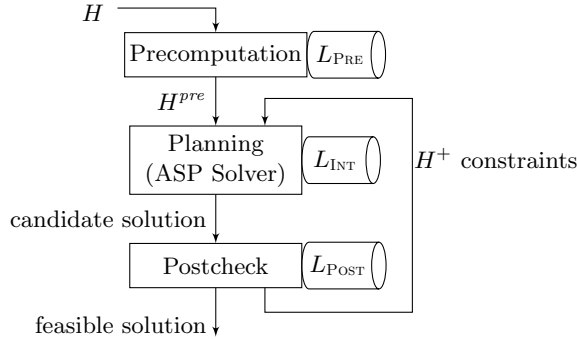


Fig. 1. Combinations of integration methods, and the data flow.

- CPU time to find the first feasible plan,
- total CPU time to find the first 10000 feasible plans, and
- total CPU time for low-level feasibility checks.

In some cases, the first feasible plan can be computed in a short amount of time. Therefore, for a better understanding of planning efficiency, we also consider the computation time of first 10000 feasible plans.

Independent from the number of low-level feasibility checks, the duration of external computations can dominate the overall planning time, or it can be negligible depending on the computational burden of performing the feasibility check. Therefore, we measure not only the number of computations of low-level feasibility checks but also the time spent for these computations.

5 Overview of the Housekeeping Domain

For a systematic empirical evaluation of integration methods, we consider various planning problem instances in housekeeping domain (like in [1, 6]) which requires hybrid planning. In the housekeeping domain [1, 6], the goal is for multiple autonomous PR2 robots to collaboratively tidy up a house, by putting items to their proper places, within a given amount of time. For example, dirty dishes are put into the dishwasher, books into the bookcase, and pillows into the cupboard. Robots can perform the following actions: they can move from one location to another location, attach to and detach from objects. Therefore, a task plan consists of these three forms of actions.

In this domain, there are two sorts of feasibility checks: checking that a plan can be completed with a given deadline, and checking whether a plan is geometrically and kinematically feasible for each robot. These checks require utilization of both high-level and low-level reasoning modules used in the housekeeping domain. In particular, **Po** and **Pe** are utilized to check whether a collision-free path exists from one coordinate to another, with and without consideration of dynamic obstacles in the environment, respectively. To ensure deadlines can be met, **Ti** is used to estimate the time required to traverse such a collision-free path. These feasibility checks are realized using bidirectional RRT

Table 2. Overview of Low-Level Checks for Housekeeping Domain

Check	Description
	Relevant Input
	Realization
Pe	Check whether a robot can move from one coordinate to another one.
	Start and goal coordinate (precomputation possible).
	RRTconnect motion planning via MoveIt! using FCL/OMPL, 3 attempts, 0.5 seconds time limit.
Po	Check whether a robot can move from one coordinate to another one, considering movable objects (obstacles) in the world.
	Start and goal coordinate for the robot, coordinate and type for k objects distinct inputs: (precomputation infeasible for $k \neq 0$).
	RRTconnect motion planning via MoveIt! using FCL/OMPL, 3 attempts, 1 second time limit.
Ti	Estimate the time a robot needs to move from one coordinate to another one by performing multiple motion plans and measuring the length of the shortest one (disregarding obstacles).
	Start and goal coordinate (precomputation possible).
	RRTconnect motion planning via MoveIt! using FCL/OMPL, 9 attempts, 1.5 seconds time limit.

Coordinates are pairs $\langle x, y \rangle$ with $x, y \in \{1, \dots, 8\}$.

(Rapidly Exploring Random Trees) [22] as implemented in the OMPL [35] library, which we use via the MoveIt! [34] and FCL [30] libraries.

As shown in Table 2, for each check, different settings are used for the motion planner. In particular, we increase the planning time limit for **Po** and increase the number of planning attempts for **Ti**. If the motion planner does not return a solution within a given amount of time (which is determined empirically) and after a preset number of trials, the corresponding move action is considered infeasible.

6 Experimental Setup

We applied different variations of integration methods to planning problem instances in the housekeeping domain, with varying size and difficulty. In particular, we performed tests on 20 Housekeeping instances (over 8×8 grid) with up to 5 robots and 8 objects, which require plans up to 12 steps (average 8.2) and up to 45 actions in a single plan (average 24.0).

All experiments were performed on a Linux server with 32 2.4GHz Intel[®] E5-2665 CPU cores and 64GB memory. We realized hybrid planning using the dlhex solver (tag 2.1.0_loi) using the backends GRINGO (3.0.4) and CLASP (2.1.1). We used ROS (groovy) to manage robot descriptions of the PR2 robots. Motion planning queries were realized using bidirectional RRT (Rapidly Exploring Random Trees) [22] as implemented in the OMPL [35] library, which we used via the MoveIt! [34] and FCL [30] libraries. Note that we used multi-threading (limited to 4 threads) for motion planning but not within GRINGO and CLASP. We

Table 3. Computational Effort of Precomputation

Domain	Check	Amount of Checks #	Precomputation Time sec
Housekeeping	Pe	4096	268.3
	Ti	4096	271.8

Table 4. Experimental Results Housekeeping: Comparison between Integration Methods

Integration			FIRST						ALL					
			Quality		Efficiency				Quality			Efficiency		
			Timeouts	Infeasible Plans	Low-Level Checks	Low-Level Time	FIRST Time		Timeouts	Feasible Plans	Feasible Plans	Low-Level Checks	Low-Level Time	ALL Time
Pe	Po	Ti	#	#	#	%	sec		#	#	%	#	%	sec
REPL	REPL	REPL	16.0	12384934	17	0.0	28865		16.0	2584	0.0	19	0.0	28893
PRE	REPL	PRE	8.3	4144063	947	0.4	15660		8.7	5603	0.1	1404	0.5	16338
GREPL	GREPL	GREPL	2.3	174148	30034	70.3	4948		2.3	8321	4.5	30156	69.6	5013
PRE	GREPL	PRE	1.0	11804	13043	67.6	2645		1.0	7472	33.8	13735	66.0	2776
PRE	INT	PRE	0.0	0	2642	34.7	1025		0.0	7599	100.0	3569	37.3	1198
INT	INT	INT	0.0	0	5873	56.2	1019		0.0	8517	100.0	6118	53.1	1112

considered a timeout of 10 hours (36000 seconds) per planning problem instance. We obtained the results of experiments, considering the metrics explained in Section 4, for

- FIRST: obtaining the first feasible plan, and for
- ALL: obtaining a maximum of 10000 feasible plans.

The measurements for enumerating up to 10000 plans reveal information about solution quality and provide a more complete picture of the behavior of each method: one method might find a feasible solutions very fast by chance, whereas finding many (or all) solutions fast by chance is unlikely.

7 Experimental Results

In the following, the reported numbers of performed low-level feasibility checks indicate *distinct* low-level reasoning tasks, as we implement a basic cache for low-level check results. In case of a timeout, a run was counted to use the full time, i.e., 36000 seconds, in order to avoid that timeouts decrease the reported average time. Precomputation effort (i.e., performing all feasibility checks in advance) is given in Table 3. The other tables do not include precomputation counts and times in the reported values.

We discuss the results of our experiments in two parts, by comparing methods of integration and by comparing levels of integration.

7.1 A Comparison of Integration Methods

The feasibility check **Po** includes all objects in the environment as inputs; therefore, the number of potential distinct low-level checks significantly exceeds the

number of other checks, rendering precomputation of these feasibility checks infeasible. Therefore, we compare integration methods described in Section 3, in two stages: first we use INT, GREPL, and REPL for all low-level checks (i.e., $L_{\text{PRE}} = \emptyset$), and then we investigate how the inclusion of PRE affects the results (i.e., for cases where $L_{\text{INT}} \cup L_{\text{POST}}$ does not include all feasibility checks, $L_{\text{PRE}} \neq \emptyset$ contains the rest).

Computational Effort Experimental results for comparing these integration methods, in terms of computational efficiency, are summarized in Table 4. According to these results, if we only look at the methods where PRE is not considered in combination with other methods, then we observe that INT by far outperforms GREPL, and GREPL outperforms REPL. We illustrate this as follows:

$$\text{REPL} < \text{GREPL} \ll \text{INT}.$$

For example, INT requires on average 1019 seconds to find the first feasible plan of an instance, while GREPL and REPL require 4948 and 28865 seconds, respectively.

If we compare methods that involve PRE, we observe that for GREPL and REPL the addition of PRE always improves efficiency significantly. For example, we require 4948 seconds for FIRST with pure GREPL, while using PRE for **Pe** and **Ti** reduces this time to 2645 seconds. For the combination of INT and PRE the picture is different: the usage of PRE increases runtimes slightly from 1019 to 1025 seconds for FIRST and from 1112 to 1198 seconds for ALL. The **Po** check eliminates many infeasible plans, so computation time does not differ much for integration scenarios with PRE. In fact, due to many constraints added as a result of PRE, the computation time may get worse. In summary, the effect of PRE on other methods depends on the relevance of low-level checks on feasibility of plans in the domain. The experimental results recommend PRE in general:

$$\begin{aligned} \text{REPL} &< \{\text{REPL}, \text{PRE}\} \\ \text{GREPL} &< \{\text{GREPL}, \text{PRE}\} \\ \text{INT} &\approx \{\text{INT}, \text{PRE}\} \end{aligned}$$

Solution Quality Experimental results for comparing integration methods, in terms of solution quality, are summarized in Table 4. These results depict solution quality measurements, including the number of feasible plans found.

According to these results (and as discussed in Section 3), the methods PRE and INT do not generate infeasible solutions, as they use all low-level checks already in search. REPL produces the fewest number of feasible plans (0.02%), GREPL creating significantly more (0.12%), and INT producing only feasible plans. Using PRE improves solution quality for REPL and GREPL. As with computational effort, the difference between FIRST and ALL is not significant from the perspective of solution quality in this domain.

Based on these results, we can conclude that, with respect to solution quality, INT and $\{\text{INT}, \text{PRE}\}$ always perform the best, followed by non-integrated methods

Table 5. Experimental Results Housekeeping: Comparison between Levels of Integration

Integration			FIRST					ALL					
			Quality		Efficiency			Quality			Efficiency		
			Timeouts	Infeasible Plans	Low-Level Checks	Low-Level Time	FIRST Time	Timeouts	Feasible Plans	Feasible Plans	Low-Level Checks	Low-Level Time	ALL Time
Pe	Po	Ti	#	#	#	%	sec	#	#	%	#	%	sec
GREPL	GREPL	GREPL	2.3	174148	30034	70.3	4948	2.3	8321	4.5	30156	69.6	5013
GREPL	INT	GREPL	1.3	5	29386	71.9	3562	1.3	8411	99.7	29615	70.8	3642
GREPL	GREPL	INT	1.0	10795	19329	77.8	2800	1.3	8415	33.4	26263	79.1	3390
INT	INT	GREPL	0.7	4	18384	68.8	2393	0.7	8466	99.8	20718	66.5	2644
GREPL	INT	INT	0.7	3	15197	78.4	2371	0.7	8466	100.0	15379	76.3	2452
INT	GREPL	INT	0.0	0	18870	66.0	2240	0.0	8517	38.7	19348	64.3	2345
INT	GREPL	GREPL	0.7	13966	19670	70.4	2077	0.7	8466	31.6	19800	68.4	2150
INT	INT	INT	0.0	0	5873	56.2	1019	0.0	8517	100.0	6118	53.1	1112

combined with PRE, followed by GREPL and then by REPL:

$$\text{REPL} < \text{GREPL} \left\langle \begin{array}{l} \{\text{GREPL}, \text{PRE}\} \\ \{\text{REPL}, \text{PRE}\} \end{array} \right\rangle \left\langle \begin{array}{l} \text{INT} \\ \{\text{INT}, \text{PRE}\} \end{array} \right\rangle$$

7.2 A Comparison of Levels of Integration

According to the results of our experiments to compare methods of integration, we can observe that the most promising two methods are GREPL and INT. With this motivation, to better understand these methods and levels of their integration, we compare all combinations of GREPL and INT for low-level feasibility checks (i.e., $L_{\text{INT}} \cup L_{\text{POST}}$ contains all feasibility checks). Thus these experiments allows us to measure the influence of integrating low-level checking results during reasoning (INT) or only after finding a solution that satisfies all checks registered so far (GREPL). Experimental results are summarized in Table 5.

Computational Effort From the results presented in Table 5, we observe that integrating more low-level checks with INT than with GREPL generally increases efficiency.

The results show a significant increase in efficiency when going from pure GREPL to any integration that uses INT. Combinations where some checks are integrated with INT and some checks are handled with GREPL do not exhibit such a strong difference in performance among one another. However, the difference between mixed integration levels and the full integration of all low-level checks with INT again shows a strong improvement of efficiency.

Solution Quality According to Table 5, integrating only **Po** with INT leads to a significantly better solution quality for ALL (99.7% feasible plans) than integrating only **Pe** (31.6%) or only **Ti** (33.4%) with INT. This is intuitively clear because **Po** subsumes the **Pe** check (if there is no path considering obstacles,

then there is no path without obstacles) and it also subsumes parts of the **Ti** check (if there is no path considering obstacles, then there is no time estimate for a path). Then we can infer that **Po** is more significant for solution quality than the other low-level checks of Housekeeping.

We can conversely observe this significance of **Po** by comparing runs where **Po** was the *only* check performed with GREPL: in this case we obtain 38.7% feasible plans for ALL while handling **Pe** or **Ti** with GREPL yields nearly 100.0% feasible plans (in absolute numbers 3 and 14 infeasible plans, respectively).

The limitation of 10000 plans for ALL is necessary to obtain comparable efficiency and solution quality results, and at the same time it shows the diversity of the properties of our experimental domain.

8 Discussion and Conclusion

We have identified four different methods of integrating high-level task planning with low-level feasibility checks, and introduced a computational method for systematically analyzing the efficacy of these methods of integration through a comprehensive empirical study. Our method also involves a detailed study of levels of integration by considering various uses of combinations of methods. Based on this systematic approach, we have performed experiments in housekeeping robotic domain with different feasibility checks. Results of these experiments suggest the following conclusions.

If low-level feasibility checks have a high influence on plan feasibility, then using full interleaved reasoning (INT) is the most promising approach. INT not only consistently achieves the best performance with respect to computation times, but also can enumerate most number of solutions compared to other approaches, since INT uses only those low-level checks which are necessary (they are computed on demand) and therefore does not overload the solver with redundant information (as PRE does). Furthermore, INT considers results of failed feasibility checks (as constraints) in the search process, and thereby never picks an action where it is known that the action will violate a low-level feasibility check.

If low-level feasibility checks necessitate a small number of inputs, such that precomputation is a feasible option, then PRE may be a good choice. However, precomputation should not be used with INT, since it can decrease its computational performance by adding many (partially irrelevant) constraints and requiring significantly more memory.

The REPL approach performs the worst, because nothing guides the search into the direction of a feasible solution; REPL is not robust and enumerates many infeasible solutions.

If both PRE and INT are not possible (e.g., if the state of inputs is large, and additionally we have to use a certain planner that does not support interleaved computation) then GREPL should be used. This approach is robust but less efficient than INT, as the integration is less tight than with INT. Nevertheless, it is a very robust approach as it is guided by its wrong choices — we can think of the use of constraints that are obtained from failed low-level feasibility checks

and added to the planning problem description, as an approach of “learning from its mistakes”.

Our ongoing work involves a similar systematic analysis for different robotic application domains, for a better understanding of the methods and the levels of integration between high-level task planning with low-level feasibility checks.

Acknowledgements

This work is partially supported by TUBITAK Grants 111E116 and 113M422.

References

1. E. Aker, V. Patoglu, and E. Erdem. Answer set programming for reasoning with semantic knowledge in collaborative housekeeping robotics. In *Proc. of IFAC SYROCO*, 2012.
2. G. Brewka, T. Eiter, and M. Truszczynski. Answer set programming at a glance. *Commun. ACM*, 54(12):92–103, 2011.
3. O. Caldiran, K. Haspalamutgil, A. Ok, C. Palaz, E. Erdem, and V. Patoglu. Bridging the gap between high-level reasoning and low-level control. In *Proc. LPNMR*, 2009.
4. T. Eiter, G. Ianni, R. Schindlauer, and H. Tompits. Effective integration of declarative rules with external evaluations for Semantic-Web reasoning. In *Proc. ESWC*, 2006.
5. T. Eiter, G. Ianni, R. Schindlauer, and H. Tompits. A Uniform Integration of Higher-Order Reasoning and External Evaluations in Answer-Set Programming. In *Proc. of IJCAI*, 2005.
6. E. Erdem, E. Aker, and V. Patoglu. Answer set programming for collaborative housekeeping robotics: representation, reasoning, and execution. *Intelligent Service Robotics*, 5:275–291, 2012.
7. E. Erdem, K. Haspalamutgil, C. Palaz, V. Patoglu, and T. Uras. Combining high-level causal reasoning with low-level geometric reasoning and motion planning for robotic manipulation. In *Proc. of ICRA*, 2011.
8. E. Erdem, D. G. Kisa, U. Oztok, and P. Schüller. A general formal framework for pathfinding problems with multiple agents. In *Proc. of AAAI*, 2013.
9. E. Erdem, V. Patoglu, Z. G. Saribatur, P. Schüller, and T. Uras. Finding optimal plans for multiple teams of robots through a mediator: A logic-based approach. *Theory and Practice of Logic Programming*, 2013.
10. E. Erdem, V. Patoglu, and P. Schüller. Levels of integration between low-level reasoning and task planning. In *Proc. AAAI 2013 Workshop on Intelligent Robotic Systems*, 2013.
11. E. Erdem, V. Patoglu, and P. Schüller. A systematic analysis of levels of integration between low-level reasoning and task planning. In *Proc. ICRA 2013 Workshop on Combining Task and Motion Planning*, 2013.
12. M. Fox and D. Long. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *J. Artif. Intell. Res. (JAIR)*, 20:61–124, 2003.
13. M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub. clasp: A conflict-driven answer set solver. In *Proc. of LPNMR*, 2007.
14. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Proc. of ICLP*, 1988.

15. M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9:365–385, 1991.
16. E. Giunchiglia, J. Lee, V. Lifschitz, N. McCain, and H. Turner. Nonmonotonic causal theories. *Artificial Intelligence*, 153:2004, 2004.
17. F. Gravot, S. Cambon, and R. Alami. *Robotics Research The Eleventh International Symposium*, volume 15 of *Springer Tracts in Advanced Robotics*, chapter aSyMov:A Planner That Deals with Intricate Symbolic and Geometric Problems. 2005.
18. K. Hauser and J.-C. Latombe. Integrating task and PRM motion planning: Dealing with many infeasible motion planning queries. In *ICAPS Workshop on Bridging the Gap between Task and Motion Planning*, 2009.
19. A. Hertle, C. Dornhege, T. Keller, and B. Nebel. Planning with semantic attachments: An object-oriented view. In *Proc. of ECAI*, 2012.
20. J. Hoffmann and B. Nebel. The FF planning system: Fast plan generation through heuristic search. *J. Artif. Intell. Res.*, 14:253–302, 2001.
21. L. P. Kaelbling and T. Lozano-Pérez. Hierarchical task and motion planning in the now. In *Proc. of ICRA*, 2011.
22. J. Kuffner Jr and S. LaValle. RRT-connect: An efficient approach to single-query path planning. In *Proc. of ICRA*, 2000.
23. F. Lagriffoul, L. Karlsson, J. Bidot, and A. Saffiotti. Combining task and motion planning is not always a good idea. In *Proc. RSS 2013 Workshop on Combined Robot Motion Planning and AI Planning for Practical Applications*, 2013.
24. V. Lifschitz. Answer set programming and plan generation. *Artificial Intelligence*, 138:39–54, 2002.
25. V. Lifschitz. What is answer set programming? In *Proc. of AAAI*, 2008.
26. V. Marek and M. Truszczyński. Stable models and an alternative logic programming paradigm. In *The Logic Programming Paradigm: a 25-Year Perspective*, pages 375–398. Springer Verlag, 1999.
27. N. McCain and H. Turner. Causal theories of action and change. In *Proc. of AAAI*, 1997.
28. I. Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence*, 25:241–273, 1999.
29. M. Nogueira, M. Balduccini, M. Gelfond, R. Watson, and M. Barry. An A-Prolog decision support system for the space shuttle. In *Proc. of PADL*, 2001.
30. J. Pan, S. Chitta, and D. Manocha. FCL: A general purpose library for collision and proximity queries. In *Proc. of ICRA*, 2012.
31. E. Plaku. Planning in discrete and continuous spaces: From LTL tasks to robot motions. In *Proc. of TAROS*, 2012.
32. E. Plaku and G. D. Hager. Sampling-based motion and symbolic action planning with geometric and differential constraints. In *Proc. of ICRA*, 2010.
33. F. Ricca, G. Grasso, M. Alviano, M. Manna, V. Lio, S. Iiritano, and N. Leone. Team-building with answer set programming in the Gioia-Tauro seaport. *Theory and Practice of Logic Programming*, 12, 2012.
34. I. A. Sucan and S. Chitta. "MoveIt!", [Online] Available: <http://moveit.ros.org>.
35. I. A. Sucan, M. Moll, and L. E. Kavraki. The open motion planning library. *IEEE Robotics & Automation Magazine*, 19(4):72–82, 2012.
36. J. Tihihonen, T. Soininen, and R. Sulonen. A practical tool for mass-customising configurable products. In *Proc. of ICED*, 2003.
37. J. Wolfe, B. Marthi, and S. Russell. Combined task and motion planning for mobile manipulation. In *Proc. of ICAPS*, 2010.