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

Esra Erdem [a,**], Volkan Patoglu [b] and Peter Schüller [c,***]

[a] *Computer Sciences and Engineering, Faculty of Engineering and Natural Sciences, Sabancı University, İstanbul, Turkey*
E-mail: esraerdem@sabanciuniv.edu
[b] *Mechatronics Engineering, Faculty of Engineering and Natural Sciences, Sabancı University, İstanbul, Turkey*
E-mail: vpatoglu@sabanciuniv.edu
[c] *Department of Computer Engineering, Marmara University, İstanbul, Turkey*
E-mail: peter.schuller@marmara.edu.tr

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 the results are 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 if the plan is found infeasible then a new plan is computed; (iv) similar to the previous strategy but the results of previous low-level checks are used during computation of a new plan. We analyze the usefulness of these strategies and their combinations by experiments on hybrid planning problems in different robotic application domains, in terms of computational efficiency and plan quality (relative to its feasibility).

Keywords: task planning — motion planning — hybrid planning — knowledge representation and reasoning — answer set programming — cognitive robotics — experimental analysis

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., planning, diagnostic reasoning) and continuous low-level feasibility checks (e.g., trajectory planning, deadline and stability enforcement, collision checks), but also their tight integration resulting in hybrid planning.

Consider, for instance, a mobile robot tidying up a house. This task requires ordering of robotic actions of moving from one location to another one, picking objects from some locations, and placing objects to their desired locations, considering the preconditions and effects of these actions as well as other changes (e.g., ramifications of actions) in the domain. Therefore, motion planning (e.g., finding a continuous trajectory from one configuration of the robot to another configuration) and other low-level feasibility checks are not sufficient to solve this problem. On the other hand, tidying up a house also requires feasibility checks, such as whether the robot will be able to move from one location to another location without colliding with any objects, whether the robot will be able to reach the object on the table and grasp it without any collisions, or whether the stack of objects will be stable once the robot puts an object at the very top. Therefore, task planning only (e.g., finding a sequence of robotic actions from an initial state to a goal state) is not sufficient to solve the problem either. These examples illustrate the necessity for a hybrid approach to planning, that integrates task planning with feasibility checks.

Motivated by the importance of hybrid planning, recently there have been some studies on integrating discrete task planning and continuous motion plan-

* Manuscript Version - The final publication is available at IOS Press through http://dx.doi.org/10.3233/AIC-150697

** Corresponding author: Faculty of Engineering and Natural Sciences, Sabancı University, İstanbul, Turkey

*** P. Schüller's work was carried out while visiting the Cognitive Robotics Lab at Sabancı University.

ning, presented at various special sessions and workshops organized at major conferences at AAAI 2010, ICAPS 2012, AAAI 2013, RSS 2013, ICRA 2013, ICLP 2013, IROS 2013, AAAI 2014, IROS 2014, ECAI 2014, ICAPS 2014, AAAI 2015, RSS 2015, IJCAI 2015, IROS 2015. 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, the related studies [32, 33, 38, 39, 42, 60, 61, 64, 70] 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, the related studies [1,4,7,11,12,24,34–36] integrate task and motion planning by considering a general interface between them, using "external atoms" (in the spirit of semantic attachments in theorem proving [69]). External atoms are predicate/function atoms whose values are computed by an external mechanism, e.g., by a C++ program. The idea is to use external atoms 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 atoms. In [4, 12, 34], this approach is applied in the action description language $\mathcal{C}+$ [31] using the causal reasoner CCALC [50]. In [1, 11, 35], it is applied in Answer Set Programming (ASP) [2, 46, 47, 49, 54] using the ASP solver CLASP [26]. Some studies [7, 36] extend the planning domain description language PDDL [23] to support external atoms and modifies the planner FF [37] and the TFD [21] accordingly; the extended PDDL is called PDDL/M, the extended versions of FF and TFD are called FF/M and TFD/M respectively. Hybrid planning is also studied [24] using the PKS planner [57, 58] whose language is an extension of STRIPS [22] with databases.

The focus of this paper is to investigate the usefulness of different methods and levels of integration between high-level (discrete) action planning and low-level (continuous) feasibility checks, at the representation level. For this purpose, (i) we identify different integration methods that are applicable to hybrid planning, (ii) propose a general representation and reasoning framework that allows us to implement all these integration methods and their combinations by modifying the representation of the robotic domain rather than by modifying the planners, solvers, and feasibility checkers, (iii) we perform a systematic empirical analysis of the usefulness of different methods and levels of integration over two robotic domains that involve a variety of navigation and manipulation tasks and feasibility checks.

Methods and levels of integration The representation-level integration of task planning with feasibility checks can be achieved with different strategies and at various levels. For instance, some related studies [1,4,11,12,35] implement some of the feasibility checks as external atoms and use these external atoms in action descriptions to guide task planning when needed. Meanwhile, 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.

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 study 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 the previous feasibility 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 provides us different levels of integration.

Hybrid planning framework To investigate the usefulness of these levels of integration at the representation level, we consider the expressive representation formalisms and the state-of-the-art efficient solvers of ASP. In particular, we use HEX programs [10] to describe actions and change, and the efficient ASP solver dlvhex [9] to compute plans.

Unlike the formalisms and solvers used in other approaches [1, 4, 7, 11, 12, 24, 34–36], that study integration at representation level, HEX and dlvhex [9] allow external atoms 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 atoms 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.

It is important to emphasize here that we do not investigate the applicability of ASP for robotic planning, nor do we claim that ASP should be used in robotic planning via HEX formalism and the dlvhex solver. Our choices of HEX and dlvhex are due to the expressivity of the formalism and the utilities of the solver, as discussed above.

Benchmarks and experiments We perform experiments on hybrid planning problems in two robotic domains: housekeeping domain (like in [1,11]) and robotic manipulation (like in [12]). The housekeeping domain involves two sorts of feasibility checks: geometric reasoning (e.g., collision checks with static objects, and with movable objects, while finding continuous trajectories) and temporal reasoning (e.g., restrictions on the total duration of plans). The robotic manipulation domain involves geometric reasoning (e.g., collision checks of robots and payloads with each other and the environment).

We analyze the usefulness of different levels of integration in these domains, 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).

Raw data on benchmarks (i.e., experimental results for each problem instance) and the ROS-dlvhex plug-in used for the experiments are available at `http://cogrobo.sabanciuniv.edu/?p=854`.

It is important to emphasize here that, since our focus is on the integration of task planning with feasibility checks, we consider offline hybrid planning (under the assumption of complete knowledge of the world) in our experiments. Therefore, execution monitoring of plans is out of the focus and the scope of this paper. Various uses of hybrid planning in the context of different execution monitoring frameworks have been studied in some other papers (e.g., [11, 15, 34]).

Summary of contributions The main contributions of our study can be summarized as follows:

- We have identified and categorized four distinct strategies to integrate a set of continuous feasibility checks into high-level reasoning.
- We have determined evaluation metrics to characterize performance of integration methods in terms of solution quality and computational efficiency.
- We have designed and implemented an experimentation platform that utilizes a common ASP solver (dlvhex), such that all integration methods can be evaluated in a fair manner. As part of this experimentation platform, we have implemented all the necessary API's to embed external computations for the feasibility checks.
- We have empirically characterized performance of each integration method, as well as their combinations, under two realistic robotic domains, by considering multiple scenarios for each domain.
- We have determined general guidelines that may help end-users select proper integration methods for their domains.

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 [18], AAAI 2013 Workshop on Intelligent Robotic Systems [17], and the 21st RCRA International Workshop on Experimental Evaluation of Algorithms for solving problems with combinatorial explosion [19]. The results presented in this manuscript are significantly more systematic and comprehensive. In particular, in our earlier workshop papers [17, 18], the experiments for direct integration were done using the ASP solver dlvhex, whereas the experiments for post-checking were done using the ASP solver CLASP. In this paper, use of a common platform for all experiments based on the ASP solver dlvhex enables a much more fair comparison of different integration approaches. Furthermore, the benchmarks used in this study involve are comprehensive: they include (i) two robotic domains that require feasibility checks with different characteristics, and (ii) more number of instances than the benchmarks used in our earlier papers.

Outline of the paper In the following, we first present some preliminaries on ASP (Section 2), hybrid planning in ASP (Section 3), and feasibility checks (Section 4). After that, we discuss different methods and levels of integration (Section 5). Then we set the stage for experiments by describing the evaluation criteria (Section 6), the robotic domain descriptions (Section 7), and the experimental setup (Section 8). We discuss the experimental results (Section 9) and the related work (Section 10), and we conclude (Section 11).

2. Answer Set Programming

Answer Set Programming (ASP) [2, 46, 47, 49, 54] 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" [28, 29]) 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 [26], 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 [55] (used by United Space Alliance), automated product configuration [68] (used by Variantum Oy), and workforce management [63] (used by Gioia Tauro seaport).

ASP has also been used for various robotic applications, such as housekeeping robotics [1, 11], cognitive factories [13, 15, 16], and multi-path finding [14].

HEX *programs* In our studies, we use HEX programs to describe actions and change. As defined in [10], a HEX program consists of rules of the form:

$$\alpha_1 \vee \cdots \vee \alpha_k \leftarrow \beta_1, \ldots, \beta_n, \mathbf{not}\, \beta_{n+1}, \ldots, \mathbf{not}\, \beta_m \quad (1)$$

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

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

where $Y_1, \ldots, Y_n$ and $X_1, \ldots, X_m$ 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 atoms are usually implemented in a programming language of the user's choice.

We refer the reader to [10] for detailed information about the semantics of HEX programs, and we give some examples of the use of external atoms.

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 it 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 atom can be implemented in C++, utilizing the bidirectional RRT (Rapidly Exploring Random Trees) [41] as implemented in the OMPL [67] library. Once the external atom 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:

$$\leftarrow at(r_1, x_1, y_1, t), goto(r_1, x_2, y_2, t), \\ \quad not\ \&path\_exists\_static[x_1, y_1, x_2, y_2]().$$

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 atom 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 atom to ensure collision-free moves of robots between locations as follows:

$$\leftarrow \mathbf{not}\ \&path\_exists\_dynamic[goto, at](). \quad (2)$$

dlvhex Given a HEX program, the ASP solver dlvhex computes an answer set for it, if there exists one.

dlvhex evaluates a HEX program as depicted in Figure 1. First, the HEX program is preprocessed: external atoms are replaced by auxiliary atoms. The resulting auxiliary program (now without external atoms) is instantiated by the ASP grounder GRINGO [27]. Next, the ASP solver CLASP searches for an answer set for the auxiliary program. As partial/full answer set candidates are examined, dlvhex uses special callbacks (propagators) of CLASP to evaluate the external atoms. This feature of dlvhex ensures that there is a one-to-one correspondence between auxiliary answer sets (i.e., answer sets for the auxiliary program) and HEX answer sets (i.e., answer sets for the HEX program), and allows transformation of the former to the latter by removing auxiliary atoms. For more details about HEX evaluation, we refer the reader to [8].

When the CLASP callbacks perform external computations (either as soon as possible, or as late as possible), and how the results of failed checks are integrated into the search process (either by just discarding the failed solution candidate, or by also adding constraints that prevent finding further candidates that contain the same failed check) are determined by the external atom evaluation configuration. Therefore, the external atom evaluation configuration plays a significant role in our study to implement different integration methods. For instance, feasibility checks can be done as late as possible after a task plan is computed, or they can be done as soon as it is possible to do so. When dlvhex performs external checks as needed, infeasible plans are eliminated 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.

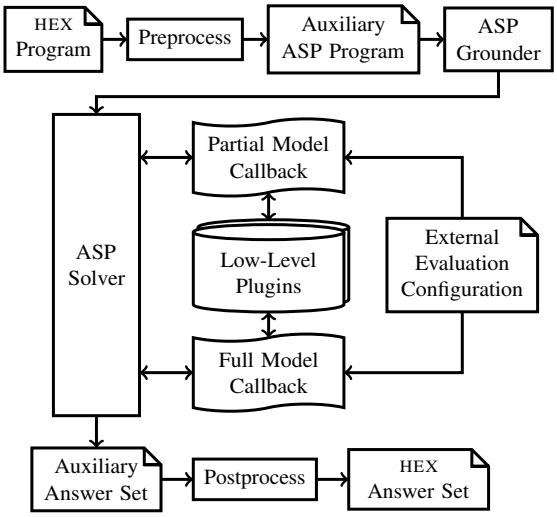

Fig. 1. Internals of HEX Evaluation.

3. ASP Planning

ASP planning [6,45,46,54,65] is based on the idea of reducing a planning problem to the problem of finding an answer set for a program and then computing a plan using an ASP solver. In that sense, it is similar to SAT planning [40], which reduces a planning problem to the problem of finding a satisfying interpretation for a set of propositional formulas and then computes a plan using a SAT solver. ASP planning differs from SAT planning in that it uses programs instead of propositional formulas, and ASP solvers instead of SAT solvers. Also, it is easier to describe some properties and constraints of robotic domains in ASP due to the special constructs its formalism and its solvers support (e.g., default negation, aggregates, optimization statements, cardinality constraints).

In ASP planning, we (i) represent the robotic domain as a program, (ii) represent the planning problem instance as a program, (iii) compute an answer set for the union of these two programs using an ASP solver, and (iv) extract the plan characterized by the answer set. Let us describe this process with an example in the robotic navigation domain mentioned in the previous section. ASP formulations of two other example robotic domains (used in our experiments) are provided in the appendix.

3.1. Representing a planning problem in ASP

A representation of an action domain in ASP consists of descriptions of actions by means of their preconditions and direct effects, descriptions of ramifications of actions, the commonsense-law of inertia, constraints (such as state, transition, noconcurrency, cardinality constraints), auxiliary definitions (such as derived predicates), optimization statements.

Fluents and actions. To describe the robotic actions and how the world state changes over time, we need to introduce atoms to represent occurrences of actions and values of fluents at a given time step. We denote occurrence of an action A at time step t by $A(t)$ and the value of a fluent F at time step t by $F(t)$. In the sample navigation domain, the robots move to their destination locations. We introduce atoms of the forms $goto(r,x,y,t)$

to denote the occurrence of action $goto(r,x,y)$ ("robot r goes to location (x,y)") at time step t. To describe the preconditions and effects of this action, fluents are needed to represent locations of robots. Thus, we introduce atoms of the form $at(r,x,y,t)$ to denote the values of boolean fluent $at(r,x,y)$ ("robot r is located at (x,y)") at time step t.

Background knowledge. In addition to the fluents and the actions, whose values may change over time, we need to represent background knowledge whose values do not change over time (e.g., locations of static objects, or common sense knowledge). For instance, to describe the locations (x,y) occupied by static obstacles, we introduce atoms of the form $occupied(x,y)$.

Direct effects of actions. Suppose that the transitions in this sample navigation domain have Markovian property: direct effects of an action executed at a state (at time step t) are observed at the next state (at time step $t+1$).

(Conditional) direct effects of an action are described by HEX rules (1) where $k \geq 1$, $n \geq 1$. The direct effects observed as changes in the fluent values are represented by $\alpha_1,\ldots,\alpha_k$. The occurrences of actions are represented by some β_i $(1 \leq i \leq n)$. The conditions under which these direct effects take place are described by the rest of the atoms β_i $(1 \leq i \leq m)$. In the sample navigation domain, as a direct effect of the action $goto(r,x,y)$, the location of the robot changes to (x,y); this direct effect is described by an ASP rule as follows:

$$at(r,x,y,t+1) \leftarrow goto(r,x,y,t).$$

Preconditions of actions. Preconditions of actions are represented by means of nonexecutability conditions. Nonexecutability conditions of an action are described by constraints (HEX rules (1) where $k=0$). The occurrences of actions are represented by some β_i $(1 \leq i \leq n)$, and the nonexecutability conditions are described by some other atoms β_i $(1 \leq i \leq m)$. In the navigation domain, a robot can neither move to an occupied location nor to its current location; these nonexecutability conditions are represented by the rules:

$$\leftarrow goto(r,x,y,t), occupied(x,y)$$
$$\leftarrow goto(r,x,y,t), at(r,x,y,t).$$

As discussed in the previous section, some preconditions of actions are determined by feasibility checks whose values are computed externally. This is possible thanks to external atoms.

Ramifications (or indirect effects) of actions. Ramifications of actions are described by HEX rules (1) where $k \geq 1$, $n \geq 1$. These rules do not mention any action atoms. A ramification of the $goto(r,x,y)$ action is that "if a robot/object is at a location then it is not anywhere else":

$$\neg at(r,x,y,t) \leftarrow at(r,x_1,y_1,t) \ \ (x \neq x_1)$$
$$\neg at(r,x,y,t) \leftarrow at(r,x_1,y_1,t) \ \ (y \neq y_1).$$

The commonsense law of inertia. Formalizing the commonsense law of inertia is proposed as a solution to the frame problem [51]. According to the commonsense law of inertia, an action can be assumed not to change the value of a fluent unless there is evidence to the contrary. ASP allows us to easily formalize the commonsense law of inertia, thanks to its nonmonotonic negation **not** . Indeed, we can describe the commonsense law of inertia by HEX rules (1) of the form

$$F(t+1) \leftarrow F(t), \mathbf{not}\ \neg F(t+1)$$
$$\neg F(t+1) \leftarrow \neg F(t), \mathbf{not}\ F(t+1)$$

for each fluent atom F. For instance, in the robotic navigation domain, we represent the commonsense law of inertia by the rules:

$$at(r,x,y,t+1) \leftarrow at(r,x,y,t), \mathbf{not}\ \neg at(r,x,y,t+1)$$
$$\neg at(r,x,y,t+1) \leftarrow \neg at(r,x,y,t), \mathbf{not}\ at(r,x,y,t+1).$$

Constraints. We can express various sorts of constraints (about states, transitions, concurrency of actions, cardinality of a set of atoms, durations of actions, etc.) using HEX rules (1) where $k=0$. For instance, the state constraint that "no two robots occupy the same location at any time step" can be expressed by the following rules:

$$\leftarrow at(r_1,x,y,t), at(r_2,x,y,t) \ \ (r_1 < r_2).$$

Planning problem. An initial state of a planning problem instance can be described as a set of facts (HEX rules (1) where $k=1$, $n=m=0$) using fluent atoms. For instance, the following facts describe an initial state (at time step 0) for two robots:

$$at(R_1,0,0,0) \leftarrow$$
$$at(R_2,0,3,0) \leftarrow .$$

The goal is described as a set of HEX rules (1) where $k=0$, $n=0$, and each goal condition is represented by β_i $(n < i \leq m)$. For instance, the following rules express

that the robots are expected to be located at $(3,4)$ and $(7,8)$ in a goal state:

$$\leftarrow \textbf{not } at(R_1,3,4,maxstep)$$
$$\leftarrow \textbf{not } at(R_1,7,8,maxstep)$$

where *maxstep* is the specified makespan for a plan.

3.2. Solving a planning problem in ASP

Suppose that the robotic actions and the change of worlds states are described by a set D of HEX rules, and a planning problem instance P is described by another set P of HEX rules, as shown by examples above. We can present the union $D \cup P$ of these two programs to the ASP solver dlvhex, in its input language. Then, for a specified plan length *maxstep* for a plan, dlvhex tries to find an answer set for the program $D \cup P$. If the program has an answer set then the action atoms in this set characterize a plan.

Once a maximum makespan *maxmakespan* for a plan is specified, an optimal plan of length *maxstep* can be computed by trying to find a plan using the ASP solver dlvhex with $maxstep = 1,2,3,...,maxmakespan$.

3.3. Discussion

ASP provides a general declarative problem solving framework for a wide variety of combinatorial search problems. Therefore, its formalism and its solvers support many utilities. In that sense, its formalism is different from action languages [30] (introduced to represent dynamic systems), planning languages [22, 52] (introduced to represent planning problems). Its solvers are different from reasoning systems (implemented to solve various reasoning problems, such as prediction, planning, postdiction) and planners (implemented to solve planning problems, such as classical planning, conditional planning, conformant planning).

4. Feasibility Checks

There are various feasibility checks to consider while planning in a robotic domain, and each feasibility check is a well-studied computational problem in robotics. For instance, feasibility of the action of a mobile robot moving from one location to another as described in Section 2, can be modeled as a motion planning problem with collision checks.

Let us formally define the motion planning problem. We first present some preliminary definitions as in [59].

A *continuous state* consists of a collection of variable values that completely describe a dynamic system at any given instant in time. The set of all continuous states constitutes the *state space* of a system.

For instance, the continuous state of a 2D polygonal robot consists of variables that specify the position (x,y) and orientation θ of the polygon with respect to a frame of reference. In this case, the state space is commonly referred to as the configuration space. In case of a simple car, the state space of the system consists of the velocity v and steering angle ϕ in addition to the configuration (x,y,θ) of the car.

Please note that, despite the identical terminology, the notion of a state in dynamic systems is different from the notion of a state in task planning. In particular, in dynamics terminology, a state of a system essentially consists of continuous variables that uniquely define the configuration of the system augmented with variables that define its velocity. In this paper, the term "state" will be reserved for task planning, and the term "continuous state" will be used for dynamic systems.

Continuous state constraints indicate a desired invariant that each continuous state should satisfy. In motion planning, common continuous state constraints include collision avoidance with obstacles and other robots, joint limits, limits on accelerations and radius of curvature avoid abrupt motions.

The motion-planning goal is specified as desired constraints that a continuous goal state should satisfy. Such constraints may include a desired position or orientation. In motion-planning with dynamics, it is also common to require that a robots velocity remain within certain bounds.

Finally, a *trajectory* indicates the evolution of a systems continuous state with respect to time. In particular, a trajectory is a function $\gamma : [0,T] \rightarrow S$, parameterized by time $T \in R \geq 0$.

Given these definitions, a *motion-planning problem* can be defined as a tuple $P = (\mathcal{S}, Valid, s_{init}, Goal)$, where

- $\mathcal{S}$ represents a state space consisting of a finite set of variables that completely describe the continuous state of the system;
- $Valid : \mathcal{S} \rightarrow \{\top, \bot\}$ is a continuous state constraint function, i.e., $Valid(s) = \top$ iff $s \in \mathcal{S}$ satisfies the continuous state constraints;
- $s_{init} \in \mathcal{S}$ is an initial continuous state;
- $Goal : \mathcal{S} \rightarrow \{\top, \bot\}$ is a goal function, i.e., $Goal(s)$ is $\top$ iff $s \in \mathcal{S}$ satisfies the motion-planning goal.

A *solution to the motion-planning problem P* is a valid trajectory $\gamma : [0,T] \rightarrow \mathcal{S}$ that starts at s_{init} and satisfies the motion-planning goal, i.e., $\gamma(0) = s_{init}$; $Goal(\gamma(T)) = \top$; and $\forall t \in [0,T] : Valid(\gamma(t)) = \top$.

The basic motion planning is proven to be PSPACE-complete [62], like propositional action planning [3,20].

There are various books that describe motion planning [5,44], and software libraries [67] that gather implementations of various motion planners. There are also software libraries [56] that include implementations of various collision check algorithms. Our methods of integration allow the use of these existing libraries as they are.

5. 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 $0,\ldots,t$, 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 $\langle A_0,A_1,\ldots,A_{n-1}\rangle$, i.e., a path $\langle S_0,A_0,S_1,A_1,\ldots,S_{n-1},A_{n-1},S_n\rangle$ 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 feasibility checks of plans for a planning problem instance H. We consider four different methods of utilizing L for computing feasible plans for H, organized into two groups: *directly integrating* reasoning L into H, and *post-checking* plans of H using L.

5.1. Direct Integration of Feasibility Checks

For *directly integrating* low-level reasoning into plan generation, we propose two methods of integration: *Precomputation* (PRE) and *Interleaved Computation* (INT).

PRE – Precomputation: This method first performs all possible feasibility checks of the low-level reasoning module L that may be required by the planning problem instance H, in advance. The results of these feasibility checks are represented by a new predicate. For instance, the results of feasibility checks of a robot r's action of moving from a location (x,y) to another location (x',y') can be represented by atoms of the form $path\_exists(x,y,x',y')$, where $1 \leq x,x' \leq n$ and $1 \leq y,y' \leq m$.

Then, these new atoms are used (like external atoms) to express constraints (like preconditions of actions). For instance, to express that a robot r cannot go from a location (x,y) to a location (x',y'), if there is no collision-free continuous trajectory between these two locations, a constraint of the form

$$\leftarrow at(r,x,y,t), goto(r,x',y',t), \\ not\ path\_exists(x,y,x',y')$$

is added to the ASP formulation of the robotic domain. Such constraints ensure that infeasible actions do not occur in a plan computed for H.

Next, the method tries to find a plan for H with respect to the modified domain description.

Clearly, in a robotic domain without movable obstacles, every plan obtained with this method satisfies all low-level feasibility checks of L. Another advantage of this method is that, since the domain description is augmented with constraints, the search space is pruned.

In robotics, it is common to precompute and cache the results of such feasibility checks to speed up computations, such as, determining robot base locations for reachability, calculating stable grasps between robot gripper and object pairs. These results can be utilized for hybrid planning in robotic domains via the method PRE.

Note that, since the results of feasibility checks are represented by atoms, they can be easily utilized for describing the preconditions of actions in other action description formalisms (like PDDL). In that sense, PRE can be applied with other formalisms and solvers.

On the other hand, in some robotic domains, PRE has severe practical limitations due to the need for too many feasibility checks. For instance, in the navigating robot example above, PRE is feasible for low-level motion planning queries if we do not consider movable obstacles in the world because the number of queries is polynomial in the size of the world: for a world with $n \times m$ locations, there are $O(n^2m^2)$ motion planning queries. However, if we consider movable obstacles in

the world and since these obstacles can cover any location at any time, then the number of queries may be beyond exponential in the size of the world (i.e., due to $O(n^2m^2(nm)!)$ motion planning queries), rendering precomputation infeasible.

Applications of the method PRE to hybrid planning can be found in various related studies [24, 35].

INT – Interleaved Computation: This method interleaves feasibility checks with high-level reasoning in the search of a plan for H, by means of external atoms that are externally evaluated by a low-level reasoning module L that implements these feasibility checks.

Consider, for instance, the navigating robot example, and suppose that there are no movable objects in the environment. This method first introduces external atoms of the form $\&path\_exists\_static[x,y,x',y']()$ whose values are computed by L as follows: L returns true if and only if there is a continuous collision-free trajectory between the locations (x,y) and (x',y').

Then, for every pair (x,y) and (x',y') of locations, a constraint of the form

$$\leftarrow at(r,x,y,t), goto(r,x',y',t), \\ not\ \&path\_exists\_static[x,y,x',y']()$$

is added to the ASP formulation of the robotic domain. This constraint ensures that, at any time step t, if there is no collision-free continuous trajectory between the locations (x,y) and (x',y'), the robot r does not move from (x,y) to (x',y') in a plan computed for H.

Next, the method tries to find a plan for H with respect to the modified domain description.

With external atoms included in the constraints, for each action considered during the search, the relevant feasibility checks implemented in L 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 the method INT satisfy all feasibility checks.

If there are movable objects in the environment, then the method INT introduces a different form of external atom, like $\&path\_exists\_dynamic[goto,at]()$, and adds a different form of constraints (i.e., constraints (2)) to the robotic domain description as described in Section 2.

As discussed in Section 2, external atoms are evaluated when needed, so are the feasibility checks in L. In this way, infeasible plans are eliminated while they are being computed, so a plan does not need to be completely computed if its infeasibility is detected by the externally defined feasibility checks.

On the other hand, despite these advantanges, the method INT can be utilized in some formalisms (e.g., $C+$, ASP, PDDL/M) and solvers (e.g., CCALC, CLASP, FF/M, TFD/M, PKS) to some extent [1, 4, 7, 11, 12, 24, 36], since external atoms whose inputs involve predicate names cannot be utilized in these formalisms and solvers. INT can be utilized completely in HEX programs and the dlvhex solver.

The applications of the method INT to hybrid planning can be found in various related studies [1, 4, 7, 11, 12, 35, 36].

5.2. Post-Checking Plan Feasibility

Alternatively, the feasibility checks of the low-level reasoning module L can be integrated with high-level reasoning in the search for a plan for the planning problem instance H, by *post-checking* the feasibility of a computed plan for H relative to L and replanning if the plan is found infeasible. We propose two methods of integration to perform post-checking on solutions: *Replanning* (REPL) and *Guided Replanning* (GREPL).

REPL – Replanning: This method first generates a plan for H, and then checks its feasibility using L. The plan is discarded if it is found infeasible and a new plan is generated for H. This process continues until a feasible plan is obtained or all possible candidates are discarded, since a feasible plan does not exist.

The method REPL provides no guidance while generating a new plan; hence, it may seem somewhat not useful. However, REPL is commonly used in robotics, mainly due to the fact that many of the planners do not allow for addition of sophisticated constraints to planning problem descriptions.

Also, it is good to include this method in our systematic analysis of different methods and levels of integration, to better understand the usefulness of providing guidance to generation of new plans.

GREPL – Guided Replanning: This method generates a plan for H, and then checks its feasibility using L. Whenever a feasibility check fails, the conditions that cause the failure are identified, and constraints are added to the formulation of H ensuring that these conditions do not occur in a plan computed for H. Then, the method tries to generate a plan for the modified planning problem instance H^+, and performs the feasibility checks in L over the generated plan (if exists). This process of modifying the planning problem description,

generating a plan for it, and checking the plan's feasibility continues until a feasible plan is computed for H, or it is find out that a feasible plan does not exist.

Consider, for instance, a robot R navigating in an environment. Suppose that a plan is computed for a planning problem H, and the plan involves the robot's action of moving from location $(1,3)$ to location $(3,3)$. Suppose that the plan is found infeasible since the feasibility check of existence of a continuous collision-free trajectory implemented in the module L failed for this action. Then, the method GREPL adds the following constraints

$$\leftarrow at(R,1,3,t), goto(R,3,3,t) \qquad (0 \leq t \leq maxstep)$$

to the ASP formulation of H, and tries to find a plan for the modified problem.

Applications of the method GREPL to hybrid planning can be found in various related studies [1,4,11,12, 34,35].

A remark about replanning: The replanning problem considered in the methods above involves the same domain description as in the original planning problem, the same initial state, the same goals, and possibly some additional constraints. Therefore, replanning may extend the original planning problem by adding constraints. In that sense, replanning as in the methods above is different from replanning in the context of execution monitoring of plans, where the replanning problem involves the same domain description, the current state, and the same goals.

5.3. Comparison of Integration Methods

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.

The methods PRE and REPL do not need any special constructs, so they are applicable with a wide range of planners and reasoners. On the other hand, INT requires special constructs like external atoms; therefore, it is applicable with planners and reasoners that support such constructs. Similarly, GREPL requires representation of planning problems with complex goals and temporal constraints; therefore, it is applicable with planners and reasoners that support representation of such problems.

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.

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	−	+	+	+

Low-level feasibility checks are triggered by task planning in all the methods except PRE, where they are all performed before planning.

5.4. Implementation of Integration Methods

In our studies, the integration methods PRE, INT, REPL, and GREPL are realized in ASP as follows.

Consider the same planning problem instance (i.e., same initial state, same goal conditions); suppose that no additional constraints are added for GREPL. The HEX program that represents the hybrid planning domain description is the same for INT, REPL, GREPL. Therefore, only the dlvhex solver's configuration for evaluation of external computations is different: for INT, we configure the solver dlvhex in a way that it checks feasibility of actions as soon as all inputs of a feasibility check are available; for GREPL, we require a fully specified solution candidate to perform this check; for REPL we additionally deactivate "external nogood learning" in the solver so that the solver can no longer remember previously failed feasibility checks.

For PRE, we replace external atoms in the domain description by new atoms, whose values are evaluated beforehand and described as a set of facts.

These minimal changes in the implementation of the four strategies aim at keeping the high-level planning engine the same while changing only the method of low-level integration. This way we know that differences in results are really just because of the type of integration and not because of other effects.

5.5. Combinations of Integration Methods

Let us denote by L_{PRE} and L_{INT} the low-level feasibility 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.

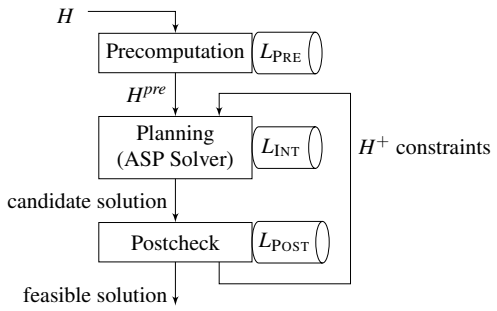

Fig. 2. Different levels of integration.

In our systematic analysis of levels of integration, we do consider the hybrid framework depicted in Fig. 2: 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., $L_{\text{INT}} = \emptyset$).

6. 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?

In this study, we are not concerned with optimization of plans and thus we do not consider other measures for plan quality.

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

- 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.

7. Overview of the Robotic Domains

For a systematic empirical evaluation of integration methods, we consider various planning problem instances in two action domains: housekeeping domain (like in [1, 11]) and robotic manipulation domain (like in [12]). Both of these domains require hybrid planning.

7.1. Housekeeping Domain

In the housekeeping domain [1, 11], 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 geometric and temporal reasoning.

Table 2 presents an overview of 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 as implemented in the OMPL library, which we use via the MoveIt! [66] and FCL [56] libraries.

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 $x, y \in \{1, \ldots, 8\}$.

Table 3
Overview of Low-Level Checks for Robotic Manipulation Domain

Check	Description
	Relevant Input
	Realization
Ro	Given two robot poses check whether they are collision-free among the robots and between robots and the environment.
	Two robot poses (precomputation possible).
	Collision check via MoveIt! using FCL.
Pa	Given two payload poses check whether they are collision-free among the payloads and between payloads and the environment.
	Two payload poses (precomputation possible).
	Collision check via MoveIt! using FCL, representing each payload as a robot.
Mo	Given start and goal positions of two robots, positions of payloads, and whether the robots are carrying one of the payloads, check whether a collision-free motion plan exists.
	Start and goal poses of two robots, poses of k payloads, carried payload (precomputation impossible).
	RRTconnect motion planning via MoveIt! using FCL/OMPL, 3 attempts, 4 seconds time limit.

Poses are triples $\langle x, y, direction \rangle$ with $x, y \in \{1, \ldots, 11\}$ and $direction \in \{n, s, w, e\}$.

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.

7.2. Robotic Manipulation Domain

We consider a mobile manipulation problem, as in [12], where two Kuka youBots arrange elongated payloads in a space that contains obstacles by performing one of the three forms of actions: moving from one location to another location, picking and dropping an endpoint of a payload. The payloads can only be carried cooperatively by both robots. Therefore, task plans in this domain are composed of these actions.

To find a feasible plan, we need to ensure that objects do not collide with each other or with the environment, and robots do not collide with each other. Some collision checks between objects can be realized in the high-level representation of the action domain, whereas most collision checks require use of geometric models of the robot and the environment. For instance, collision-free paths for robots for particular collaborative actions can only be determined using low-level geometric reasoning and cannot be represented at the high-level.

Table 3 presents an overview of low-level reasoning components used in the robotic manipulation domain. In particular, given position and orientation of two robots, **Ro** verifies if this configuration is collision-free, while **Pa** checks whether two payloads can be located in particular positions and orientations without collisions with environment or each other. Furthermore, **Mo** verifies if it is possible to generate a motion plan for moving robots and payloads from one configuration to another one.

For this domain, we perform motion planning and collision checking for one or two independently moving Kuka youBot bases, or for two Kuka youBot bases connected by a payload. Similar to the housekeeping domain, we realize motion planning using bidirectional RRT using the MoveIt!, OMPL, and FCL libraries.

7.3. Discussion

The domains used as benchmarks have been carefully designed, taking into account the following aspects.

They represent real world robotic domains, such that they can be dynamically simulated and/or physically implemented using real robots, as in [1, 11, 12].

The domains involve a variety of robotic actions, involving both navigation [1, 11] and manipulation [12].

The problem sizes are reasonable not only from the perspective of the real world applications, but also from the point of view of related work on robotic planning. For instance, in planning problem instances of our housekeeping domain, we view the workspace as an 8×8 grid, which indicates 64 abstract locations of interest for the robots. We consider up to 5 robots and 8 objects of interest. The plans require up to 45 robotic actions. The robotic manipulation instances are even larger in size. The related work [64] considers a domain where a robot transfers some objects from one place to another, one at a time, as part of the experimental evaluation. The largest instance involves up to 40 objects, and the maximum makespan is 40. The other related works [7, 24, 36, 42] consider smaller instances.

The domains involve a variety of feasibility checks whose computations necessitate the use of different algorithms or external computations of different complexity. For instance, in the housekeeping domain, all the low-level reasoning modules perform motion planning to check feasibility of actions, but the motion planners are used with different parameters. In robotic manipulation domain, there are two types low-level reasoning: (i) object collision checks **Pa** and **Ro** operate on 3D models of robots, objects, and the environment; and (ii) the motion planning check **Mo** performs motion planning and internally performs many collision checks as in (i). Hence, for **Pa** and **Ro** checks, a collision checker is used while a motion planner is used for **Mo**.

To be able to use the method PRE, enumeration of feasibility checks is required. The types of feasibility checks considered in our domains require different computational effort for enumeration. The feasibility checks **Pe** and **Ti** in the housekeeping domain take a pair of coordinates as input, similarly checks **Ro** and **Pa** in the robotic manipulation domain take a pair of poses. If there are $n \times m$ coordinates, in the housekeeping domain, we need to perform the checks **Pe** and **Ti** $O(n^2m^2)$ times; in the robotic manipulation domain, since there are 4 possible orientations for a robot and 2 possible orientations for a payload endpoint, we need to perform **Ro** $O(n^2m^2)$ number of times and **Pa** $O(n^2m^2)$ number of times.

On the other hand, the checks **Po** and **Mo** include all objects in the environment as inputs; therefore, the number of potential distinct feasibility checks significantly exceeds the number of the checks discussed in the previous paragraph, rendering precomputation of these feasibility checks infeasible. For instance, in the housekeeping domain, if we consider all movable objects, then the number of the check **Po** increases to $O(n^2m^2(nm)!)$. Similarly, the number of feasibility checks **Mo** increases by a factor of $(nm)!$. Performing a high number of feasibility checks as in these domains can require a large amount of time; but even if that is not the case, the results of these checks must be represented in the domain to constrain high-level reasoning, and this requires a large amount of space.

8. 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 steps) and up to 45 actions (average 24.0 actions) in a single plan. We experimented with 20 Robotic Manipulation instances (over a 11×11 grid) that require plans of up to 21 (average 8.3) steps, and involving up to 63 (average 23.4) actions.

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 dlvhex 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 and Kuka robots. Motion planning queries were realized using bidirectional RRT as implemented in the OMPL library, which we used via the MoveIt! and FCL libraries. Note that we used multi-threading (limited to 4 threads) for motion planning but not within GRINGO and CLASP. We considered a timeout of 10 hours (36000 seconds) per planning problem instance. We obtained the results of experiments, considering the metrics explained in Section 6, 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.

Table 4
Computational Effort of Precomputation

Domain	Check	# of Checks	Time (sn)
Housekeeping	**Pe**	4096	268.3
	Ti	4096	271.8
Robotic Manipulation	**Ro**	125840	35.4
	Pa	9669	2.1

9. 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 4. 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.

9.1. A Comparison of Integration Methods

The feasibility checks **Po** and **Mo** include 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 5, 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).

9.1.1. Computational Effort

Experimental results for comparing these integration methods, in terms of computational efficiency, are summarized in Tables 5 and 6, and Figure 3.

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 in Housekeeping and has similar performance in Robotic Manipulation. We illustrate this as follows:

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

For example, in Housekeeping, INT requires on average 1019 seconds to find the first feasible plan of an instance, while GREPL and REPL require 4948 seconds and 28865 seconds, respectively. For Robotic Manipulation, INT requires on average 267 seconds for FIRST, while GREPL and REPL require 1937 seconds and 2000 seconds, respectively.

If we compare methods that involve PRE, we observe that for GREPL and REPL the addition of PRE always increases efficiency significantly. For example, in Housekeeping, 4948 seconds is required for FIRST with pure GREPL, while using PRE for **Pe** and **Ti** reduces this time to 2645 seconds. In Robotic Manipulation, 1937 seconds is required with GREPL and 661 seconds is required when PRE is used for **Ro** and **Pa**.

For the combination of INT and PRE the picture is different: in Housekeeping, the usage of PRE increases runtimes slightly from 1019 seconds to 1025 seconds for FIRST and from 1112 seconds to 1198 seconds for ALL; in Robotic Manipulation, the usage of PRE increases runtimes from 267 seconds to 663 seconds for FIRST and from 1977 seconds to 2691 seconds for ALL. For Housekeeping, 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. For Robotic Manipulation, we observe that all methods using PRE for checks **Ro** and **Pa** perform very similar — whichever way we integrate **Mo**. Both effects can be explained by a low relevance of the **Mo** check on solutions of Robotic Manipulation. Once precomputation has eliminated infeasible plans, few infeasible plans remain for **Mo** to eliminate so the method of integration makes no big difference, hence all methods with PRE show similar performance. Furthermore, adding all check results of PRE to the Robotic Manipulation domain introduces many irrelevant constraints which can make the search slower, while using INT instead of PRE will only use relevant constraints.

In summary, the effect of PRE on other methods depends on the relevance of low-level checks on feasibility of plans in the domain. 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 be used cautiously with INT, since it can decrease its computational performance by adding many (partially irrelevant) constraints and requiring significantly more memory.

Table 5
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

Table 6
Robotic Manipulation: 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
Ro	**Pa**	**Mo**	#	#	#	%	sec	#	#	%	#	%	sec
REPL	REPL	REPL	1.0	1925	1925	82.6	2000	1.0	2549	24.2	7988	88.7	3748
GREPL	GREPL	GREPL	0.7	1564	1564	33.4	1937	1.0	2559	27.1	6872	59.6	3363
PRE	PRE	INT	0.0	0	42	1.8	663	0.0	2527	100.0	3613	72.2	2691
PRE	PRE	GREPL	0.0	4	4	0.2	661	0.0	2527	45.3	3047	70.4	2582
PRE	PRE	REPL	0.0	4	4	0.2	655	0.0	2527	45.4	3035	70.4	2552
INT	INT	INT	0.0	0	171	3.7	267	0.0	2755	100.0	5800	82.8	1977

$$\text{REPL} < \{\text{REPL}, \text{PRE}\}$$

$$\text{GREPL} < \{\text{GREPL}, \text{PRE}\}$$

$$\text{INT} \approx \{\text{INT}, \text{PRE}\} \qquad \text{(Housekeeping)}$$

$$\{\text{INT}, \text{PRE}\} < \text{INT} \qquad \text{(Robotic Manipulation)}$$

9.1.2. Solution Quality

Experimental results for comparing integration methods, in terms of solution quality, are summarized in Tables 5 and 6, and Figure 4. These results depict solution quality measurements, including the number of feasible plans found. As the number of infeasible plans varies greatly between methods (cf. Table 6), we use a logarithmic scale in Figure 4.

According to these results (and as discussed in Section 5), the methods PRE and INT do not generate infeasible solutions, as they use all low-level checks already in search. Therefore, we cannot show these zero values on the logarithmic scale (recall that $\log(0) = -\infty$).

In the Housekeeping domain, we obtain a clear order: REPL produces the fewest number of feasible plans (0.0%), GREPL creating significantly more number of feasible plans (4.5%), and INT producing only feasible plans (100%). 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.

In the Robotic Manipulation domain, REPL is also the worst method (24.2% feasible plans), followed by GREPL (27.1% feasible plans) with INT being the best method (only feasible plans). Similar to the analysis of computational effort, PRE makes GREPL and REPL behave in a similar way and without a clear winner; the precomputed low-level checks manage to eliminate most infeasible solutions already without considering the **Mo** check.

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

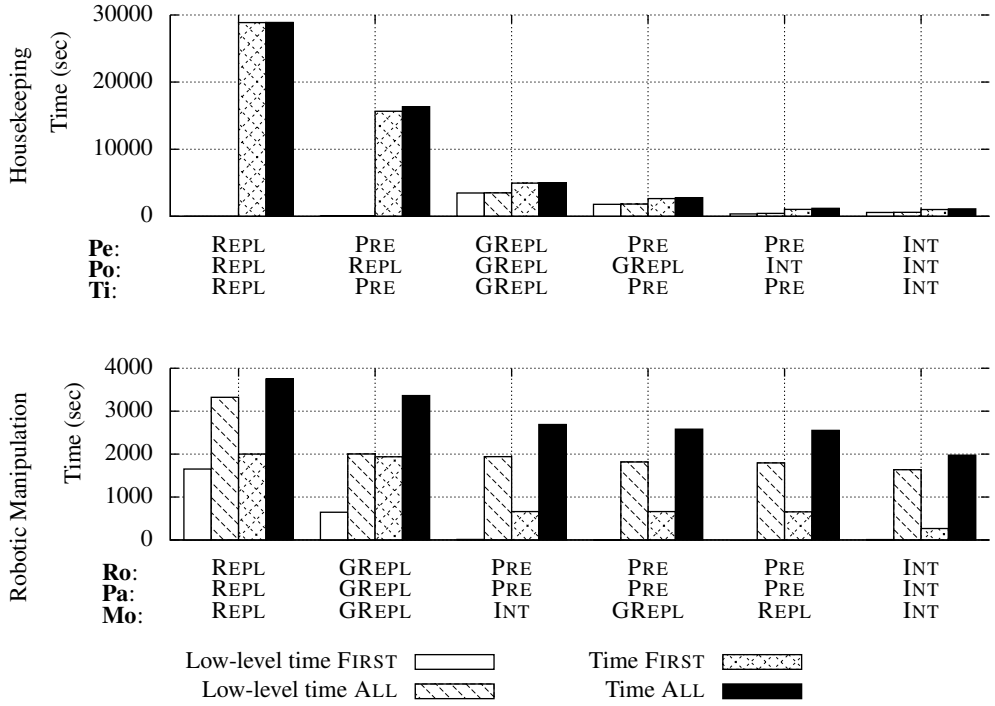

Fig. 3. Comparison of Computational Effort depending on Integration Method

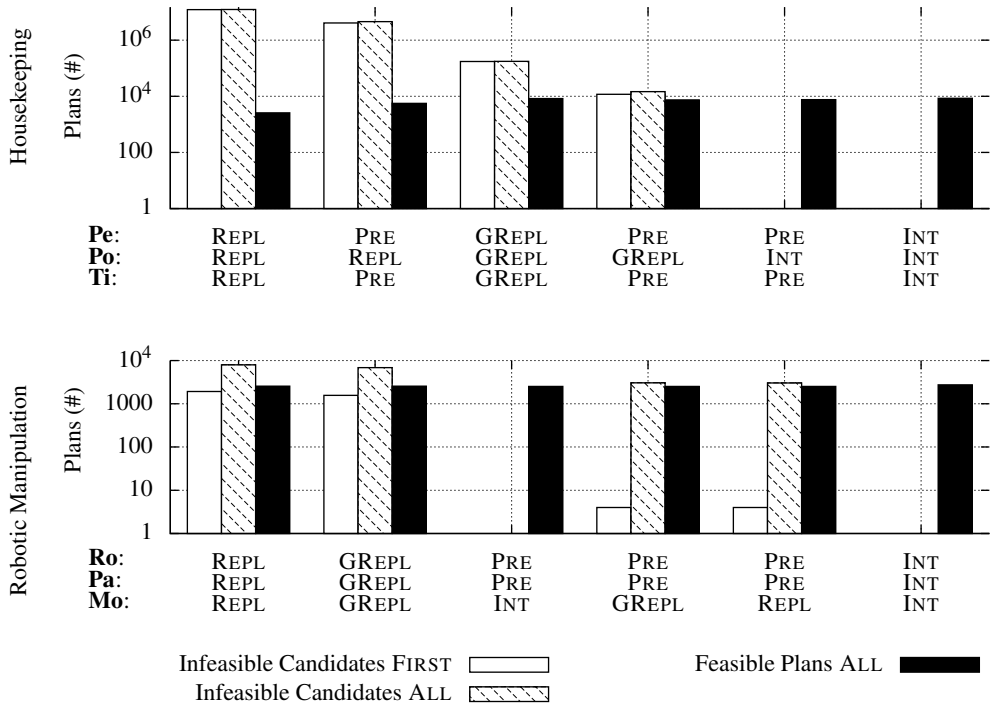

Fig. 4. Comparison of Solution Quality depending on Integration Method

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

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

9.1.3. Additional Observations

Number of low-level feasibility checks. When we compare REPL with REPL + PRE in Tables 5 and 6, we see that the number of low-level checks in Housekeeping increases while it decreases for Robotic Manipulation.

In Housekeeping, REPL requires only very little effort in low-level checks. This might seem unintuitive at first; however, both **Pe** and **Ti** depend on a small set of inputs. As REPL is not guided by earlier failed checks and the solver generates many similar solutions, similar and infeasible solutions can be discarded without considerable effort by few distinct low-level checks (recall that we use a basic cache).

In Robotic Manipulation, the importance of low-level checks for solution feasibility is not as high as in Housekeeping. Therefore, few infeasible plans are produced without low-level checks, and using PRE in this domain achieves that all runs terminate within the time limit. The checks integrated with PRE do not need to be performed anymore with REPL. Therefore, the number of checks decreases in the Robotic Manipulation domain.

For GREPL, adding PRE decreases the number of low-level checks for both domains, because GREPL is a method where checks guide the search process so no failed check is encountered again.

Difference between FIRST *and* ALL If we compare the time required to find solutions and the number of infeasible solutions between the two domains in Figures 3 and 4, we observe that in Housekeeping there is only a small difference between FIRST and ALL, while in Robotic Manipulation initial feasible plans are found much earlier than enumeration ends, and the number of infeasible plans shows a similar difference.

This observation can be explained by the different levels of relevance of low-level checks for these domains. In Housekeeping, it takes significant effort to find an initial solution that does not violate any low-level check, and similar solutions can be found easily afterwards. On the contrary, in Robotic Manipulation, low-level checks play a less important role so finding the next feasible plan is as hard as finding the first feasible plan.

Memory Usage We measured peak memory usage over the whole runtime of each instance. The maximum memory usage for Housekeeping stayed below 2GB, for Robotic Manipulation below 5GB.

Memory usage increases drastically when using precomputation, as a larger amount of constraints must be processed by the solver. Apart from that, we could not observe significant differences in memory usage between different methods of integration.

In Housekeeping, memory usage of integration scenarios that include PRE uses around 750MB ($\pm$50) while not using PRE brings down memory usage to 425MB ($\pm$25). Similarly for Robotic Manipulation, using PRE requires on average 1900MB ($\pm$100) while not using PRE requires merely 600MB ($\pm$50).

9.2. 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 allow 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 Tables 7 and 8, and Figures 5 and 6.

9.2.1. Computational Effort

From the results presented in Tables 7 and 8 and Figure 5, we observe that integrating more low-level checks with INT than with GREPL generally increases efficiency.

The results of Housekeeping show a significant increase in computational 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.

For Robotic Manipulation there is also a significant increase in computational efficiency when integrating any low-level check with INT, compared to integrating all checks with GREPL. However, between the remaining checks and full integration, the picture is slightly

Table 7
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

Table 8
Robotic Manipulation: 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
Ro	**Pa**	**Mo**	#	#	#	%	sec	#	#	%	#	%	sec
GREPL	GREPL	GREPL	0.7	1564	1564	33.4	1937	1.0	2559	27.1	6872	59.6	3363
GREPL	GREPL	INT	0.0	156	1662	53.2	771	0.0	2589	77.9	5841	82.1	2680
INT	GREPL	INT	0.0	56	1151	44.8	736	0.0	2922	66.8	7839	82.7	2570
INT	INT	GREPL	0.0	188	336	13.3	300	0.3	2989	33.0	6971	85.3	2359
GREPL	INT	GREPL	0.0	111	224	11.3	296	0.0	2405	33.3	5818	84.5	2316
INT	GREPL	GREPL	0.0	120	151	5.6	279	0.0	3220	81.9	6084	83.7	2095
GREPL	INT	INT	0.0	6	169	5.8	272	0.0	2412	33.7	4795	83.3	2064
INT	INT	INT	0.0	0	171	3.7	267	0.0	2755	100.0	5800	82.8	1977

different from the one in Housekeeping. Full integration with INT is the most efficient combination; however, the computational efficiency does not improve much over levels of integration that are mixed between INT and GREPL. This observation can be explained by the structure of low-level checks in this domain: **Mo** requires motion planning that involves collision checks **Ro** and **Pa**. Motion planning in **Mo** is much more expensive than collision checks. Therefore, integrating only **Mo** with INT is the slowest level of integration; all other combinations improve upon that.

9.2.2. Solution Quality

Figure 6 depicts measurements from Tables 7 and 8 regarding solution quality compared to levels of integration.

In Housekeeping, according to Table 7, integrating only **Po** with INT leads to a significantly better solution quality for ALL (99.7% feasible plans) than integrating only **Pe** (31.6% feasible plans) or only **Ti** (33.4% feasible plans) 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

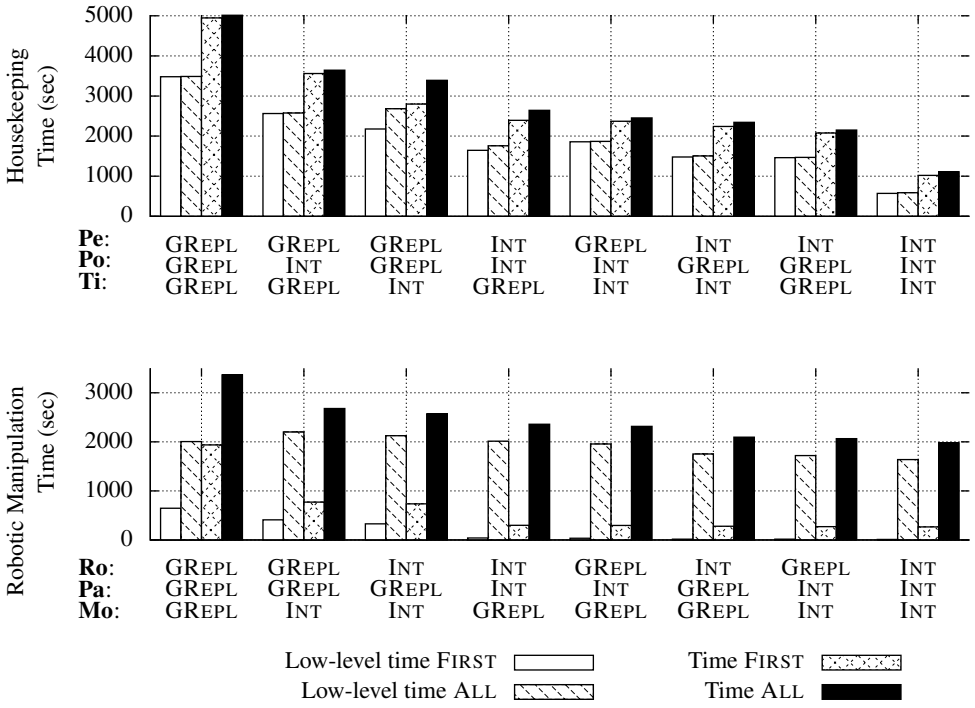

Fig. 5. Comparison of Computational Effort vs. Levels of Integration

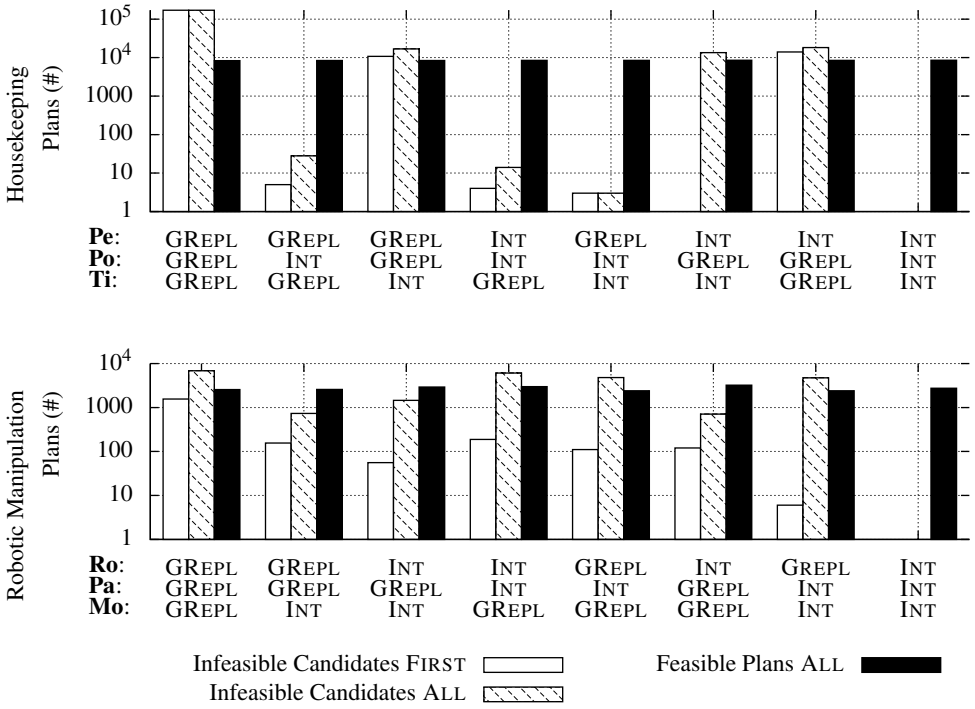

Fig. 6. Comparison of Solution Quality vs. Levels of Integration

nearly 100.0% feasible plans (in absolute numbers 3 and 14 infeasible plans, respectively).

For Robotic Manipulation, we observe that handling only **Mo** with GREPL, i.e., not integrating it, yields 33.0% feasible plans; only pure GREPL yields less number of feasible plans. From this observation, we can infer that **Mo** has the highest significance for solution quality among the Robotic Manipulation low-level checks.

If we compare the variation in number of infeasible plans between both domains, we see that Housekeeping has either small (< 30) or large (> 10000) values, while Robotic Manipulation shows a more gradual difference between numbers of infeasible plans. The reason for that is our limit of enumerating only 10000 plans for ALL: in Robotic Manipulation most instances have fewer feasible plans, hence also the lower average of feasible plans; on the contrary, several Housekeeping instances have much more than 10000 feasible plans, and these plans exhibit similar subsequences of actions. Therefore, in Housekeeping we can enumerate many similar plans in a short time, hence once a feasible plan is found, similar feasible plans can be found easily. On the other hand, in Robotic Manipulation, every enumerated plan requires significant additional search effort and additional verification of low-level checks.

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

10. Related Work

We discuss the related work in two parts: related work that integrates task planning with feasibility checks at the representation level, in a robotic domain; and related work that studies the usefulness of integration of task planning with feasibility checks, but at the search level.

10.1. Integrating Task Planning with Motion Planning at the Representation Level

Recently, there have been some studies [1,4,7,11,12, 24,34–36] on integrating discrete task planning and continuous motion planning, at the representation level. For each one of these studies, Table 9 provides information about the representation formalism, reasoner/planner, robotic domain, feasibility checks considered in the study.

The studies [4, 12, 34] consider the action description language $\mathcal{C}+$ [31] to represent robotic domains and use the causal reasoner CCALC [50] to compute hybrid plans. Caldiran et al. [4] studies mobile manipulation where two mobile robots manipulate a payload in a maze-like environment. In this domain, robots need to move concurrently among the obstacles while holding the endpoints of a payload. Two sorts of feasibility checks are considered: collision checks between robots/payloads and the obstacles at every state, and collision avoidance while the robots are moving from one state to another. The former checks are similar to **Pa** and **Ro**; INT is applied to integrate these checks into planning. The latter feasibility check is similar to **Mo**, but the pose of the payload is not considered since $C+$ and CCALC do not allow external atoms whose inputs include predicate names; so GREPL is applied to integrate this check into task planning. Erdem et al. [12] study robotic manipulation (as in our benchmarks), where two robots change the configurations of multiple payloads with respect to the specified goal. Feasibility checks of [4] (i.e., **Pa** and **Ro**) are considered with similar integration methods. Havur et al. [34] apply hybrid planning to a variation of Tower of Hanoi, where disks have some orientations, have to be carried by two robots, and their initial and goal configurations may be any possible configuration. Here, while the robots rotate a disk they are carrying, there may be some collisions between them. Therefore, the authors consider collision checks between robots, and apply GREPL to integrate these checks in task planning using CCALC.

The studies [1, 11, 35] formulate the robotic domains in ASP; Aker et al. [1, 11] use the ASP solver ICLINGO [25], whereas Havur et al. [35] use dlvhex to compute hybrid plans. Aker et al. [1,11] study the housekeeping domain (as in our benchmarks), considering the feasibility checks **Pe**, **Ti** and **Po**. INT is applied for **Ti** and **Pe**. Since ICLINGO (like CCALC) does not allow external atoms whose inputs include predicate names, INT could not be applied for **Po**; instead, GREPL is applied to integrate **Po** into task planning. Havur et al. [35] apply hybrid planning to rearrangement of objects on a cluttered surface with a mobile manipulator. Authors consider reachability of the robot arm, finding a proper location for the mobile base and feasibility of grasps using PRE; and collision checks among objects and the robot arm, collision free trajectories for placing and/or pushing objects among movable obstacles utilizing INT, since dlvhex allows external atoms whose inputs include predicate names. Furthermore, GREPL is used to ensure proper propagation of object locations from one state to another.

Table 9
Related work on integration of task planning with feasibility checks at the representation level.

Study	Formalism	Solver	Robotic domain	Feasibility checks (Integration method)
Caldiran et al. [4]	$C+$	CCALC	Mobile manipulation (with concurrency)	Collision checks (INT) Collision free trajectory with moveable objects (GREPL)
Erdem et al. [12]	$C+$	CCALC	Manipulation (with concurrency)	Collision checks (INT) Collision free trajectory with moveable objects (GREPL)
Havur et al. [34]	$C+$	CCALC	Tower of Hanoi (manipulation with concurrency)	Collision free trajectory for a plan (GREPL)
Aker et al. [1, 11]	ASP	ICLINGO	Housekeeping domain (navigation and manipulation with concurrency)	Collision free trajectory without moveable objects (INT) Temporal constraints over trajectory (INT) Collision free trajectory with moveable objects (GREPL)
Havur et al. [35]	ASP	dlvhex	Geometric rearrangement (mobile manipulation with concurrency)	Reachability and force closure checks (PRE) Collision free trajectory with moveable objects (INT) Collision free trajectory for a plan (GREPL)
Dornhege et al. [7]	PDDL/M	FF/M	Manipulation (without concurrency)	Collision checks (INT)
Hertle et al. [36]	PDDL/M	TFD/M	Room scanning domain (navigation)	Collision free trajectory without moveable objects (INT)
Gaschler et al. [24]	STRIPS extension	PKS	Bartenter domain (manipulation)	Collision free trajectory without moveable objects (PRE)

The studies [7, 36] extend the planning domain description language PDDL [23] to support external atoms, and modify the planners FF [37] and TFD [21] accordingly to compute hybrid plans; the extended PDDL is called PDDL/M, the extended versions of FF and TFD are called FF/M and TFD/M, respectively. Dornhege et al. [7] use PDDL/M and FF/M for hybrid planning in a manipulation domain where a robot aims to transfer objects from a table to a shelf. Depending on the continuous grasp direction, collisions may occur. The authors thus integrate collision checks into task planning; they use INT method for that. Hertle et al. [36] use PDDL/M and TFD/M for hybrid planning in a room scanning domain where a robot aims to scan all target locations in every room in a minimal amount of time. Checking whether there is collision-free path from one location to another one is integrated into task planning, using the INT method.

Gaschler et al. [24] use the PKS planner [57, 58] to apply hybrid planning to a manipulation domain, where a bartender robot detects bottles located on the bar and removes all empty bottles to a special dishwasher location at the right of the robot. The language of PKS is an extension of STRIPS [22]. To be able to grasp the bottles without any collisions, the feasibility check of collision-free trajectory is integrated in task planning. Due to the small number of bottles, the authors use PRE method for integration.

In the studies above, we see various applications of hybrid planning. However, there is no experimental evaluation over different integration methods.

10.2. A Systematic Analysis of Integration at the Search Level

Recently, Lagriffoul et al. [43] have investigated the usefulness of integration of task planning with geometric reasoning at the search level. In that study, geometric checks are embedded into symbolic search trees (for task planning) up to some depth D of the symbolic search tree. After depth D of the symbolic search tree, no geometric reasoning is considered until a goal state is reached. When a goal state is reached then the plan's feasibility is checked; if the plan is not feasible, then the search over the symbolic search tree continues. In such a platform, levels of integration is characterized by this depth. Effects of level of integration is studied by measuring the number of visited symbolic search states and the number of visited geometric configurations until a feasible plan is found.

For their experiments, the authors modify the planner JSHOP2 [53] according to the experimental framework described above. They consider a robotic domain that consists of picking cups from a table and putting them into a box on the table. The problem instances they experiment with involve two or three cups; the maximum makespan for a task plan (and thus maximum value for

D) is 8 and the maximum threshold for the number of geometric configurations is 200.

From the results of their experiments, the authors make several interesting observations. In particular, they observe that, as the value of *D* changes, there is a trade-off between the number of visited symbolic search states and the number of visited geometric configurations until a feasible plan is found. They suggest (a) a tight integration (i.e., a larger *D*) for domains with lower geometric success rate (i.e., the ratio of the average branching factor of the symbolic search tree with geometric checks to the average branching factor of the symbolic search tree without any geometric search), and (b) no tight integration for domains with high geometric success rate.

Lagriffoul et al.'s observations (a) are inline with our experimental results for interleaved method (INT). According to our results, if a geometric check plays a significant role on finding a feasible plan, then it is better to interleave this check in task planning. Indeed, as discussed above, the geometric check **Po** plays a significant role on the feasibility of a plan in Housekeeping, and integrating **Po** only with INT, leads to a significantly better solution quality (99.7% feasible plans, Table 7).

Lagriffoul et al.'s observations (b), on the other hand, do not confirm with our experimental results for INT. According to our results, it is still better to interleave a geometric check in task planning, even if it does not play as much significant role on finding a feasible plan. Indeed, although the geometric check **Pe** plays a less significant role on the feasibility of a plan in Housekeeping compared to **Po** (31.7% feasible plans, Table 7), integrating this check with INT only, leads to a significant improvement on computational efficiency (reducing the computation time from 4948 seconds to 2077 seconds, Table 7). Similar observations are made for the Manipulation domain as well, suggesting a tight integration with INT.

The different observations are not surprising, considering that the experiments are conducted with two different integration approaches (search level vs representation level). Also, in Lagriffoul et al.'s analysis, experiments are performed over one robotic domain and over instances with short makespans, effect of levels of integration on computational efficiency (computation time) is not considered, and effects of various combinations of levels of integration for feasibility checks are not investigated.

11. 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 and robotic manipulation domains 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 be used cautiously 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".

Acknowledgements

This work is partially supported by TUBITAK Grants 111E116, 113M422, 114E491 (ChistEra COACHES), 114E430.

Appendix

A. Housekeeping Domain Description

The Housekeeping domain used in our experiments is essentially the one described in [1, 11]. Let us briefly review the formulation of this domain in ASP.

Planning problem: input and output. An initial state of a planning problem instance is described as a set of facts using simple fluents of the forms

- $at(th,x,y,t)$ ("robot/object th is at location (x,y) at time step t") and
- $connected(r,e,t)$ ("robot r is attached to the endpoint e of an object at time step t")

for $t = 0$.

The goal is described as a set of constraints

$$\leftarrow \textbf{not } tidy(maxstep)$$
$$\leftarrow \textbf{not } free(maxstep)$$
$$\leftarrow \textbf{not } elapsed\_time(maxtime, maxstep)$$

using auxiliary fluents of the forms

- $tidy(t)$ ("the house is tidy at time step t"),
- $free(t)$ ("no robot is attached to any object, so they are all free, at time step t") and
- $elapsed\_time(g,t)$ ("estimated elapsed time from time step 0 till time step t is g time units")

where *maxstep* is the makespan and *maxtime* is the estimated total plan duration. The auxiliary fluents are defined in terms of the simple fluents as in [1, 11]. The definition of *tidy* utilizes commonsense knowledge extracted from the commonsense knowledge base CONCEPTNET [48] by means of an external atom.

A plan is a sequence of actions of the forms

- $goto(r,x,y,t_a)$ ("robot r moves to location (x,y) at time step t_a")
- $attach(r,t_a)$ with the attribute $attach\_point(r,e,t_a)$ ("robot r attaches to the endpoint e of an object at time step t_a")
- $detach(r,t_a)$ ("robot r detaches from the object that it is attached to, at time step t_a")

Once a maximum makespan *maxmakespan* for a plan is specified, an optimal plan of length *maxstep* is computed by trying to find a plan using the solver dlvhex with $maxstep = 1,2,3,...,maxmakespan$.

Preconditions and direct effects of actions. We describe direct effects and preconditions of the action $goto(r,x,y,t_a)$ by the following rules:

$$at(r,x,y,t_a+1) \leftarrow goto(r,x,y,t_a)$$
$$\leftarrow goto(r,x,y,t_a), occupied(x,y)$$
$$\leftarrow goto(r,x,y,t_a), at(r,x,y,t_a).$$

The first rule above describes the change in the robot's location as a direct effect of this action, whereas the other rules describe the preconditions of this action (i.e., a robot can neither move to an occupied location nor to its current location).

The direct effects and preconditions of the action *attach* are defined by the following rules:

$$connected(r,e,t_a+1) \leftarrow attach(r,t_a), attach\_point(r,e,t_a)$$
$$\leftarrow attach(r,t_a), connected(r,e,t_a)$$
$$\leftarrow attach(r,t_a), attach\_point(r,e,t_a), different\_loc(r,e,t_a).$$

The first rule above described that the robot is connected to the endpoint of the object after the execution of this action. The other rules describe the preconditions of executing this action (i.e., a robot cannot attach to an object if it is already connected to some object, or if the object is at a remote location different from the location of the robot).

Similarly the action *detach* is described by the following rules:

$$\neg connected(r,e,t_a+1) \leftarrow detach(r,t_a), connected(r,e,t_a)$$
$$\leftarrow detach(r,t_a), free\_robot(r,t_a).$$

Ramifications. Indirect effects of actions can be defined by rules as well. For instance, an indirect effect of the action $attach(r,t_a)$ with $attach\_point(r,e,t_a)$ is that the location of the endpoint e is equal to the location of the robot r. This ramification can be expressed by the rules:

$$at(e,x,y,t) \leftarrow connected(r,e,t), at(r,x,y,t).$$

Note that these rules also describe a ramification of the $goto(r,x,y,t_a)$ action.

Another ramification of the $goto(r,x,y,t_a)$ action is that if a robot/object is at a location that it is not anywhere else:

$$\neg at(th,x,y,t) \leftarrow at(th,x_1,y_1,t) \ \ (x \neq x_1)$$
$$\neg at(th,x,y,t) \leftarrow at(th,x_1,y_1,t) \ \ (y \neq y_1).$$

Commonsense law of inertia. We describe the commonsense law of inertia by the following rules:

$$\begin{array}{l} at(th,x,y,t_a+1) \leftarrow \textbf{not}\ \neg at(th,x,y,t_a+1), \\ \quad at(th,x,y,t_a) \\ \neg at(th,x,y,t_a+1) \leftarrow \textbf{not}\ at(th,x,y,t_a+1), \\ \quad \neg at(th,x,y,t_a) \\ connected(r,e,t_a+1) \leftarrow \textbf{not}\ \neg connected(r,e,t_a+1), \\ \quad connected(r,e,t_a) \\ \neg connected(r,e,t_a+1) \leftarrow \textbf{not}\ connected(r,e,t_a+1), \\ \quad \neg connected(r,e,t_a). \end{array}$$

State constraints. The state constraint that no two robots/objects occupy the same location at any time step can be expressed by the following rules:

$$\leftarrow at(e_1,x,y,t), at(e_2,x,y,t)\ \ (e_1 < e_2).$$

We can describe the constraint that a robot cannot be attached to multiple objects at the same time as follows:

$$\leftarrow connected(r,e_1,t), connected(r,e_2,t)\ \ (e_1 < e_2).$$

We can describe the state constraint that the endpoints of the objects should be aligned horizontally or vertically as follows:

$$\begin{array}{l} \leftarrow at(e_1,x_1,y_1,t), at(e_2,x_2,y_2,t), \\ \quad belongs(e_1,o), belongs(e_2,o) \\ \quad (e_1 < e_2,\ (x_1-x_2)^2+(y_1-y_2)^2 \neq 1). \end{array}$$

Concurrency constraints. A robot cannot simultaneously move, and attach or detach:

$$\begin{array}{l} \leftarrow goto(r,x,y,t_a), attach(r,t_a) \\ \leftarrow goto(r,x,y,t_a), detach(r,t_a). \end{array}$$

Temporal constraints. To be able to ensure that the elapsed time is less than a given specific time unit, we introduce further auxiliary fluents of the form $robot\_time(r,d_e,t_a+1)$ ("estimated time for a robot r to complete its action started at time step t_a id d_e time units").

For $goto(r,x,y,t_a)$ action, among all possible values for its duration,

$$\begin{array}{l} robot\_time(r,d_e,t_a+1) \vee \neg robot\_time(r,d_e,t_a+1) \leftarrow \\ \quad goto(r,x,y,t_a) \\ \neg robot\_time(r,d_1,t) \leftarrow robot\_time(r,d_2,t)\ \ (d_1 \neq d_2). \end{array}$$

the duration of the action is estimated over the length of a continuous trajectory from the robot's current location to (x,y). The estimated duration is computed externally using a motion planner and the result of this external computation is embedded into the following constraint via the external atom $\&time\_estimate[goto,at,robot\_time]()$ to define the duration of the $goto(r,x,y,t_a)$ action:

$$\leftarrow \textbf{not}\ \&time\_estimate[goto,at,robot\_time]().$$

Note that this constraint realizes the low-level check **Ti**.

For other actions, we define their durations as constants:

$$\begin{array}{l} robot\_time(r,1,t_a+1) \leftarrow detach(r,t_a) \\ robot\_time(r,1,t_a+1) \leftarrow attach(r,t_a). \end{array}$$

Once durations of each action is defined, we accumulate the maximum of the durations of actions executed at time step t_a to find the elapsed time so far at t_a+1. As noted above, the elapsed time is denoted by an auxiliary fluent of the form $elapsed\_time(g,t_a)$. This fluent is defined recursively as follows:

$$\begin{array}{l} nmax\_robot\_time(d_1,t) \leftarrow robot\_time(r_1,d_1,t), \\ \quad robot\_time(r_2,d_2,t)\ \ (d_1 < d_2) \\ elapsed\_time(g_1+d,t_a+1) \leftarrow elapsed\_time(g_1,t_a), \\ \quad robot\_time(r,d,t_a+1), \textbf{not}\ nmax\_robot\_time(d,t_a). \end{array}$$

Geometric constraints. As described in Section 2, the following constraint ensures that, 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.

$$\leftarrow \textbf{not}\ \&path\_exists[goto,at]().$$

Note that this geometric constraint utilizes the low-level feasibility check **Pe**.

The following geometric constraint utilizes the low-level feasibility check **Po**.

$$\leftarrow \textbf{not}\ \&path\_exists\_obstacles[goto,at]().$$

In addition to the functionality provided by *path_exists*, the function *path_exists_obstacles* consider obstacles specified in predicate *at* in motion planning. This makes the check more complex and dependent on a larger set of fluents in the plan candidate, therefore caching is not as effective and precomputation is not feasible.

B. Robotic Manipulation Domain Description

The Manipulation domain used in our experiments is essentially the one described in [12]. Let us briefly review the formulation of this domain in ASP.

Planning problem: input and output. An initial state of a planning problem instance is described by a set of facts using the simple fluents of the form

- $xpos(r,x,t)$, $ypos(r,y,t)$, and $dir(r,d,t)$ that specify position (x,y) and rotation d of robot r at time step t,
- $xpay(p,x,t)$ and $ypay(p,y,t)$ that specify positions (x,y) of payload endpoint p at time step t (note that each payload k has two endpoints labeled as $2k$ and $2k-1$), and
- $holding(r,p,t)$ that specifies whether robot r is holding payload endpoint p at time step t.

We specify the goal by constraints using auxiliary fluents. For example, consider the goal that payload 2 is located at $(3,2)$ with endpoint 3 and at $(8,2)$ with endpoint 4 and that no robot is holding any object. This goal is described as follows:

$$\begin{array}{l}\leftarrow \textbf{not}\, goal\\ goal \leftarrow xpay(1,3,t), ypay(1,2,t), xpay(2,8,t),\\ \quad ypay(2,2,t), \textbf{not}\, holdingsth(t)\\ holdingsth(t) \leftarrow holding(r,p,t).\end{array}$$

A plan is a sequence of actions of the forms

- $move(r,d,n,t_a)$ ("robot r moves in direction d by n units at time step t_a")
- $pick(r,p,t_a)$ ("robot r picks payload p at time step t_a")
- $drop(r,t_a)$ ("robot r drops the payload it is holding, at time step t_a")

Once a maximum makespan *maxmakespan* for a plan is specified, an optimal plan of length *maxstep* is computed by trying to find a plan using the solver dlvhex with $maxstep = 1,2,3,...,maxmakespan$.

Preconditions and direct effects of actions. The direct effects of moving can be defined by the following rules:

$$\begin{array}{l}xpos(r,x_1+n,t_a+1) \leftarrow move(r,right,n,t_a),\\ \quad xpos(r,x_1,t_a)\\ xpos(r,x_1-n,t_a+1) \leftarrow move(r,left,n,t_a),\\ \quad xpos(r,x_1,t_a)\\ ypos(r,y_1+n,t_a+1) \leftarrow move(r,up,n,t_a),\\ \quad ypos(r,y_1,t_a)\\ ypos(r,y_1-n,t_a+1) \leftarrow move(r,down,n,t_a),\\ \quad ypos(r,y_1,t_a).\end{array}$$

A robot cannot move if it is the only one holding a payload. This precondition is described by the following constraint

$$\begin{array}{l}\leftarrow mover(r,t_a), holding(r,p,t_a), samePayload(p,p_1),\\ \quad \#count\{r_1 : holding(r_1,p_1,t_a)\} \leq 0.\end{array}$$

that utilizes the auxiliary fluents $mover(r,t)$ ("robot r moves at time step t") and $samePayload(p,p_1)$ ("p and p_1 are the endpoints of the same payload") defined as follows:

$$\begin{array}{l}mover(r,t_a) \leftarrow move(r,d,n,t_a)\\ samepayload(2k,2k+1)\\ samepayload(2k+1,2k).\end{array}$$

We define the direct effects and preconditions of a robot's action of picking up an endpoint of a payload as follows:

$$\begin{array}{l}holding(r,p,t_a+1) \leftarrow pick(r,p,t_a)\\ \leftarrow holding(r,p,t_a), pick(r,p,t_a)\\ \leftarrow pick(r,p,t_a), xpos(r,x,t_a), xpay(p,x_1,t_a)\ \ (x \neq x_1)\\ \leftarrow pick(r,p,t_a), ypos(r,y,t_a), ypay(p,y_1,t_a)\ \ (y \neq y_1).\end{array}$$

We define the direct effects and preconditions of a robot's action of dropping the endpoint of a payload that it is holding, as follows:

$$\begin{array}{l}\neg holding(r,p,t_a+1) \leftarrow drop(r,t_a), holding(r,p,t_a)\\ \leftarrow drop(r,t_a), \#count\{p : holding(r,p,t_a)\} \leq 0.\end{array}$$

Ramifications. The position of a payload's endpoint p is the same as the position of the robot r who is holding p. Representing this statement formally also helps describe the ramifications of actions (e.g., whenever a robot moves to a new location then the position of the payload it is holding also changes accordingly). Here are the rules representing these ramifications:

$$\begin{array}{l}xpay(p,x,t) \leftarrow holding(r,p,t), xpos(r,x,t)\\ ypay(p,y,t) \leftarrow holding(r,p,t), ypos(r,y,t).\end{array}$$

Furthermore, holding an endpoint of a payload entails that the robot does not hold any other endpoint:

$$\neg holding(r,p_2,t) \leftarrow holding(r,p_1,t)\ \ (p_1 \neq p_2).$$

Commonsense law of inertia. We define inertia for fluents *holding*, *xpos*, *ypos*, *xpay*, and *ypay* by rules similar to the ones . For efficiency reasons we use true negation only for the fluent *holding*:

$$holding(r,p,t_a+1) \leftarrow holding(r,p,t_a), \\ \quad \mathbf{not}\, \neg holding(r,p,t_a+1)$$
$$\neg holding(r,p,t_a+1) \leftarrow \neg holding(r,p,t_a), \\ \quad \mathbf{not}\, holding(r,p,t_a+1).$$

Concurrency constraints. We disallow concurrency of picking, dropping, and moving.

$$picker(r,t_a) \leftarrow pick(r,p,t_a)$$
$$\leftarrow mover(r,t_a), picker(r,t_a)$$
$$\leftarrow mover(r,t_a), drop(r,t_a).$$

State constraints. We require that payloads are located horizontally or vertically on the grid.

$$\leftarrow \mathbf{not}\, beingCarried(k,t), \#count\{o : orientation(k,o,t)\} \leq 0$$

where the auxiliary fluent $beingCarried(k,t)$ is defined as follows:

$$beingCarried(k,t) \leftarrow holding(r,p,t), endpointOf(k,p), \\ holding(r,p_1,t), endpointOf(k,p_1) \;\; (p < p_1).$$

Furthermore we ensure that robots do not hold different payloads at the same time, and that a payload is held by two robots or by none.

$$\leftarrow holding(r,p_1,t), samePayload(p_1,p_2), \\ \#count\{r_2 : holding(r_2,p_2,t)\} \leq 0 \;\; (p_1 \neq p_2)$$
$$\leftarrow holding(r,p_1,t), holding(r,p_2,t) \;\; (p_1 < p_2).$$

Geometric constraints. We require that payloads do not overlap with each other. Note that this constraint excludes many cases where low-level feasibility checks (i.e., collision checks) would fail.

$$\leftarrow orientation(k_1,v,t), \mathbf{not}\, beingCarried(k_1,t), \\ min_X(k_1,x,t), min_Y(k_1,y_1,t), \\ orientation(k_2,v,t), \mathbf{not}\, beingCarried(k_2,t), \\ min_X(k_2,x,t), min_Y(k_2,y_2,t) \\ (1 \leq k_1 < k_2 \leq maxP, abs(y_2 - y_1) < lengthP)$$
$$\leftarrow orientation(k_1,h,t), \mathbf{not}\, beingCarried(k_1,t), \\ min_X(k_1,x_1,t), min_Y(k_1,y,t), \\ orientation(k_2,h,t), \mathbf{not}\, beingCarried(k_2,t), \\ min_X(k_2,x_2,t), min_Y(k_2,y,t) \\ (1 \leq k_1 < k_2 \leq maxP, abs(x_2 - x_1) < lengthP)$$
$$\leftarrow orientation(k_1,v,t), \mathbf{not}\, beingCarried(k_1,t), \\ min_X(k_1,x_1,t), min_Y(k_1,y_1,t), \\ orientation(k_2,h,t), \mathbf{not}\, beingCarried(k_2,t), \\ min_X(k_2,x_2,t), min_Y(k_2,y_2,t) \\ (k_1 \neq k_2, x_2 \leq x_1 \leq x_2 + lengthP, \\ y_1 \leq y_2 \leq y_1 + lengthP).$$

Here the auxiliary fluents are defined as follows:

$$min_X(k,x_1,t) \leftarrow endpointOf(k,p), endpointOf(k,p_1), \\ xpay(p,x_1,t), xpay(p_1,x_2,t) \;\; (p \neq p_1, x_1 \leq x_2)$$
$$min_Y(k_1,y_1,t) \leftarrow endpointOf(k,p), endpointOf(k,p_1), \\ ypay(p,y_1,t), ypay(p_1,y_2,t) \;\; (p \neq p_1, y_1 \leq y_2)$$
$$orientation(k,h,t) \leftarrow ypay(p,y_1,t), ypay(p_1,y_1,t), \\ endpointOf(k,p), endpointOf(k,p_1) \;\; (p < p_1)$$
$$orientation(k,v,t) \leftarrow xpay(p,x_1,t), xpay(p_1,x_1,t), \\ endpointOf(k,p), endpointOf(k,p_1) \;\; (p < p_1).$$

To make physical feasibility more likely, we restrict movements of robots when holding the payload in a non-axis-aligned way: the robots do not have to hold the payloads from their endpoints exactly but "sufficiently close to" the endpoints within some "tolerance" value.

$$\leftarrow xpay(p_1,x_1,t), xpay(p_2,x_2,t), ypay(p_1,y_1,t), \\ ypay(p_2,y_2,t), samePayload(p_1,p_2)$$
$$\leftarrow xpay(p_1,x_1,t), xpay(p_2,x_2,t), ypay(p_1,y_1,t), \\ ypay(p_2,y_2,t), samePayload(p_1,p_2).$$

where $(x_1 \cdot x_2)^2 + (y_1 \cdot y_2)^2 > lengthP^2 + tolerance$ and $p_1 < p_2$.

The low-level feasibility check **Ro** is ensured by the following constraint:

$$\leftarrow \mathbf{not}\, \&robot\_collision\_free[xpos, ypos, dir]().$$

Here the external function *robot_collision_free* gets as input the set of atoms $xpos(r,x,t)$, $ypos(r,y,t)$, and $dir(r,d,t)$ that are true in the current plan candidate, extracts for each time step t all coordinate/direction triples (x,y,d) of a robot r; and returns true if and only if for each time step the specified robot poses are collision-free with other robot poses and with the environment.

Similarly, the following constraint realizes **Pa**:

$$\leftarrow \mathbf{not}\, \&payload\_collision\_free[xpay, ypay]().$$

Finally the **Mo** low-level check is realized in the following constraint

$$\leftarrow \mathbf{not}\, motion\_feasible[xpos, ypos, dir, xpay, ypay, \\ holding, move]().$$

From the atoms in the extensions of predicates *xpos*, *ypos*, *dir*, *xpay*, *ypay*, *holding*, *move*, the low-level feasibility check function *motion_feasible* extracts at every time step t_a the positions and orientations of the robots, the positions of the payloads, which robot is

holding which payload, and which robot is moving to where. This information provides the initial configuration and the goal configuration for a motion planner; note that the information extracted from *holding* fluent determines whether motion planning should be done with two robots carrying a payload, with one robot, or with two robots that are not connected to payloads. Then, using a motion planner, $motion\_feasible$ returns true if the robots' move actions are feasible with respect to the motion planner's output.

C. Experimental Results with the ASP Solver CLASP

We have performed some experiments using the ASP solver CLASP with GRINGO, over the housekeeping domain (with the feasibility checks **Pe** and **Ti**) and the robotic manipulation domain (with the feasibility checks **Ro** and **Pa**), to compare PRE with GREPL. Experiments were performed one run on a Linux server with 32 2.4GHz Intel® E5-2665 CPU cores and 64GB memory.

The experimental results are summarized in Table 10. Due to the more coarse interface of CLASP compared to dlvhex, only some parts of the domains could be modeled; thus, some integration methods could not be tested. However, the results confirm the conclusions given in Section 11 suggested by the experimental results presented in Section 9.

References

[1] E. Aker, V. Patoglu, and E. Erdem. Answer set programming for reasoning with semantic knowledge in collaborative housekeeping robotics. In *Proc. of SYROCO*, 2012.

[2] G. Brewka, T. Eiter, and M. Truszczynski. Answer set programming at a glance. *Commun. ACM*, 54(12):92–103, 2011.

[3] T. Bylander. The computational complexity of propositional STRIPS planning. *Artificial Intelligence*, 69(1–2):165–204, 1994.

[4] 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. of LPNMR*, 2009.

[5] H. Choset, K. M. Lynch, S. Hutchinson, G. A. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun. *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, 2005.

[6] Y. Dimopoulos, B. Nebel, and J. Koehler. Encoding planning problems in nonmonotonic logic programs. In *Proc. of ECP*, 1997.

[7] C. Dornhege, P. Eyerich, T. Keller, S. Trüg, M. Brenner, and B. Nebel. Semantic attachments for domain-independent planning systems. In *Proc. of ICAPS*, 2009.

[8] T. Eiter, M. Fink, G. Ianni, T. Krennwallner, C. Redl, and P. Schüller. A Model Building Framework for Answer Set Programming with External Computations. *Theory and Practice of Logic Programming*, 2015 (in press).

[9] T. Eiter, G.Ianni, R.Schindlauer, and H.Tompits. Effective integration of declarative rules with external evaluations for Semantic-Web reasoning. In *Proc. of ESWC*, 2006.

[10] 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.

[11] 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.

[12] 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.

[13] E. Erdem, K. Haspalamutgil, V. Patoglu, and T. Uras. Causality-based planning and diagnostic reasoning for cognitive factories. In *Proc. 17th IEEE Int. Conf. Emerging Technologies and Factory Automation (ETFA)*, 2012.

[14] 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.

[15] E. Erdem, V. Patoglu, and Z. G. Saribatur. Integrating hybrid diagnostic reasoning in plan execution monitoring for cognitive factories with multiple robots. In *Proc. of ICRA*, 2015. Finalist for Best Conference Paper Award, Finalist for Best Cognitive Robotics Paper Award.

[16] 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*, 13(4–5):831–846, 2013.

[17] 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.

[18] 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.

[19] E. Erdem, V. Patoglu, and P. Schüller. A systematic analysis of levels of integration between high-level task planning and low-level feasibility checks. In *Proc. of RCRA*, 2014.

[20] K. Erol, D. S. Nau, and V. S. Subrahmanian. Complexity, decidability and undecidability results for domain-independent planning. *Artif. Intell.*, 76(1–2):75–88, 1995.

[21] P. Eyerich, R. Mattmüller, and G. Röger. Using the context-enhanced additive heuristic for temporal and numeric planning. In *Proc. of ICAPS*, 2009.

[22] R. E. Fikes and N. J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. In *Proc. of IJCAI*, 1971.

[23] 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.

[24] A. Gaschler, R. P. A. Petrick, M. Giuliani, M. Rickert, and A. Knoll. KVP: A knowledge of volumes approach to robot task planning. In *Proc. of IROS*, 2013.

Table 10
Experiments with CLASP and GRINGO

Integration	FIRST			ALL					
	Quality		Efficiency	Quality			Efficiency		
	Timeouts	Infeasible Plans	FIRST Time	Timeouts	Feasible Plans	Feasible Plans	Low-Level Checks	Low-Level Time	ALL Time
Housekeeping **Pe** & **Ti**	#	#	sec	#	#	%	#	sec	sec
GREPL	2	44	3126	13	1575	93.6	657	307	5641
PRE	0	0	3479	12	43790	100.0	8192	3256	6272
Robotic Manipulation **Ro** & **Pa**	#	#	sec	#	#	%	#	sec	sec
GREPL	2	11	2007	3	621	94.2	129	7	3242
PRE	0	0	888	0	652	100.0	29282	238	974

[25] M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, and S. Thiele. Engineering an incremental ASP solver. In *Proc. of ICLP*, 2008.

[26] M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub. clasp: A conflict-driven answer set solver. In *Proc. of LPNMR*, 2007.

[27] M. Gebser, T. Schaub, and S. Thiele. Gringo : A new grounder for answer set programming. In *Proc. of LPNMR*, 2007.

[28] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Proc. of ICLP*, 1988.

[29] M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9:365–385, 1991.

[30] M. Gelfond and V. Lifschitz. Action languages. *ETAI*, 2:193–210, 1998.

[31] E. Giunchiglia, J. Lee, V. Lifschitz, N. McCain, and H. Turner. Nonmonotonic causal theories. *Artificial Intelligence (AIJ)*, 153:2004, 2004.

[32] 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.

[33] K. Hauser and J.-C. Latombe. Integrating task and PRM motion planning: Dealing with many infeasible motion planning queries. In *Workshop on Bridging the Gap between Task and Motion Planning at ICAPS*, 2009.

[34] G. Havur, K. Haspalamutgil, C. Palaz, E. Erdem, and V. Patoglu. A case study on the tower of hanoi challenge: Representation, reasoning and execution. In *Proc. of ICRA*, 2013.

[35] G. Havur, G. Ozbilgin, E. Erdem, and V. Patoglu. Geometric rearrangement of multiple movable objects on cluttered surfaces: A hybrid reasoning approach. In *Proc. of ICRA*, 2014.

[36] A. Hertle, C. Dornhege, T. Keller, and B. Nebel. Planning with semantic attachments: An object-oriented view. In *Proc. of ECAI*, 2012.

[37] J. Hoffmann and B. Nebel. The FF planning system: Fast plan generation through heuristic search. *J. Artif. Intell. Res.*, 14:253–302, 2001.

[38] L. P. Kaelbling and T. Lozano-Pérez. Hierarchical task and motion planning in the now. In *Proc. of ICRA*, 2011.

[39] L. P. Kaelbling and T. Lozano-Pérez. Integrated task and motion planning in belief space. *International Journal of Robotics Research*, 32(9–10):1194–1227, 2013.

[40] H. A. Kautz and B. Selman. Planning as satisfiability. In *Proc. of ECAI*, 1992.

[41] J. Kuffner Jr and S. LaValle. RRT-connect: An efficient approach to single-query path planning. In *Proc. of ICRA*, volume 2, 2000.

[42] F. Lagriffoul, D. Dimitrov, J. Bidot, A. Saffiotti, and L. Karlsson. Efficiently combining task and motion planning using geometric constraints. *International Journal of Robotics Research*, 2014.

[43] F. Lagriffoul, L. Karlsson, J. Bidot, and A. Saffiotti. Combining task and motion planning is not always a good idea. In *Proc. RSS 2013 on Combined Robot Motion Planning and AI Planning for Practical Applications*, 2013.

[44] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006. Available at http://planning.cs.uiuc.edu/.

[45] V. Lifschitz. Action languages, answer sets and planning. In *In The Logic Programming Paradigm: a 25-Year Perspective*, pages 357–373. Springer Verlag, 1999.

[46] V. Lifschitz. Answer set programming and plan generation. *Artificial Intelligence*, 138:39–54, 2002.

[47] V. Lifschitz. What is answer set programming? In *Proc. of AAAI*, 2008.

[48] H. Liu and P. Singh. ConceptNet: A practical commonsense reasoning toolkit. *BT Technology Journal*, 22, 2004.

[49] 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.

[50] N. McCain and H. Turner. Causal theories of action and change. In *Proc. of AAAI/IAAI*, 1997.

[51] J. Mccarthy and P. J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In *Machine Intelligence*, pages 463–502. Edinburgh University Press, 1969.

[52] D. Mcdermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins. Pddl - the planning domain definition language. Technical Report TR-98-003, Yale Center for Computational Vision and Control,, 1998.

[53] D. S. Nau, H. Muñoz-Avila, Y. Cao, A. Lotem, and S. Mitchell. Total-order planning with partially ordered subtasks. In *Proc. of IJCAI*, 2001.

[54] I. Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence*, 25:241–273, 1999.

[55] 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.

[56] J. Pan, S. Chitta, and D. Manocha. FCL: A general purpose library for collision and proximity queries. In *Proc. of ICRA*, 2012.

[57] R. P. A. Petrick and F. Bacchus. A knowledge-based approach to planning with incomplete information and sensing. In *Proc. of AIPS*, 2002.

[58] R. P. A. Petrick and F. Bacchus. Extending the knowledge-based approach to planning with incomplete information and sensing. In *Proc. of ICAPS*, 2004.

[59] E. Plaku. *From High-Level Tasks to Low-Level Motions: Motion Planning for High-Dimensional Nonlinear Hybrid Robotic Systems*. PhD dissertation, Rice University, TX, USA, July 2008.

[60] E. Plaku. Planning in discrete and continuous spaces: From LTL tasks to robot motions. In *Proc. of TAROS*, 2012.

[61] E. Plaku and G. D. Hager. Sampling-based motion and symbolic action planning with geometric and differential constraints. In *Proc. of ICRA*, 2010.

[62] J. Reif. Complexity of the mover's problem and generalizations. In *Proc. of SOCS*, 1979.

[63] 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.

[64] S. Srivastava, E. Fang, L. Riano, R. Chitnis, S. Russell, and P. Abbeel. Combined task and motion planning through an extensible planner-independent interface layer. In *Proc. of ICRA*, 2014.

[65] V. S. Subrahmanian and C. Zaniolo. Relating stable models and AI planning domains. In *Proc. of ICLP*, 1995.

[66] I. A. Sucan and S. Chitta. "MoveIt!", [Online] Available: http://moveit.ros.org.

[67] I. A. Sucan, M. Moll, and L. E. Kavraki. The open motion planning library. *Robotics & Automation Magazine, IEEE*, 19(4):72–82, 2012.

[68] J. Tiihonen, T. Soininen, and R. Sulonen. A practical tool for mass-customising configurable products. In *Proc. of ICED*, 2003.

[69] R. W. Weyhrauch. Prolegomena to a theory of formal reasoning. Technical report, Stanford University, 1978.

[70] J. Wolfe, B. Marthi, and S. Russell. Combined task and motion planning for mobile manipulation. In *Proc. of ICAPS*, 2010.