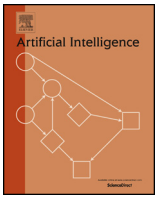
Artificial Intelligence 319 (2023) 103902

Contents lists available at ScienceDirect

Artificial Intelligence

journal homepage: www.elsevier.com/locate/artint



# Hybrid planning for challenging construction problems: An Answer Set Programming approach

Faseeh Ahmad [a,1], Volkan Patoglu [b], Esra Erdem [c,*]

[a] Lund University, Robotics and Semantic Systems, Lund, Sweden
[b] Sabanci University, Mechatronics Engineering, Istanbul, Turkey
[c] Sabanci University, Computer Science and Engineering, Istanbul, Turkey

ARTICLE INFO

*Article history:*
Received 12 December 2020
Received in revised form 2 February 2023
Accepted 3 March 2023
Available online 9 March 2023

*Keywords:*
Construction problems
Hybrid planning
Answer Set Programming

ABSTRACT

We study construction problems where multiple robots rearrange stacks of prefabricated blocks to build stable structures. These problems are challenging due to ramifications of actions, true concurrency, and requirements of supportedness of blocks by a surface or a robot and stability of the overall structure at all times. We propose a general elaboration tolerant method to solve a wide range of construction problems, based on the knowledge representation and reasoning paradigm of Answer Set Programming. This method not only (i) determines a stable final configuration of the structure, but also (ii) computes the order of manipulation tasks for multiple autonomous robots to build the structure from an initial configuration, (iii) while simultaneously ensuring the requirements of supportedness and stability at all times. We prove the soundness and completeness of our method with respect to these properties. We introduce a set of challenging construction benchmark instances, including construction of (uneven) bridges and overhangs, and discuss the usefulness of our framework over these instances. Furthermore, we perform experiments to investigate the computational performance of our hybrid method, and demonstrate the applicability of our method using a bimanual Baxter robot.



## 1. Introduction

The construction industry relies on manual labor as its primary source of productivity, while robots promise to dramatically improve the speed and quality of construction work by automating repetitive and labor intensive tasks [7]. Even though automation can improve the efficiency and the productivity of certain construction tasks, the design of the structure to be built, planning of the robot motions, and a proper ordering of robot actions are still decided manually in these approaches. Robotics will have a major impact on the construction industry, if these reasoning tasks can also be performed automatically. For instance, it would be very beneficial if a group of autonomous search and rescue robots could automatically build bridges in a disaster zone, by rearranging stacks of prefabricated building materials that are accessible to them.

We view construction problems as hybrid planning problems where discrete/logical task planning is combined with continuous/probabilistic feasibility checkers: find a plan (i.e., a sequence of feasible actions) to obtain a final stable configuration of prefabricated objects satisfying some goal conditions, from a given initial configuration.

* Corresponding author.
*E-mail addresses:* faseeh.ahmad@cs.lth.se (F. Ahmad), volkan.patoglu@sabanciuniv.edu (V. Patoglu), esra.erdem@sabanciuniv.edu (E. Erdem).

[1] F. Ahmad's work was carried out during his graduate studies at Sabanci University.

https://doi.org/10.1016/j.artint.2023.103902
0004-3702/

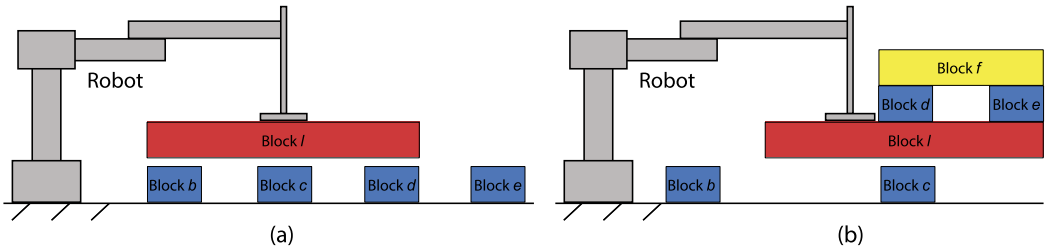



**Fig. 1.** Ramifications involved in construction problems.

### 1.1. Motivating challenges

Robot construction problems involve several challenges from the perspective of planning, knowledge representation and reasoning, and robotics. These changes include (i) representation of sophisticated ramifications of actions, (ii) reasoning about global constraints to eliminate spurious structures during planning, (iii) integrating low-level feasibility checks into planning to ensure stability of constructions and feasibility of plan executions, and (iv) elaboration tolerant representation of variations of robot construction problems.

*Recursive ramifications* Construction problems involve a variety of sophisticated ramifications that are challenging to represent.

For instance, consider Fig. 1(a), where a robot places a large block $l$ on top of a small block $b$. As a direct effect of this action, $l$ becomes on $b$. As a ramification of this action, $l$ is located on top of other small blocks close to $b$, such as $c$ and $d$.

Let us now consider Fig. 1(b), where a robot is moving a subassembly of blocks $l$, $d$, $e$ and $f$ from one location $b$ to another location $c$. As a direct of effect of this action, the base block $l$ becomes on $c$. As ramifications of this action, block $l$ is not on $b$ anymore, none of the blocks included in the subassembly are supported by $b$ anymore, and all blocks in the subassembly are supported by $c$ now.

Representing these ramifications require concepts that are recursively defined. For instance, in the first example, to identify which blocks are close to $b$ and will become under $l$, the relative positions of blocks are not sufficient. It is necessary to define the global positions of blocks from their relative positions; such a definition is recursive. In the second example, it is necessary to define which blocks are supported by which other blocks in a construction, like a subassembly being carried; such a definition is recursive as well. The capability of defining recursive concepts, like the transitive closure of "being immediately on top of another block", is needed for representing sophisticated ramifications.

Representing such ramifications is challenging, e.g., for PDDL-based planning: Thiebaux et al. [93, Theorem 3] prove that "Unless EXPTIME = PSPACE, there is no compilation scheme from PDDLX (even restricted to DATALOG axioms) to PDDL preserving plan size polynomially."

*Global recursive constraints* Global state constraints are necessary to eliminate spurious configurations of blocks, such as unsupported flying blocks or circular configurations of blocks. These constraints are necessary to ensure soundness of solutions, but are very challenging to model, since recursive concepts are required to define such spurious configurations.

Although some global constraints are supported by PDDL (e.g., state trajectory constraints in PDDL3 [43]), global recursive constraints as mentioned in the examples above are not supported by the current PDDL-based planners, to the knowledge of the authors and the experts they have consulted.[2] For instance, in the blocks world, suppose that we define *above*/2 recursively (as the transitive closure of *on*/2 predicate) as a derived predicate, as suggested by Edelkamp and Hoffman [22] and by Thiebaux et al. [93]. Then expressing the global constraint "for every block $x$, $x$ is not above $x$" is not possible in PDDL. As the curious reader may consider, one can get around this problem by encoding all global constraints in the preconditions of each operator and in the goal conditions; such a method is suggested by Haslum et al. [50] to compile a type of state constraints into action descriptions. However, this method leads to a domain description that is not elaboration tolerant[3]: a modification of a global constraint requires updating the preconditions of all actions and the goal condition.

*Low-level feasibility checks* For robotic applications, feasibility of computed plans is a pre-requisite such that these plans can be used for real-life implementations. Along these lines, integration of stability checks into abstract planning is required to obtain feasible construction plans.

For instance, for a feasible construction plan, based on an abstract description of goal conditions, stable goal configurations should be determined for planning. In addition to the stable goal configuration, at every state reached during the execution of a plan, the structure being constructed should be ensured to be stable. Furthermore, during each transition

[2] Personal communication, Stefan Edelkamp and Robert Mattmueller, March 2019.

[3] According to McCarthy [67] (http://jmc.stanford.edu/articles/elaboration.html), a formalism is elaboration tolerant to the extent that it is convenient to modify a set of facts expressed in the formalism to take into account new phenomena or changed circumstances.

from one state to another, it should be guaranteed that the structure stays stable. Therefore, the robot construction problems cannot be addressed using solely a task planner, but necessities proper use of feasibility checks together with the planner. While essential for real-life applicability, it is challenging to combine task planning studied over discrete domains, with feasibility checks performed in continuous domains.

*Elaboration tolerance* For real-world applications, the flexibility of methods is important for the users to be able to address different variations of the problem. In robot construction, there exists a wide variety of problems that involve construction and transfer of subassemblies, use scaffolding and counterweights to temporarily balance structures, build overhangs and bridges that connect two surfaces. Each of these construction problems need to consider and model new phenomena. For instance, involving subassemblies requires a new type of stability check (i.e., stability of a construction being held by the robot), whereas building bridges requires definition of connectivity of both sides (i.e., of a river). It is challenging to represent the construction problem in such a general way that is tolerant to all these elaborations, as much as possible.

Elaboration tolerance is an important challenge in knowledge representation and reasoning [67]. Having a flexible framework that can address variations of a problem is also important for robotic applications, such as the robot construction problems.

### 1.2. Our contributions

We propose a formal hybrid planning framework for robot construction problems, where multiple autonomous robots rearrange stacks of prefabricated blocks to build stable structures, including stacks, bridges or overhangs. This framework can address all of the challenges discussed above, thanks to the underlying knowledge representation formalism and efficient automated reasoners of Answer Set Programming (ASP) [9]. Our ASP-based framework

- provides an elaboration-tolerant representation for a wide range of construction problems, utilizing nonmonotonicity (e.g., for common sense law of inertia) and recursive definitions (e.g., transitive closure),
- determines a stable final configuration of blocks, utilizing semantic attachments in logical formulas (e.g., by external atoms), and
- computes the order of manipulation tasks for multiple robots to build it from an initial configuration, in the spirit of hybrid planning where discrete/logical task planning is combined with continuous/probabilistic feasibility checkers (e.g., simulation-based physics engines).

Our hybrid framework is general and flexible, in the sense that many variations of construction problems, including optimal stacks and bridges, can be handled with the simplest kind of elaboration, i.e., the addition of new formulas [67]. It is interesting to note that our framework also provides a solution to the infamous maximum overhang puzzle [47,76,75, 77]—the problem of finding maximum overhangs with counterweights.[4]

Our ASP-based formal framework for robot construction problems prevents nonsensical configurations, like circular configuration of blocks (Proposition 3) or flying blocks (Proposition 4), and undesired occurrences of some actions concurrently (Proposition 5), like picking a block while placing another block on top of it.

It solves the ramification problem through recursive definitions of global locations of blocks from their relative locations (Proposition 1), and definitions of empty spaces not occupied by any blocks (Proposition 2). Our ASP-based formal framework also guarantees desired properties, like the stability of a construction (Proposition 6) and the connectedness of two sides of a bridge (Proposition 7).

In addition to these soundness results, our ASP-based framework guarantees completeness by ensuring the computation of all valid construction plans subject to such properties and constraints and whose lengths are less than a given maximum makespan (Proposition 8).

To investigate the applicability of our hybrid ASP-based framework to solve a variety of problems, we introduce a diverse set of challenging robot construction benchmark instances (Figs. B.27–B.47, Appendix B) where multiple autonomous robots rearrange stacks of prefabricated blocks to build stable structures, including stacks, bridges or overhangs by making use of counterweights, scaffolding, subassemblies, and true concurrency of manipulations. Such a benchmark set of different types of construction is also a useful contribution to advance studies on robot construction problems, hybrid planning, and knowledge representation.

Using these benchmarks, experimental evaluations are performed to understand the scalability of our hybrid ASP-based method, and the effect of granularity of goal specification and the integration of stability checks on computational efficiency in terms of time.

Furthermore, to verify the executability of the plans computed by our method and to show their applicability with real robots, we perform dynamic simulations and physical implementations of several benchmark scenarios.

[4] A relatively recent solution [75,77] to this 150 year old puzzle, honored with the prestigious David P. Robbins Prize in mathematics, has introduced the use of blocks as counterbalance to improve upon the well-established solution.

### 1.3. Organization of the manuscript

The rest of the paper is organized as follows. Section 2 defines the robot construction problem, Section 3 provides illustrative examples, while Section 4 presents the formal representation of the problem in Answer Set Programming and provides the soundness and completeness results. Section 5 details integration of stability checks for hybrid planning and presents examples demonstrating the importance of such integration for hybrid planning of construction problems. Section 6 discusses the results of experimental evaluations to investigate the computational performance of our hybrid ASP-based method, as the input size and the granularity of goal specification change, and when the integration of stability checks is disabled. Section 7 discusses several challenging benchmark problems and demonstrates applicability of the proposed approach through executions with a bimanual robot. Section 8 presents a comprehensive review of related works. Section 9 concludes the study and presents future research directions. Detailed proofs are presented in Appendix A, while challenging benchmarks problems and their solutions are given in Appendix B.

## 2. Robot construction problems

A robot construction problem comprises

- a final stable configuration of different types of prefabricated blocks stacked on each other on the ground, that satisfies some goal conditions, and
- a feasible stack rearrangement plan to obtain that final configuration from a specified initial configuration of the blocks.

Figs. 5, 6 and 19 present such stable final configurations, together with feasible construction plans to achieve them.

Initially, regular shaped blocks are stacked on the ground/table as specified by the problem instance. The ground consists of a set of flat surfaces (disconnected surfaces are required for bridges) and each surface has limited space for construction.

We use unit spaces, within a discrete model of the problem, to identify how much space is available on a block/surface and where to locate a block. A single unit space is set to be equal to the size of the smallest block. To describe our approach, we consider three types of prefabricated blocks in the form of regular-shaped blocks: small blocks with one unit space, medium blocks with three unit spaces, and large blocks with five unit spaces. We assume that the blocks are placed in the same row, i.e., other orientations are not considered. We also assume that the width and the height of all the blocks are the same, while their weights and mass distributions may vary based on the problem instance. These assumptions allow us to model discrete/logical task planning in two-dimensional space, while continuous/probabilistic feasibility checks are conducted in three-dimensional space where each block is considered as a three-dimensional object with a predefined mass distribution.

We consider construction tasks performed by multiple autonomous robots, such as bimanual manipulators. The robots can pick and place blocks. We assume that the orientations of the blocks remain the same during the plan, so that the robots do not have to rotate the blocks.

Our approach does not rely on any assumptions about the weight distribution of the blocks. For clarity of presentation and without loss of generality, we only focus on the stability of the structures as the feasibility check performed in the continuous domain. In particular, we ensure the stability of each step of the plan by testing it with a physics engine. Other feasibility checks, such as motion planning queries, reachability and graspability checks for manipulation actions, can be similarly integrated to our hybrid reasoning framework [31,81,71,79], as illustrated with an example in Section 4.1.

The goal conditions can be described in an abstract manner to capture important aspects of specific structures. For instance,

- for a simple stack, height can be maximized,
- for a bridge, the ground on one side should be connected to the one on the other side,
- for an overhang, constraints can be provided about the desired length of the overhang.

If necessary, further goal conditions may be specified in an abstract manner (e.g., lightweight blocks should be placed on top of heavy ones), with more details (e.g., blocks 3 and 4 must be placed on block 5), or even with further details (e.g., block 1 must be placed on block 2, ensuring that unit space 1 of block 1 is on unit space 3 of block 2).

In general, there exist multiple final configurations that satisfy the goal conditions, but only the ones that are stable and that can be achieved with a feasible construction plan are of interest. In that sense, the robot construction problem not only aims for a plan that reaches a goal configuration, but also ensures that this configuration and all intermediate steps are stable. Under these assumptions, we model the robot construction problems as a hybrid planning problem.

Construction problems are challenging from the perspective of planning, since they involve incorporation of preexisting structure into the final design, pre-assembly of movable substructures, and use of extra blocks as temporary supports or counterweights during construction. These problems are challenging from the perspective of geometric reasoning as well, since they these involve stability checks of complex structures.

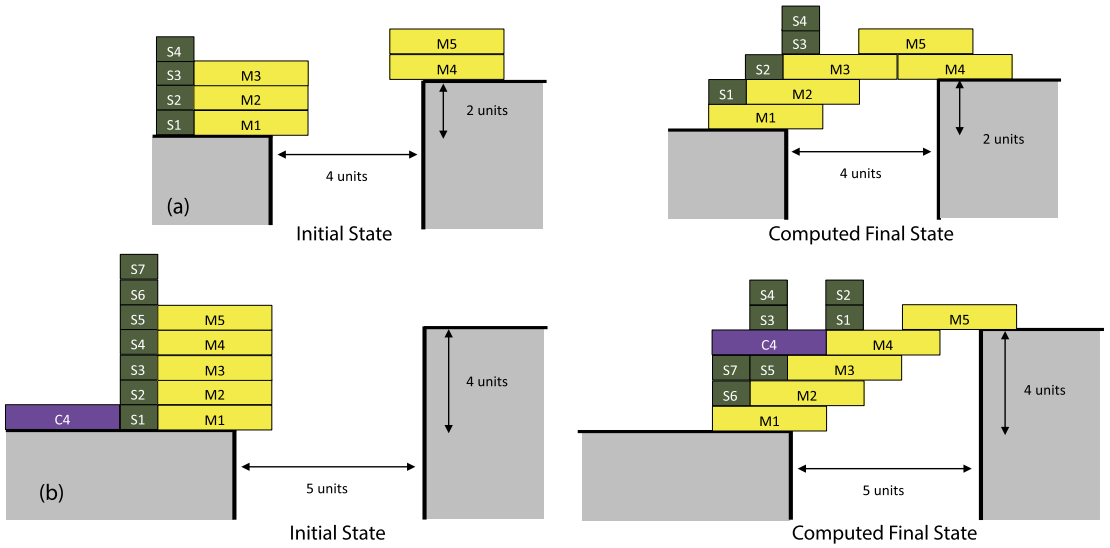



**Fig. 2.** Stable construction of asymmetric bridges: Given an initial state (left figure), a stable bridge is constructed (right figure).

Note that construction problems inherit the intractability of elementary Blocks World (EBW) problems, as the latter is a specific type of construction problems. Thanks to Gupta and Nau [46], we know that, given an EBW problem and a positive integer $L$, it is NP-complete to decide the existence of a plan whose makespan is less than or equal to $L$ [46, Theorem B.5].

Consider a variation of EBW (called VLBW) where the table can hold on a limited number of blocks, the blocks are of different sizes, and a large block cannot be placed on a smaller block. Given a VLBW problem, it is possible that the shortest plan has exponential length [46, Theorem E.1]. In that sense, if we consider a stability checker that returns "unstable" when a large block is placed on a smaller block, then it is possible that a construction problem (like Tower of Hanoi) has an exponential length shortest plan.

## 3. Illustrative examples

Consider the robot construction problems shown in Fig. 2 specified by their initial states; in each problem, the goal is to build a stable bridge. The construction area is limited, and an upper bound is given on the length of a plan. The problem shown in Fig. 2(b) further requires construction to start from the left side and proceed towards the right side. Solutions to these bridge construction problems require

- finding stable goal configurations of prefabricated blocks so that they connect the two sides (e.g., as in the final states shown in Fig. 2),
- handling ramifications of actions (e.g., when C4 is placed on S7, it becomes on S5 as an indirect effect as in Fig. 2(b)),
- construction and incorporation of subassemblies (e.g., the subassembly of S4, S3, M3 in the final state shown in Fig. 2(a)),
- using blocks or subassemblies as counterweights (e.g., all the small blocks and C4 in the final state in Fig. 2(b)), and
- maintaining stability of the structure at all times.

Construction of overhangs (Figs. 3 and 19), symmetric bridges (Fig. 4), and other interesting structures (Figs. 5 and 6) demonstrate further challenges, such as

- the need for concurrency of actions (e.g., moving $S1$ and $S2$ onto $L1$ at the same time as in Fig. 6(b)).

## 4. Modeling robot construction problems

We use Answer Set Programming (ASP) [9]—a form of knowledge representation and reasoning paradigm in AI—for hybrid planning. The idea is to represent the hybrid action domain by a set of logical formulas (called "rules"), whose models (called "answer sets" [41]) correspond to plans and can be computed by special implemented systems called answer set solvers, like DLVHEX [25], making calls to relevant feasibility checkers as needed.

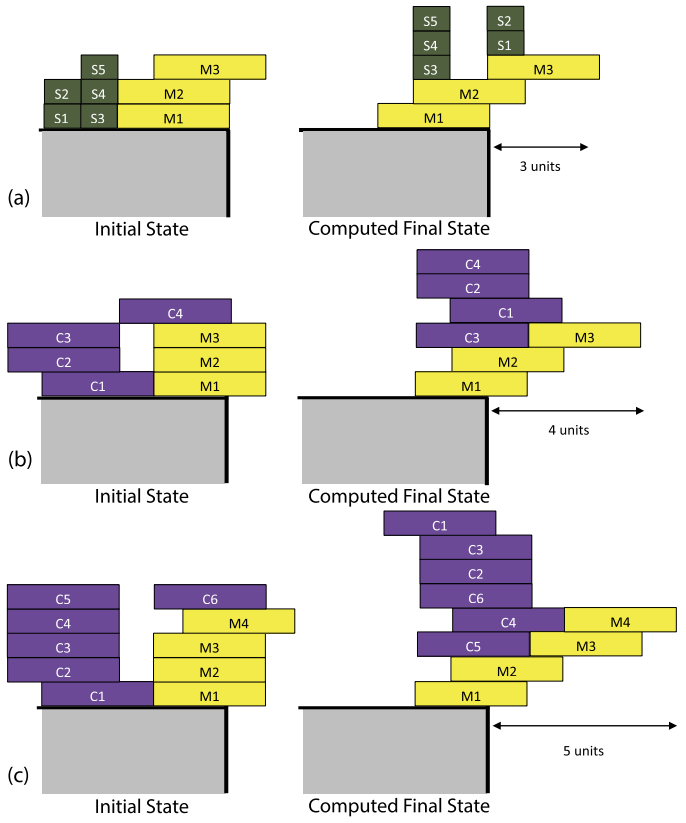



**Fig. 3.** Stable construction of overhangs: Given an initial state (left figure), a stable overhang of a given size is constructed (right figure).

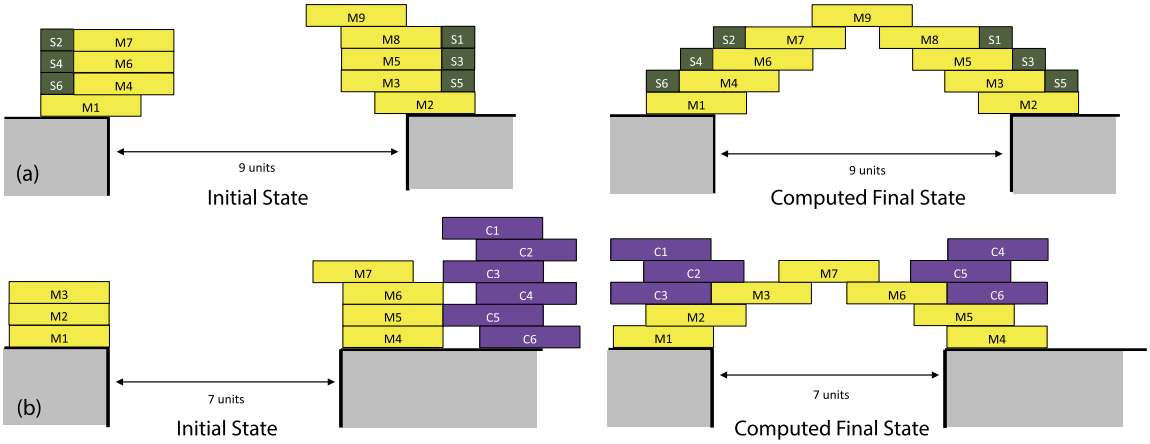



**Fig. 4.** Stable construction of symmetric bridges: Given an initial state (left figure), a stable bridge is constructed (right figure).

### 4.1. Formulas in ASP

We consider disjunctive ASP rules of the form:

$$\alpha_1 \vee \cdots \vee \alpha_k \leftarrow \beta_1, \ldots, \beta_n, \mathit{not}\ \beta_{n+1}, \ldots, \mathit{not}\ \beta_m$$

where $m, k \geq 0$, each $\alpha_i$ is a literal, and each $\beta_i$ is a literal or an external literal. Here, $\alpha_1 \vee \cdots \vee \alpha_k$ is called the head, $\beta_1, \ldots, \beta_n, \mathit{not}\ \beta_{n+1}, \ldots, \mathit{not}\ \beta_m$ is called the body of a rule. Intuitively, a rule expresses that if all $\beta_i$ $(1 \leq i \leq n)$ holds but no $\beta_i$ $(n+1 \leq i \leq m)$ holds then some $\alpha_i$ $(1 \leq i \leq k)$ holds as well. When $k = 0$, the rule is a constraint; when $n = m = 0$, it is a fact.

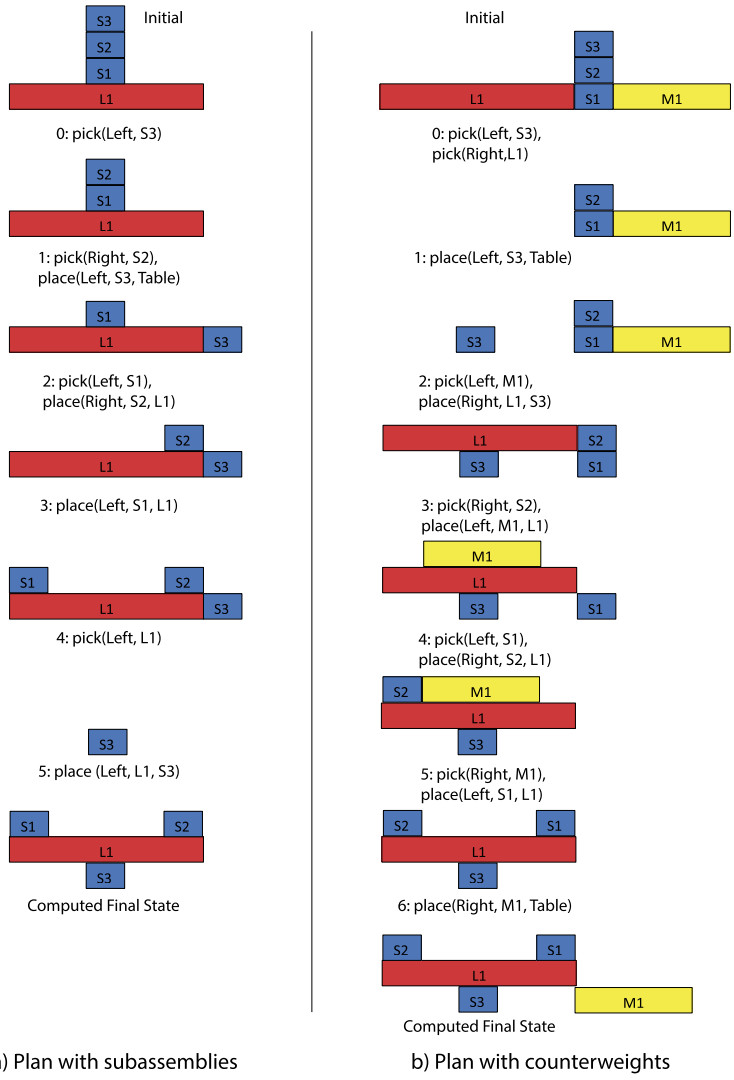



**Fig. 5.** Plans for two challenging robot construction scenarios that includes a) manipulation of subassemblies and b) use of counterweights. In these examples, the initial states are specified completely in detail (e.g., block $S1$ is on the unit space 3 of block $L1$), while the goal states are specified partially in an abstract way (e.g., blocks $S1$ and $S3$ are on block $L1$).

For instance, the following rule expresses that pick actions may occur at any time in a plan:

$$pick(a, b, t) \vee \neg pick(a, b, t) \leftarrow$$

where $a$ denotes a gripper of a robot, $b$ denotes a block to be picked up, and $t$ is a time step. Note that $pick(a, b, t) \vee \neg pick(a, b, t)$ is not a tautology as in classical logic, but expresses a choice of occurrence for an action. This is due to the nonmonotonic semantics of ASP formulas.

An external atom $\&g[Y_1, \ldots, Y_n](X_1, \ldots, X_m)$ is defined by its name $g$, input $Y_1, \ldots, Y_n$ and output $X_1, \ldots, X_m$. Intuitively, $g$ takes the input $Y_1, \ldots, Y_n$, passes it to an external computation (like a stability checker), and conveys the results $X_1, \ldots, X_m$ into the rules in the spirit of semantic attachments in theorem proving [100]. Different from the semantic attachments in planning [21,53], the arguments $Y_1, \ldots, Y_n$ passed to external computation do not need to be object constants or variables; they can be predicate extensions.

For instance, the following rule prevents the occurrence of a pick action in a plan, if the block that the robot wants to pick is not reachable:

$$\leftarrow pick(a, b, t), not\ \&reachable[on, a, b]().$$

Here the external computation *reachable* takes as arguments the arm $a$, the block $b$, and the extension of the predicate *on*, which describes the location of every block in the environment. Given this information, *reachable* applies a motion planning

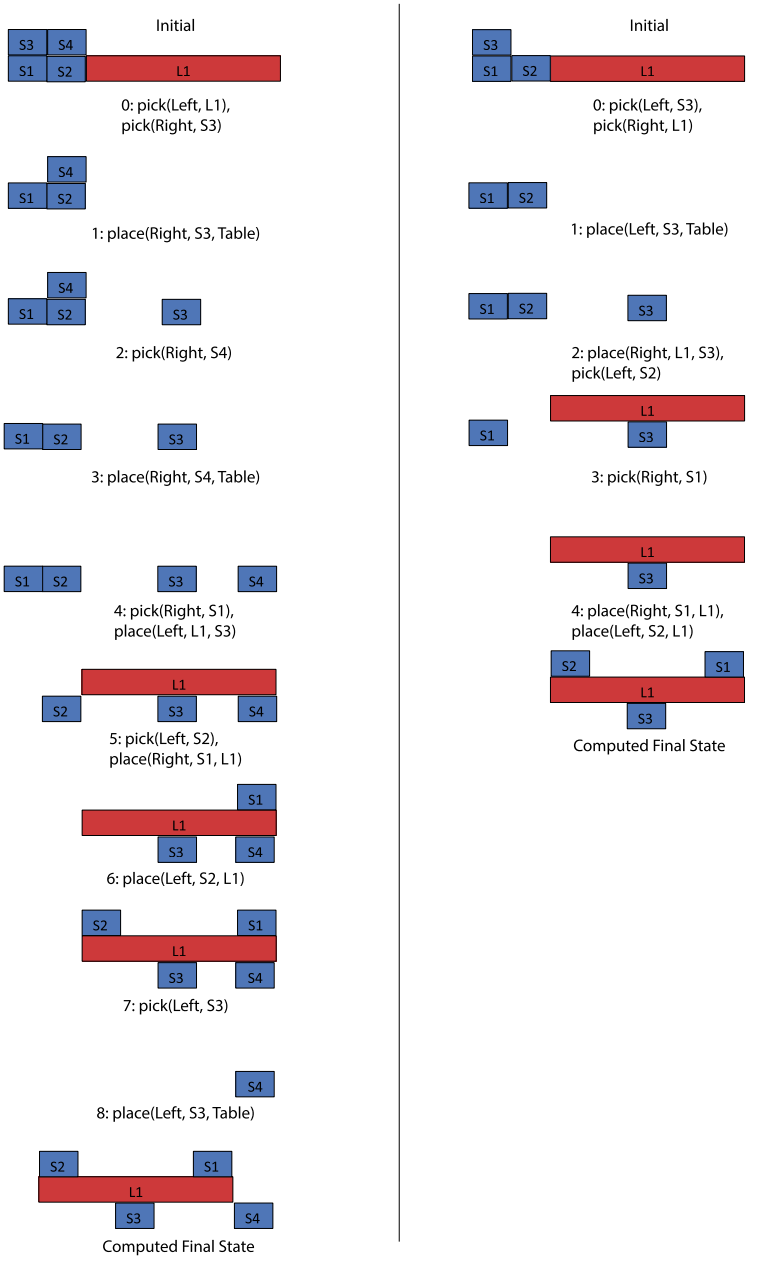



**Fig. 6.** Plans for two challenging robot construction scenarios that includes a) use of scaffolds and b) true concurrency of actions. In these examples, the initial states are specified completely in detail (e.g., block $S1$ is on the unit space 1 of the table), while the goal states are specified partially in an abstract way (e.g., blocks $S1$ and $S3$ are on block $L1$).

algorithm to check whether the arm $a$ can reach the block $b$ without colliding with any other objects in the environment. External atoms allow us to embed feasibility checks into task planning [31]. DLVHEX evaluates external atoms as needed [23].

ASP offers some useful constructs to concisely represent knowledge. For instance, aggregate atoms are expressions of the form [34]:

$$s_1 \prec_1 \alpha\{t_1, ..., t_n : \beta_1, ..., \beta_m\} \prec_2 s_2.$$

Here $t_i$ are terms, $\beta_i$ are literals, $\alpha$ is a function that evaluates the numerical value of the aggregate, and $\prec_1$ and $\prec_2$ are predicates that compare the resulting value with the terms $s_1$ and $s_2$. An aggregate atom holds if the comparison is true with respect to evaluating $\alpha$ on the tuples $\langle t_1, ..., t_n \rangle$ for which $\beta_1, ..., \beta_m$ hold. For instance, the aggregate atom

Occurrences/nonoccurrences: Pick actions may occur at any time.

$$pick(a, b, t) \vee \neg pick(a, b, t) \leftarrow$$

Direct effects: After a robot arm $a$ picks a block $b$, the robot arm is holding that block.

$$holding(a, b, t + 1) \leftarrow pick(a, b, t)$$

Preconditions: A robot arm $a$ cannot pick a block $b$ if it is holding it or another block, or if another robot arm is holding it.

$$\leftarrow pick(a, b, t), holding(a, b', t)$$
$$\leftarrow pick(a, b, t), holding(a', b, t) \qquad (a \neq a')$$

**Fig. 7.** ASP formalization $\Pi$ of construction problems: Occurrences/nonoccurrences, direct effects and preconditions of pick actions.

$$\#count\{b : box(b), holding(a, b, t)\} = 0$$

describes that the number of boxes $b$ that a robot is holding with its arm $a$ at time step $t$ is 0.

Weak constraints are expressions of the following form [10]:

$$\overset{\sim}{\leftarrow} Body(t_1, ..., t_n)[w@p, t_1, ..., t_n].$$

Here, $Body(t_1, ..., t_n)$ is a formula (as in the body of a rule) with the terms $t_1, ..., t_n$. Intuitively, whenever an answer set for a program satisfies $Body(t_1, ..., t_n)$, the tuple $\langle t_1, ..., t_n \rangle$ contributes a cost of $w$ to the total cost function relative to its priority $p$. The ASP solver tries to find an answer set with the minimum total cost. For instance, the following weak constraint

$$\overset{\sim}{\leftarrow} blockHeight(b, h, T). \ [h@1, b]$$

instructs DLVHEX to compute an answer set where the total height of every block $b$ in a tower at the end of the plan, at time step $T$, is minimized.

### 4.2. Fluents and actions

The objects in a robot construction domain consist of a set $\mathcal{A}$ of robotic grippers, a set $\mathcal{B}$ of blocks, and a set $\mathcal{L}$ of locations ($\mathcal{B} \subseteq \mathcal{L}$). The positions on each location $l \in \mathcal{L}$ (and thus each block $b \in \mathcal{B}$) are represented by its unit spaces $1, 2, ..., n_l$ for some positive integer $n_l$ that denotes the length of that location. Moreover, nonnegative integers $0, 1, ..., T-1$ describe time steps for a task plan, where $T$ specifies the maximum makespan (i.e., the length) of a plan.

In the following, the schematic variable $t$ ranges between 0 and $T$, $a$ and $a'$ range over all grippers, $b$ and $b'$ range over all blocks, $l$ and $l'$ range over all locations (e.g., blocks and *Table*), and $u$, $u'$, $v$ and $v'$ range over relevant unit spaces.

We consider two fluents to describe the states of the world:

- $holding(a, b, t)$ (robot's gripper $a$ is holding block $b$ at step $t$ of the plan), and
- $on(b, v, l, u, t)$ (box $b$ is at location $l$ at time step $t$, in such a way that the unit space $v$ of $b$ is on the unit space $u$ on $l$).

We consider two actions:

- $pick(a, b, t)$ (pick the block $b$ with the gripper $a$ at step $t$) and
- $place(a, l, t)$ (place the block being held by the gripper $a$, on the location $l$ at step $t$) with the attribute $placeOn(a, b, v, l, u, t)$ (place the block $b$ being held by the gripper $a$ such that the unit space $v$ of $b$ is on the unit space $u$ of $l$).

Here, the variable $t$ ranges between 0 and $T-1$.

Using these fluent and action constants, the preconditions and direct effects of pick and place actions, and the commonsense law of inertia can be formalized in ASP (shown in Figs. 7–9) following the guidelines described by Erdem et al. [30,26]. This formulation considers pick and place actions from a surface, and takes into account the following desired conditions about the nonexecutability of pick and place actions:

- A robot arm cannot pick a block if it is already holding it or another block.
- A robot arm cannot pick a block if another robot arm is holding it.
- A robot arm cannot place a block (onto any location) if it is not holding any blocks.
- A robot arm cannot place (any block) onto a block that is being held by another robot arm.

Occurrences/nonoccurrences: Place actions may occur at any time.

$$placeOn(a, b, v, l, u, t) \vee \neg placeOn(a, b, v, l, u, t) \leftarrow$$
$$place(a, l, t) \leftarrow placeOn(a, b, v, l, u, t)$$

Direct effects: After a robot places unit $v$ of a block $b$ that it is holding, onto unit $u$ of a location $l$, the unit $v$ of block $b$ becomes on the unit $u$ of location $l$.

$$on(b, v, l, u, t+1) \leftarrow placeOn(a, b, v, l, u, t), holding(a, b, t)$$

Preconditions: A robot cannot place onto location $l$ if it is not holding any blocks.

$$\leftarrow place(a, l, t), \#count\{b : box(b), holding(a, b, t)\} = 0$$

A robot arm $a$ cannot place onto a block $b$ if another robot arm is holding $b$.

$$\leftarrow place(a, b, t), holding(a', b, t) \qquad (a \neq a')$$

**Fig. 8.** ASP formalization $\Pi$ of construction problems: Occurrences/nonoccurrences, direct effects and preconditions of place actions.

If a block $b$ is (resp. not) at a location $l$ at time step $t$ then $b$ remains to be (resp. not) at $l$ at the next step $t+1$ by default (i.e., unless some action changes its location directly/indirectly).

$$on(b, v, l, u, t+1) \leftarrow not\ \neg on(b, v, l, u, t+1), on(b, v, l, u, t)$$
$$\neg on(b, v, l, u, t+1) \leftarrow not\ on(b, v, l, u, t+1), \neg on(b, v, l, u, t)$$

If a block is (resp. not) being held then it remains to be (resp. not) held at the next state by default.

$$holding(a, b, t+1) \leftarrow not\ \neg holding(a, b, t+1), holding(a, b, t)$$
$$\neg holding(a, b, t+1) \leftarrow not\ holding(a, b, t+1), \neg holding(a, b, t)$$

**Fig. 9.** ASP formalization $\Pi$ of construction problems: Commonsense law of inertia.

If a block is on some location then the block is not being held by a robot, and vice versa.

$$onAux(b, l, t) \leftarrow on(b, v, l, u, t)$$
$$\neg holding(a, b, t) \leftarrow onAux(b, l, t)$$
$$\neg on(b, v, l, u, t) \leftarrow holding(a, b, t)$$

Further uniqueness conditions on locations of blocks:

$$\neg on(b, v, l', u', t) \leftarrow on(b, v, l, u, t) \qquad (\langle l, u\rangle \neq \langle l', u'\rangle)$$
$$\neg holding(a, b', t) \leftarrow holding(a, b, t) \qquad (b \neq b')$$
$$\neg holding(a', b, t) \leftarrow holding(a, b, t) \qquad (a \neq a')$$

**Fig. 10.** ASP formalization $\Pi$ of construction problems: Straightforward negative ramifications of pick and place actions.

In the following, let us discuss how the further challenges of robot construction problems are addressed using ASP.

### 4.3. Ramifications

The pick and place actions of a robot have many interesting indirect effects (or ramifications), as also considered in the Blocks World domain. Some of these ramifications are quite straightforward, as shown in Fig. 10. For instance, if a block $b$ is placed on some location $l$, then, as a direct effect of this action, $b$ becomes on $l$; as an indirect effect, the robot's gripper becomes empty:

$$\neg holding(a, b, t) \leftarrow onAux(b, l, t).$$

Positive ramifications (Fig. 13):

$$on(b, v+i, l, u+i, t) \leftarrow on(b, v, l, u, t)$$
$$(1 \leq i \leq min\{size(b)-v, size(l)-u\})$$
$$on(b, v-j, l, u-j, t) \leftarrow on(b, v, l, u, t) \qquad (1 \leq i \leq min\{v, u\})$$

Positive ramifications (Fig. 14):

$$on(b, v, b', u, t) \leftarrow globalPos(b, v, h, x, t), globalPos(b', u, h-1, x, t) \quad (h > 0)$$

Negative ramifications (Fig. 15):

$$\neg on(b, v+i, l', u', t) \leftarrow on(b, v, l, u, t), globalPos(b, v+i, h, x_i, t),$$
$$empty(h-1, x_i, t) \qquad (size(b)-v > size(l)-u, size(l)-u < i \leq size(b))$$
$$\neg on(b, v-j, l', u', t) \leftarrow on(b, v, l, u, t), globalPos(b, v-j, h, x_j, t),$$
$$empty(h-1, x_j, t) \qquad (min\{size(b)-v, size(l)-u\} \leq j < v)$$

**Fig. 11.** ASP formalization $\Pi$ of construction problems: Ramifications of placing a long block on another block, as illustrated in Figs. 13–15.

Global positions of locations, defined vertically:

$$globalPos(b, v, 1, x, t) \leftarrow on(b, v, Table, x, t) \quad (1 \leq x \leq n_{Table})$$
$$globalPos(b, v, h, x, t) \leftarrow globalPos(b', u, h-1, x, t), on(b, v, b', u, t) \quad (h > 1)$$

Global positions of locations, defined horizontally:

$$globalPos(b, v+1, h, x+1, t) \leftarrow globalPos(b, v, h, x, t) \quad (v < size(b))$$
$$globalPos(b, v-1, h, x-1, t) \leftarrow globalPos(b, v, h, x, t) \quad (v > 1, x > 1)$$

Empty spaces above the table:

$$empty(h, x, t) \leftarrow \#count\{b : box(b), globalPos(b, v, h, x, t)\} = 0 \qquad (h > 0)$$

**Fig. 12.** ASP formalization $\Pi$ of construction problems: Auxiliary definitions for global positions and empty spaces.

Here, *onAux* is a projection of *on*. Furthermore, if the unit space $v$ of block $b$ is on the unit space $u$ of location $l$, then $(b, v)$ is not on any other unit $(l', u')$.

$$\neg on(b, v, l', u', t) \leftarrow on(b, v, l, u, t)$$

where the variable $u$ and $u'$ range over the unit spaces of $l$, and $l \neq l'$ or $u \neq u'$.

If a robot's gripper $a$ picks a block $b$, then as its direct effect $a$ is holding $b$; as an indirect effect, $b$ is not on any block or the table:

$$\neg on(b, v, l, u, t) \leftarrow holding(a, b, t).$$

Furthermore, as indirect effects, the gripper $a$ is not holding any other blocks $b'$ $(b \neq b')$:

$$\neg holding(a, b', t) \leftarrow holding(a, b, t)$$

and no other gripper $a'$ is holding $b$ $(a \neq a')$:

$$\neg holding(a', b, t) \leftarrow holding(a, b, t)$$

There are further ramifications of pick and place actions, pertaining to the robot construction problems that are not considered in the Blocks World.

An interesting positive ramification occurs (Fig. 13) when a unit space $v$ of a large block $b$ is placed on a unit $u$ of another block $l$: the block $b$ occupies as many unit spaces as its size allows on $l$. In particular, block $b$ occupies the right part of $l$, starting from the unit space $u$ of $l$:

$$on(b, v+i, l, u+i, t) \leftarrow on(b, v, l, u, t) \tag{1}$$

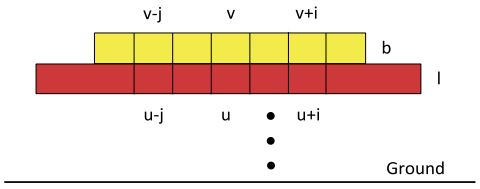



**Fig. 13.** As indirect effects of placing unit $v$ of $b$ on unit $u$ of $l$, unit $v+i$ (resp. $v-j$) of $b$ is on unit $u+i$ (resp. $u-j$) of $l$.

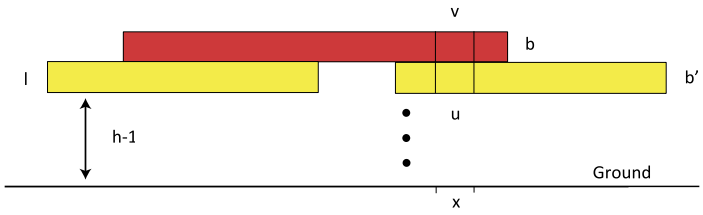



**Fig. 14.** As indirect effects of placing $b$ on $l$, unit $v$ of $b$ becomes on unit $u$ of $b'$, and unit $v+i$ (resp. $v-j$) of $b$ becomes on unit $u+i$ (resp. $u-j$) of $b'$.

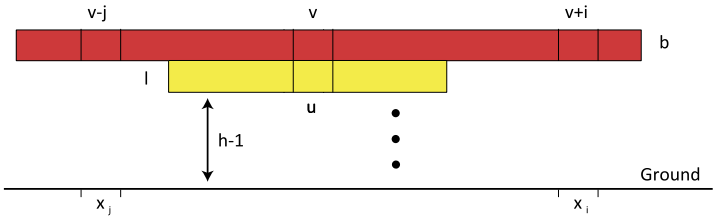



**Fig. 15.** As indirect effects of placing unit $v$ of $b$ on unit $u$ of $l$, unit $v+i$ (resp. $v-j$) of $b$ is on unit $u+i$ (resp. $u-j$) of $l$.

where $i$ ranges between 1 and $min\{size(b)-v, size(l)-u\}$, and $size(block)$ denotes the length of $block$. Similarly, block $b$ occupies the left part of $l$ starting from the unit space $v$ of $l$:

$$on(b, v-j, l, u-j, t) \leftarrow on(b, v, l, u, t) \tag{2}$$

where $j$ ranges between 1 and $min\{v, u\}$.

Another interesting positive ramification occurs when a large block $b$ is placed on top of block $l$ (Fig. 14): block $b$ is also placed on another block $b'$ that is not too far from block $l$.

Such a sophisticated ramification is represented as follows. Suppose that atoms of the form $globalPos(b, v, h, x, t)$ express that the unit space $v$ of block $b$ at time step $t$ is globally located at $(h, x)$ (i.e., $x$ units to the right of the leftmost side of the table, and $h$ units above the surface of the table). After placing block $b$ on block $l$, if the unit space $v$ of $b$ and the unit space $u$ of $b'$ are both globally located horizontally $x$ units from the leftmost side of the table, and $b$ and $b'$ are globally located vertically $h$ and $h'$ units above the table, respectively, then as a ramification the unit $v$ of $b$ is on the unit $u$ of $b'$. This ramification is described by the following rule:

$$on(b, v, b', u, t) \leftarrow globalPos(b, v, h, x, t), globalPos(b', u, h-1, x, t) \quad (h > 0)$$

An interesting negative ramification occurs (Fig. 15) when a large block $b$ is placed on another block, and some parts of $b$ are not on any location as they happen to be over an empty space.

Such a sophisticated ramification is represented as follows. Suppose that atoms of the form $empty(h, x, t)$ express that the unit space globally located at $(h, x)$ is empty (i.e., not occupied by any block) at time step $t$.

After placing unit space $v$ of a block $b$ on unit space $u$ of another block $l$, if the unit space $v+i$ of $b$ is globally located at $(h, x_i)$ and the global position $(h-1, x_i)$ is empty, then as a ramification the unit $v+i$ of $b$ is not on any location. This negative ramification is described by the following rule:

$$\begin{aligned} \neg on(b, v+i, l', u', t) \leftarrow\ & on(b, v, l, u, t), globalPos(b, v+i, h, x_i, t), \\ & empty(h-1, x_i, t) \end{aligned} \tag{3}$$

where $size(b)-v > size(l)-u$, $i$ ranges between $size(l)-u+1$ and $size(b)$, and $h>1$.

Similarly, a negative ramification occurs if the unit space $v-j$ of $b$ is globally located at $(h, x_j)$ and the global position $(h-1, x_j)$ is empty, and is described by the following rule:

$$\neg on(b, v-j, l', u', t) \leftarrow on(b, v, l, u, t), globalPos(b, v-j, h, x_j, t), \\ empty(h-1, x_j, t) \tag{4}$$

where $j$ ranges between $min\{size(b)-v, size(l)-u\}$ and $v-1$, and $h>1$.

We define global locations of blocks recursively. For every time step $t$ and for every block $b$ (within a tower) that is located $x$ units to the right of the leftmost side of the table, first we recursively define the height $h$ of every unit $v$ of block $b$:

$$\begin{aligned} &globalPos(b, v, 1, x, t) \leftarrow on(b, v, Table, x, t) \quad (1 \leq x \leq n_{Table}) \\ &globalPos(b, v, h, x, t) \leftarrow globalPos(b', u, h-1, x, t), on(b, v, b', u, t) \quad (h>1). \end{aligned} \tag{5}$$

Next, we recursively define the locations of other units of block $b$ horizontally to the right and to the left of that tower:

$$\begin{aligned} &globalPos(b, v+1, h, x+1, t) \leftarrow globalPos(b, v, h, x, t) \quad (v<size(b)) \\ &globalPos(b, v-1, h, x-1, t) \leftarrow globalPos(b, v, h, x, t) \quad (v>1) \end{aligned} \tag{6}$$

Note that the recursive definition of *globalPos* characterizes a form of reachability where immediate-connectedness between a block unit $\langle b, v\rangle$ and a block/table unit $\langle b', v'\rangle$ is understood as one of the following conditions:

(i) $\langle b, v\rangle$ is located immediately on $\langle b', v'\rangle$ (specified by atoms of the form $on(b, v, b', v', t)$),
(ii) $b = b'$ and $v' = v + 1$, or
(iii) $b = b'$ and $v' = v - 1$.

The last two conditions describe horizontal immediate-connectedness of unit spaces of a block.

At every time step $t$, for every table unit $\langle Table, x\rangle$, the rules (5) $\cup$ (6) identify the block units $\langle b, v\rangle$ reachable from $\langle Table, x\rangle$ at time $t$ relative to such immediate-connectedness relation, and furthermore define their global locations recursively with respect to their height $h$ from the table and the distance $x$ from the leftmost side of the table.

**Proposition 1.** *Let $\Pi'$ be the disjunctive program shown in Figs. 7–11. Then, for every time step $t$ and for every block $b$ supported by the table, rules (5) $\cup$ (6), when added to $\Pi'$, correctly describe the global position of $b$ at time $t$ with respect to its height $h$ from the table and the distance $x$ from the leftmost side of the table.*

The proof of Proposition 1 follows from the observation that the disjunctive program $\Pi'$ can be equivalently transformed into a nondisjunctive program (Theorem 1 of [28]), $\Pi'$ does not contain atoms of the form $globalPos(b, v, h, x, t)$ in the heads of its rules, and the correctness of the recursive reachability definition (see Lemma 2), as explained in Appendix A.

After defining the global positions, the empty spaces above the table are defined as the global positions $(h, x)$ that are not occupied by any blocks:

$$empty(h, x, t) \leftarrow \#count\{b : box(b), globalPos(b, v, h, x, t)\} = 0 \quad (h > 0). \tag{7}$$

**Proposition 2.** *Let $\Pi'$ be the disjunctive program shown in Figs. 7–12, except for rules (7). Then, for every time step $t$ and for every block $b$ supported by the table, rules (7), when added to $\Pi'$, correctly describe the global positions $(h, x)$ of empty spaces at time $t$.*

The proof of Proposition 2 follows from the observation that the disjunctive program $\Pi'$ can be equivalently transformed into a nondisjunctive program (Theorem 1 of [28]), and does not contain atoms of the form $empty(h, x, t)$ in the heads of its rules, as explained in Appendix A.

**Remark.** In our experiments, we assume that there is no block overhanging from the leftmost side of the table as the ASP solver DLVHEX does not allow negative integers. To ensure this assumption, we add further constraints as shown below.

$$\begin{aligned} &\leftarrow on(b, v, Table, x, t) \quad (v > x) \\ &\leftarrow on(b, v, b', u, t), globalPos(b', u, h, x, t) \quad (h > 0, v > x) \end{aligned}$$

### 4.4. Supportedness constraints

*Supportedness of blocks* At every state of the world (including the initial state and goal state), it is desired that

D1 No block is supported by itself (i.e., no circular configurations).

The formulation of this constraint is challenging because it requires the transitive closure of a binary relation *onAux* that describes which block is on which location.

Recall that *onAux* is obtained from *on* by projection, as part of Π (Fig. 10):

$$onAux(b, l, t) \leftarrow on(b, v, l, u, t).$$

For every step $t$, we recursively define the supportedness of a block $b$ by a location $l$ as follows:

$$\begin{array}{l} supported(b, l, t) \leftarrow onAux(b, l, t) \\ supported(b, l, t) \leftarrow onAux(b, l', t), supported(l', l, t) \quad (b \neq l'). \end{array} \tag{8}$$

After that, we add a constraint to ensure that no block $b$ is supported by itself:

$$\leftarrow supported(b, b, t). \tag{9}$$

**Proposition 3.** *Let* Π *be the disjunctive program shown in Figs. 7–12. Then, rules (8), when added to* Π*, correctly describe the supportedness of blocks by other blocks and by the table. Furthermore, adding constraints (9) to* Π ∪ (8) *guarantees the desired feature D1 (i.e., no circular configuration of blocks occurs in construction at any time step* $0, 1, ..., T-1$*).*

The proof follows from (i) Theorem 1 of [28] about an equivalent representation of disjunctive rules about occurrences/nonoccurrences of actions by nondisjunctive rules, (ii) the observation that Π does not contain atoms of the form $supported(b, l, t)$ in the heads of its rules, and (iii) Propositions 4 and 5 of [29] about the correctness and well-foundedness of the transitive closure of a relation defined recursively in ASP. The proof is presented in Appendix A.

*Supportedness of subassemblies* Note that in the Blocks World domain, every block is supported by the table unless it is being held by a robot. In robot construction problems, since we would like to allow robots to pick and place subassemblies, we cannot simply enforce this constraint. Instead, we can enforce that

D2 Every block $b$ is supported by either the table or a block being held by a robot unless the block $b$ itself is being held by the robot.

For that, for every time step $t$, we define an auxiliary concept to identify blocks 1) being supported by the table, 2) being supported by a block being held by a robot, or 3) being held by a robot:

$$\begin{array}{l} supportedAux(b, 1, t) \leftarrow supported(b, Table, t), \\ \quad \#count\{a : arm(a), holding(a, b, t)\} = 0 \\ supportedAux(b, 2, t) \leftarrow supported(b, b', t), holding(a, b', t) \qquad (b \neq b') \\ supportedAux(b, 3, t) \leftarrow holding(a, b, t) \end{array} \tag{10}$$

and add the following constraint to ensure that exactly one of these three cases should hold for each block $b$ (i.e., no flying blocks, no blocks supported by both the table and a robot):

$$\begin{array}{l} \leftarrow \#count\{x : supportedAux(b, x, t)\} < 1 \\ \leftarrow \#count\{x : supportedAux(b, x, t)\} > 1 \end{array} \tag{11}$$

**Proposition 4.** *Let* Π *be the disjunctive program shown in Figs. 7–12. Then, rules (10), when added to* Π ∪ (8) ∪ (9)*, correctly describe the supportedness of blocks and subassemblies by other blocks, by the table, or by a robot. Furthermore, adding constraints (11) to* Π ∪ (8) ∪ (9) ∪ (10) *guarantees the desired feature D2 (i.e., no flying blocks, and no blocks supported by both the table and a robot at any time step* $0, 1, ..., T-1$*).*

The proof follows from (i) the observation that Π ∪ (8) ∪ (9) does not contain atoms of the form $supportedAux(b, x, t)$ in the heads of its rules, and (ii) from Proposition 3 and 2 of [33] about the conservative extensions of models by adding a definition, and elimination of models by adding constraints, respectively. The proof is presented in Appendix A.

### 4.5. Constraints about occurrences of action

*Constraints for manipulating subassemblies in hand* Regarding manipulation of subassemblies in robot construction problems, it may be desired that

D3 No robot picks/places subassemblies from/to some other subassemblies being held by a robot.

This is guaranteed by the following constraints:

$$\begin{array}{l} \leftarrow pick(a, b, t), not\ supported(b, Table, t) \\ \leftarrow place(a, b, t), not\ supported(b, Table, t) \end{array} \tag{12}$$

Supportedness of a block by another block or the table:

$$supported(b, l, t) \leftarrow onAux(b, l, t)$$
$$supported(b, l, t) \leftarrow onAux(b, l', t), supported(l', l, t) \quad (b \neq l').$$

No circular configurations of blocks:

$$\leftarrow supported(b, b, t).$$

Supportedness of a block (or a subassembly) 1) by the table, 2) by another block being held by a robot, or 3) by a robot:

$$supportedAux(b, 1, t) \leftarrow supported(b, Table, t),$$
$$\#count\{a : arm(a), holding(a, b, t)\} = 0$$
$$supportedAux(b, 2, t) \leftarrow supported(b, b', t), holding(a, b', t) \qquad (b \neq b')$$
$$supportedAux(b, 3, t) \leftarrow holding(a, b, t)$$

No flying blocks:

$$\leftarrow \#count\{x : supportedAux(b, x, t)\} < 1$$

No block supported by the table and a robot:

$$\leftarrow \#count\{x : supportedAux(b, x, t)\} > 1$$

**Fig. 16.** Supportedness of blocks and subassemblies.

*Noconcurrency constraints* Unless specified otherwise, the ASP modeling of the construction problem allows true concurrency. We can explicitly specify noconcurrency constraints, based on the capabilities of manipulators or desired conditions of the construction process. For instance, it may be desired that

D4 No robot picks a block using both of its grippers.

This is guaranteed by the following constraint:

$$\leftarrow \#count\{a : arm(a), pick(a, b, t)\} > 1. \tag{13}$$

For instance, it may be desired that

D5 A block $b$ cannot be picked by a gripper $a$ while another gripper $a'$ $(a \neq a')$ is placing a block on $b$.

This is guaranteed by the following constraint:

$$\leftarrow pick(a, b, t), place(a', b, t). \tag{14}$$

The following proposition ensures that the constraints above guarantee the desired properties D3–D5, while preserving the correctness of the ASP program for robot construction problems discussed in the previous sections.

**Proposition 5.** *Let $\Pi'$ be the program obtained from the disjunctive program $\Pi$ shown in Figs. 7–12, by adding the program shown in Fig. 16 about the supportedness of blocks/subassemblies. Then, adding constraints (12) ∪ (13) ∪ (14) to $\Pi'$ further ensures the desired features D3–D5 about occurrences of actions.*

Propositions 3 and 4 ensure constructions that satisfy supportedness constraints. Then the proof follows from application of Proposition 2 of [33] about the elimination of models by adding constraints (12) ∪ (13) ∪ (14). The proof is explained in Appendix A.

*Other constraints* Depending on the capabilities of the robotic manipulator and the difficulties of execution of concurrent actions, we can include further constraints about the occurrences of actions.

For instance, if the robotic manipulator can place a block to a location $l$ from the above only (so it cannot place a block to an empty space under another block), then we can add the following constraint to our program above:

$$\leftarrow placeLU(l, u, t), not\ clearAbove(l, u, t)$$

where *placeLU* is a projection of *placeOn* and *clearAbove* describes that there is no part of any block above a unit space:

$$placeLU(l,u,t) \leftarrow placeOn(a,b,v,l,u,t)$$
$$clearAbove(l,u,t) \leftarrow \#count\{b : globalPos(b,v,h',x,t), h' > h\} = 0,$$
$$\quad globalPos(l,u,h,x,t).$$

In another case, considering the difficulty of synchronization of actions, it may be desired that a robotic manipulator does not replace a block with another block at the same time. This is guaranteed by adding the following constraint:

$$\leftarrow pick(a,b,t), on(b,v,l,u,t), placeLU(l,u,t).$$

Considering the stability of the structure, it may be desired that two blocks cannot be picked at the same time if there exists another block that is supported by both of them. This is guaranteed by adding the following constraint:

$$\leftarrow pick(a,b,t), pick(a',b',t), commonSupported(b,b',t) \quad (b \neq b')$$

where *commonSupported* is defined as follows:

$$commonSupported(b_1,b_2,t) \leftarrow supported(b,b_1,t), supported(b,b_2,t) \quad (b_1 \neq b_2).$$

Similarly, it may be desired that two blocks cannot be placed at the same time if they will support another block together afterwards. This is guaranteed by adding the following constraint:

$$\leftarrow placeB(a,b,t), placeB(a',b',t),$$
$$\quad not\ commonSupported(b,b',t), commonSupported(b,b',t+1)$$

where *placeB* is a projection of *placeOn*:

$$placeB(a,b,t) \leftarrow placeOn(a,b,v,l,u,t).$$

**Remark.** In our experiments, we take into account these constraints about occurrences of actions.

### 4.6. Stability constraints

Stability checks are desired to ensure the stability of the overall assembly and each subassembly at every state of the construction plan. These checks can be performed externally by a module utilizing state-of-the-art physics engines, and their results can be embedded in the ASP formulation using external atoms. This modular approach enables our framework to be independent from any particular implementation of the stability checking algorithm, thus the stability checker can be treated as a black-box. Note that same approach is commonly employed for collision-checking during motion planning [61].

Let $\Gamma$ be a stability checking algorithm that returns *True* if the given structure is stable, and *False* otherwise.

We consider two external atoms to embed stability checks into our ASP formulation:

- $\&stable[on,t]()$ gets as input the relative positions of all the blocks supported by the table at step $t$ (described by the *on* predicate); and
- $\&hStable[holding,on,t]()$ gets as input the relative positions of all the blocks being carried by a manipulator at step $t$ (described by *holding* and *on* predicates).

Both external atoms utilize the stability checker $\Gamma$, and return the outputs (i.e., *True* for stable or *False* for unstable) accordingly.

We embed the outcomes of stability checks into our domain description by constraints as follows:

$$\begin{aligned} &\leftarrow not\ \&stable[on,t]() \\ &\leftarrow holding(a,b,t), onAux(b',b,t), not\ \&hStable[holding,on,t](). \end{aligned} \tag{15}$$

Note that the external atom $\&stable[on,t]()$ gets as input the extension of *on* predicate at step $t$, that describes the relative positions of the blocks; it returns *False* if the structure is not stable. The external atom $\&hStable[holding,on,t]()$ works in the similar fashion but for assemblies that are being carried by a robot.

The following proposition ensures that the constraints (15) guarantee the stability of the assembly being constructed by a robot.

**Proposition 6.** *Let $\Pi'$ be the program obtained from the disjunctive program $\Pi$ shown in Figs. 7–12, and by adding the program shown in Fig. 16 about supportedness of blocks/subassemblies. Suppose that the stability checking algorithm $\Gamma$ is correct (i.e., the construction is stable iff $\Gamma$ returns True). Then adding rules (15) to $\Pi'$ ensures that, at every time step t, every configuration of blocks assembled on a flat surface (e.g., table) or being carried by a gripper is stable.*

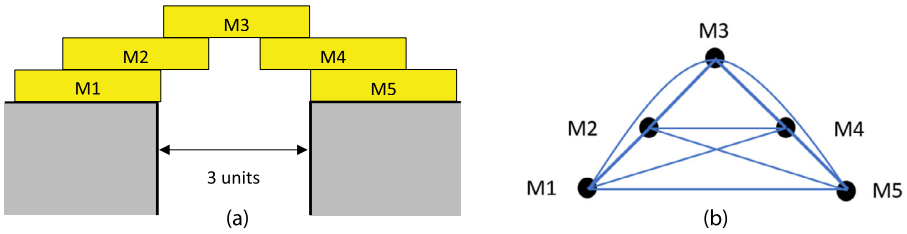



**Fig. 17.** Connectedness of blocks in a construction, illustrated as a graph. a) A bridge consisting of 5 blocks, b) connectedness graph where nodes represent blocks and edges represent connectedness.

Propositions 3 and 4 ensure constructions that satisfy supportedness constraints and thus prevent spurious structures, like flying blocks/subassemblies, circular configurations of blocks, or blocks being supported by both the table and a robot. Then the proof of Proposition 6 follows from an application of Proposition 2 of [33] about the elimination of spurious models by adding constraints. The proof is presented in Appendix A.

### 4.7. Bridges and overhangs

In bridge construction scenarios, instead of one whole surface, there are two surfaces apart from each other: e.g., one on the left side and the other on the right side. One of the required conditions for a bridge is that, it connects these two sides:

D6 At the end of the construction, there exists a block $x$ supported by the left side of the bridge and another block $y$ supported by the right hand side of the bridge such that $x$ and $y$ are connected to each other.

First we define immediate-connectedness as symmetric-supportedness:

$$\begin{aligned} &symSupported(x, y, t) \leftarrow supported(x, y, t) \\ &symSupported(x, y, t) \leftarrow supported(y, x, t) \end{aligned} \tag{16}$$

Then we recursively define connectedness of blocks as the transitive closure of immediate-connectedness (as illustrated by Fig. 17 by a graph), using an auxiliary atom of the form $connected(x, y, t)$:

$$\begin{aligned} &connected(x, y, t) \leftarrow symSupported(x, y, t) \\ &connected(x, y, t) \leftarrow symSupported(x, z, t), connected(z, y, t) \end{aligned} \tag{17}$$

and add a constraint to ensure the required condition D6 for the last time step $T$:

$$\begin{aligned} \leftarrow \#count\{x, y : &connected(x, y, T), \\ &side(x, Left, T), side(y, Right, T)\} = 0. \end{aligned} \tag{18}$$

Here, $side(x, Left, T)$ expresses that there exists a block $x$ supported by the left side of the bridge at time step $T$.

**Proposition 7.** *Suppose that the stability checking algorithm $\Gamma$ is correct (i.e., the construction is stable iff $\Gamma$ returns True). Let $\Pi'$ be the program obtained from the disjunctive program $\Pi$ shown in Figs. 7–12, by adding the program shown in Fig. 16 about supportedness of blocks/subassemblies, and by adding the constraints (15) about stability of the construction. Then, adding rules (16)$\cup$(17)$\cup$(18) to $\Pi'$ ensures a stable symmetric bridge (i.e., a construction that satisfies the condition D6) at time step $T$.*

Propositions 3–6 ensure constructions that satisfy supportedness and stability constraints. Then the proof follows from Proposition 3 and 2 of [33] about the conservative extensions of models by adding a definition, and elimination of models by adding constraints, respectively.

*Uneven bridges* In order to solve scenarios involving uneven bridges as shown in Fig. 18, the heights of both sides (*Left* and *Right*) should be specified in the base cases of the recursive definition of the global position of a unit space of a block.

Note that, in the program shown in Fig. 12, since all blocks are assumed to be on the same surface (i.e., *Table*), the base case of the recursive definition (5) is expressed by the following rule:

$$globalPos(b, v, 1, x, t) \leftarrow on(b, v, Table, x, t) \quad (1 \leq x \leq n_{Table}).$$

Now we have two surfaces, *Left* and *Right* sides, we need to consider their heights and how far the *Right* side is from the *Left* side. Suppose that the height of the left side of the bridge is $H$, the height of the right side is $H'$, and the leftmost

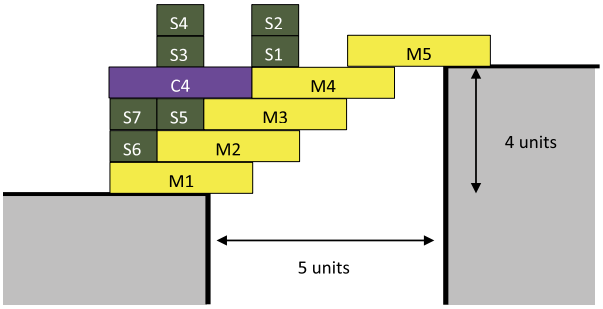



**Fig. 18.** The height difference between the left and right side is 4 units.

side of the right side is $L'$ units from the leftmost side of the left side. Then, instead of the rule above, we define the base cases by the following rules:

$$globalPos(b, v, H+1, x, t) \leftarrow on(b, v, Left, x, t) \quad (1 \leq x \leq n_{Left})$$
$$globalPos(b, v, H'+1, x+L'-1, t) \leftarrow on(b, v, Right, x, t) \quad (1 \leq x \leq n_{Right}).$$

We also need to update the definition of empty spaces, to include parts of the sides and what is between the sides. Instead of the following rule:

$$empty(h, x, t) \leftarrow \#count\{b : box(b), globalPos(b, v, h, x, t)\} = 0 \qquad (h > 0)$$

we can include the following rules:

$$partOfSide(h', x) \leftarrow globalPos(s, v, h, x, 0) \quad (h > 0, 0 \leq h' < h, s = Left, Right)$$
$$emptyOther(h, x, t) \leftarrow \#count\{b : box(b), globalPos(b, v, h, x, t)\} = 0,$$
$$\quad not\ partOfSide(h, x) \quad (0 \leq h)$$
$$empty(h, x, t) \leftarrow partOfSide(h, x)$$
$$empty(h, x, t) \leftarrow emptyOther(h, x, t).$$

*Overhangs* In overhang scenarios (Fig. 19), one of the required conditions about a final structure is that

- there exists a block $b$ supported by *Table* such that unit $v$ of $b$ is globally located at $x$ units from the leftmost side of the table, and
- the difference between the maximum overhang $z$ and the size of *Table* is equal to $x$.

$$\leftarrow \#count\{b : supported(b, Table, T), globalPos(b, v, \_, x, T),$$
$$\quad x = z - size(Table), overhang(z)\} = 0.$$

### 4.8. Maximizing/minimizing the heights of stacks

In block stacking scenarios, the goal may involve maximizing/minimizing the heights of stacks. A stack can be a tower or a tightly packed structure with or without holes, supported by a base (e.g., a block, blocks, table or sides of a bridge). For instance, Fig. 20 presents a stack with a height of 8 units.

*Maximizing the height* To maximize the height of a specified block $B$ in a stack, first we define a penalty for each block $b$ at the end of the plan, at time step $T$, inversely proportional to its height $h$ in a stack (e.g., $C - h$ where $C$ is the maximum height of a structure built by the existing blocks). Then, we add a weak constraint that minimizes the penalty for the specified block $B$, at the end of the plan, at time step $T$.

$$\overset{\sim}{\leftarrow} globalPos(B\_, h, \_, T), c = C - h.\ [c@1] \tag{19}$$

In Fig. 21, the height of the block $L2$ is maximized. Note that $L2$ is a large block of 4 units, whereas $S1, \ldots, S4$ are small blocks of size 1. Therefore, $L2$ cannot be supported by a tower of small blocks. Fortunately, $L2$ can be supported by two small blocks, as shown in the figure.

In Fig. 22, the height of block $L1$ is maximized by making use of the available blocks. It is important to note that, here the goal is not to connect the two sides.

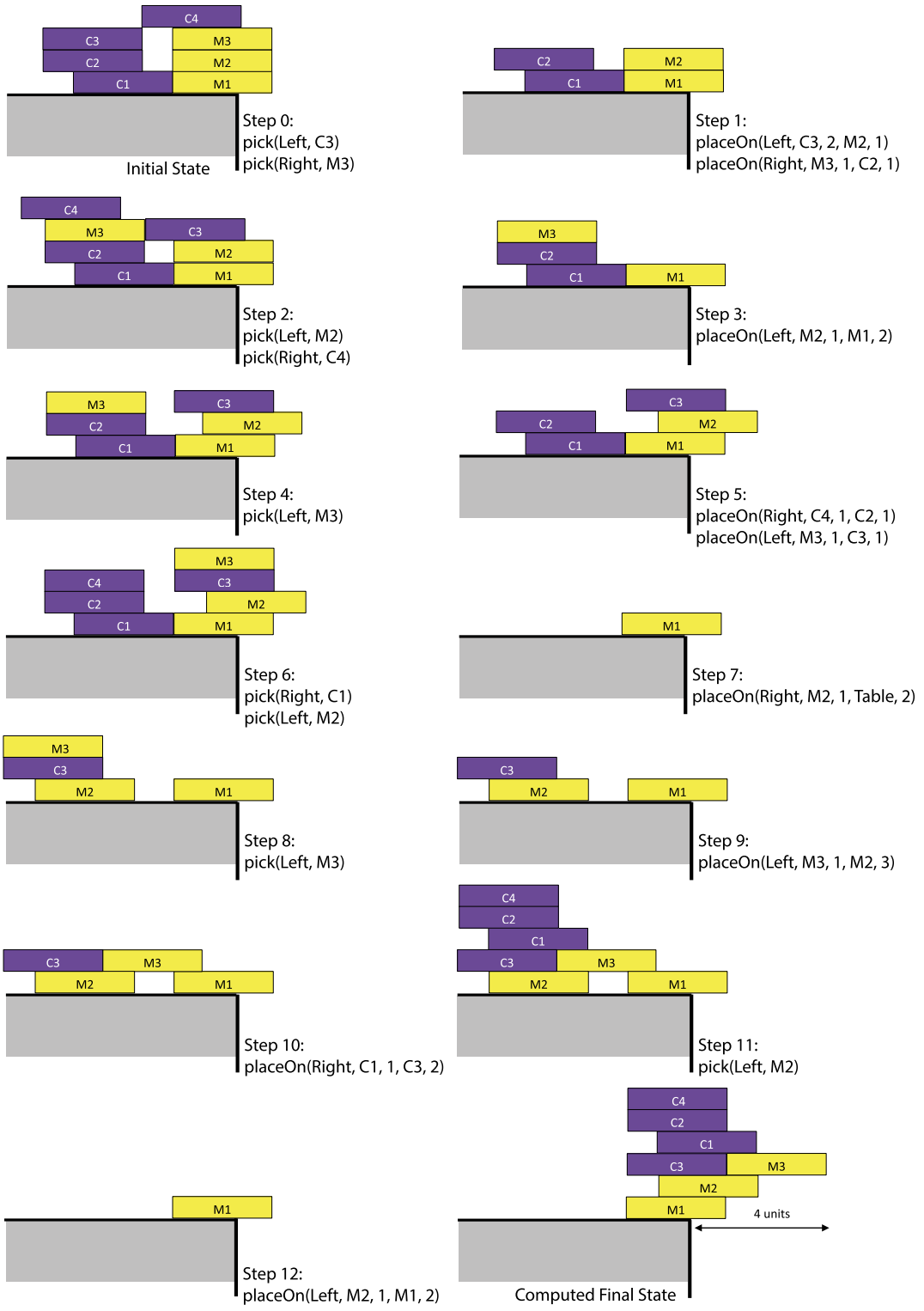



**Fig. 19.** A sample plan for a stable overhang construction.

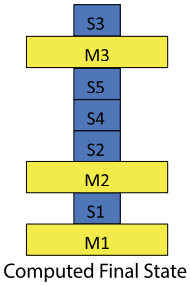



**Fig. 20.** Maximizing the height of block $S3$ in a stack of 8 blocks.

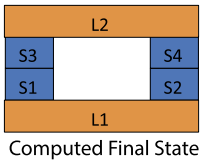



**Fig. 21.** Maximizing the height of block $L2$ in a stack of 6 blocks.

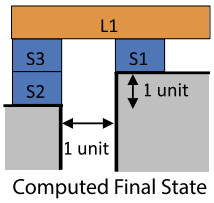



**Fig. 22.** Maximizing the height of block $L1$ in a stack of 4 blocks when there are two uneven sides with a height difference of 1 unit.

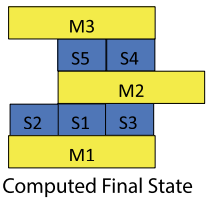



**Fig. 23.** Minimizing the height of block $S3$ in a stack of 8 blocks. Here, $M1$ is the base of the stack.

*Minimizing the height* Alternatively, the height of a specified block $B$ in a stack can be minimized using a penalty that is directly proportional to the height of the block.

$$\overset{\frown}{\leftarrow} globalPos(B\_, h, \_, T).\ [h@1] \tag{20}$$

Here, we can ensure that all blocks (including the one whose height is to be minimized) should be part of a stack built on block $L$.

$$\leftarrow not\ supported(b, L, T)\ \ (b{\neq}L). \tag{21}$$

In Fig. 23, the height of the block $S3$ is minimized, in a stack that involves all the blocks supported by the base block $M1$ on the table.

### 4.9. Soundness and completeness

The soundness of the proposed method with respect to the desired properties is provided by Propositions 3–7. The following proposition shows its completeness.

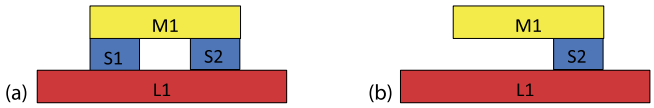



**Fig. 24.** (a) A final stable configuration. (b) An intermediate configuration of blocks obtained when feasibility checks are not used.

**Proposition 8.** *Suppose that the stability checking algorithm* $\Gamma$ *is correct (i.e., the construction is stable iff* $\Gamma$ *returns True). Let* $\Pi'$ *be the ASP program obtained from* $\Pi$ *by adding the supportedness and stability constraints (and, in case of bridge construction, also the connectedness constraints), and the ramification rules as described above. Then every robot construction plan that satisfies these desired properties and whose makespan is at most* $T-1$ *is characterized by an answer set for* $\Pi'$.

The proof follows from the representation methodology of the program: no constraint added to the program eliminates a valid robot construction plan.

## 5. Importance of feasibility check and hybrid planning for construction problems

Maintaining the stability of a structure is of prime importance in construction tasks. The structure should be stable during all the steps of construction. Ensuring stability at all times is challenging from both planning and geometric points of view.

As discussed in Section 4, in order to perform the continuous domain checks, we utilize external atoms *stable* and *hStable* implemented as Python functions.

In particular, we test stability numerically using the PyBullet physics engine. To ensure robust stability of our assemblies even under bounded disturbances, we adapt the notion of dynamic stability, in the sense of Fourier's inequality, which requires all objects to assume zero acceleration within a local neighborhood of their initial configuration, under the action of gravitational and friction forces [74]. In addition to gravitational forces, we consider small disturbance forces to the assembly and check for its configuration after some finite time interval. If the configuration of each object in the assembly stays within an empirically determined threshold from their initial location, we consider the assembly as dynamically stable.

As an example, consider the construction problem shown in Fig. 24, with two small blocks $S1$ and $S2$, a medium block $M1$, and a large block $L1$. All the blocks are initially on the table.

The goal conditions for a final configuration are specified by a set of facts:

$$goal(S1, L1).\ goal(S2, L1).\ goal(M1, S1).\ goal(M1, S2).\ goal(L1, Table).$$

According to this description, the small blocks $S1$ and $S2$ are on the long block $L1$, the block $M1$ is on $S1$ and $S2$, and the block $L1$ is on the table.

These goal conditions are ensured at a specified maximum step $T$ by constraints as follows:

$$\leftarrow goal(b, l), not\ onAux(b, l, T).$$

Note that since *onAux* atoms are projections of *on*, the final positions of objects are selected nondeterministically. We include the constraint above in the following examples where the goal is specified by atoms of the form $goal(B, L)$.

A possible final stable configuration is shown in Fig. 24(a). Such a configuration is achievable by the following hybrid plan of length 4 (with 6 actions), computed by DLVHEX as:

$$pick(Left, S2, 0), pick(Right, S1, 0),$$
$$placeOn(Right, S1, 1, L1, 2, 1),$$
$$placeOn(Left, S2, 1, L1, 4, 2), pick(Right, M1, 2),$$
$$placeOn(Right, M1, 3, S2, 1, 3).$$

According to this plan, first the smaller blocks are picked and placed on the long block; afterwards, the medium block is picked and placed on top of the two small blocks.

Now, let us consider a domain description without any feasibility checks. Then, DLVHEX computes the following non-hybrid task plan of length 4 (with 6 actions):

$$pick(Right, M1, 0), pick(Left, S2, 0),$$
$$placeOn(Left, S2, 1, L1, 4, 1),$$
$$placeOn(Right, M1, 3, S2, 1, 2), pick(Left, S1, 2),$$
$$placeOn(Left, S1, 1, L1, 2, 3).$$

According to this non-hybrid plan, first the medium block $M2$ and the small block $S2$ are picked, then the small block is placed on the longer block $L1$, then the medium block $M2$ is placed on $S2$. At this point, an unstable configuration is obtained, as shown in Fig. 24(b). Therefore, integration of feasibility checks is crucial to ensure generation of stable plans.

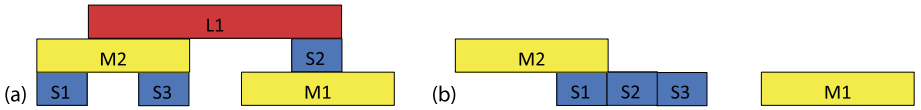



**Fig. 25.** (a) A final stable configuration. (b) An intermediate configuration of blocks, obtained when feasibility checks are not used.

Let us present another example. Consider the construction problem in Fig. 25 with six blocks on the table, $S1$, $S2$, $S3$, $M1$, $M2$, and $L1$, with the following goal conditions:

$$goal(M2, S1).\ goal(S2, M1).$$
$$goal(L1, S2).\ goal(L1, M2).$$
$$goal(S1, Table).\ goal(S3, Table).\ goal(M1, Table).$$

A possible final stable configuration is shown in Fig. 25. Such a configuration is achievable by the following hybrid plan of length 4 (with 6 actions), computed by DLVHEX:

$$pick(Left, M2, 0), pick(Right, S2, 0),$$
$$placeOn(Left, M2, 3, S3, 1, 1), placeOn(Right, S2, 2, M1, 1, 1),$$
$$pick(Right, L1, 2),$$
$$placeOn(Right, L1, 5, S2, 1, 3).$$

Without any feasibility checks, the following plan of length 4 (with 6 actions) is computed:

$$pick(Right, M2, 0)$$
$$pick(Left, L1, 1), placeOn(Right, M2, 3, S1, 1, 1),$$
$$pick(Right, S2, 2), placeOn(Left, L1, 1, M1, 3, 2),$$
$$placeOn(Right, S2, 1, M2, 1, 3).$$

This non-hybrid task plan is not feasible since it leads to an unstable intermediate configuration, where the medium block $M2$ is placed on the small block $S1$ at one end, leaving the center of mass outside the small block.

## 6. Benchmark instances and experimental evaluation

In this section, we introduce a diverse set of challenging robot construction benchmark instances. Precise definitions of these 21 benchmark instances, and a feasible construction plan computed by our method for each benchmark instance are provided in Appendix B, not to interfere with the readability of the manuscript.

The benchmark instances address different types of construction problems: Scenarios 1–8 necessitate multiple autonomous robots rearrange stacks of prefabricated blocks to build stable structures by making use of counterweights, scaffolding, subassemblies, and true concurrency of manipulations. Scenarios 9–11 focus on construction of stable overhangs with proper use of counterweights. Scenarios 12 and 13 and Scenarios 14 and 15 emphasize the connectivity of two sides to build stable symmetric and asymmetric (uneven) bridges, respectively. Finally, Scenarios 16–21 provide examples of challenging optimization problems while building stable stacks of prefabricated blocks or bridges.

While our main concern in this study is not computational efficiency, we have performed three sets of experiments using the domain description presented in the paper (without any further optimizations) to investigate the following aspects:

- Experiments 1: The scalability of the proposed hybrid method using the automated reasoner DLVHEX and PyBullet physics engine on table-top, overhang and bridge instances, in terms of computation time.
- Experiments 2: The effect of specifying goal description in more detail, on the computation time.
- Experiments 3: The effect of integrating stability checks on the computation time.

For all experiments, we have used the ASP solver DLVHEX 2.5 with PyBullet 2.4.1 physics engine installed on a Linux server with 16 2.4 GHz Intel® Xeon® E5-2665 CPU cores and 64 GB memory. For each benchmark scenario, six runs were performed in parallel with different solver settings (i.e., with configurations `handy` or `jumpy`); we report the CPU times of all six runs and highlight the fastest one in the tables below.

*Experiments 1: scalability* Table 1 presents the computational times for Scenarios 1–9 with one surface (i.e., the table). Table 1 indicates that Scenarios 1–9 (with 4–10 blocks of different sizes and weights, where the table size is restricted to 4–15 units of space) can be consistently solved within about 5 minutes. The makespans of plans range between 2 and 9. From these results, we observe the input size (cf. the product of the number of blocks and the size of the table) has a significant effect on the computation time: the computation time increases as the input size increases.

While a feasible solution of a 4 unit overhang is presented for Scenario 10 (with 7 blocks and surface space of 7 units) in Fig. B.36 of Appendix B, the computational time for this instance is not reported in Table 1 since a solution for a 4 unit

**Table 1**
Computation time for benchmark Scenarios 1–9.

| Instance | Plan Length | # Rules | Solver Configuration | # Runs | CPU Time [sec] |
|---|---|---|---|---|---|
| Scenario 1 (Fig. B.27) 10 blocks (table size 11) | 2 | 989316 | handy | 1 | 29.9 |
| | | | | 2 | 24.0 |
| | | | | 3 | 25.6 |
| | | | jumpy | 1 | **23.8** |
| | | | | 2 | 23.9 |
| | | | | 3 | 26.2 |
| Scenario 2 (Fig. B.28) 4 blocks (table size 6) | 6 | 285581 | handy | 1 | **6.8** |
| | | | | 2 | 7.2 |
| | | | | 3 | 9.5 |
| | | | jumpy | 1 | 7.0 |
| | | | | 2 | 8.2 |
| | | | | 3 | 8.4 |
| Scenario 3 (Fig. B.29) 4 blocks (table size 4) | 5 | 44697 | handy | 1 | 2.0 |
| | | | | 2 | 2.0 |
| | | | | 3 | 2.1 |
| | | | jumpy | 1 | **1.8** |
| | | | | 2 | 2.1 |
| | | | | 3 | 2.1 |
| Scenario 4 (Fig. B.30) 6 blocks (table size 7) | 4 | 259191 | handy | 1 | 7.6 |
| | | | | 2 | 10.8 |
| | | | | 3 | **6.1** |
| | | | jumpy | 1 | 6.8 |
| | | | | 2 | 7.2 |
| | | | | 3 | 6.5 |
| Scenario 5 (Fig. B.31) 5 blocks (table size 9) | 7 | 1745933 | handy | 1 | 108.2 |
| | | | | 2 | 83.0 |
| | | | | 3 | **78.0** |
| | | | jumpy | 1 | 186.2 |
| | | | | 2 | 175.8 |
| | | | | 3 | 96.4 |
| Scenario 6 (Fig. B.32) 5 blocks (table size 7) | 9 | 1194702 | handy | 1 | 2697.9 |
| | | | | 2 | 32.0 |
| | | | | 3 | 32.2 |
| | | | jumpy | 1 | 168.2 |
| | | | | 2 | 53.2 |
| | | | | 3 | **30.7** |
| Scenario 7 (Fig. B.33) 8 blocks (table size 15) | 4 | 6027747 | handy | 1 | 819.2 |
| | | | | 2 | 707.1 |
| | | | | 3 | 695.7 |
| | | | jumpy | 1 | **204.5** |
| | | | | 2 | 244.2 |
| | | | | 3 | 321.8 |
| Scenario 8 (Fig. B.34) 9 blocks (table size 14) | 4 | 6910706 | handy | 1 | 611.4 |
| | | | | 2 | 547.0 |
| | | | | 3 | 666.3 |
| | | | jumpy | 1 | **210.6** |
| | | | | 2 | 368.3 |
| | | | | 3 | 337.6 |
| Scenario 9 (Fig. B.35) 8 blocks (table size 5, overhang size 3) | 6 | 871040 | handy | 1 | **18.6** |
| | | | | 2 | 18.9 |
| | | | | 3 | 18.7 |
| | | | jumpy | 1 | 27.4 |
| | | | | 2 | 27.7 |
| | | | | 3 | 32.6 |

overhang cannot be computed within the time threshold of 5000 seconds. Similarly, a solution for Scenario 11 cannot be computed within the threshold. As expected, the computation time for overhang scenarios quickly become intractable as the maximum overhang size increases.

Table 2 provides the results of evaluations for symmetric and asymmetric bridge scenarios. In particular, the plans for Symmetric Bridge Benchmarks 1 and 2 characterize stable bridges over 3 and 5 units of gaps between two surfaces of the same height, respectively. The plans for Asymmetric Bridge Benchmarks 1 and 2 characterize stable bridges over 4 units of gaps between two surfaces of different heights with 2 and 3 units of height differences, respectively. Larger symmetric

**Table 2**
Computation time for symmetric and asymmetric bridge benchmarks.

| Instance | Plan Length | # Rules | Clasp Configuration | CPU Time [sec] | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | # Runs | With Stability Checks | | | Without Stability Checks | |
| | | | | | Connectivity Constraint | Abstract Goal | Exact Goal | Abstract Goal | Exact Goal |
| Symmetric Bridge Benchmark 1 (Fig. 17(a)) | 2 | 45498 | handy | 1 | 2.0 | 1.7 | 1.7 | **0.9** | 1.2 |
| | | | | 2 | 2.0 | 1.8 | **1.6** | 1.0 | 1.1 |
| | | | | 3 | 2.0 | 1.7 | 2.1 | **0.9** | **0.9** |
| | | | jumpy | 1 | 1.8 | 1.7 | 2.4 | **0.9** | 1.2 |
| | | | | 2 | **1.7** | **1.6** | **1.6** | **0.9** | 1.2 |
| | | | | 3 | 1.8 | 1.7 | 2.5 | **0.9** | 1.2 |
| Symmetric Bridge Benchmark 2 | 4 | 259191 | handy | 1 | 537.4 | 114.7 | **46.7** | 48.9 | 36.8 |
| | | | | 2 | 133.1 | 118.8 | 53.1 | 77.7 | **34.9** |
| | | | | 3 | 459.5 | 150.1 | 51.8 | 95.4 | 37.8 |
| | | | jumpy | 1 | **71.9** | **61.2** | 54.7 | **47.5** | 37.8 |
| | | | | 2 | 398.4 | 68.4 | 58.4 | 60.4 | 38.0 |
| | | | | 3 | 540.8 | 76.5 | 57.4 | 60.4 | 35.2 |
| Asymmetric Bridge Benchmark 1 (Fig. B.40) | 6 | 1745933 | handy | 1 | 2693.8 | 548.2 | **145.9** | 459.5 | 108.0 |
| | | | | 2 | 434.9 | 430.6 | 292.1 | 484.9 | 117.0 |
| | | | | 3 | 790.6 | 585.6 | 180.1 | 297.3 | 139.6 |
| | | | jumpy | 1 | **163.9** | 438.3 | 161.7 | 292.3 | **72.3** |
| | | | | 2 | 605.7 | **409.6** | **145.9** | 312.1 | 105.8 |
| | | | | 3 | 298.2 | 461.3 | 173.0 | **261.9** | 113.3 |
| Asymmetric Bridge Benchmark 2 | 6 | 1688641 | handy | 1 | **1023.0** | 423.9 | 209.2 | 83.6 | 52.1 |
| | | | | 2 | 3877.6 | 897.4 | 310.3 | 117.2 | 68.3 |
| | | | | 3 | 3688.5 | 776.5 | 188.5 | 115.0 | 90.4 |
| | | | jumpy | 1 | 6967.0 | **92.7** | 171.3 | 94.7 | 53.2 |
| | | | | 2 | 5209.5 | 361.4 | **113.0** | **79.2** | 60.3 |
| | | | | 3 | timeout | 297.5 | 203.9 | 101.4 | **50.4** |

and asymmetric bridge scenarios shown in Figs. B.38, B.39, and B.41 are not included in the evaluations, as these instances cannot be solved within the 5000 second time threshold.

*Experiments 2: granularity of goal specifications* There may be multiple goal configurations of blocks depending on the desired conditions about the final structure, and all of them may not be stable. In such cases, finding a stable goal configuration is a challenge.

To evaluate the effect of the granularity of goal specifications on the computational efficiency, three different descriptions of a goal configuration are tested:

- Case (i) only the connectivity of the two sides of a river is provided as a constraint to describe the goal;
- Case (ii) in addition to the connectivity constraint, an abstract description of which block is on top of which other block is specified; and
- Case (iii) in addition to the connectivity constraint, for each block, an exact definition of which unit is on top of which unit of other blocks is provided.

Intuitively, Case (i) simply asks the system to "build a bridge that connects two sides", Case (ii) specifies to "build a bridge where block $M1$ is on block $M3$, etc.", and Case (iii) specifies to "build a bridge where the unit space 2 of block $M1$ is on the unit space 3 of the block $M3$, etc.".

The results of our experiments are presented in Table 2. The results indicate that, similar to the overhang scenarios, the computation time for bridge scenarios quickly becomes intractable as the gap and the height difference between the sides increases.

Moreover, Table 2 provides evidence that the computation time decreases, as the goal specifications become more precise. In particular, Case (iii) takes the least amount of time, as a stable goal configuration is provided as an input to the automated reasoner.

*Experiments 3: stability checks* To evaluate the effect of integrating stability checks on the computational efficiency, the experiments for Case (ii) and Case (iii) were also run when the feasibility checks were turned off (i.e., the stability checks return *True* for all instances). It is important to underline that we have not modified the domain description. Instead, we have modified the stability checkers so that they always return a positive response. We have also ensured that, for each scenario, the length of the plan is the same as the length of the plan computed with stability checks. In this way, the sizes of the programs are very close to each other, with or without stability checks.

Please note that, the results of computations without the stability checks are highly likely to be infeasible as the stability of the intermediate and/or goal states are not ensured. Along these lines, these results need to be verified with a physics engine and some sort of replanning needs to be performed until a plan is verified to be feasible [31].

Table 2 shows that the integration of the stability checks increases the computation time, especially for larger scenarios. For instance, the results for Asymmetric Bridge Benchmark 2 indicate that, when an abstract goal configuration is provided, the computation of a solution with the feasibility checks takes 2 to 5 times longer. However, please note that since the feasibility of a plan cannot be ensured when the stability checks are turned off, it is likely that more than 5 different plans need to be computed until a plan can be verified to be feasible. Hence, integration of the stability checks, in general, presents a more efficient means to compute a feasible plan as discussed in [31].

*Further discussions* Let us discuss the results of these experiments in the light of how DLVHEX handles external atoms. DLVHEX replaces external atoms with auxiliary atoms, and introduces auxiliary rules so that the answer sets of the original program correspond to a subset of the answers sets of the resulting program where the auxiliary atoms faithfully represent the values of the external atoms. While this compatibility check is done, DLVHEX learns "nogoods" describing conditions which contradict the semantics of external atoms, and utilizes nogoods to restrict the search space.

While the strategy of learning nogoods improves the computational performance for certain domains, it has been observed by Eiter et al. [24] in a robotic domain that this strategy may lead to inefficient computations: since the external atoms depend on a large part of the interpretation (e.g., locations of all blocks), learning nogoods from evaluations of external atoms cannot simply cut away significant portions of the search space. We make similar observations in robot construction problems.

Furthermore, in particular in the bridge and overhang scenarios, we observe an increase in the computation time when the goal condition is provided as a set of connectivity constraints, leading to a very large number of possible goal configurations for the blocks. Note that the computation times decrease as the goal conditions become more restrictive, as can be observed under the abstract goal and exact goal conditions.

## 7. Solving robot construction problems with hybrid planning: implementation and execution

To verify the executability of the plans computed by our method and to show their applicability with real robots, we have completed dynamic simulations and physical implementations of several benchmark scenarios.

We have considered the scenarios shown in Figs. 5, 6 and 19 as they present several interesting scenarios to show the applicability of our formal approach to address challenging robot construction problems.

Figs. 5 and 6 present four scenarios that demonstrate the need for (a) manipulation of subassemblies, (b) utilization of counterweights, (c) use of scaffolds, and (d) true (non-serializable) concurrency to ensure stable construction of certain structures.

For instance, Fig. 5a demonstrates manipulation of subassemblies, where a sub-assembly comprises of two or more blocks being manipulated together by a single manipulator. The stability of the assembly and the subassemblies are ensured throughout the plan. Note that, in this scenario, not only it is challenging to decide on which sub-assemblies may be used for efficient construction, but also to ensure the stability of the sub-structures and represent the effects of manipulating a sub-assembly, due to ramifications.

Fig. 5b presents a scenario where counterweights are introduced temporarily to balance the weight of the structure, so that it remains stable during the construction. At the time step 3, the robot places $M1$ with its $Left$ arm on $L1$ to be used as a temporary counterweight. Note that this counterweight ensures the stability of the construction at the next time step. In the end, the robot places $M1$ on the table, as $M1$ is used only as a counterweight to temporarily to balance the structure and is not part of the final stable configuration. Note that deciding the use of counterweights as part of a hybrid plan is challenging.

Fig. 6a presents a scenario where scaffolding is used to temporarily support the construction. In scaffolding, instead of supporting the structure from above by introducing a heavy object, the structure is supported from below. Note that, at time step 3, $S4$ is placed as a scaffold to support the structure. In the end, the robot moves $S4$ away from the structure as it is not part of the final stable configuration. Note that deciding the use of scaffolds as part of a hybrid plan is challenging.

Fig. 6b demonstrates a scenario where non-serializable true concurrency is required. In this scenario, we assume that counterweights, scaffolds or subassembly manipulations are not allowed. At time step 4, the robot concurrently places $S1$ and $S2$ on either end of $L1$ to achieve a final stable configuration. Note that if the last actions are not executed concurrently, the structure would become unstable. Allowing true concurrency in planning is a challenging problem, from the perspectives of both representation and reasoning.

Fig. 19 depicts a plan to construct an overhang of 4 units from the edge of the table by a bimanual manipulator. Here, blocks $C1$–$C4$ are used as counterweights and blocks $M1$–$M3$ are used to create the overhang.

### 7.1. Dynamic simulation

Dynamic simulations are implemented in Gazebo [56] version 7. The positions of all the blocks are known initially. After computing a task plan, OMPL [90] is used to generate a motion plan for every action in the construction plan, and MoveIt [14] commander is used to follow that motion plan.

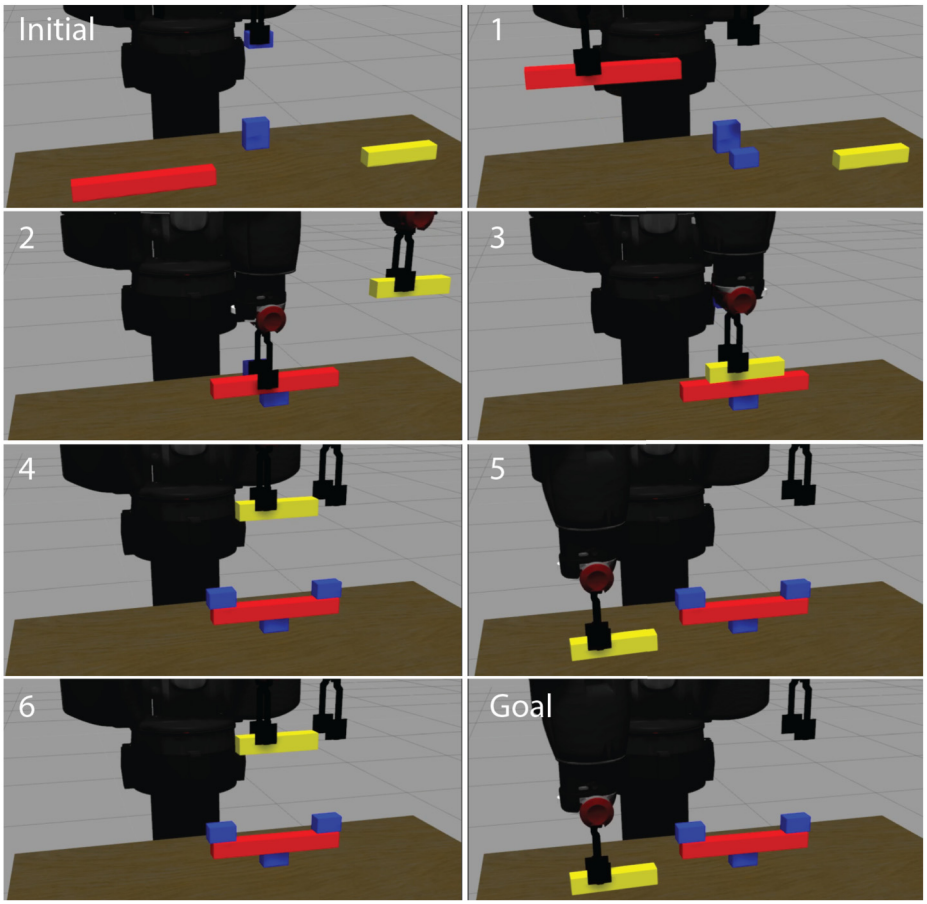



**Fig. 26.** Snapshots present dynamic simulation of a sample construction problem with a bimanual Baxter robot.

Fig. 26 presents snapshots from a dynamic simulation of a sample construction problem with a bimanual Baxter robot. In this scenario, the medium block is used as a counter weight to prevent the structure from falling.

Initially all the blocks are on the table and the robot picks up a small block, as in Snapshot Initial. The robot places the small block on the table and picks the large block in Snapshot 1. Robot places the large block on the small block and picks the medium block in Snapshot 2. The robot places the medium block on the large block in Snapshot 3. The robot places the small block on the large block and picks up another small block in Snapshot 4. The robot places the small block on the large block in Snapshot 5. The robot picks up the medium block in Snapshot 6 and places it on the table in Snapshot Goal.

Dynamic simulations of several interesting construction planning scenarios are presented at https://www.youtube.com/watch?v=LDJIH_dViOU.

### *7.2. Execution*

After a construction plan is computed for a particular scenario, and continuous motion trajectories for robot manipulators are computed using OMPL, a physical execution is performed.

We have used a Baxter robot, a bimanual robot with two seven degrees of freedom arms, for execution of construction plans. Three kinds of blocks are utilized: small, medium and large. The size of the smallest block is $30 \times 30 \times 30$ mm$^3$, the size of a medium-sized block is $90 \times 30 \times 30$ mm$^3$, and the size of the largest block is $150 \times 30 \times 30$ mm$^3$. All blocks are machined from aluminum, and are spray-painted with different colors to simplify object identification.

A video of a sample physical execution of construction plans is also presented at https://www.youtube.com/watch?v=RyUeMaBERtw.

## 8. Related work

To the best of authors' knowledge, this is the first robotic construction study that addresses a variety of multi-robot stack rearrangement planning problems for building stable structures of different sorts. In the literature, there exist several studies that focus on different specific aspects of the robotic construction task: deciding for the stability of a given structure

(e.g., from an image obtained from Angry Birds), deciding for the existence of a specified stable structure (e.g., a maximum overhang) from a given set of identical blocks or an unspecified stack from a given set of different sizes of objects (e.g., like stones), planning for towers of identical blocks (e.g., the blocks world) ignoring stability, etc. Let us go over them to better emphasize the challenges of the robot construction problems that we study.

**The blocks world:** The well-known (elementary) blocks world problems [105] have been widely studied by AI community. It is proven to be NP-complete to decide the existence of a plan whose makespan is less than a given positive integer [46, Theorem B.5]. Blocks world problems are quite restricted compared to robot construction problems, since while proposing the problem, Winograd's interest was in language rather than in construction problems. For instance, the blocks world deals with identical blocks and allows a block to be placed on a flat surface or on another block, but not on multiple blocks as necessitated by the robot construction problems. It does not allow manipulation of subassemblies, use of counterweights and scaffolds, or concurrent placements of blocks, either. Also, there is no consideration of feasibility checks to ensure the stability of the stack at each step of a plan.

Later, Fahlman [36] has introduced a set of robot construction problems where the goal is for a robot to build specified structures out of simple blocks of different shapes and sizes. These problems allow incorporation of subassemblies into the final design, and the use of extra blocks as temporary supports or counterweights during construction; they also consider collisions of blocks and instability of the structures, but not motion planning. Fahlman's main interest was in maximizing common sense (rather than soundness, completeness or optimality). He implemented a planning system (called BUILD) to solve some of these problems. BUILD utilizes the programming language CONNIVER [68,91] to control and invoke procedures (rather than backtracking search based on failure), guided with heuristics. Therefore, as Fahlman notes in his paper [36], most of the effort in BUILD is spent to implement procedures (e.g., for constructing a subassembly, or for checking stability check of a subassembly). Fahlman's robot construction problems have not been investigated with a formal approach since then.

**Maximum overhang puzzle:** Mathematicians and theoretical computer scientists have studied a classic puzzle that aims to determine the maximum overhang achievable by a stack of identical blocks [47,76,75,77]. A relatively recent solution [75, 77] to this 150 year old puzzle, honored with the prestigious David P. Robbins Prize in mathematics, has introduced the use of blocks as counterbalance to improve upon the well-established solution. While the maximum overhang problem focuses on the determination of a stable and optimal final configuration of identical blocks, the planning aspects of the construction problem to attain the goal configuration is not considered within the scope of these studies.

**Image understanding and qualitative reasoning in games:** Applications in scene understanding from 2D pictures and computer games require inferring physical relations among objects [46,54,86]. Determination of stability of stacked objects and supportedness among objects have been studied, commonly with qualitative reasoning approaches [97]. Determination of stable final configuration of constructions has also been studied in computer games, like Angry Bird [37,86,13]. These studies focus on the physical relations of a given final configuration and do not address the block rearrangement problem to build stable constructions.

**Stability of assemblies:** In robotics, static stability [6,63,64,82,66,106,102,69] and dynamic stability [74,84] of assemblies with and without friction have been thoroughly studied. The computational complexity of determining the assembly stability in 2D is established in [73]. The stability determination techniques have been utilized in several robotic applications, that include a Jenga playing robot [99], multiple robots building a ramp [70], an autonomous robot stacking a balancing vertical tower out of irregularly shaped stones [38], and a robot dry stacking irregular objects to build large piles [92]. Note that, in these studies, the challenging task planning aspect of construction planning has not been addressed.

Toussaint [95] has utilized stability checks for building some tallest stable tower from a set of unlabeled cylinders and blocks; no goal condition is specified. His method applies a restricted version of task planning to decide for the order of manipulation actions, based on simple Strips operators and Monte Carlo tree search, and considers a restricted form of stability check that depends on whether the objects are placed on support areas of other objects. Due to these restrictions, his method is limited to building towers with sequential plans.

Note that for sophisticated constructions that involve temporary scaffolding, counterweights, and subassemblies, it is necessary to express ramifications of actions as well as true concurrency. However, as discussed in the introduction, expressing ramifications directly by simple Strips operators is challenging [93, Theorem 3] due to lack of logical inference. Also, expressing true concurrency directly is challenging as the domain description needs to be extended with exponential number of new operators, where each operator characterizes a concurrent action. Due to these theoretical results, other studies [32,48,96] that rely on simple Strips operators, do not present general methods for such sophisticated constructions either.

It is important to note that these methods do not cover sophisticated structures, like bridges or overhangs, since objects are not necessarily placed on support areas of other objects. Such sophisticated structures require definition of transitive closure to ensure supportedness or connectedness. Transitive closure is not definable in first-order logic [35, Theorem 5]; it is not directly supported by Strips either [93], in general.

**Assembly planning:** In automated manufacturing, assembly plans aim to determine the proper order of assembly operations to build a coherent object. During assembly planning, the goal configuration is well-defined and the problem is generally approached by starting with the goal configuration and working backwards to disassemble all parts. Object stability has also been considered within this context [8,62,83,104,80,5,98]. The assembly planning problem is significantly different from the robotic construction problems: on the one hand, it allows assembly of irregular objects; on the other

hand, the goal configuration is pre-determined and solutions are commonly restricted to monotone plans where an object is moved at most once.

**Rearrangement planning:** Geometric rearrangement with multiple movable objects and its variations (like navigation among movable obstacles [88,87]) have been studied in literature. Since even a simplified variant with only one movable obstacle has been proved to be NP-hard [103,19], many studies introduce several important restrictions to the problem, like monotonicity of plans [16,20,89,2,72,58,57]. While a few can handle nonmonotone plans [52,59]; these studies do not allow stacking either. Recently, Han et al. [49] study rearrangement of objects in stack-like containers (by pushes and pops); these problems do not require stability checks.

**Hybrid planning:** Hybrid planning has been generally concerned about the problem of combining task planning with feasibility checks. Recent work on hybrid planning usually considers feasibility checks based on motion planning, and can be described in three groups with respect to their computational approaches: (i) by developing/modifying search algorithms for task planning that utilize motion planning [45,51,3,55,60,85,1], (ii) by utilizing formal methods and relevant solvers [78, 17,18], or (iii) by formally embedding motion planning as part of representations of actions (in the spirit of semantic attachments [101]) and using relevant automated reasoners [11,21,53,39,27,31,4,40,94]. The studies in the third group are further extended to HTN planning [65] and conditional planning [71].

Our approach to hybrid planning for construction problems is similar to the studies in the third group since we embed feasibility checks in action descriptions, and utilize ASP solvers to solve hybrid planing problems. On the other hand, our study is different from these studies in two important ways, as necessitated by the challenges of construction problems. First, we consider stability checks as feasibility checks. The stability checks require the locations of all blocks in the environment, including the ones on surfaces and the ones being held. This requirement brings about the second important difference: embeddings of terms (i.e., object constants and variables) in semantic attachments as in the related approaches are not sufficient anymore, since the stability checks necessitate extensions of predicates (e.g., the predicates *on* and *holding*). Currently, there exists one automated reasoner (i.e., DLVHEX) that supports embeddings of predicate extensions in semantic attachments and that can be used for planning.

**Our earlier studies:** We have been studying on various hybrid reasoning tasks (e.g., assembly planning [11,27], rearrangement planning [52], conditional planning [79,71], diagnosis [81], explanation generation [15]) for robotics applications. In these studies, we have considered different feasibility checks based on motion planning, such as reachability and graspability checks, and investigated hybrid reasoning methods based on action languages [42,44] or ASP languages [12] to find solutions. From the perspective of hybrid planning applied to manipulation tasks, this study is different from our earlier studies in the following ways. First, manipulation tasks for a construction involve not only table-top picks and places but also stacking, subassemblies, bridges and overhangs. These possibilities lead to global constraints and goal conditions that require recursive definitions. Also, due to different sizes of blocks, manipulation actions lead to many sophisticated positive and negative ramifications that utilize recursive definitions. As discussed above, construction problems require stability checks whose embeddings in action descriptions necessitate extensions of predicates; this is challenging for existing automated reasoners.

## 9. Conclusion

We have studied multi-robot construction problems that are challenging for both AI and Robotics not only due to modeling challenges (e.g., ramifications of manipulation actions, true concurrency of actions, supportedness of blocks by other blocks), but also due to necessity of stability checks of constructions as they are being built. We have addressed these challenges by a hybrid planning framework developed over the logic-based formalism and automated reasoners of Answer Set Programming (ASP): ASP allows true concurrency, embedding outcomes of stability checks into state constraints by external atoms, recursive definitions of sophisticated concepts, like supportedness and connectedness, and nested recursive definitions of global positions of blocks from their relative positions. Thanks to the expressive knowledge representation languages of ASP, our hybrid planning framework is general enough to solve a variety of construction problems.

We have provided theoretical guarantees on soundness and completeness for our formal framework, with respect to the desired properties of constructions (e.g., absence of nonsensical structures such as circular configurations of blocks, connectedness of the two sides of a bridge, stability of constructions).

We have introduced a diverse set of challenging robot construction benchmark instances that include bridges and overhangs constructed with counterweights, scaffolding and true concurrency of manipulations. We have discussed the usefulness of our framework over these instances, performed experiments to investigate the computational performance of our hybrid method, and demonstrated the applicability of our method using a bimanual Baxter robot. Such a benchmark set of different types of construction and such demonstrations with real robots are useful to advance studies on robot construction problems, hybrid planning, and knowledge representation.

*Future work* Robot construction problems present further interesting challenges to knowledge representation and reasoning, planning, and robotics. For instance, allowing blocks of different heights, width, shapes, or orientations would lead to more sophisticated ramifications. Allowing blocks to be manipulated by multiple robots, or considering environments with restricted spaces would require more involved feasibility checks.

*Discussion* This study has been highly interdisciplinary and involved handling challenges from the perspectives of knowledge representation and reasoning, planning, and robotics. It has led to synergies not only between planning and robotics, but also between knowledge representation (in particular, reasoning about actions and change) and planning.

We believe that the challenges involved in formally representing the construction domain and devising methods that can provide solutions to construction problems with soundness and completeness guarantees, are inspiring for further studies in knowledge representation and reasoning, planning, and robotics.

In this study, we have addressed the challenges of construction problems using the knowledge representation and reasoning paradigm of ASP, since it has provided us with an expressive language and an efficient solver with the capability of integrating external computations. Note that ASP languages have not been introduced specifically to reason about actions and change or planning; ASP is applied to solve problems in a variety of applications [42]. Therefore, investigating the challenges of construction problems using languages particularly designed for reasoning about actions and change and for planning is an open research problem.

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Data availability

Data will be made available on request.

## Acknowledgements

We thank Stefan Edelkamp and Robert Mattmueller for useful discussions about the problem of representing global constraints in PDDL, and methods to handle it with the current state-of-the-art planners. We thank Scott Fahlman for sharing his experiences on robot construction problems and for useful discussions; they have provided a better understanding of these problems for us. We also thank the Editor-in-Chief, the Associate Editor, and the anonymous reviewers for their useful suggestions and remarks to improve the manuscript.

## Appendix A. Proofs

Let us first present the definitions and theorems that we use in the proofs of Propositions 3–7.

### A.1. Transitive closure in ASP

Erdem and Lifschitz [29] consider nondisjunctive programs that consist of rules of the form

$$Head \leftarrow Body,$$

where *Head* is a literal or $\bot$ and *Body* is a formula, and consider the following definition *Def* of the transitive closure of $tc$ of a binary relation $p$:

$$\begin{array}{l} tc(x,y) \leftarrow p(x,y) \\ tc(x,y) \leftarrow p(x,v), tc(v,y). \end{array} \tag{A.1}$$

Then they prove its correctness:

**Proposition 4 of [29].** Let $\Pi$ be a program that does not contain atoms of the form $tc(x,y)$ in the heads of rules. If $X$ is an answer set for $\Pi \cup Def$ then $\{\langle x,y\rangle : tc(x,y) \in X\}$ is the transitive closure of $\{\langle x,y\rangle : p(x,y) \in X\}$.

Moreover, they provide a syntactic condition to ensure the well-foundedness of $p$:

**Proposition 5 of [29].** If $\Pi$ contains a constraint

$$\leftarrow tc(x,x)$$

and $C$ is finite, then for every set $X$ of literals that is closed under $\Pi \cup Def$, the set $\{\langle x,y\rangle : p(x,y) \in X\}$ is well-founded.

### A.2. Equivalent transformations in ASP

Erdem and Lifschitz [28] consider disjunctive programs that consist of rules of the form:

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

where $m, k \geq 0$, each $\alpha_i$ and each $\beta_i$ are literals. They prove that, in such programs, some disjunctive rules can be replaced by nondisjunctive rules, preserving the answer sets for the program.

**Theorem 1 of [28].** For any program $\Pi$ and any atom $p$, the programs

$$p \vee \neg p \leftarrow$$

and the program

$$p \leftarrow not\ \neg p$$
$$\neg p \leftarrow not\ p$$

have the same answer sets.

### A.3. Definitions and constraints in ASP

Erdogan and Lifschitz [33] consider a more general form of programs that consist of rules of the form

$$Head \leftarrow Body,$$

where *Head* and *Body* are formulas, and prove that adding a definition to such a program produces a conservative extension.

**Proposition 3 of [33].** Let $\Pi_1$ be a program and $Q$ be a set of atoms that donot occur in $\Pi_1$. Let $\Pi_2$ be a program that consists of rules of the form

$$q \leftarrow F$$

where $q \in Q$, and $F$ does not contain any element of $Q$ in the scope of negation as failure. Then $Z \mapsto Z \setminus Q$ is a 1-1 correspondence between the answer sets for $\Pi_1 \cup \Pi_2$ and the answer sets for $\Pi_1$.

They also prove that adding constraints eliminate undesired answer sets.

**Proposition 2 of [33].** For any program $\Pi$ and a formula $F$, a set $Z$ of literals is an answer set for $\Pi \cup \{\leftarrow F\}$ iff $Z$ is an answer set for $\Pi$ and does not satisfy $F$.

### A.4. Defining reachability in ASP

A binary relation $p$ over a finite set $C$ of object constants, can be viewed as a digraph where the vertices denote elements of $C$ and the edges denote the relation $p$. In this graph, the reachability of a vertex $x$ from a given vertex *Source* by a path whose length is at most $k$ can be defined in ASP as follows:

$$\begin{array}{l} reachable(1, x) \leftarrow p(Source, x) \\ reachable(n+1, x) \leftarrow p(y, x), reachable(n, y). \qquad (1 \leq n < k) \end{array} \tag{A.2}$$

Let us denote this definition of reachability by *Def*. The following lemmas state that this definition is correct.

Consider programs that consist of rules of the form

$$Head \leftarrow Body,$$

where *Head* is a literal or $\bot$ and *Body* is a formula.

For every $n$ $(1 \leq n < k)$, let $R_n$ denote the set of vertices reachable from *Source* by a path defined with respect to $p$ and whose length is at most $n$.

**Lemma 1.** *Let $\Pi$ be a positive program that does not contain atoms of the form reachable$(n, y)$ in the heads of rules, and that defines the binary relation $p$. Let $X$ be the answer set for $\Pi \cup Def$. For every $n$ $(1 \leq n < k)$ and for every vertex $y$, reachable$(n, y) \in X$ iff $y \in R_n$.*

**Lemma 2.** *Let $\Pi$ be a program that does not contain atoms of the form reachable$(n, y)$ in the heads of rules, and that defines the binary relation $p$. If $X$ is an answer set for $\Pi \cup Def$ then every element of the set $\{y : reachable(n, y) \in X, 1 \leq n < k\}$ is reachable from Source by a path whose length is at most $n$. Furthermore, for every vertex $x$ reachable from Source by a path whose length is at most $n$ $(1 \leq n < k)$, there is some answer set $X$ for $\Pi \cup Def$ such that $x$ is in $\{y : reachable(n, y) \in X, 1 \leq n < k\}$.*

**Proof of Lemma 1.** Note that, since $\Pi$ is a positive program, it does not contain negation as failure. Let $X$ be the answer set for $\Pi \cup Def$. Take $n$ to be any number such that $1 \leq n < k$, and take $x$ to be any vertex. We need to prove that $reachable(n, y) \in X$ iff $y \in R_n$.

*Left-to-right.* Since there is no negation as failure in $\Pi$, $X$ can be characterized as the union of $\bigcup_{i=0}^{k} X_i$ of the sequence of sets of literals defined as follows: $X_0 = \emptyset$; $X_{i+1}$ is the set of all literals $L$ such that $\Pi \cup Def$ contains a rule $L \leftarrow Body$ with $Body$ satisfied by $X_i$.

We show by induction on $i$ that $reachable(i, y) \in X_i$ implies $y \in R_i$. If $i = 0$, the assertion holds since $X_0 = \emptyset$. Assume that, for every $x$ and $i$, $reachable(i, y) \in X_i$ iff $y \in R_i$. Take any atom $reachable(i+1, y)$ from $X_{i+1}$. Take a rule $reachable(i+1, y) \leftarrow Body$ in $\Pi \cup Def$ such that $X_i$ satisfies $Body$. Since $\Pi$ does not contain atoms of the form $reachable(n, y)$ in the heads of rules, this rule belongs to $Def$. Since $i > 0$, $Body = p(y, x), reachable(i, y)$. Then $p(y, x), reachable(i, y) \in X_i \subseteq X$. By induction hypothesis, $y \in R_i$. Then, $x$ is reachable from $Source$ by a path of length $i + 1$.

*Right-to-left.* Since $X = \bigcup_{i=0}^{k} X_i$, it is sufficient to prove that, for every $i > 0$, $y \in R_i$ implies $reachable(i, y) \in X$. The proof is by induction on $i$. When $i = 1$, $y \in R_1$ so that $p(Source, y) \in X$. Since $X$ is closed under $Def$, it follows that $reachable(1, y) \in X_1$. Assume that, for every $i > 1$, $y \in R_i$ implies $reachable(i, y) \in X$. Take $x$ from $R_{i+1}$. Then there is a vertex $y$ in $R_i$ such that there is an edge from $y$ to $x$. Then $p(y, x) \in X$ and, by induction hypothesis, $reachable(i, y) \in X$. Since $X$ is closed under $Def$, it follows that $reachable(i + 1, y) \in X$. □

**Proof of Lemma 2.** Let $\Pi$ be a program that may contain negation as failure but that does not contain atoms of the form $reachable(n, y)$ in the heads of rules. Suppose that $\Pi$ defines a binary relation $p$. That is, every answer set $Y$ for $\Pi$ characterizes a binary relation $p$. For every $n$ $(1 \leq n < k)$ and $Y$, let $R_{Y,n}$ denote the set of vertices reachable from $Source$ by a path defined with respect to $p$ defined in $Y$, and whose length is at most $n$.

Let $X$ be an answer set for $\Pi \cup Def$. Then $X$ is the answer set for $\Pi^X \cup Def^X = \Pi^X \cup Def$. Let $Q$ be the set of all atoms of the form $reachable(n, y)$. By Proposition 3 of [33], $X \setminus Q$ is the answer set for $\Pi^X$ and thus defines $p$. By Lemma 1 applied to $\Pi^X$, for every $n$ and $y$, $reachable(n, y) \in X$ iff $y \in R_{X \setminus Q, n}$. □

### A.5. Proof of Proposition 1

Let us recall the definition (5) of the global location of a block vertically in a tower on the table:

$$globalPos(b, v, 1, x, t) \leftarrow on(b, v, Table, x, t)$$
$$globalPos(b, v, h, x, t) \leftarrow globalPos(b', u, h{-}1, x, t), on(b, v, b', u, t),$$

and the definition (6) of the global location of a block horizontally with respect to a block at the same height:

$$globalPos(b, v{+}1, h, x{+}1, t) \leftarrow globalPos(b, v, h, x, t) \quad (v{<}size(b))$$
$$globalPos(b, v{-}1, h, x{-}1, t) \leftarrow globalPos(b, v, h, x, t) \quad (v{>}1)$$

**Proposition 1.** *Let $\Pi'$ be the disjunctive program shown in Figs. 7–11. Then, for every block $b$ supported by the table, rules (5) ∪ (6), when added to $\Pi'$, correctly describe its global position with respect to its height $h$ from the table and the distance $x$ from the leftmostside of the table.*

The proof of Proposition 1 follows from the observation that $\Pi'$ does not contain atoms of the form $globalPos(b, v, h, x, t)$ in the heads of its rules, that the disjunctive program $\Pi'$ can be equivalently transformed into a nondisjunctive program (Theorem 1 of [28]), and the correctness of the recursive reachability definition (Lemma 2).

**Proof.** Let $\Pi'$ be the disjunctive program shown in Figs. 7–11. First, observe that, thanks to Theorem 1 of [28], each disjunctive rule like

$$pick(a, b, t) \vee \neg pick(a, b, t) \leftarrow$$

describing the occurrences of an action can be replaced a pair of nondisjunctive rules as follows

$$pick(a, b, t) \leftarrow not\ \neg pick(a, b, t)$$
$$\neg pick(a, b, t) \leftarrow not\ pick(a, b, t).$$

Let $\Pi''$ denote this nondisjunctive program.

With the program (5) ∪ (6), at every time step $t$ and for every block $b$ supported by the table, we identify the global locations of a unit space $v$ of a block $b$ by its height $h$ from the table, and by the number $x$ of unit spaces it is from the leftmostside of the table.

In fact, at every time step $t$, we consider an immediate-connectedness relation $p$ between a block unit $\langle b, v\rangle$ and a block/table unit $\langle b', v'\rangle$ iff one of the following three conditions hold: (i) $\langle b, v\rangle$ is located on $\langle b', v'\rangle$ (specified by atoms of

the form $on(b, v, b', v', t)$), (ii) $b = b'$ and $v' = v + 1$, or (iii) $b = b'$ and $v' = v - 1$. Then, at every time step $t$, for every table unit $\langle Table, x\rangle$, we identify the block units reachable from it with a program similar to (A.2) and define their global locations recursively with respect to height and width. *Base case:* Take any block $b$ on the table and any unit space $v$ of that block. Suppose that the block unit $\langle b, v\rangle$ is located $x$ units from the leftmostside of the table. Then the global location of $\langle b, v\rangle$ is $(0, x)$. This is expressed in the first rule of (5). *Inductive step:* Suppose that the global location of $\langle b', v'\rangle$ is $(h, x)$. If (i) holds then the global location of $\langle b, v\rangle$ is $(h + 1, x)$. This is expressed in the second rule of (5). If (ii) holds, then the global location of $\langle b, v\rangle$ is $(h, x + 1)$. This is expressed in the first rule of (6), respecting the size of the block. If (iii) holds, then the global location of $\langle b, v\rangle$ is $(h, x - 1)$. This is expressed in the second rule of (6), respecting the size of the block.

Then this definition of reachability of block units from table/box units based on the immediate-connectedness is correct thanks to Lemma 2 applied to $\Pi''$. For every answer set $X$ for $\Pi'' \cup (5) \cup (6)$, the following holds:

- For every time $t$, for every unit space $v'$ of a block $b'$ globally located at a distance $x$ from the leftmostside of the table at time $t$, $\langle b', v'\rangle \in \{\langle b, v\rangle : globalPos(b, v, h, x, t) \in X\}$ iff the unit $\langle b', v'\rangle$ is reachable from the unit $\langle Table, x\rangle$ with a tower of height $h$ based on the immediate-connectedness relation.
- For every time $t$, for every unit space $v'$ of a block $b'$ globally located at a height $h$ from the table at time $t$, $\langle b', v'\rangle \in \{\langle b, v\rangle : globalPos(b', v, h, x, t) \in X\}$ iff the unit $\langle b', v'\rangle$ is reachable from the unit $\langle b', v\rangle$ with the immediate-connectedness relation. □

### A.6. Proof of Proposition 2

Let us recall that the empty spaces above the table are defined in rules (7) as the global positions $(h, x)$ not occupied by any blocks:

$$empty(h, x, t) \leftarrow \#count\{b : box(b), globalPos(b, v, h, x, t)\} = 0 \quad (h > 0).$$

**Proposition 2.** *Let $\Pi'$ be the disjunctive program shown in Figs. 7–12, except for rules (7). Then, for every time step $t$ and for every block $b$ supported by the table, rules (7), when added to $\Pi'$, correctly describe the global positions $(h, x)$ of empty spaces at time $t$.*

**Proof.** Let $\Pi'$ be the disjunctive program shown in Figs. 7–12, except for rules (7). First, observe that, thanks to Theorem 1 of [28], each disjunctive rule like

$$pick(a, b, t) \vee \neg pick(a, b, t) \leftarrow$$

describing the occurrences of an action can be replaced a pair of nondisjunctive rules as follows

$$pick(a, b, t) \leftarrow not\ \neg pick(a, b, t)$$
$$\neg pick(a, b, t) \leftarrow not\ pick(a, b, t).$$

Let $\Pi''$ denote this nondisjunctive program.

Note that $\Pi''$ does not contain atoms of the form $empty(h, x, t)$ in the heads of its rules. Then we can replace atoms of the form $empty(h, x, t)$ in the bodies of rules in $\Pi''$ by expressions of the form $\#count\{b : box(b), globalPos(b, v, h, x, t)\} = 0$, where $h > 0$. Let $\Pi'''$ denote this nondisjunctive program.

Then, by Proposition 3 of [33], adding the definition (7) to $\Pi'''$ ensures that the answer sets for $\Pi'''$ are conservatively extended to define empty spaces. □

### A.7. Proof of Proposition 3

Let us recall the definition (8) of supportedness,

$$supported(b, l, t) \leftarrow onAux(b, l, t).$$
$$supported(b, l, t) \leftarrow onAux(b, l', t), supported(l', l, t) \quad (b \neq l'),$$

and the acyclicity constraint (9).

$$\leftarrow supported(b, b, t).$$

**Proposition 3.** *Let $\Pi$ be the disjunctive program shown in Figs. 7–12. Then, rules (8), when added to $\Pi$, correctly describe the supportedness of blocks by other blocks or by the table. Furthermore, adding constraints (9) to $\Pi \cup (8)$ guarantees the desired feature D1 (i.e., no circular configuration of blocks occurs in construction at any time step $0..T - 1$).*

The proof follows from (i) Theorem 1 of [28] about an equivalent representation of disjunctive rules about occurrences/nonoccurrences of actions by nondisjunctive rules, (ii) the observation that $\Pi$ does not contain atoms of the form $supported(b, l, t)$ in the heads of its rules, and (iii) Propositions 4 and 5 of [29] about the correctness and well-foundedness of the transitive closure of a relation defined recursively in ASP.

**Proof.** The proof consists of two parts: correctness of the supportedness definition, and guarantee of no circular configurations.

*Part 1.* The correctness of supportedness definition follows from Proposition 4 of [29] about the correctness of the transitive closure of a relation defined recursively in ASP.

Let $\Pi$ be the disjunctive program that describes the construction domain, as shown in Figs. 7–12. First, observe that, thanks to Theorem 1 of [28], each disjunctive rule like

$$pick(a, b, t) \vee \neg pick(a, b, t) \leftarrow$$

describing the occurrences of an action can be replaced a pair of nondisjunctive rules as follows

$$pick(a, b, t) \leftarrow not\ \neg pick(a, b, t)$$
$$\neg pick(a, b, t) \leftarrow not\ pick(a, b, t).$$

Let $\Pi'$ denote this nondisjunctive program.

Let *Def* be the supportedness definition (8). Then, by Proposition 4 of [29], for every answer set $X$ for $i' \cup Def$, the following holds:

At each time step $t$, $\{\langle b, l\rangle : supported(b, l, t) \in X\}$ is the transitive closure of $\{\langle x, y\rangle : onAux(b, l, t) \in X\}$.

Therefore, the supportedness definition (8) is correct.

*Part 2.* By Proposition 5 of [29], adding the constraint

$$supported(b, b, t)$$

to $\Pi' \cup Def$ ensures the well-foundedness of the set $\{\langle x, y\rangle : onAux(x, y) \in X\}$, hence, spurious circular configurations of blocks. □

### A.8. Proof of Proposition 4

Let us recall the definition (10) that identify blocks being supported by the table, being supported by a block being held by the robot, or being held by the robot:

$$supportedAux(b, 1, t) \leftarrow supported(b, Table, t),$$
$$\quad \#count\{a : arm(a), holding(a, b, t)\} = 0$$
$$supportedAux(b, 2, t) \leftarrow supported(b, b', t), holding(a, b', t) \qquad (b \neq b')$$
$$supportedAux(b, 3, t) \leftarrow holding(a, b, t)$$

and add the constraints (11) to ensure that one of these three cases should hold for each block $b$:

$$\leftarrow \#count\{x : supportedAux(b, x, t)\} < 1$$
$$\leftarrow \#count\{x : supportedAux(b, x, t)\} > 1$$

**Proposition 4.** *Let $\Pi$ be the disjunctive program shown in Figs. 7–12. Then, rules (10), when added to $\Pi \cup (8) \cup (9)$, correctly describe the supportedness of blocks and subassemblies by other blocks, by the table, or by the robot. Furthermore, adding constraints (11) to $\Pi \cup (8) \cup (9) \cup (10)$ guarantees the desired feature D2 (i.e., no flying blocks, and no blocks supported by both the table and the robot at any time step $0..T-1$).*

**Proof.** Observe that the program $\Pi \cup (8) \cup (9)$ does not contain atoms of the form $supportedAux(b, t)$ in the heads of its rules.

Then, by Proposition 3 of [33], rules (10), when added to $\Pi \cup (8) \cup (9)$, correctly describe the supportedness of blocks and subassemblies by other blocks: answer sets for $\Pi \cup (8) \cup (9) \cup (10)$ are conservative extensions of $\Pi \cup (8) \cup (9)$.

Furthermore, by Proposition 2 of [33], adding constraints (11) to $\Pi \cup (8) \cup (9) \cup (10)$ eliminates answer sets for $\Pi \cup (8) \cup (9) \cup (10)$ that do not satisfy the desired feature D2. Therefore, adding constraints (11) guarantees D2. □

### A.9. Proof of Proposition 5

**Proposition 5.** *Let $\Pi'$ be the program obtained from the disjunctive program $\Pi$ shown in Figs. 7–12, by adding the program shown in Fig. 16 about supportedness of blocks/subassemblies. Then adding constraints (12) ∪ (13) ∪ (14) to $\Pi'$ further ensures the desired features D3–D5 about occurrences of actions.*

**Proof.** Propositions 3 and 4 ensure constructions that satisfy supportedness constraints. By Proposition 2 of [33], adding constraints (12) ∪ (13) ∪ (14) to $\Pi'$ eliminates its answer sets that do not satisfy any of D3–D5. Therefore, adding these constraints guarantees D3–D5. □

### A.10. Proof of Proposition 6

**Proposition 6.** *Let $\Pi'$ be the program obtained from the disjunctive program $\Pi$ shown in Figs. 7–12, and by adding the program shown in Fig. 16 about supportedness of blocks/subassemblies. Suppose that the stability checking algorithm $\Gamma$ is correct (i.e., the construction is stable iff $\Gamma$ returns True). Then adding rules (15) to $\Pi'$ ensures that, at every time step $t$, every configuration of blocks assembled on a flat surface (e.g., table) or being carried by a gripper is stable.*

**Proof.** Propositions 3 and 4 ensure constructions that satisfy supportedness constraints and thus prevent spurious structures, like flying blocks/subassemblies, circular configurations of blocks, or blocks being supported by both the table and the robot. Then the proof of Proposition 6 follows from an application of Proposition 2 of [33] about the elimination of spurious models by adding constraints: adding rules (15) to $\Pi'$ eliminates its answer sets where the blocks assembled on a flat surface are not stable or where the blocks being carried are not stable. □

### A.11. Proof of Proposition 7

**Proposition 7.** *Suppose that the stability checking algorithm $\Gamma$ is correct (i.e., the construction is stable iff $\Gamma$ returns True). Let $\Pi'$ be the program obtained from the disjunctive program $\Pi$ shown in Figs. 7–12, by adding the program shown in Fig. 16 about supportedness of blocks/subassemblies, and by adding the constraints (15) about stability of the construction. Then, adding rules (16) ∪ (17) ∪ (18) to $\Pi'$ ensures a stable symmetric bridge (i.e., a construction that satisfies the condition D6) at time step $T$.*

**Proof.** Propositions 3–6 ensure constructions that satisfy supportedness and stability constraints. By Theorem 1 of [28], the program $\Pi' \cup (16)$ can be transformed into an equivalent nondisjunctive program $\Pi''$. By Proposition 3 of [33], adding the definition (16) to $\Pi''$ ensures that the answer sets for $\Pi'$ are conservatively extended to define symmetric supportedness relation. Proposition 4 of [29] ensures that the transitive closure of symmetric supportedness, i.e., definition *connectedness*, is correct. Then, further adding (17) extends the answer sets for $\Pi'' \cup (16)$ by connectedness relation. By Proposition 2 of [33], the answer sets that violate D6 at time step $T$ are eliminated. Therefore, adding rules (16) ∪ (17) ∪ (18) to $\Pi''$ ensures D6, and thus a stable symmetric bridge, at time step $T$. □

### A.12. Proof of Proposition 8

**Proposition 8.** *Suppose that the stability checking algorithm $\Gamma$ is correct (i.e., the construction is stable iff $\Gamma$ returns True). Let $\Pi'$ be the ASP program obtained from $\Pi$ by adding the supportedness and stability constraints (and, in case of bridge construction, also the connectedness constraints), and the ramification rules as described above. Then every robot construction plan that satisfies these desired properties and whose makespan is at most $T-1$ is characterized by an answer set for $\Pi'$.*

**Proof.** Due to the representation methodology of the program, candidate robot construction plans are generated by the part of the program that does not include any constraints. Supported by Proposition 2 of [33], each constraint then eliminates some of these candidates that violate desired features of a construction and its plan. No constraint in $\Pi'$ eliminates a valid robot construction plan. □

## Appendix B. Benchmark instances for construction problems and their solutions

In this appendix, we present a set of benchmark instances associated with challenging construction problems. Some of these benchmarks have been introduced by Fahlman [36]. We introduce an extended set of construction instances as changing benchmarks and present solutions to these problems computed using our hybrid planning approach.

For some problems, a stable final state and a plan cannot be computed by DLVHEX within the time threshold of 5000 seconds, as mentioned in Section 6. In such cases, we also provide a partial plan (e.g., first few actions) as part of input to further guide the search so that a stable final state and a plan are computed.

As discussed in Section 4.5, unless specified otherwise, the ASP modeling of the construction problem allows true concurrency. We can explicitly specify noconcurrency constraints, based on the capabilities of manipulators or desired conditions

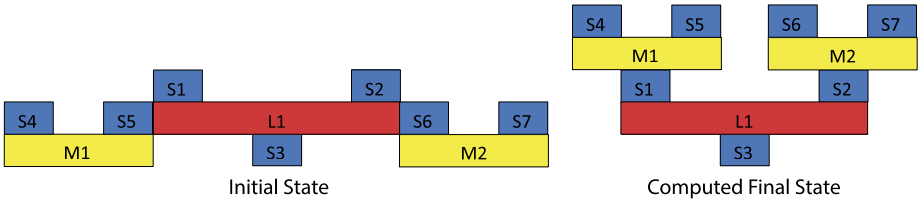



**Fig. B.27.** Scenario 1 – Incorporation of sub-assemblies into the final design.

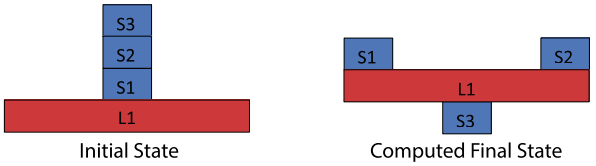



**Fig. B.28.** Scenario 2 [36] [Fig. 1.4] – Pre-assembly of movable a stable sub-structure.

of the construction process. For instance, considering the difficulty of synchronization of actions, it may be desired that a robotic manipulator does not replace a block with another block at the same time. In our experiments, we take into account such constraints about occurrences of actions. Without noconcurrency constraints, the plans usually get shorter and thus their computation times decrease further; this allows computation of solutions for scenarios within time threshold.

DLVHEX is a sophisticated system with many options that can be fine tuned for better performance. As mentioned in Section 6, in our experiments, we have tried different settings of the solver's configuration (i.e., `handy` or `jumpy`). Further fine tuning of DLVHEX might also allow computation of solutions for scenarios within time threshold.

In Scenarios 1–8, the weights of the uniform blocks are assumed to be as follows: $W_{small} = 1$ unit, $W_{medium} = 3$ units, and $W_{large} = 5$ units.

### *B.1. Sub-assembly manipulation*

Sub-assembly manipulation comprises of two or more blocks being manipulated together. In Fig. B.27, a sub-assembly consisting of the blocks $M1$, $S4$ and $S5$ is being manipulated, as the robot picks the block $M1$ and places it on top of block $S1$. As part of hybrid planning, it is challenging to decide for sub-assembly construction, and to ensure the stability of the structures. Note that it is also challenging to represent effects of manipulating a sub-assembly, due to ramifications.

*Scenario 1* (Fig. B.27)

The construction problem in Fig. B.27 involves incorporation of sub-assemblies into the final design.

The initial state of all the blocks is specified by the following facts:

$$init(M1, 1, Table, 1).init(S4, 1, M1, 1).init(S5, 1, M1, 3).init(S3, 1, Table, 6).$$
$$init(L1, 3, S3, 1).init(S1, 1, L1, 1).init(S2, 1, L1, 5).init(M2, 1, Table, 9).$$
$$init(S6, 1, M2, 1).init(S7, 1, M2, 3, 1).$$

Goal conditions for a final configuration are specified by the set of following facts:

$$goal(S3, Table).goal(L1, S3).goal(S1, L1).goal(S2, L1).goal(M1, S1).$$
$$goal(M2, S2).goal(S4, M1).goal(S5, M1).goal(S6, M2).goal(S7, M2).$$

For such an instance, a plan for a bimanual robot, reads as follows:

$0 : pick(Left, M1), pick(Right, M2).$
$1 : placeOn(Left, M1, 2, S1, 1), placeOn(Right, M2, 2, S2, 1).$

*Scenario 2* (Fig. B.28)

The construction problem in Fig. B.28 requires first the pre-assembly of movable a stable sub-structure on the table.

The initial state of all the blocks is specified by the following facts:

$$init(L1, 1, Table, 1).init(S1, 1, L1, 3).init(S2, 1, S1, 1).init(S3, 1, S2, 1).$$

Goal conditions for a final configuration are specified by the set of following facts:

$$goal(S3, Table).goal(L1, S3).goal(S1, L1).goal(S2, L1).$$

For such a problem instance, our planner generates the following plan:

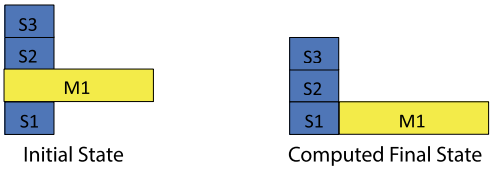



**Fig. B.29.** Scenario 3 [36][Fig. 1.9] – Disassembly of pre-assembled parts.

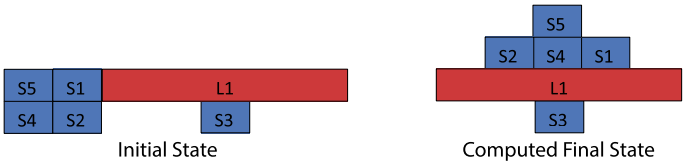



**Fig. B.30.** Scenario 4 – The use of permanent counter weights to balance out the structure.

0 : $pick(Left, S3)$.
1 : $placeOn(Left, S3, 1, Table, 6), pick(Right, S2)$.
2 : $pick(Left, S1), placeOn(Right, S2, 1, L1, 1)$.
3 : $placeOn(Left, S1, 1, L1, 5)$.
4 : $pick(Left, L1)$.
5 : $placeOn(Left, L1, 3, S3, 1)$.

Note that special attention needs to be paid as to where blocks are placed on $L1$ to ensure stability.

### *B.2. Disassembly*

Construction problems not only involve building new structures by stacking blocks or sub-assemblies, but also may require a disassembly of pre-assembled parts to reach the goal.

*Scenario 3* (Fig. B.29)

The construction problem in Fig. B.29 cannot be solved by moving one block at a time as in the Blocks World, since the stability of the overall structure needs to be preserved while executing the plan. It is required to first move the block $M1$ together with the blocks above it.

Initial state of all the boxes is specified by the following facts:

$$init(S1, 1, Table, 1).init(M1, 1, S1, 1).init(S2, 1, M1, 1).init(S3, 1, S2, 1).$$

Goal conditions for a final configuration are specified by the set of following facts:

$$goal(M1, Table).goal(S1, Table).goal(S3, S1).goal(S2, S3).$$

Generated plan is as follows:

0 : $pick(Left, M1)$.
1 : $placeOn(Left, M1, 1, Table, 2)$.
2 : $pick(Right, S3)$.
3 : $placeOn(Right, S3, 1, S1, 1), pick(Left, S2)$.
4 : $placeOn(Left, S2, 1, S3, 1)$.

### *B.3. Counter weights*

Counter weights may be required to temporarily or permanently balance a structure, so that it remains stable during and at the end of the construction. Therefore, deciding for the use of counter weights as part of a hybrid plan is challenging.

*Scenario 4* (Fig. B.30)

The construction problem in Fig. B.30 requires use of permanent counter weights to balance out the structure.

The initial configuration is expressed by the following set of facts:

$$init(S4, 1, Table, 1).init(S2, 1, Table, 2).init(S3, 1, Table, 5).$$
$$init(S5, 1, S4, 1).init(S1, 1, S2, 1).init(L1, 3, S3, 1).$$

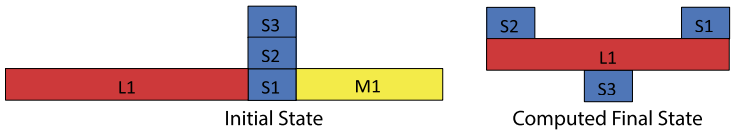



**Fig. B.31.** Scenario 5 – The use of a temporary counter weight.

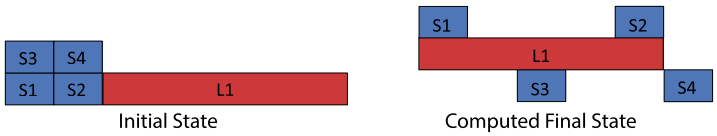



**Fig. B.32.** Scenario 6 – Scaffolding.

Goal conditions for a final configuration are specified by the set of following facts:

$$goal(S3, Table).goal(L1, S3).goal(S2, L1).$$
$$goal(S4, L1).goal(S1, L1).goal(S5, S4).$$

Generated plan is as follows:

0 : $pick(Left, S4)$.
1 : $placeOn(Left, S4, 1, L1, 3), pick(Right, S1)$.
2 : $pick(Left, S2), placeOn(Right, S1, 1, L1, 4)$.
3 : $placeOn(Left, S2, 1, L1, 2)$.

It is interesting to observe that the block $S4$ (and the block $S5$ above it) is moved onto $L1$ as a counter weight, so that the blocks $S2$ and $S1$ can be moved onto $L1$ appropriately.

*Scenario 5* (Fig. B.31)

The construction problem in Fig. B.31 requires use of a temporary counter weight $M1$ to balance the structure.

The initial configuration is expressed by the following set of facts:

$$init(L1, 1, Table, 1).init(S1, 1, Table, 6).init(M1, 1, Table, 7).$$
$$init(S2, 1, S1, 1).init(S3, 1, S2, 1).$$

Goal conditions for a final configuration are specified by the set of following facts:

$$goal(S3, Table).goal(L1, S3).goal(S2, L1).$$
$$goal(S1, L1).goal(M1, Table).$$

The generated plan is:

0 : $pick(Left, S3), pick(Right, L1)$.
1 : $placeOn(Left, S3, 1, Table, 3)$.
2 : $pick(Left, M1), placeOn(Right, L1, 3, S3, 1)$.
3 : $placeOn(Left, M1, 1, L1, 2), pick(Right, S2)$.
4 : $pick(Left, S1), placeOn(Right, S2, 1, L1, 1)$.
5 : $placeOn(Left, S1, 1, L1, 5), pick(Right, M1)$.
6 : $placeOn(Right, M1, 1, Table, 6)$.

### *B.4. Temporary scaffolding*

Similar to counter weights, scaffolding may be needed to temporarily support a construction. In scaffolding, instead of supporting the structure from above by introducing a heavy object, the structure is supported from below. Deciding for temporary use of scaffolds as a part of a hybrid plan is challenging.

*Scenario 6* (Fig. B.32)

The construction problem in Fig. B.32 necessitates scaffolding, since there does not exist a heavy box or multiple boxes that can be used as a counter weight in this scenario.

The initial configuration is expressed by the following set of facts:

$$init(S1, 1, Table, 1).init(S2, 1, Table, 2).init(L1, 1, Table, 3).$$
$$init(S3, 1, S1, 1).init(S4, 1, S2, 1).$$

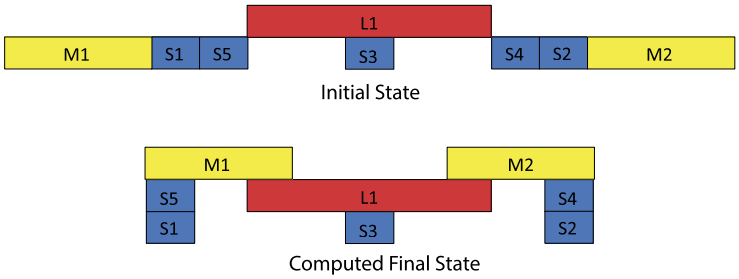



**Fig. B.33.** Scenario 7 – True concurrency.

Goal conditions for a final configuration are specified by the set of following facts:

$goal(S3, Table).goal(L1, S3).goal(S2, L1).$
$goal(S1, L1).goal(S4, Table).$

Generated plan is:

0 : $pick(Left, L1), pick(Right, S3).$
1 : $placeOn(Right, S3, 1, Table, 5).$
2 : $pick(Right, S4).$
3 : $placeOn(Right, S4, 1, Table, 7).$
4 : $placeOn(Left, L1, 3, S3, 1), pick(Right, S1).$
5 : $placeOn(Right, S1, 1, L1, 1), pick(Left, S2).$
6 : $placeOn(Left, S2, 1, L1, 5).$
7 : $pick(Left, S3).$
8 : $placeOn(Left, S3, 1, Table, 4).$

### *B.5. True concurrency of actions*

For some construction problems, multiple robots are required to perform truly concurrent (non-serializable) actions to achieve a task. In particular, it may happen that if some blocks or sub-assemblies are not placed concurrently, the structure becomes unstable. Allowing true concurrency in planning is a challenging problem, from the perspectives of both representation and reasoning.

*Scenario 7* (Fig. B.33)
Consider the construction problem in Fig. B.33, where $M1$ and $M2$ are placed by two concurrent actions. This problem shows the importance of true concurrency.

The initial configuration is expressed by the following set of facts:

$init(M1, 1, Table, 1).init(S1, 1, Table, 4).init(S5, 1, Table, 5).init(S3, 1, Table, 8).$
$init(S4, 1, Table, 11).init(S2, 1, Table, 12).init(M2, 1, Table, 13).init(L1, 3, S3, 1).$

Goal conditions for a final configuration are specified by the set of following facts:

$goal(S1, Table).goal(S2, Table).goal(S3, Table).$
$goal(S5, S1).goal(L1, S3).goal(S4, S2).$
$goal(M1, S5).goal(M1, L1).goal(M2, L1).goal(M2, S4).$

Generated plan is:

0 : $pick(Left, S5), pick(Right, S4).$
1 : $placeOn(Left, S5, 1, S1, 1), placeOn(Right, S4, 1, S2, 1).$
2 : $pick(Left, M1), pick(Right, M2).$
3 : $placeOn(Left, M1, 1, S1, 1), placeOn(Right, M2, 3, S4, 1).$

When the box $M2$ is placed on $S4$, as a direct effect the third unit space of $M2$ is on the first unit space of $S4$; as its indirect effects, the first unit space of $M2$ is on the fifth unit space of $L1$. Same effect happens when $M1$ is placed on $S5$.

### *B.6. Ramifications of actions*

Representing and reasoning about indirect effects (ramifications) of actions are challenging for planning.

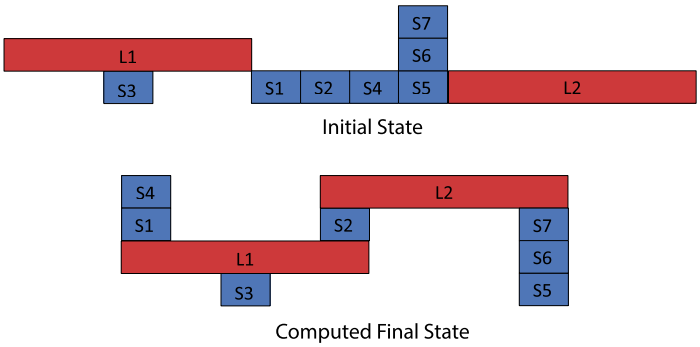



**Fig. B.34.** Scenario 8 – Challenging ramifications.

*Scenario 8* (Fig. B.34)

The construction problem in Fig. B.34 shows the importance of properly handling ramifications. Here, when $L2$ is placed on $S2$ at the final step, as an indirect effect of this action, $L2$ becomes on top of the block $S7$ as well.

The initial configuration is expressed by the following set of facts:

$$init(S3, 1, Table, 3).init(S1, 1, Table, 6).init(S2, 1, Table, 7).$$
$$init(S4, 1, Table, 8).init(S5, 1, Table, 9).init(L2, 1, Table, 10)$$
$$init(L1, 3, S3, 1).init(S6, 1, S5, 1).init(S7, 1, S6, 1).$$

Goal conditions for a final configuration are specified by the set of following facts:

$$goal(S3, Table).goal(L1, S3).goal(S1, L1).$$
$$goal(S2, L1).goal(S4, S1).goal(S5, Table).$$
$$goal(S6, S5).goal(S7, S6).goal(L2, S2).goal(L2, S7).$$

The generated plan is:

$0 : pick(Left, S1), pick(Right, S2).$
$1 : placeOn(Left, S1, 1, L1, 1), placeOn(Right, S2, 1, L1, 5).$
$2 : pick(Left, S4), pick(Right, L2).$
$3 : placeOn(Left, S4, 1, S1, 1), placeOn(Right, L2, 1, S2, 1, 1).$

### B.7. Overhang scenarios

There may be multiple goal configurations of blocks depending on the desired conditions about the final structure, and all of them may not be stable. In such cases, finding a stable goal configuration is a challenge. For instance, the determination of the maximum overhang achievable by a stack of identical blocks [47,76] with some blocks used as counterweights [75,77] is a 150 years old mathematical puzzle with recent solutions.

In the maximum overhang scenarios, in addition to the determination of a stable and optimal final configuration of blocks, we also address the planning aspect of the construction problem required to attain the goal configuration.

In these sample scenarios listed below, the yellow blocks are used for construction purposes, while the green and purple blocks are used as counter weights. The yellow and purple blocks occupy 3 unit spaces and green blocks occupy 1 unit space. The purple and green blocks are denoted by the letters 'C' and 'S', respectively. In these examples, the purple blocks are assumed to be 10 times heavier than the yellow blocks, while the green blocks are assumed to be 3 times heavier than the yellow blocks. In particular, in Scenarios 9–11, the weights of the uniform blocks and counterweights are assumed to be as follows: $W_{medium} = 3$ units, $W_{small_{counter}} = 30$ units, and $W_{medium_{counter}} = 30$ units.

*Scenario 9* (Fig. B.35)

Consider the construction problem in Fig. B.35 that involves 8 blocks with 5 of them as counter weights. The goal here is to achieve a maximum overhang of 3 units. This requires careful balancing of weights to stabilize the structure.

The initial configuration of blocks is given as:

$$init(S1, 1, Table, 1).init(S2, 1, S1, 1).init(S3, 1, Table, 2).init(S4, 1, S3, 1).$$
$$init(S5, 1, S4, 1).init(M1, 1, Table, 3).init(M2, 1, M1, 1).init(M3, 1, M2, 2).$$

A stable final state and a plan are computed as follows:

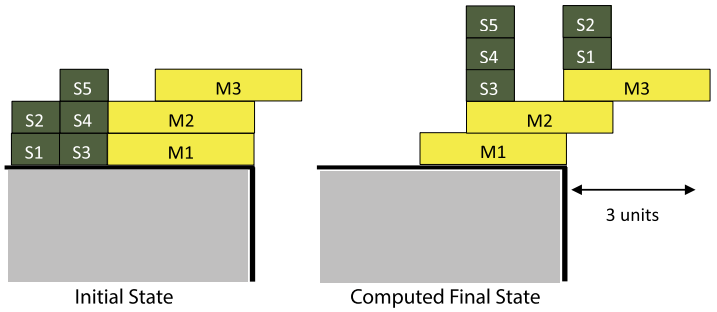



**Fig. B.35.** Scenario 9 – Stable construction of a 3 unit overhang. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

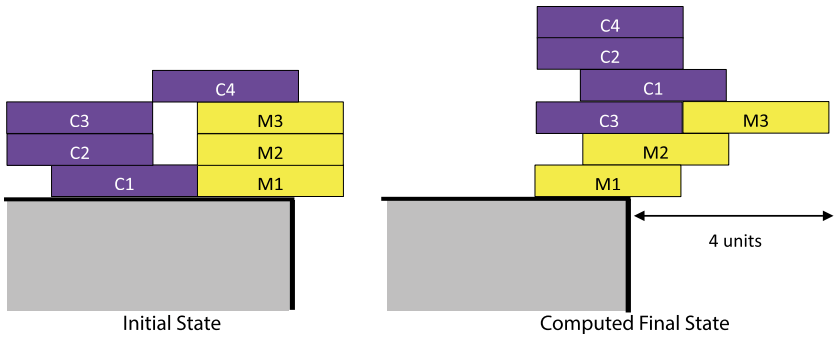



**Fig. B.36.** Scenario 10 – Stable construction of a 4 unit overhang.

0 : $pick(Left, S3)$, $pick(Right, S1)$.
1 : $placeOn(Left, S3, 1, M2, 1)$, $placeOn(Right, S1, 1, M3, 1)$.
2 : $pick(Left, M2)$.
3 : $placeOn(Left, M2, 1, M1, 2)$.
4 : $pick(Right, M3)$.
5 : $placeOn(Right, M3, 1, M2, 3)$.

*Scenario 10* (Fig. B.36)

The construction problem in Fig. B.36 involves 7 blocks with 4 of them as counter weights. Here the goal is to achieve an overhang of 4 units.

The initial configuration of blocks is given as:

$$init(C1, 1, Table, 3).init(C2, 2, C1, 1).init(C3, 1, C2, 1).init(M1, 1, Table, 6).$$
$$init(M2, 1, M1, 1).init(M3, 1, M2, 1).init(C4, 2, M3, 1).$$

A stable final state and a plan are computed as follows:

0 : $pick(Left, C3)$, $pick(Right, M3)$.
1 : $placeOn(Left, C3, 2, M2, 1)$, $placeOn(Right, M3, 1, C2, 1)$.
2 : $pick(Left, M2)$, $pick(Right, C4)$.
3 : $placeOn(Left, M2, 1, M1, 2)$.
4 : $pick(Left, M3)$.
5 : $placeOn(Right, C4, 1, C2, 1)$, $placeOn(Left, M3, 1, C3, 1)$.
6 : $pick(Right, C1)$, $pick(Left, M2)$.
7 : $placeOn(Right, M2, 1, Table, 2)$.
8 : $pick(Left, M3)$.
9 : $placeOn(Left, M3, 1, M2, 3)$.
10 : $placeOn(Right, C1, 1, C3, 2)$.
11 : $pick(Left, M2)$.
12 : $placeOn(Left, M2, 1, M1, 2)$.

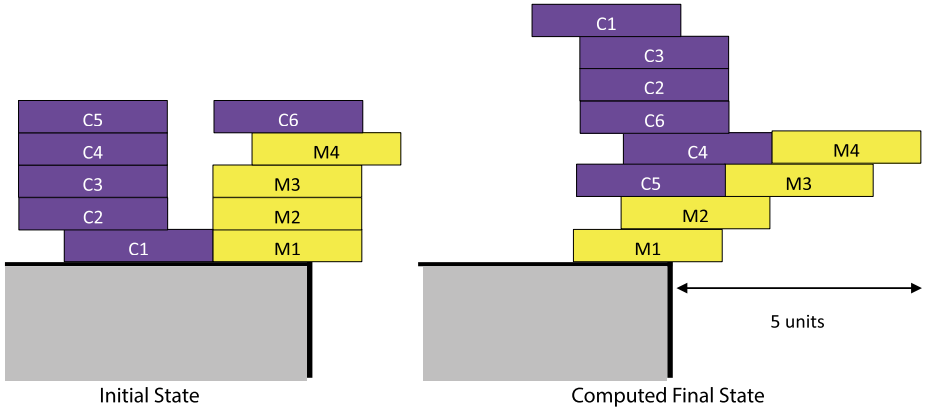



**Fig. B.37.** Scenario 11 – Stable construction of a 5 unit overhang.

*Scenario 11* (Fig. B.37)

In the construction problem in Fig. B.37, an overhang of 5 units is aimed.

The initial configuration of blocks is as follows:

$$init(C1, 1, Table, 9).init(C2, 2, C1, 1).init(C3, 1, C2, 1).$$
$$init(C4, 1, C3, 1).init(C5, 1, C4, 1).init(M1, 1, Table, 12).$$
$$init(M2, 1, M1, 1).init(M3, 1, M2, 1).init(M4, 1, M3, 2).init(C6, 2, M4, 1).$$

A stable final state and a plan are computed as follows:

$0 : pick(Left, M2), pick(Right, C2).$
$1 : placeOn(Left, M2, 1, Table, 6), placeOn(Right, C2, 1, M1, 1).$
$2 : pick(Left, M3), pick(Right, C5).$
$3 : placeOn(Left, M3, 1, M2, 3).$
$4 : pick(Left, M4), placeOn(Right, C5, 2, M2, 1).$
$5 : placeOn(Left, M4, 1, Table, 2), pick(Right, C4).$
$6 : pick(Left, C2), placeOn(Right, C4, 1, C5, 2).$
$7 : pick(Right, M2).$
$8 : placeOn(Left, C2, 1, C6, 1), placeOn(Right, M2, 1, M1, 2).$
$9 : pick(Left, C6).$
$10 : placeOn(Left, C6, 2, C4, 1), pick(Right, M4).$
$11 : pick(Left, C1).$
$12 : placeOn(Left, C1, 2, C3, 1), placeOn(Right, M4, 1, M3, 2).$

### *B.8. Symmetric bridge scenarios*

In bridge construction scenarios, the goal is to join both sides (e.g., of a river) by a stable construction of blocks. These scenarios require connectedness, as well as the stability. The resulting bridges may be required to be symmetric or asymmetric, depending on the requirements of the task.

In Scenarios 12–15, the weights of the uniform blocks and counterweights are assumed to be as follows: $W_{medium} = 3$ units, $W_{small_{counter}} = 30$ units, and $W_{medium_{counter}} = 30$ units.

*Scenario 12* (Fig. B.38)

The bridge construction problem in Fig. B.38 involves a distance of 9 units between the two sides. There total 15 blocks with 6 of them being counter weights.

The initial configuration is expressed by following set of facts:

$$init(M1, 1, LeftSide, 1).init(S6, 1, M1, 1).init(M4, 1, M1, 2).init(S4, 1, S6, 1).$$
$$init(M6, 1, M4, 1).init(S2, 1, S4, 1).init(M7, 1, M6, 1).init(M2, 2, RightSide, 1).$$
$$init(M3, 2, M2, 1).init(S5, 1, M2, 3).init(M5, 1, M3, 1).init(S3, 1, S5, 1).$$
$$init(M8, 1, M5, 1).init(S1, 1, S3, 1).init(M9, 2, M8, 1).$$

A stable final state and a plan are computed as follows:

$0 : pick(Left, M6), pick(Right, S4).$

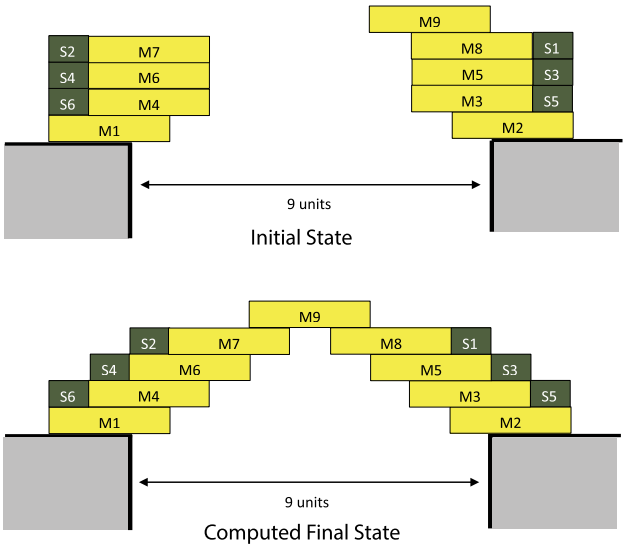



**Fig. B.38.** Scenario 12 – Stable construction of a 9 unit bridge.

1 : $placeOn(Left, M6, 1, M4, 2).placeOn(Right, S4, 1, M4, 1).$
2 : $pick(Left, M5), pick(Right, S3).$
3 : $placeOn(Left, M5, 3, M3, 2).placeOn(Right, S3, 1, M3, 3).$
4 : $pick(Left, M7), pick(Right, S2).$
5 : $placeOn(Left, M7, 1, M6, 2).placeOn(Right, S2, 1, M6, 1).$
6 : $pick(Left, M8), pick(Right, S1).$
7 : $placeOn(Left, M8, 3, M5, 2).placeOn(Right, S1, 1, M5, 3).$
8 : $pick(Left, M9).$
9 : $placeOn(Left, M9, 1, M7, 3).$

*Scenario 13* (Fig. B.39)

The bridge construction problem in Fig. B.39 has 7 units distance between the sides. There are 13 blocks with 6 of them being counter weights.

The initial configuration is given by the following set of facts:

$$init(M1, 1, LeftSide, 1).init(M2, 1, M1, 1).init(M3, 1, M2, 1).init(M4, 1, RightSide, 1).$$
$$init(M5, 1, M4, 1).init(M6, 1, M5, 1).init(M7, 2, M6, 1).init(C6, 1, RightSide, 5).$$
$$init(C5, 2, C6, 1).init(C4, 1, C5, 2).init(C3, 2, C4, 1).init(C2, 1, C3, 2).init(C1, 2, C2, 1).$$

A stable final state and a plan are computed as follows:

0 : $pick(Left, M2), pick(Right, M5).$
1 : $placeOn(Left, M2, 1, M1, 2), placeOn(Right, M5, 2, M4, 1)$
2 : $pick(Left, M3), pick(Right, C3).$
3 : $placeOn(Left, M3, 1, M2, 3), placeOn(Right, C3, 2, M2, 1).$
4 : $pick(Left, M6), pick(Right, C6).$
5 : $placeOn(Left, M6, 3, M5, 1), placeOn(Right, C6, 2, M5, 3).$
6 : $pick(Left, M7).$
7 : $placeOn(Left, M7, 3, M6, 1).$

### *B.9. Asymmetric bridge scenarios*

Asymmetric bridge scenarios are similar to the bridge construction scenarios in terms of their goal conditions. The only difference in these scenarios is that both sides have different heights.

*Scenario 14* (Fig. B.40)

This bridge construction scenario in Fig. B.40 involves 9 blocks with 4 being counter weights. The right side is 2 units higher than left side and the sides are 4 units apart.

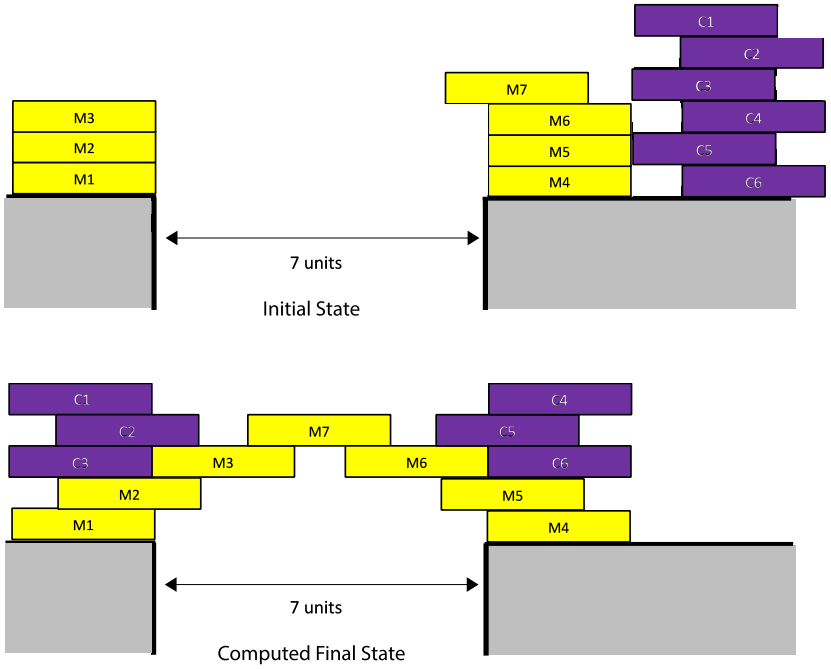



**Fig. B.39.** Scenario 13 – Stable construction of a 7 unit bridge.

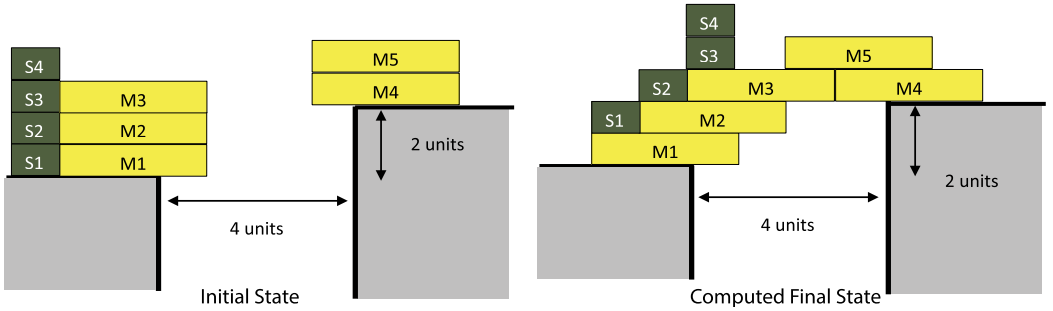



**Fig. B.40.** Scenario 14 – Stable construction of a 4 unit asymmetric bridge.

The initial configuration is given by the following set of facts:

$$init(S1, 1, LeftSide, 1).init(S2, 1, S1, 1).init(M2, 1, M1, 1).init(S3, 1, S2, 1).$$
$$init(M3, 1, M2, 1).init(S4, 1, S3, 1).init(M4, 2, RightSide, 1).init(M5, 1, M4, 1).$$

A stable final state and a plan are computed as follows:

$0 : pick(Left, S1), pick(Right, M2).$
$1 : placeOn(Left, S1, 1, M1, 1).placeOn(Right, M2, 1, M1, 2).$
$2 : pick(Left, S2), pick(Right, M3).$
$3 : placeOn(Left, S2, 1, M2, 1).placeOn(Right, M3, 1, M2, 2).$
$4 : pick(Left, S3), pick(Right, M5).$
$5 : placeOn(Left, S3, 1, M3, 1).placeOn(Right, M5, 1, M3, 3).$

*Scenario 15* (Fig. B.41)

The bridge construction scenario in Fig. B.41 has 9 blocks with 4 being counter weights. The right side is 4 units higher than left side and the sides are 5 units apart.

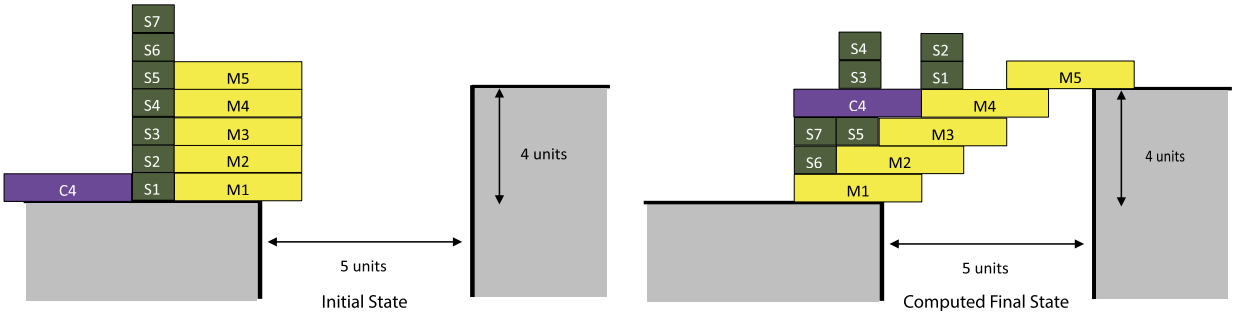



**Fig. B.41.** Scenario 15 – Stable construction of a 5 unit asymmetric bridge.

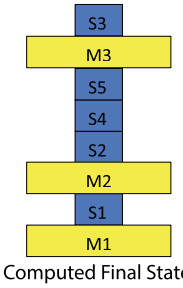



**Fig. B.42.** Scenario 16 – Maximizing the height of block S3 in a stack of 8 blocks.

The initial configuration is given by the following set of facts:

$$init(C4, 1, LeftSide, 1).init(S1, 1, LeftSide, 4).init(S2, 1, S1, 1).$$
$$init(S3, 1, S2, 1).init(S4, 1, S3, 1).init(S5, 1, S4, 1).$$
$$init(S6, 1, S5, 1).init(S7, 1, S6, 1).init(M1, 1, LeftSide, 5).$$
$$init(M2, 1, M1, 1).init(M3, 1, M2, 1).init(M4, 1, M3, 1).init(M5, 1, M4, 1).$$

A stable final state and a plan are computed as follows:

$0 : pick(Left, S6), pick(Right, M2).$
$1 : placeOn(Left, S6, 1, M1, 1), placeOn(Right, M2, 1, M1, 2).$
$2 : pick(Left, S5), pick(Right, M3).$
$3 : placeOn(Left, S5, 1, M2, 1), placeOn(Right, M3, 1, M2, 2).$
$4 : pick(Right, S3).$
$5 : placeOn(Right, S3, 1, C4, 2).$
$6 : pick(Left, C4), pick(Right, M4).$
$7 : placeOn(Left, C4, 1, S7, 1), placeOn(Right, M4, 1, M3, 2).$
$8 : pick(Left, M5), pick(Right, S1).$
$9 : placeOn(Left, M5, 1, M4, 3), placeOn(Right, S1, 1, M4, 1).$

### *B.10. Tower stacking*

In tower staking benchmarks, the goal is to maximize or minimize the height of a particular block in a stack of a given number of blocks.

In Scenarios 16–21, the weights of the uniform blocks are assumed to be as follows: $W_{small} = 3$ units, $W_{medium} = 3$ units, and $W_{large} = 5$ units.

*Scenario 16* (Fig. B.42)

The construction problem in Fig. B.42 aims to maximize the height of block $S3$ in a stack. There are 8 blocks in the stack. Initially, all the blocks are on the table.

A stable final state and a plan are computed as follows:

$0 : pick(Left, S1), pick(Right, M2).$
$1 : placeOn(Left, S1, 1, M1, 2).$

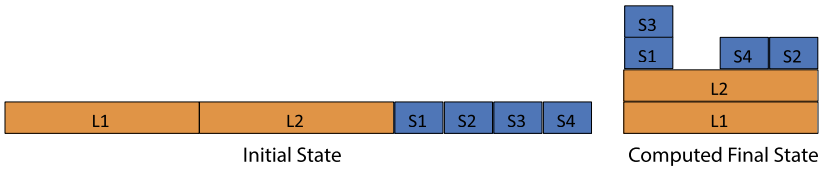



**Fig. B.43.** Scenario 17 – Maximizing the height of block L2 in a stack of 6 blocks.

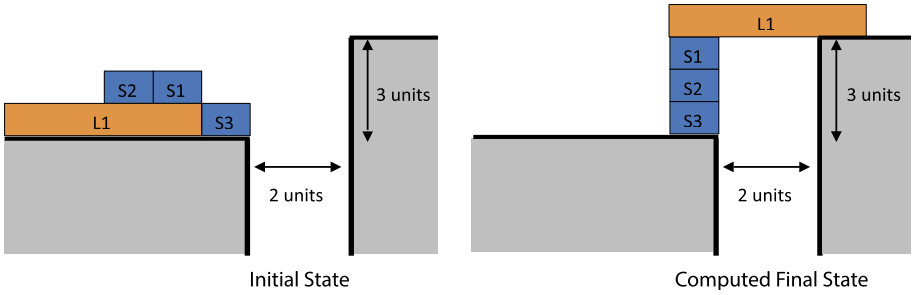



**Fig. B.44.** Scenario 18 – Maximizing the height of block L1 in a stack of 4 blocks.

$2 : pick(Left, S2), placeOn(Right, M2, 2, S1, 1).$
$3 : placeOn(Left, S2, 1, M2, 2), pick(Right, S4).$
$4 : placeOn(Right, S4, 1, S2, 1, 1), pick(Left, S5).$
$5 : placeOn(Left, S5, 1, S4, 1), pick(Right, M3).$
$6 : placeOn(Right, M3, 2, S5, 1), pick(Left, S3).$
$7 : placeOn(Left, S3, 1, M3, 2).$

*Scenario 17* (Fig. B.43)

The construction problem in Fig. B.43 aims to maximize the height of block $L2$ in a stack. There are 6 blocks in the stack. Initial state and the computed goal state is given in Fig. B.43.

A stable final state and a plan are computed as follows:

$0 : pick(Left, S1), pick(Right, S2).$
$1 : placeOn(Left, S1, 1, L1, 1), placeOn(Right, S2, 1, L1, 4)$
$2 : pick(Left, S3), pick(Right, S4).$
$3 : placeOn(Left, S3, 1, S1, 1), placeOn(Right, S4, 1, S2, 1).$
$4 : pick(Left, L2).$
$5 : placeOn(Left, L2, 1, S3, 1).$

*Scenario 18* (Fig. B.44)

The construction problem in Fig. B.44 aims to maximize the height of block $L1$ in a stack. There are 4 blocks in the stack. Here, $L1$ cannot be directly placed on the right side, as there is not enough space, that is, the single unit space on the left side would cause block $L1$ to fall without proper support. Note that in this problem specification, there is no requirement that the two sides should be connected. Initial state and the computed goal state are given in Fig. B.44.

A stable final state and a plan are computed as follows:

$0 : pick(Left, S2), pick(Right, S1).$
$1 : placeOn(Left, S2, 1, S3, 1).$
$2 : pick(Left, L1), placeOn(Right, S1, 1, S2, 1).$
$3 : placeOn(Left, L1, 1, S1, 1).$

*Scenario 19* (Fig. B.45)

The construction problem in Fig. B.45 aims to maximize the height of block $L1$ in a stack. There are 4 blocks in the stack. Here, there exist two unit spaces available on the right side, as opposed to only one unit space in the previous benchmarks. Once again, there is no requirement that the two sides should be connected. Initial state and the computed goal state are given in Fig. B.45.

A stable final state and a plan are computed as follows:

$0 : pick(Left, S1), pick(Right, S2).$

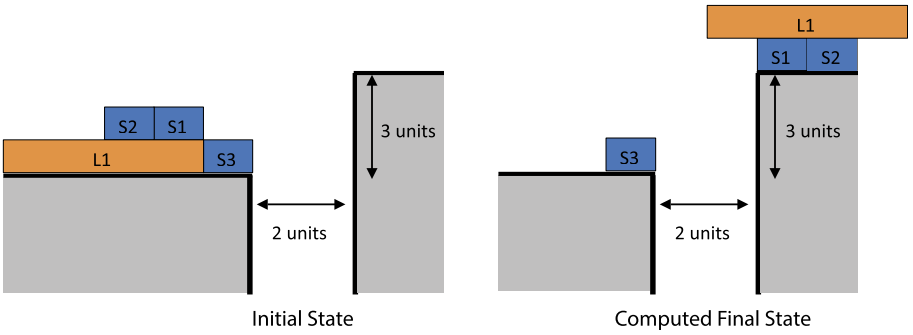



**Fig. B.45.** Scenario 19 – Maximizing the height of block L1 in a stack of 4 blocks.

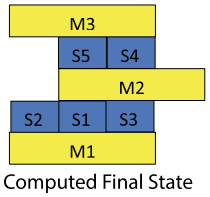



**Fig. B.46.** Scenario 20 – Minimizing the height of block S3 in a stack of 8 blocks.

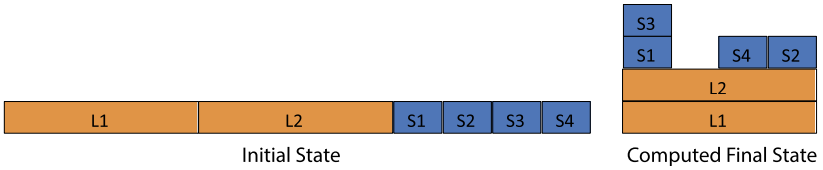



**Fig. B.47.** Scenario 21 – Minimizing the height of block L2 in a stack of 7 blocks.

$1 : placeOn(Left, S1, 1, RightSide, 1), placeOn(Right, S2, 1, RightSide, 2).$
$2 : pick(Left, L1).$
$3 : placeOn(Left, L1, 2, S1, 1).$

*Scenario 20* (Fig. B.46)

This construction problem in Fig. B.46 aims to minimize the height of block $S3$ in a stack. There are total 8 blocks in the stack. Please note that, in the problem specification its is required that $S3$ is part of the stack, that is, it cannot be placed directly on the ground/table. Initially, all the boxes are on the table.

A stable final state and a plan are computed as follows:

$0 : pick(Left, S1), pick(Right, S2).$
$1 : placeOn(Left, S1, 1, M1, 2), placeOn(Right, S2, 1, M1, 1).$
$2 : pick(Left, S3), pick(Right, M2).$
$3 : placeOn(Left, S3, 1, M1, 3).$
$4 : placeOn(Right, M2, 1, S1, 1).$
$5 : pick(Left, S5), pick(Right, S4).$
$6 : placeOn(Left, S5, 1, M2, 1), placeOn(Right, S4, 1, M2, 2).$
$7 : pick(Left, M3).$
$8 : placeOn(Left, M3, 2, S5, 1).$

*Scenario 21* (Fig. B.47)

This construction problem in Fig. B.47 aims to minimize the height of block $L2$ in a stack. There are total 7 blocks in the stack. In the problem specification its is required that $L2$ is part of the stack. Initially, all the boxes are on the table.

A stable final state and a plan are computed as follows:

$0 : pick(Left, S1), pick(Right, S2).$
$1 : placeOn(Left, S1, 1, L2, 1), placeOn(Right, S2, 1, L2, 4).$
$2 : pick(Left, S3), pick(Right, S4).$

3 : $placeOn(Left, S3, 1, S1, 1)$, $placeOn(Right, S4, 1, L2, 3)$.
4 : $pick(Left, L2)$.
5 : $placeOn(Left, L2, 1, L1, 1)$.

## References


[1] A. Akbari, Muhayyuddin, J. Rosell, Knowledge-oriented task and motion planning for multiple mobile robots, J. Exp. Theor. Artif. Intell. 31 (1) (2019) 137–162.
[2] J. Barry, K. Hsiao, L.P. Kaelbling, T. Lozano-Pérez, Manipulation with multiple action types, in: Proc. of ISER, 2013, pp. 531–545.
[3] M. Beetz, D. Jain, L. Mosenlechner, M. Tenorth, L. Kunze, N. Blodow, D. Pangercic, Cognition-enabled autonomous robot control for the realization of home chore task intelligence, Proc. IEEE 100 (8) (2012) 2454–2471.
[4] S. Bernardini, M. Fox, D. Long, C. Piacentini, Boosting search guidance in problems with semantic attachments, in: Proc. of ICAPS, 2017.
[5] L. Beyeler, J.-C. Bazin, E. Whiting, A graph-based approach for discovery of stable deconstruction sequences, in: Proc. of AAG, 2015, pp. 145–157.
[6] M. Blum, A. Griffith, B. Neumann, A stability test for configurations of blocks, Technical report, MIT Artificial Intelligence Laboratory, 1970.
[7] T. Bock, Construction automation and robotics, in: Robotics and Automation in Construction, InTech, 2008.
[8] N. Boneschanscher, H. van der Drift, S.J. Buckley, R.H. Taylor, Subassembly stability, in: Proc. of AAAI, vol. 88, 1988, pp. 780–785.
[9] G. Brewka, T. Eiter, M. Truszczynski, Answer set programming: an introduction to the special issue, AI Mag. 37 (3) (2016) 5–6.
[10] F. Buccafurri, N. Leone, P. Rullo, Enhancing disjunctive datalog by constraints, IEEE Trans. Knowl. Data Eng. 12 (5) (2000) 845–860.
[11] O. Caldiran, K. Haspalamutgil, A. Ok, C. Palaz, E. Erdem, V. Patoglu, Bridging the gap between high-level reasoning and low-level control, in: Proc. of LPNMR, 2009, pp. 342–354.
[12] F. Calimeri, W. Faber, M. Gebser, G. Ianni, R. Kaminski, T. Krennwallner, N. Leone, F. Ricca, T. Schaub, ASP-Core-2 input language format, https://www.mat.unical.it/aspcomp2013/files/ASP-CORE-2.03c.pdf, 2013.
[13] F. Calimeri, M. Fink, S. Germano, A. Humenberger, G. Ianni, C. Redl, D. Stepanova, A. Tucci, A. Wimmer, Angry-HEX: an artificial player for angry birds based on declarative knowledge bases, IEEE Trans. Comput. Intell. AI Games 8 (2) (2016) 128–139.
[14] S. Chitta, I.A. Sucan, S. Cousins, Moveit! [ROS topics], IEEE Robot. Autom. Mag. 19 (1) (2012) 18–19.
[15] G. Coruhlu, E. Erdem, V. Patoglu, Explainable robotic plan execution monitoring under partial observability, IEEE Trans. Robot. (2021) 1–21.
[16] A. Cosgun, T. Hermans, V. Emeli, M. Stilman, Push planning for object placement on cluttered table surfaces, in: Proc. of IEEE/RSJ IROS, 2011, pp. 4627–4632.
[17] N.T. Dantam, Z.K. Kingston, S. Chaudhuri, L.E. Kavraki, Incremental task and motion planning: a constraint-based approach, in: Proc. of RSS, 2016.
[18] N.T. Dantam, Z.K. Kingston, S. Chaudhuri, L.E. Kavraki, An incremental constraint-based framework for task and motion planning. I, J. Robot. Res. 37 (10) (2018).
[19] E.D. Demaine, M.L. Demaine, M. Hoffmann, J. O'Rourke, Pushing blocks is hard, Comput. Geom. 26 (1) (2003) 21–36.
[20] M.R. Dogar, S.S. Srinivasa, A planning framework for non-prehensile manipulation under clutter and uncertainty, Auton. Robots 33 (3) (2012) 217–236.
[21] C. Dornhege, P. Eyerich, T. Keller, S. Trüg, M. Brenner, B. Nebel, Semantic attachments for domain-independent planning systems, in: Proc. of ICAPS, 2009.
[22] S. Edelkamp, J. Hoffmann, PDDL2.2: the language for the classical part of the 4th international planning competition, Technical Report 195, University of Freiburg, 2004.
[23] T. Eiter, M. Fink, G. Ianni, T. Krennwallner, C. Redl, P. Schüller, A model building framework for answer set programming with external computations, Theory Pract. Log. Program. 16 (4) (2016) 418–464.
[24] T. Eiter, M. Fink, T. Krennwallner, C. Redl, P. Schüller, Efficient HEX-program evaluation based on unfounded sets, J. Artif. Intell. Res. 49 (2014) 269–321.
[25] T. Eiter, G. Ianni, R. Schindlauer, H. Tompits, dlvhex: a system for integrating multiple semantics in an answer-set programming framework, in: Workshop on Logic Programming and Constraint Systems, 2006, pp. 206–210.
[26] E. Erdem, M. Gelfond, N. Leone, Applications of answer set programming, AI Mag. 37 (3) (2016) 53–68.
[27] E. Erdem, K. Haspalamutgil, C. Palaz, V. Patoglu, T. Uras, Combining high-level causal reasoning with low-level geometric reasoning and motion planning for robotic manipulation, in: Proc. of IEEE ICRA, 2011, pp. 4575–4581.
[28] E. Erdem, V. Lifschitz, Transformations of logic programs related to causality and planning, in: Proc. of LPNMR, 1999, pp. 107–116.
[29] E. Erdem, V. Lifschitz, Tight logic programs, Theory Pract. Log. Program. 3 (4–5) (2003) 499–518.
[30] E. Erdem, V. Patoglu, Applications of ASP in robotics, Künstl. Intell. 32 (2–3) (2018) 143–149.
[31] E. Erdem, V. Patoglu, P. Schüller, A systematic analysis of levels of integration between high-level task planning and low-level feasibility checks, AI Commun. 29 (2) (2016) 319–349.
[32] C. Erdogan, M. Stilman, Planning in constraint space: automated design of functional structures, in: Proc. of IEEE ICRA, 2013.
[33] S.T. Erdogan, V. Lifschitz, Definitions in answer set programming, in: Proc. of LPNMR, 2004, pp. 114–126.
[34] W. Faber, G. Pfeifer, N. Leone, Semantics and complexity of recursive aggregates in answer set programming, Artif. Intell. 175 (1) (2011) 278–298.
[35] R. Fagin, Monadic generalized spectra, Math. Log. Q. 21 (1) (1975) 89–96.
[36] S.E. Fahlman, A planning system for robot construction tasks, Artif. Intell. 5 (1) (1974) 1–49.
[37] L. Ferreira, C. Toledo, Generating levels for physics-based puzzle games with estimation of distribution algorithms, in: Proc. of ACM ACE, 2014, p. 25.
[38] F. Furrer, M. Wermelinger, H. Yoshida, F. Gramazio, M. Kohler, R. Siegwart, M. Hutter, Autonomous robotic stone stacking with online next best object target pose planning, in: Proc. of IEEE ICRA, 2017, pp. 2350–2356.
[39] A. Gaschler, R.P.A. Petrick, M. Giuliani, M. Rickert, A. Knoll, KVP: a knowledge of volumes approach to robot task planning, in: Proc. of IEEE/RSJ IROS, 2013, pp. 202–208.
[40] A. Gaschler, R.P.A. Petrick, O. Khatib, A. Knoll, KABouM: knowledge-level action and bounding geometry motion planner, J. Artif. Intell. Res. 61 (2018) 323–362.
[41] M. Gelfond, V. Lifschitz, Classical negation in logic programs and disjunctive databases, New Gener. Comput. 9 (1991) 365–385.
[42] M. Gelfond, V. Lifschitz, Action languages, Electron. Trans. Artif. Intell. 2 (1998) 193–210.
[43] A. Gerevini, P. Haslum, D. Long, A. Saetti, Y. Dimopoulos, Deterministic planning in the fifth international planning competition: PDDL3 and experimental evaluation of the planners, Artif. Intell. 173 (5) (2009) 619–668.
[44] E. Giunchiglia, J. Lee, V. Lifschitz, N. McCain, H. Turner, Nonmonotonic causal theories, Artif. Intell. 153 (1–2) (2004) 49–104.
[45] F. Gravot, S. Cambon, R. Alami, aSyMov: a planner that deals with intricate symbolic and geometric problems, in: Proc. of ISRR, 2003, pp. 100–110.
[46] N. Gupta, D.S. Nau, On the complexity of blocks-world planning, Artif. Intell. 56 (2–3) (1992) 223–254.
[47] J.F. Hall, Fun with stacking blocks, Am. J. Phys. 73 (12) (2005) 1107–1116.
[48] S.D. Han, N.M. Stiffler, K.E. Bekris, J. Yu, Autonomous design of functional structures, Adv. Robot. 29 (9) (2015) 625–638.
[49] S.D. Han, N.M. Stiffler, K.E. Bekris, J. Yu, Efficient, high-quality stack rearrangement, IEEE Robot. Autom. Lett. 3 (3) (2018) 1608–1615.
[50] P. Haslum, F. Ivankovic, M. Ramírez, D. Gordon, S. Thiébaux, V. Shivashankar, D.S. Nau, Extending classical planning with state constraints: heuristics and search for optimal planning, J. Artif. Intell. Res. 62 (2018) 373–431.

[51] K. Hauser, 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.
[52] G. Havur, G. Ozbilgin, E. Erdem, V. Patoglu, Geometric rearrangement of multiple movable objects on cluttered surfaces: a hybrid reasoning approach, in: Proc. of IEEE ICRA, 2014, pp. 445–452.
[53] A. Hertle, C. Dornhege, T. Keller, B. Nebel, Planning with semantic attachments: an object-oriented view, in: Proc. of ECAI, 2012, pp. 402–407.
[54] Z. Jia, A.C. Gallagher, A. Saxena, T. Chen, 3d reasoning from blocks to stability, IEEE Trans. Pattern Anal. Mach. Intell. 37 (5) (2015) 905–918.
[55] L.P. Kaelbling, T. Lozano-Pérez, Integrated task and motion planning in belief space. I, J. Robot. Res. 32 (9–10) (2013) 1194–1227.
[56] N.P. Koenig, A. Howard, Design and use paradigms for Gazebo, an open-source multi-robot simulator, in: Proc. of IEEE/RSJ IROS, 2004, pp. 2149–2154.
[57] A. Krontiris, K.E. Bekris, Dealing with difficult instances of object rearrangement, in: Proc. of RSS, 2015.
[58] A. Krontiris, K.E. Bekris, Efficiently solving general rearrangement tasks: a fast extension primitive for an incremental sampling-based planner, in: Proc. of IEEE ICRA, 2016, pp. 3924–3931.
[59] A. Krontiris, R. Shome, A. Dobson, A. Kimmel, K. Bekris, Rearranging similar objects with a manipulator using pebble graphs, in: Proc. of IEEE-RAS Humanoids, 2014, pp. 1081–1087.
[60] F. Lagriffoul, D. Dimitrov, J. Bidot, A. Saffiotti, L. Karlsson, Efficiently combining task and motion planning using geometric constraints. I, J. Robot. Res. 33 (14) (2014) 1726–1747.
[61] S.M. LaValle, Planning Algorithms, Cambridge University Press, 2006.
[62] S. Lee, Y.G. Shin, Assembly planning based on geometric reasoning, Comput. Graph. 14 (2) (1990) 237–250.
[63] R.K. Livesley, Limit analysis of structures formed from rigid blocks, Int. J. Numer. Methods Eng. 12 (12) (1978) 1853–1871.
[64] R.K. Livesley, A computational model for the limit analysis of three-dimensional masonry structures, Meccanica 27 (3) (1992) 161–172.
[65] M.C. Magnaguagno, F. Meneguzzi, Semantic attachments for HTN planning, in: Proc. of AAAI, 2020, pp. 9933–9940.
[66] R. Mattikalli, D. Baraff, P. Khosla, Finding all stable orientations of assemblies with friction, IEEE Trans. Robot. Autom. 12 (2) (1996) 290–301.
[67] J. McCarthy, Elaboration tolerance, in: Working Papers of the Fourth International Symposium on Logical Formalizations of Commonsense Reasoning, 1998.
[68] D. McDermott, G. Sussman, The Conniver Reference Manual, MIT AI Memo, vol. 259, 1972.
[69] R. Mojtahedzadeh, A. Bouguerra, E. Schaffernicht, A.J. Lilienthal, Support relation analysis and decision making for safe robotic manipulation tasks, Robot. Auton. Syst. 71 (2015) 99–117.
[70] N. Napp, R. Nagpal, Distributed amorphous ramp construction in unstructured environments, Robotica 32 (2) (2014) 279–290.
[71] A. Nouman, V. Patoglu, E. Erdem, Hybrid conditional planning for robotic applications, Int. J. Robot. Res. 40 (2–3) (2021) 594–623.
[72] K. Okada, A. Haneda, H. Nakai, M. Inaba, H. Inoue, Environment manipulation planner for humanoid robots using task graph that generates action sequence, in: Proc. of IEEE/RSJ IROS, vol. 2, 2004, pp. 1174–1179.
[73] R.S. Palmer, Computational complexity of motion and stability of polygons, Technical report, Cornell University, 1989.
[74] J.-S. Pang, J. Trinkle, Stability characterizations of rigid body contact problems with Coulomb friction, J. Appl. Math. Mech. (Z. Angew. Math. Mech.) 80 (10) (2000) 643–663.
[75] M. Paterson, Y. Peres, M. Thorup, P. Winkler, U. Zwick, Maximum overhang, Am. Math. Mon. 116 (9) (2009) 763–787.
[76] M. Paterson, U. Zwick, Overhang, in: ACM-SIAM Symposium on Discrete Algorithm, 2006, pp. 231–240.
[77] M. Paterson, U. Zwick, Overhang, Am. Math. Mon. 116 (1) (2009) 19–44.
[78] E. Plaku, Planning in discrete and continuous spaces: from LTL tasks to robot motions, in: Joint Proc. of TAROS and FIRA RoboWorld, 2012, pp. 331–342.
[79] M. Rizwan, V. Patoglu, E. Erdem, Human robot collaborative assembly planning: an answer set programming approach, Theory Pract. Log. Program. 20 (6) (2020) 1006–1020.
[80] F. Röhrdanz, H. Mosemann, F. Wahl, Generating and evaluating stable assembly sequences, Adv. Robot. 11 (2) (1996) 97–126.
[81] Z.G. Saribatur, V. Patoglu, E. Erdem, Finding optimal feasible global plans for multiple teams of heterogeneous robots using hybrid reasoning: an application to cognitive factories, Auton. Robots 43 (2019) 213–238.
[82] J.M. Schimmels, M.A. Peshkin, Force-assembly with friction, IEEE Trans. Robot. Autom. 10 (4) (1994) 465–479.
[83] B. Schmult, Autonomous robotic disassembly in the blocks world, Int. J. Robot. Res. 11 (5) (1992) 437–459.
[84] P.D. Spanos, P.C. Roussis, N.P. Politis, Dynamic analysis of stacked rigid blocks, Soil Dyn. Earthq. Eng. 21 (7) (2001) 559–578.
[85] S. Srivastava, E. Fang, L. Riano, R. Chitnis, S.J. Russell, P. Abbeel, Combined task and motion planning through an extensible planner-independent interface layer, in: Proc. of IEEE ICRA, 2014, pp. 639–646.
[86] M. Stephenson, J. Renz, Procedural generation of complex stable structures for angry birds levels, in: Proc. IEEE CIG, 2016, pp. 1–8.
[87] M. Stilman, J. Kuffner, Planning among movable obstacles with artificial constraints, Int. J. Robot. Res. 27 (11–12) (2008) 1295–1307.
[88] M. Stilman, J.J. Kuffner, Navigation among movable obstacles: real-time reasoning in complex environments, Int. J. Humanoid Robot. 2 (04) (2005) 479–503.
[89] M. Stilman, J.-U. Schamburek, J. Kuffner, T. Asfour, Manipulation planning among movable obstacles, in: Proc. of IEEE ICRA, 2007, pp. 3327–3332.
[90] I.A. Sucan, M. Moll, L.E. Kavraki, The open motion planning library, IEEE Robot. Autom. Mag. 19 (4) (2012) 72–82.
[91] G.J. Sussman, D.V. McDermott, From planner to conniver: a genetic approach, in: Proc. of ACM Fall Joint Computer Conference, Part II, 1972, pp. 1171–1179.
[92] V. Thangavelu, Y. Liu, M. Saboia, N. Napp, Dry stacking for automated construction with irregular objects, in: Artif. Intell., 2018.
[93] S. Thiébaux, J. Hoffmann, B. Nebel, In defense of PDDL axioms, Artif. Intell. 168 (1–2) (2005) 38–69.
[94] A. Thomas, S. Amatya, F. Mastrogiovanni, M. Baglietto, Task-assisted motion planning in partially observable domains, CoRR, arXiv:1908.10227, 2019.
[95] M. Toussaint, Logic-geometric programming: an optimization-based approach to combined task and motion planning, in: Proc. of IJCAI, 2015, pp. 1930–1936.
[96] M. Toussaint, M. Lopes, Multi-bound tree search for logic-geometric programming in cooperative manipulation domains, in: Proc. of IEEE ICRA, 2017, pp. 4044–4051.
[97] P.A. Wałega, M. Zawidzki, T. Lechowski, Qualitative physics in angry birds, IEEE Trans. Comput. Intell. AI Games 8 (2) (2016) 152–165.
[98] W. Wan, K. Harada, K. Nagata, Assembly sequence planning for motion planning, Assem. Autom. 38 (2) (2018) 195–206.
[99] J. Wang, P. Rogers, L. Parker, D. Brooks, M. Stilman, Robot jenga: autonomous and strategic block extraction, in: Proc. of IEEE/RSJ IROS, 2009, pp. 5248–5253.
[100] R.W. Weyhrauch, Prolegomena to a theory of formal reasoning, Technical report, Stanford University, 1978.
[101] R.W. Weyhrauch, Prolegomena to a theory of mechanized formal reasoning, Artif. Intell. 13 (1–2) (1980) 133–170.
[102] E.J.W. Whiting, Design of structurally-sound masonry buildings using 3D static analysis, PhD thesis, Massachusetts Institute of Technology, 2012.
[103] G. Wilfong, Motion planning in the presence of movable obstacles, in: Proc. of SCG, 1988, pp. 279–288.
[104] R.H. Wilson, J.-C. Latombe, Geometric reasoning about mechanical assembly, Artif. Intell. 71 (2) (1994) 371–396.
[105] T. Winograd, Understanding natural language, Cogn. Psychol. 3 (1) (1972) 1–191.
[106] U. Zwick, Jenga, in: Proc. of ACM SIAM SODA, 2002, pp. 243–246.