2013 IEEE International Conference on Robotics and Automation (ICRA)
Karlsruhe, Germany, May 6-10, 2013

A Case Study on the Tower of Hanoi Challenge: Representation, Reasoning and Execution

Giray Havur, Kadir Haspalamutgil, Can Palaz, Esra Erdem, Volkan Patoglu

Abstract— **The Tower of Hanoi puzzle, has recently been established as a robotics challenge as a part of EU Robotics coordination action in 2011 and IEEE IROS Conference in 2012. It provides a good standardized test bed to evaluate integration of high-level reasoning capabilities of robots together with their manipulation and perception aspects. We address this challenge within a general planning and monitoring framework: we represent the puzzle in a logic-based formalism, integrate task planning and motion planning, solve this hybrid planning problem with a state-of-the-art automated reasoner (e.g., a SAT solver), execute the computed plans under feedback control while also monitoring for failures, and recover from failures as required. We show the applicability of this framework by implementing it using two robotic manipulators on a physical experimental setup.**

I. Introduction

The famous Tower of Hanoi problem [1], has been challenging computer scientists and computational mathematicians over a century. In particular, this puzzle is shown to be a provably hard problem, with exponential time complexity [2]. Furthermore, many variations of the problem have been introduced [3], where the initial and final configurations, number of pegs, meaning of the colors of disks, and the rules of the puzzle are modified. Excellent surveys for Tower of Hanoi and its variations are given in [4], [5], while an extensive bibliography is available in [6]. Work on this problem still goes on [7], studying properties of solution instances, as well as variants of the original problem. Each variation of the puzzle introduces new challenges necessitating new algorithms to be developed for its solution. Even though the classical Tower of Hanoi puzzle can be solved using a relatively simple recursive algorithm, no such algorithms are known for several variations of the puzzle. Achieving optimal solutions is even harder. For instance, the optimality of the Frame and Stewart algorithm [8], [9] that addresses the 4 peg version of the puzzle [10] has only recently been proven, more than 70 years after it has been proposed [11].

Tower of Hanoi provides challenges to robotics as well. As the robotic systems make their transition from repetitive tasks in highly-structured industrial settings to loosely defined tasks in unstructured human environments, the complexity of the tasks and the variability of environment place high demands on the robots' intelligence and autonomy. Consequently, a pressing need becomes apparent to furnish the robotic systems with high-level cognitive capabilities. The Tower of Hanoi puzzle, has recently been established as a robotics challenge as a part of EU Robotics coordination action in 2011 and IEEE IROS Conference in 2012, since it provides a good standardized test method to evaluate integration of high-level reasoning capabilities of robots together with their manipulation and perception aspects.

As a first challenge, solutions to variations of Tower of Hanoi puzzle cannot be hard-coded, since these problems are provably hard and with no known algorithms to solve them. Hence, the problem necessitates the robots to be furnished with generic planners/reasoners to attack the problem. Such an approach is preferable, since it is generalizable to more practical tasks instead of being limited to the puzzle at hand. In addition, the successful competition of the puzzle requires a tight integration between the the high level planning capabilities of the robot with its geometric reasoning, perception, low-level control aspects. In particular, manipulation planning, grasp planning (and even navigation planning in case of mobile manipulation) need to be properly integrated with the high level task plans so that the robots can compute continuous plans that can be executed by the robot controllers. Execution requires low-level feedback control and tight coupling with perception, since otherwise uncertainties will prohibit execution of the plan during physical implementations.

In this paper, we introduce a general framework that integrates task planning, motion planning and feedback control to find and execute optimal solutions to Tower of Hanoi puzzle and its variations, while monitoring execution of the plans. We demonstrate the applicability of this framework with two robotic manipulators solving variations of Tower of Hanoi on a physical experimental setup.

II. Proposed Framework

State-of-the-art automated reasoners can be used to find optimal solutions to variations of Tower of Hanoi problem. Among these, we propose to utilize two sorts of automated reasoners: SAT solvers (e.g., MANYSAT [12]) developed for the satisfiability problem (SAT), and ASP solvers (e.g., CLASP [13]) developed for reasoning in Answer Set Programming (ASP). According to SAT [14], the idea is to formalize (in general NP-hard) computational problems as a set of formulas in propositional logic so that the models of this set of formulas characterize the solutions of the given problem; and compute the models using SAT solvers.

This work is supported by Sabancı University IRP Grant IACF09-00643, and TUBITAK Grant 111E116.

G. Havur, K. Haspalamutgil, C. Palaz, E. Erdem, V. Patoglu are with Faculty of Engineering and Natural Sciences, Sabancı University, İstanbul, Turkey {havurgiray,kadirhas,canpalaz}@sabanciuniv.edu {esraerdem,vpatoglu}@sabanciuniv.edu

According to ASP [15], [16], the idea is similar; though the problems are represented in a different logical formalism where the formulas have a non-monotonic meaning and the models (called answer sets [17]) are computed using ASP solvers. Both SAT and ASP have been applied in various real-world applications. For instance, SAT has been used for software and hardware verification [18] and planning [19]; ASP has been applied to biomedical informatics [20], space shuttle control [21], workforce management [22], etc.

Although both SAT and ASP provide efficient solvers and their formalisms are general enough to represent various kinds of computational problems, their formalisms do not provide special structures for a systematic formalization of dynamic domains. To facilitate the use of such general knowledge representation formalisms and their solvers, some other form of logic-based formalisms specifically designed for representing dynamic domains in a structured way, called action description languages [23], have been introduced. Further, to be able to use SAT/ASP solvers for reasoning about actions and change, sound transformations from action description languages to SAT and ASP have been developed [24], [25], [26].

For task planning, we propose to use the action description language $\mathcal{C}+$ [26] and its reasoner CCALC [27]. CCALC transforms an action domain description in $\mathcal{C}+$, into the input language of a SAT solver; and then calls a SAT solver to find a task plan. With the expressive formalism of $\mathcal{C}+$ and the computational efficiency of SAT solvers, CCALC can handle many challenges: concurrency, the frame problem [28], the qualification problem [29], ramifications [30], numeric-valued fluents [31], [32], nondeterminism, etc. With CCALC, one can solve planning problems, which may involve temporal complex goals. Furthermore, while computing a (discrete) plan, some geometric constraints on continuous trajectories can be embedded in domain description, to ensure feasible (collision-free) plans.

A collision-free trajectory for the task plan (if one exists) is computed by a motion planning algorithm, for instance, using Rapidly exploring Random Trees (RRT) [33] or Probabilistic Road Maps (PRM) [34]. In our framework, we consider Bi-RRTs since they conduct a bidirectional probabilistic search starting from both initial and goal configurations, for a more efficient sampling of the narrow passages at the initial and goal configurations.

The overall architecture of our planning and monitoring framework that integrates automated reasoners with motion planners is illustrated in Figure 1.

We start with a description of an action domain in the action description language $\mathcal{C}+$. The idea is, based on this description, to plan the actions of two robots to achieve a common goal. For that, we use the reasoner CCALC with the SAT solver MANYSAT. Given an initial state and goal conditions, CCALC computes a discrete task plan to reach a goal state, and displays the complete history (including the state information). Given such a history, we calculate the continuous trajectories of the robots (including the positions and the orientations of the joints of the robots, as well as

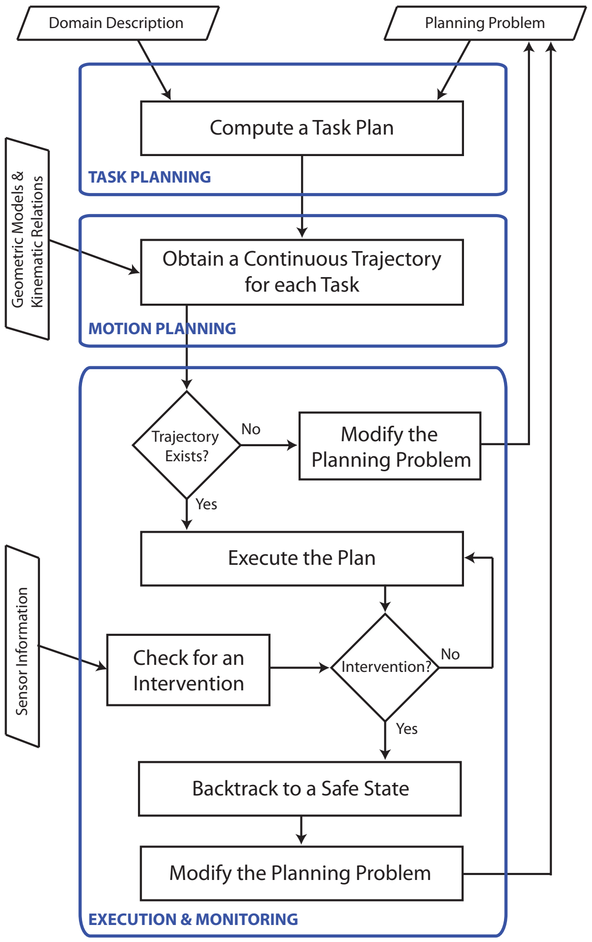
Domain Description
Planning Problem
Compute a Task Plan
TASK PLANNING
Geometric Models & Kinematic Relations
Obtain a Continuous Trajectory for each Task
MOTION PLANNING
Trajectory Exists?
No
Modify the Planning Problem
Yes
Execute the Plan
Sensor Information
Check for an Intervention
Intervention?
No
Yes
Backtrack to a Safe State
Modify the Planning Problem
EXECUTION & MONITORING

Fig. 1. The overall system architecture.

their velocities) using motion planning; these trajectories are obtained from a history automatically with a C++ program. Sometimes, the motion planner cannot find such a trajectory for the task plan because no collision-free trajectory exists for some task in the plan. Then a new plan is computed by prohibiting that task. Otherwise, we pass these trajectories to the robot controllers, for low-level feedback tracking of the trajectories. All these tasks are automatically performed on a PC using a Python program.

However, execution of the plan may fail due to some external intervention. Therefore, plan execution is monitored to detect human interventions in a dynamic environment, and then to recover from such plan failures. In particular, sensor information is requested at predefined intervals. When an intervention is detected, first the current state after intervention is identified and a safe state for positions of the disks are extracted from this state information. Second, a motion plan is calculated and applied to robots to move the disks to these positions. Next, a new plan is computed to reach a goal state. Finally, this new plan is executed and monitored.

III. PROBLEM DESCRIPTION

We consider a modified version of the Tower of Hanoi problem. Recall that the original puzzle consists of a number of different-sized rings and three identical pegs which the rings can slide onto; the pegs are attached to and perpendicular to a platform. Initially all the rings are stacked on a single peg in decreasing order so that the largest ring is the base of the tower and the smallest ring is the tip of the tower. The aim is to move all the rings to another peg with respect to the following two constraints: Only the topmost rings can be moved, and one at a time; and no ring can be stacked on top of a smaller one. The mathematical version of this puzzle has a solution for n rings and the solution can be found in $2^n - 1$ number of steps.

We extend this puzzle by adding the possibility of rotating rings about the peg axis with respect to the following constraints: Only the topmost rings can be rotated, in place or while being moved from one peg to another, and one at a time; and a ring can be rotated in multiples of 90° about the peg axis. Therefore, a configuration of rings include also their orientations. We suppose that there is a mark (e.g., an arrow) put on each ring, and that the arrow can be directed in four different directions with 90° intervals at least one direction being along the line connecting the pegs together; then there are four possible orientations of a ring. Also we assume that the initial and the goal configurations can be any legitimate configuration of rings. The goal is to move and/or rotate the rings in such a way as to transform the initial configuration to the goal configuration.

In addition to finding a solution to this problem, we want to execute the solution on a physical platform by using two pantograph robots with linear actuators attached to their end effectors. We assume that a ring can be moved or rotated by two robots only if they hold it at opposite ends. Therefore, it is not possible for two robots to move two different rings at the same time, or move a ring while rotating another ring. However, the robots can rotate a ring while moving it.

We solve this problem using both task planning and motion planing: we compute the shortest sequence of actions to transform the initial configuration to the goal configuration, using task planning; and then we compute a continuous trajectory from the task plan, using motion planning, so that the robots can follow the trajectory to achieve the goal.

It is important to note here that this version of Tower of Hanoi has challenges for both sorts of planning. A shortest task plan may involve concurrencies (e.g., a ring can be rotated while being moved from one peg to another), making task planning more challenging. Rotations of rings necessitate collision checks between the robot arms, making motion planning more challenging. Furthermore, a collision-free continuous trajectory may not be computed from the task plan, using motion planning (e.g., a task plan that involves a 180° rotation in one step sometimes leads to a collision of robot arms), making hybrid planning more challenging.

Suppose that we compute a solution (i.e., a trajectory to be followed by the two robots to obtain the goal configuration from the initial configuration of rings). Its execution by the robots gets challenging as well, if we allow external interventions: what if someone rotates/moves one of the topmost rings while the robots are carrying another ring? We handle such cases by monitoring execution of the plans.

IV. Task Planning

We describe the domain of the problem in the action description language $\mathcal{C}+$, and use the reasoning system CCalc with the SAT solver ManySAT to compute a task plan. Our choice of this particular formalism and the system to solve the problem is due to the following:

- The version of the Tower of Hanoi problem requires concurrency (e.g., rotating a ring while moving it from one peg to another), to find a shortest plan. $\mathcal{C}+$ allows concurrency of actions unless stated otherwise, and a shortest task plan involving concurrent actions can be computed by CCalc.
- If execution of the plan fails (due to external intervention), then the monitoring agent modifies the planning problem and asks the task planner to find a different plan to execute from that point on. In most cases the modified planning problems involve a set of possible initial states rather than one particular initial state. For instance, suppose that execution of the plan fails while a ring is being moved to another peg. Since the ring is not located at any of the pegs, it can be stacked on top of some peg (if possible) to obtain a legitimate configuration of rings; such a legal configuration specifies the initial state for the modified planning problem. However, there may be more than one possible configuration of rings, each configuration possibly leading to plans of different lengths; then an initial state can be described by a disjunction of these configurations. CCalc allows such a modification of the problem.
- If execution of the task plan is not possible (e.g., the robot arms collide with each other), the motion planner finds which action of the task plan is not executable and at which state. Then the monitoring agent asks the task planner to compute a different task plan by modifying the planning problem. This modification involves adding some constraints to the planning problem (as shown in the example in Section 5) to ensure that the task planner does not compute a plan that involves the same kind of failure (i.e., execution of the failed action at the detected state). CCalc allows such a modification of the planning problem.

In this problem, pegs are denoted by unique constants $p1, p2, p3$ and rings are identified by unique numbers $1, 2, 3, 4, ..., n$ such that Ring i is smaller than Ring j if $i < j$. We suppose that there is a mark (e.g., an arrow) on each ring, and that the arrow can be directed in four different directions with 90° intervals at least one direction being along the line connecting the pegs together; these four possible orientations are uniquely denoted by $1, 2, 3, 4$. Each ring D has a location and an orientation, specified by the functional fluents $loc(D)$ and $ort(D)$: a ring D is located either on top of another ring D' in which case $loc(D) = D'$, or on the base of a peg P in which case $loc(D) = P$.

We also consider two auxiliary functions: $base(D) = P$ expresses that ring D is somewhere on peg P, and $clear(L)$ expresses that ring/peg L has no other ring on top of it.

The former is declared as a functional fluent, and defined recursively in terms of $loc(D)$. The first causal law above defines the base of each peg as itself, and the second one states that the base of a disk is the base of its location. We define $clear(L)$ as a macro in terms of $loc(R)$.

We consider two kinds of actions: $move(P1, P2)$ (move the ring at the top of the stack on peg $P1$ to the top of peg $P2$) and $rotate(P, O)$ (rotate the ring at the top of the stack on peg P so that its orientation becomes O). Notice that we do not introduce an action for rotating a disk while moving it; this will be handled by concurrency.

We describe the legitimate states of the world, and the preconditions and the effects of actions by causal laws.

A. *Direct effects of actions*

The direct effects of moving the topmost ring D of the stack at peg $P1$, to the topmost location L of the stack at peg $P2$ can be expressed by the causal laws

```
move(P1,P2) causes loc(D)=L
  if base(D)=P1 & clear(D) &
     base(L)=P2 & clear(L).
```

Here `base(D)=P1 & clear(D)` expresses that ring D is on top of the stack at peg $P1$, whereas `base(L)=P2 & clear(L)` expresses that L is the topmost location of peg $P2$ (i.e., L is either the base of $P2$ or the ring at the top of the stack on $P2$).

Similarly, the effect of rotating the topmost ring D at peg P to the orientation O can be described by the causal laws

```
rotate(P,O) causes ort(D)=O
  if base(D)=P & clear(D).
```

B. *Preconditions of actions*

It is not possible to move a ring from an empty peg:

```
nonexecutable move(P,P1) if clear(P).
```

It is not possible to execute some actions concurrently. For instance, only one ring can be moved at each step:

```
nonexecutable move(P,P1) & move(P2,P3)
  if P@<P2.
```

Only one ring can be rotated at each step:

```
nonexecutable rotate(P,O) & rotate(P1,O1)
  if P@<P1.
```

A ring cannot be rotated while moving another ring:

```
nonexecutable move(P,P1) & rotate(P2,O)
  if P2\=P.
```

C. *Constraints*

The following constraint does not allow two rings to occupy the same location:

```
constraint loc(D)\=loc(D1) where D<D1.
```

This constraint prevents moving a ring to two different pegs at the same time.

The following constraint does not allow a ring to be on top of a smaller one:

```
constraint loc(D)\=D1 where D>=D1.
```

D. *Computing a Task Plan using* CCALC

Consider an instance of the problem described above, where there are 4 rings. Initially Rings 1 and 2 are at Peg 3, Ring 3 is at Peg 2, and Ring 4 is at Peg 1; the orientations of Rings 1–4 are 1, 2, 1, 2 respectively. We want to move all the rings to Peg 1, with orientation 2. This planning problem (Planning Problem 1) can be described in the language of CCALC by a "query" as follows:

```
:- query
maxstep :: 0..infinity;
0: % initial state
% position of the rings
loc(1)=2, loc(2)=p3, loc(3)=p2, loc(4)=p1,
% orientation of the rings
ort(1)=1, ort(2)=2, ort(3)=1, ort(4)=2;
maxstep: % goal
% position of the rings
loc(1)=2, loc(2)=3, loc(3)=4, loc(4)=p1,
% orientation of the rings
ort(1)=2, ort(2)=2, ort(3)=2, ort(4)=2.
```

Given the domain description and the planning problem above, CCALC computes the following plan (Plan 1) of length 4:

```
0: rotate(p2,2) move(p2,p1)
1: move(p3,p2)
2: move(p3,p1)
3: rotate(p2,2) move(p2,p1)
```

Note that the first action of the plan is concurrent: the robots move the topmost ring from Peg 2 to Peg 1, and meanwhile they rotate it 90° changing its orientation from 1 to 2.

V. Motion Planning

Given an initial state and goal conditions, CCALC computes not only a high-level discrete task plan to reach a goal state from the initial state, but also its complete history (including the state information at each step). Given such a history, a motion planner is employed to calculate continuous trajectories for the robots (including the positions and the orientations of the joints of the robots, as well as their velocities) for each step of the task plan, so that the plan can be executed by the robots by following these trajectories.

In the physical setup of our example, the clearance between the rings and the pegs is tight (about 0.5mm). Such tight clearances are challenging for motion planners, since they map to narrow passages in the configuration space of the robots. In our example, at each step of the task plan, a ring is moved from one peg (narrow passage in the configuration space) to another, while possibly changing its orientation. To efficiently solve the motion planning problem that has both the initial and the goal configurations in a narrow passage, we choose to use Rapidly exploring Random Trees [33]. RRTs are suitable for our problem since they conduct a *bidirectional* probabilistic search starting from both initial and goal configurations. Using this approach, efficient sampling of the narrow passage is ensured as the algorithm does not include a point from the free region of the configuration space until a trajectory is found to reach this free region. Once both the initial and goal configurations are connected to the free region of the configuration space, they are usually connected with a single intermediate point, resulting in a smooth trajectory with good runtime performance of the motion planner. For motion planning, we have also experimented with our problem using Probabilistic Road Maps [34], but the computational performance was not as good as RRTs.

Our implementation of Rapidly exploring Random Trees (RRT) [33] is similar to [35]. We start with two trees initially rooted at the start and goal configurations. We try to expand and connect these trees in succession. The algorithm iterates until a connection is made or a cutoff time has been reached. In each iteration we sample a point p from our configuration space and if p is a valid configuration, we perform the following for each tree i ($i = 1, 2$): We select the node p_i from tree i that is closest to p, and find the point q_i which is d distance away from p_i in the direction of p, where d is a parameter for our program that stands for branch length. If the direct path connecting q_i and p_i is collision free, we add q_i to tree i by connecting it with p_i. In each iteration, we also try to connect the two trees with the sampled point p, we check if both the paths p_i from p are collision free, if so we connect the two trees at point p by connecting p with both p_1 and p_2 and return the trajectory connecting the start and goal configurations. This is a greedy approach with little regard to optimality but we observe that the solutions obtained this way in our case are not far from optimal. The motion planning algorithm is implemented in C++.

Figure 2(a) depicts the top view of the trajectories calculated and traced for the first task `rotate(p2,2)` `move(p2,p1)` of the plan described in the section on Task Planning, while Figure 2(b) presents several snapshots taken during their execution. In particular, the task is to move the blue ring from the second peg to the first peg, while concurrently rotating it by a 90° counterclockwise. In Figure 2(a) dashed blue and red lines depict the trajectories calculated by the motion planner for each robot, while the solid blue and red lines represent the trajectories traced by these robots during the execution of the plan. Numbers are used to denote order of the some intermediate configurations returned by the motion planner. Note that even though the task planner returns the shortest task plan in terms of the number of discrete task steps, the trajectory calculated using the probabilistic motion planner is far from being optimal.

VI. Execution and Monitoring

Once the motion planner computes a continuous trajectory from the discrete task plan computed by CCALC, the trajectories is passed to the pantograph robots controllers by means of messages via Ethernet communication. While the robots execute these trajectories, there may be unexpected external interventions: for instance, some external agent may move one of the topmost rings from one peg to a different peg while the robots are moving/rotating another ring. In such cases, once the monitoring agent notices that there has been an intervention (via sensor information), it stops the execution of the plan. First, the monitoring agent tries to find all legitimate states considering all possible locations onto which the robots can put the ring they are holding. If such states exist, then the monitoring agent asks CCALC to find a shortest plan from one of these legitimate states to a goal state. If such a task plan exists, then it asks the motion planner to compute collision-free continuous trajectories and monitors the execution of the new plan. The whole execution monitoring process is based on Algorithm 1, which can be applied to problems that require simultaneous/integrated use of AI planning and motion planning.

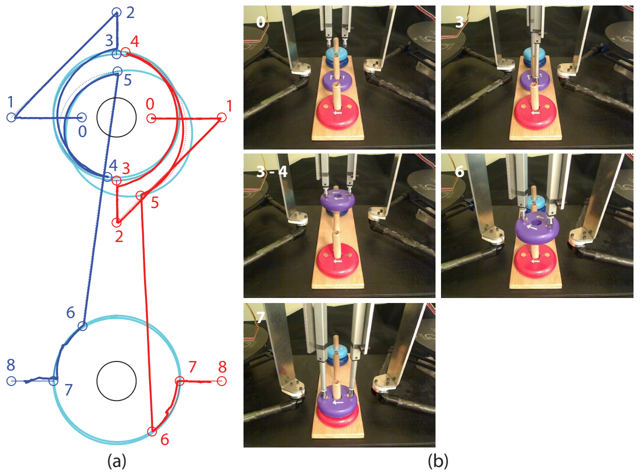

Fig. 2. (a) Top view of the trajectories calculated and traced for the task plan step `rotate(p2,2)` `move(p2,p1)`. Dashed blue and red lines depict the trajectories calculated by the motion planner for each robot, while the solid blue and red lines represent the trajectories traced by these robots during the execution of the plan. (b) Several snapshots taken during execution.

Algorithm 1 EXECUTE&MONITOR

Input: An action domain description $\mathcal{D}$, a planning problem $\mathcal{P}$ with the initial state(s) S and the goal G

```
$plan, P, H, \Pi \leftarrow$ TASK&MOTION_PLAN($\mathcal{D}$,$\mathcal{P}$);
if $\neg plan$ then
    Abort;    // no plan to execute
$completed := false$;
while $\neg completed$ do
    $A, R \leftarrow$ Extract from $P$ the next action to be executed, and the ring being moved/rotated;
    $\pi_A \leftarrow$ Extract from $\Pi$ the trajectory for $A$;
    $C \leftarrow$ Extract from $H$ the current state;
    $intervened \leftarrow$ Execute $A$ by following $\pi_A$ at the current state $C$, and meanwhile check for any interventions;
    if $intervened$ then
        $L_R \leftarrow$ Find the legitimate locations to put the ring $R$ onto;
        $I \leftarrow$ Find the legitimate states obtained by putting $R$ onto a location in $L_R$;
        if $I == \emptyset$ then
            Abort;    // no solution
        else
            $\mathcal{P} \leftarrow$ Modify the planning problem $\mathcal{P}$, where the initial state is a disjunction of the states in $I$ and the goal is $G$;
            EXECUTE&MONITOR($\mathcal{D}$,$\mathcal{P}$);
    else
        if $A$ is the last action of $P$ then
            $completed := true$;
return ;    // plan successfully executed
```

Algorithm 2 TASK&MOTION_PLAN

```
Input: Action domain description $\mathcal{D}$, and planning problem $\mathcal{P}$ with
  the initial state(s) $S$ and the goal $G$

  while true do
    plan, $P \leftarrow$ Compute a shortest task plan $P$ of length $n$ (within
    a history $H$) using CCALC with $\mathcal{D}$ and $\mathcal{P}$ (if there exists such
    a plan);
    if $\neg$plan then
      return false;
    $\Pi := \langle\rangle$;    // The complete trajectory
    motion_plan := true;
    while motion_plan do
      $A \leftarrow$ Extract from $P$ the next action without a trajectory;
      $C_I \leftarrow$ Extract from $H$ the current state;
      $C_G \leftarrow$ Extract from $H$ the next state;
        // Find a trajectory for $A$
      motion_plan, $\pi_A \leftarrow$ MOTION_PLAN($C_I$,$C_G$);
      if $\neg$motion_plan then
        $\mathcal{P} \leftarrow$ Modify the planning problem $\mathcal{P}$ ensuring that $A$ is
        not executed at state $C_I$;
      else
        $\Pi \leftarrow$ Append $\pi_A$ to $\Pi$
        if $A$ is the last action then
          return true, $P, H, \Pi$;
```

The controllers of the robots are implemented on a PC-based architecture, that comprises of a PCI I/O card and workstation, simultaneously running RTX real-time operating system and Windows XP SP2. For robust trajectory tracking of the end-effectors, feedback controllers are implemented in a real-time framework. The robots are planar parallel mechanisms (pantographs) with two degrees-of-freedom. To enable pick and drop actions, the end-effectors of the pantograph robots are equipped with linear servo motors with magnetic tips acting out of plane. These servo motors are controlled via their dedicated amplifiers driven by the analog outputs of the control card.

Consider for instance the planning problem described in the section on Task Planning. The motion planner computes collision-free trajectories for Plan 1 computed by CCALC. The robots start executing the plan by following these trajectories. At time step 3, Ring 1 is at Peg 2, Ring 2, 3 and 4 are at Peg 1; the orientations of Rings 1–4 are 4, 2, 2, 2 respectively. At this state the robots start executing the concurrent execution of moving Ring 1 on top of the stack at Peg 1. However, while the robots are carrying the ring, some external agent moves Ring 2 to Peg 2.

The monitoring agent notices this intervention and identifies two legitimate states, where Ring 1 is on Ring 2 (at Peg 2) or at Peg 3. Then it modifies Planning Problem 1 as follows:

```
:- query
maxstep :: 0..infinity;
0: % initial states
loc(1)=3, (loc(2)=p2 ++ loc(2)=p3),
loc(3)=4, loc(4)=p1,
ort(1)=4, ort(2)=1, ort(3)=2, ort(4)=2;
maxstep: % goal
loc(1)=2, loc(2)=3, loc(3)=4, loc(4)=p1,
ort(1)=2, ort(2)=2, ort(3)=2, ort(4)=2.
```

Note that the definition of an initial state involves a disjunction (++) of the two legitimate states. CCALC computes the following plan (Plan 2) of length 3 for this problem, by deciding for one of these two legitimate states (e.g., where Ring 2 is at Peg 3) as the initial state:

```
0: move(p1,p2)
1: rotate(p3,2) move(p3,p1)
2: rotate(p2,2) move(p2,p1)
```

However, the motion planner fails to find a continuous trajectory for this plan, since the robots collide with each other (between time stamps 2 and 3) while rotating Ring 1 from Orientation 2 to 4 (which is a 180° rotation).

Then the monitoring agent modifies Planning Problem 2 (obtaining Planning Problem 3) to ensure that the concurrent action `rotate(p2,2) move(p2,p1)` is not executed at the state where Ring 1 is at Peg 2 and Rings 2–4 are at Peg 1, by adding the following constraints:

```
:- query
maxstep :: 0..infinity;
0:  ...; % Initial states
maxstep: ...; % Goal
T<maxstep ->> ( % Constraints 1
  ((T: ort(1)=4) && (T: ort(2)=2) &&
   (T: ort(3)=2) && (T: ort(4)=2) &&
   (T: loc(1)=p2) && (T: loc(2)=3) &&
   (T: loc(3)=4) && (T: loc(4)=p1) &&
   (T: base(1)=p2) && (T: base(2)=p1) &&
   (T: base(3)=p1) && (T: base(4)=p1))
   ->> -((T: rotate(p2,2)) &&
         (T: move(p2,p1)))).
```

CCALC then computes the following plan (Plan 3) of length 3 for this problem:

```
0: rotate(p1,2) move(p1,p2)
1: rotate(p3,2) move(p3,p1)
2: move(p2,p1)
```

However, the motion planner fails to find a continuous trajectory for this plan, since the robots collide with each other (between time stamps 0 and 1) while rotating Ring 1 from Orientation 4 to 2 (which is a 180° rotation).

Then the monitoring agent modifies Planning Problem 3 by adding further constraints to ensure that the concurrent action `rotate(p1,2) move(p1,p2)` is not executed at the state where Rings 1, 3 and 4 are at Peg 1 and Ring 2 is at Peg 3:

```
:- query
maxstep :: 0..infinity;
0: ...; % Initial states
maxstep: ...; % Goal
...; % Constraints 1
T<maxstep ->> (
  ((T: ort(1)=4) && (T: ort(2)=1) &&
   (T: ort(3)=2) && (T: ort(4)=2) &&
   (T: loc(1)=3) && (T: loc(2)=p3) &&
   (T: loc(3)=4) && (T: loc(4)=p1) &&
   (T: base(1)=p1) && (T: base(2)=p3) &&
   (T: base(3)=p1) && (T: base(4)=p1))
   ->> -((T: rotate(p1,2)) &&
         (T: move(p1,p2)))).
```

Then CCALC computes the following plan (Plan 4) of length 3 for this problem:

```
0: rotate(p1,3) move(p1,p2)
1: rotate(p3,2) move(p3,p1)
2: rotate(p2,2) move(p2,p1)
```

Fortunately, the motion planner finds a continuous trajectory for this task plan, and the plan can be executed by the robots as in Figure 3.

VII. RELATED WORK

There have been various uses of logic-based formalisms in cognitive robotics [36], by implementing high-level robot control systems based on different families of formalisms for reasoning about actions and change. For instance, [37], [38] have presented a formal framework for high level reasoning in the style of cognitive robotics, using the action description language $\mathcal{C}+$ with the reasoner CCALC on LEGO MINDSTORMS NXT robots. There are also related work on the use of state-of-the-art automated reasoners, such as SAT solvers and ASP solvers, for high-level reasoning in various robotics applications [39] (e.g., cognitive factories [40], service robotics [41], [42], assembly planning [43]). Our work can be viewed as an extension of these related work, by tightly integrating motion planning with reasoning about actions and change, in particular, task planning.

There have been various studies to integrate task planning and motion planning. For instance, [44] describes an algorithm that combines task and motion planning techniques: the task planner is employed to find an admissible heuristic cost to speed up motion planning. [45] follows a similar approach of using task planning to speed up motion planning queries; however, does so by avoiding inexecutable subtasks as much as possible. In particular, the algorithm proposed in [45] incrementally builds a subtask graph, with each increment containing at least one more path from the start to the goal and tries to find a motion plan that supports the task plan represented by the subtask graph. Other studies that integrate task and motion planning techniques include [46], where navigation among movable obstacles is addressed for a class of problems in which disconnected free spaces can be connected by moving a single obstacle. These authors propose using a graph search at the task planning level to find the necessary actions to connect free spaces while moving least number of obstacles and a manipulation search to find plans that require the least amount of work. [47] uses assembly graphs to keep valid assembly configurations and feasible assembly steps, and make use of repositioning the structures to make the workspace less constrained. In [48], this approach is extended to include failure recovery at three distinct levels: at the lowest level contingency responses are employed, at the motion planning level plan repair for a failed assembly step is attempted, and at the task planning level the assembly sequence is replanned. [49] utilizes a tree-based search for the motion planner to obtain dynamically feasible trajectories, guided by the task planner relative to a utility heuristic. [50] and [51] combine motion planning with some form of hierarchical task planning.

Our work is different from these related work in that, thanks to the expressiveness of the action language $\mathcal{C}+$ and the computational efficiency of the state-of-the-art automated reasoners, our framework can address various challenges (e.g., concurrency, disjunctive initial states, complex goal conditions) substantially extending the set of problems that can be handled by the traditional approaches. Also there is a strong bilateral relation between task planning and motion planning in our framework: the task planner guides the motion planner to compute a continuous trajectory, whereas the motion planner guides the task planner to compute a collision-free task plans as detailed in [43].

VIII. CONCLUSIONS

We have demonstrated integration of task planning and motion planning in a general execution monitoring framework, which can be applied to domains that involves concurrencies at the task planning level, narrow passages at the motion planning level and external interventions at the execution level. We have shown how to automatically compute hybrid plans to solve an extended version of the Tower of Hanoi puzzle (with two robots) as well as how to intelligently monitor the execution of these plans in a dynamic environment, detecting and recovering from possible interventions. Our framework can be extended to multiple robots as in [42], [52], [40]. Both a dynamic simulation (with Gazebo) and a physical implementation (with Kuka youBots) of our approach applied to Tower of Hanoi are available at `http://cogrobo.sabanciuniv.edu/?page_id=706`.

REFERENCES

[1] E. Lucas, *Recreations Mathematiques*. Paris: Gauthier-Villars, 1893, vol. II, reprinted several times by Albert Blanchard.

[2] C. Gerety and P. Cull, "Time complexity of the towers of hanoi problem," *SIGACT News*, vol. 18, no. 1, pp. 80–87, 1986.

[3] P. K. Stockmeyer, "Variations on the four-post tower of hanoi puzzle," in *Proc. of Southeastern International Conference on Combinatorics, Graph Theory and Computing*, 1994, pp. 3–12.

[4] A. M. Hinz, "The tower of hanoi," *Enseign. Math.*, vol. 35, pp. 289–321, 1989.

[5] P. K. Stockmeyer and F. Lunnon, "New variations on the tower of hanoi," in *Proc. of International Conference on Fibonaci Numbers and Their Applications*, 2008.

[6] P. K. Stockmeyer, "The tower of hanoi: A bibliography," Department of Computer Science, College of William and Mary, Tech. Rep., 2005. [Online]. Available: http://www.cs.wm.edu/~pkstoc/biblio2.pdf

[7] H. Ahrabian, C. Badamchi, and A. Nowzari-Dalini, "On the solution of the towers of hanoi problem," *International Journal of Mathematical and Computer Sciences*, vol. 7, no. 1, pp. 5–8, 2011.

[8] J. S. Frame, "Solution to advanced problem 3918," *Amer. Math. Monthly*, vol. 48, pp. 216–217, 1941.

[9] B. M. Stewart, "Solution to advanced problem 3918," *Amer. Math. Monthly*, vol. 48, pp. 217–219, 1941.

[10] H. Dudeney, *The Canterbury Puzzles*. London: Thomas Nelson and Sons, 1907.

[11] R. Demontis, "What is the least number of moves needed to solve the 4-peg towers of hanoi problem?" *CoRR*, vol. abs/1203.3280, 2012.

[12] Y. Hamadi, S. Jabbour, and L. Sais, "Control-based clause sharing in parallel sat solving," in *Proc. of IJCAI*, 2009, pp. 499–504.

[13] M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub, "clasp: A conflict-driven answer set solver," in *Proc. of LPNMR*, 2007, pp. 260–265.

[14] C. P. Gomes, H. Kautz, A. Sabharwal, and B. Selman, "Cognitive robotics," in *Handbook of Knowledge Representation*. Elsevier, 2008.

[15] V. Lifschitz, "What is answer set programming?" in *Proc. of AAAI*, 2008, pp. 1594–1597.

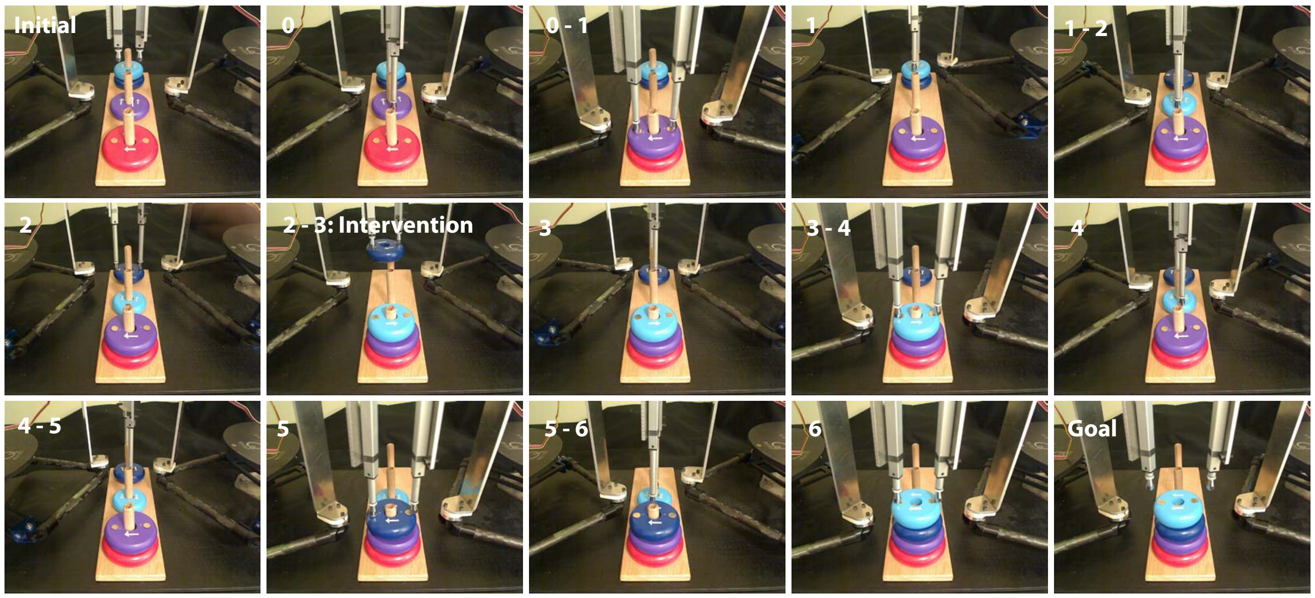

Fig. 3. Snapshots taken during the execution with two pantograph robots with magnetic tipped linear actuators attached to their end effectors.

[16] G. Brewka, T. Eiter, and M. Truszczynski, “Answer set programming at a glance,” *Commun. ACM*, vol. 54, no. 12, pp. 92–103, 2011.

[17] M. Gelfond and V. Lifschitz, “Classical negation in logic programs and disjunctive databases,” *New Generation Computing*, vol. 9, pp. 365–385, 1991.

[18] M. N. Velev and R. E. Bryant, “Effective use of boolean satisfiability procedures in the formal verification of superscalar and vliw microprocessors,” *J. Symb. Comput.*, vol. 35, no. 2, pp. 73–106, 2003.

[19] H. A. Kautz and B. Selman, “Planning as satisfiability,” in *Proc. of ECAI*, 1992, pp. 359–363.

[20] E. Erdem, Y. Erdem, H. Erdogan, and U. Oztok, “Finding answers and generating explanations for complex biomedical queries,” in *Proc. of AAAI*, 2011.

[21] M. Nogueira, M. Balduccini, M. Gelfond, R. Watson, and M. Barry, “An a-prolog decision support system for the space shuttle,” in *Proc. of PADL*, 2001, pp. 169–183.

[22] F. Ricca, G. Grasso, M. Alviano, M. Manna, V. Lio, S. Iiritano, and N. Leone, “Team-building with answer set programming in the Gioia-Tauro seaport,” *Theory and Practice of Logic Prog.*, vol. 12, 2012.

[23] M. Gelfond and V. Lifschitz, “Action languages,” *Electronic Transactions on Artificial Intelligence*, vol. 2, pp. 193–210, 1998.

[24] N. McCain and H. Turner, “Satisfiability planning with causal theories,” in *Proc. of KR*, 1998, pp. 212–223.

[25] V. Lifschitz and H. Turner, “Representing transition systems by logic programs,” in *Proc. of LPNMR*, 1999, pp. 92–106.

[26] E. Giunchiglia, J. Lee, V. Lifschitz, N. McCain, and H. Turner, “Nonmonotonic causal theories,” *Artificial Intelligence*, vol. 153, no. 1–2, pp. 49–104, 2004.

[27] N. McCain and H. Turner, “Causal theories of action and change,” in *Proc. of AAAI/IAAI*, 1997, pp. 460–465.

[28] J. McCarthy and P. Hayes, “Some philosophical problems from the standpoint of artificial intelligence,” *Machine Intelligence*, vol. 4, pp. 463–502, 1969.

[29] J. McCarthy, “Circumscription—a form of non-monotonic reasoning,” *Artificial Intelligence*, vol. 13, pp. 27–39,171–172, 1980.

[30] J. Finger, “Exploiting constraints in design synthesis,” Ph.D. dissertation, Stanford University, 1986, phD thesis.

[31] J. Lee and V. Lifschitz, “Describing additive fluents in action language $\mathcal{C}$+,” in *Proc. of IJCAI*, 2003.

[32] E. Erdem and A. Gabaldon, “Cumulative effects of concurrent actions on numeric-valued fluents,” in *Proc. of AAAI*, 2005, pp. 627–632.

[33] S. M. Lavalle, “Rapidly-exploring random trees: A new tool for path planning,” Tech. Rep., 1998.

[34] L. Kavraki, P. Svestka, J. Latombe, and M. Overmars, “Probabilistic roadmaps for path planning in high-dimensional configuration spaces,” *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[35] J. J. Kuffner Jr. and S. M. Lavalle, “RRT-Connect: An efficient approach to single-query path planning,” in *Proc. of ICRA*, 2000, pp. 995–1001.

[36] H. Levesque and G. Lakemeyer, “Cognitive robotics,” in *Handbook of Knowledge Representation*. Elsevier, 2007.

[37] O. Caldiran, K. Haspalamutgil, A. Ok, C. Palaz, E. Erdem, and V. Patoglu, “Bridging the gap between high-level reasoning and low-level control,” in *Proc. of LPNMR*, 2009.

[38] ——, “From discrete task plans to continuous trajectories,” in *Proc. of BTAMP*, 2009.

[39] E. Erdem and V. Patoglu, “Applications of action languages in cognitive robotics,” in *Correct Reasoning*, 2012, pp. 229–246.

[40] E. Erdem, K. Haspalamutgil, V. Patoglu, and T. Uras, “Causality-based planning and diagnostic reasoning for cognitive factories,” in *Proc. of IEEE Int. Conf. Emerging Tech. & Factory Automation (ETFA)*, 2012.

[41] E. Aker, V. Patoglu, and E. Erdem, “Answer set programming for reasoning with semantic knowledge in collaborative housekeeping robotics,” in *Proc. Int. IFAC Symp. Robot Control (SYROCO)*, 2012.

[42] E. Aker, A. Erdogan, E. Erdem, and V. Patoglu, “Causal reasoning for planning and coordination of multiple housekeeping robots,” in *Proc. of LPNMR*, 2011, pp. 311–316.

[43] E. Erdem, K. Haspalamutgil, C. Palaz, V. Patoglu, and T. Uras, “Combining high-level causal reasoning with low-level geometric reasoning and motion planning for robotic manipulation,” in *Proc. of ICRA*, 2011.

[44] S. Cambon, R. Alami, and F. Gravot, “A hybrid approach to intricate motion, manipulation and task planning,” *The International Journal of Robotics Research*, vol. 28, no. 1, pp. 104–126, 2009.

[45] K. Hauser and J. Latombe, “Integrating task and PRM motion planning: Dealing with many infeasible motion planning queries,” in *Proc. of BTAMP*, 2009.

[46] M. Stilman and J. Kuffner, “Navigation among movable obstacles: Real-time reasoning in complex environments,” in *Proc. of Humanoids*, vol. 1, December 2004, pp. 322–341.

[47] F. Heger, “Generating robust assembly plans in constrained environments,” in *Proc. of ICRA*, 2008, pp. 4068–4073.

[48] F. Heger and S. Singh, “Robust robotic assembly through contingencies, plan repair and re-planning,” in *Proc. of ICRA*, 2010.

[49] E. Plaku and G. D. Hager, “Sampling-based motion and symbolic action planning with geometric and differential constraints,” in *Proc. of ICRA*, 2010, pp. 5002–5008.

[50] J. Wolfe, B. Marthi, and S. Russell, “Combined task and motion planning for mobile manipulation,” in *Proc. of ICAPS*, 2010.

[51] L. P. Kaelbling and T. Lozano-Pérez, “Hierarchical task and motion planning in the now,” in *Proc. of ICRA*, 2011, pp. 1470–1477.

[52] E. Erdem, E. Aker, and V. Patoglu, “Answer set programming for collaborative housekeeping robotics: representation, reasoning, and execution,” *Intelligent Service Robotics*, vol. 5, pp. 275–291, 2012.