PhD DISSERTATION

# Hybrid Conditional Planning for Service Robotics

by

Ahmed Nouman

Submitted to the Graduate School of Sabancı University
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

Sabancı University

January, 2019

PhD THESIS DISSERTATION

Hybrid Conditional Planning

for Service Robotics

APPROVED BY:

| | |
|---|---|
| Assoc. Prof. Dr. Volkan Patoğlu (Thesis Advisor) | [illegible] |
| Assoc. Prof. Dr. Güllü Kızıltaş Şendur | [illegible] |
| Assist. Prof. Dr. Kamer Kaya | [illegible] |
| Assist. Prof. Dr. Orkunt Sabuncu | [illegible] |
| Assist. Prof. Dr. Reyyan Yeniterzi | [illegible] |

DATE OF APPROVAL: 3rd Dec, 2018

© Ahmed Nouman, 2018
All Rights Reserved

# Hybrid Conditional Planning for Service Robotics

Ahmed Nouman

Mechatronics Engineering, Doctor of Philosophy, 2019

Thesis Advisors: Assoc. Prof. Volkan Patoğlu, Assoc. Prof. Esra Erdem Patoğlu

**Keywords**: Planning under incomplete knowledge and partial observability, Conditional planning, Hybrid planning, Motion planning, Task planning, Plan execution monitoring, Cognitive robotics, Action languages

## Abstract

Planning is an indispensable ability for intelligent service robots operating in unstructured environments. Given service robots commonly have incomplete knowledge about and partial observability of handle such uncertainty. Moreover, the plans they compute should be feasible for real-world execution.

Conditional planning is concerned with reaching goals from an initial state, in the presence of incomplete knowledge and partial observability; by utilizing sensing actions. Since all contingencies are considered in advance, a conditional plan is essentially a tree of actions where the root represents the initial state, leaves represent goal states, and each branch of the tree from the root to a leaf represents a possible execution of (deterministic) actuation actions and (non-deterministic) sensing actions to reach a goal state. Hybrid conditional planning extends conditional planning further by integrating low-level feasibility checks into executability conditions of actuation actions in conditional plans.

We introduce a *parallel offline* algorithm called HCPlan, for computing hybrid conditional plans in robotics applications. HCPlan relies on modeling actuation actions and sensing actions in the causality-based action description language $\mathcal{C}+$, and computation of the branches of a conditional plan in parallel using a SAT solver. In particular, thanks to external atoms, continuous feasibility checks (such as collision and reachability checks) are embedded into causal laws representing actuation actions and sensing actions; and thus each branch of a hybrid conditional plan describes a feasible execution of actions to reach their goals. Utilizing causal laws that describe non-deterministic effects of actions, sensing actions can be explicitly formalized; and thus each branch of a conditional plan can be computed without necessitating an ordering of sensing actions in advance.

Furthermore, we introduce two different extensions of our hybrid conditional planner HCPlan: HCPlan-Anytime and HCPlan-Reactive. HCPlan-Anytime computes a *partial* hybrid conditional plan within a given time, by generating the branches with respect to their probability of execution. HCPlan-Reactive computes a hybrid conditional plan with a receding horizon. These extensions trade-off completeness of hybrid conditional plans for improved computation time, and provide useful important variations towards real-time use of the hybrid conditional planning.

We develop comprehensive benchmarks for service robotics domain and evaluate our approach over these benchmarks with extensive experiments in terms of computational efficiency and plan quality. We compare HCPlan with other related conditional planners and approaches. We further demonstrate the usefulness of our approach in service robotics applications through dynamic simulations and physical implementations.

Hizmet Robotları için Melez Koşullu Planlama

Ahmed Nouman

Mekatronik Mühendisliği, Doktora, 2019

Tez Danışmanları: Doç. Dr. Volkan Patoğlu, Doç. Dr. Esra Erdem Patoğlu

**Keywords**: Eksik bilgi ve kısmi gözlemlenebilirlik altında planlama, Koşullu planlama, Melez planlama, Hareket planlaması, Görev planlaması, İcra takibi, Bilişsel robotik, Eylem dilleri

## Abstract

Planlama, yapılandırılmamış ortamlarda çalışan akıllı servis robotları için vazgeçilmez bir özelliktir. Hizmet robotları, çevrelerini genelde kısmi olarak gözlemleyebilirler ve çevreleri hakkında eksik bilgiye sahiptirler. Otonom hizmet robotları, gerçek dünyada uygulanabilir planlar oluşturabilmek için, eksik bilgiyi, kısmi gözlemlenebilirliği ve geometrik uygulanabilirlik kriterlerini hesaba katabilen, klasik planlamanın ötesinde, bilişsel yeteneklerle donatılmalıdır.

Koşullu planlama, eksik bilginin ve algılama eylemlerinin varlığında, başlangıç durumundan hedeflere ulaşmayı amaçlar. Çevrimdışı koşullu bir planda tüm olasılıklar önceden değerlendirilir; koşullu bir plan, kökün başlangıçtaki durumu, yaprakların hedef durumlarını temsil ettiği ve kökten yaprağa her bir dalın muhtemel bir plan uygulamasını temsil ettiği, deterministik harekete geçirme eylemleri ve deterministik olmayan algılama eylemlerinden oluşan bir ağaçtır. Melez koşullu planlama, fiziksel uygulanabilirlik kontrollerini de koşullu planlamaya entegre eder.

Bu tezde, robot uygulamalarına yönelik melez koşullu planlama için yeni bir *paralel çevrimdışı* algoritma (HCPlan) öneriyoruz. HCPlan, harekete geçirme ve algılama eylemlerinin nedensellik temelli eylem tanım dili $\mathcal{C}+$'ta modellenmesine ve koşullu planı oluşturan ağacın her bir dalının bir SAT çözücü kullanarak paralel olarak hesaplanmasına dayanır. Robotun eylemlerinin sürekli uzayda uygulanabilirliğinine ait testler (çarpışma ve ulaşılabilirlik testleri gibi) harici atomlar kullanarak nedensel kurallara entegre edilmekte ve böylece melez koşullu planın her bir dalı, hedef durumlara ulaşmak için uygulanabilir bir eylem sıralamasını ve icrasını temsil etmektedir. Önerilen yaklaşımda algılama eylemlerinin deterministik olmayan etkilerini nedensel kurallar ile formal olarak biçimlendirmekte, böylece koşullu planın her bir dalı, algılama eylemlerinin önceden sıralanması gerekmeksizin hesaplanabilmektedir.

Bu tezde ayrıca, melez koşullu planlama algoritmamızın iki farklı uzantısını sunuyoruz: HCPlan-Anytime ve HCPlan-Reactive. HCPlan-Anytime, ayrılan süre içerisinde melez koşullu plana ait dallardan icra sırasında en olası olanları önceliklendirerek *kısmi* melez koşullu plan hesaplayabilmektedir. HCPlan-Reactive uzaklaşan bir ufka dair melez koşullu planlar hesaplar. Bu uzantılar, daha verimli hesaplama süresi için melez koşullu planların tamlığından ödün vermekte, melez koşullu planlamanın gerçek zamanlı kullanımına yönelik önemli seçenekler sunmaktadır.

Değerlendirme için, HCPlan'ı diğer ilgili koşullu planlayıcılar ve yaklaşımlarla karşılaştırmalarını sunuyor, hizmet robotiği için kapsamlı kıyaslama senaryoları önerip, bu senaryolar üzerinde deneylerle yaklaşımımızın hesaplama verimliliğini ve plan kalitesini değerlendiriyoruz. Ayrıca, yaklaşımımızın servis robotik uygulamalarındaki başarımını dinamik simülasyonlar ve fiziksel uygulamalar ile gösteriyoruz.

# Acknowledgements

*I would first like to thank my thesis advisors Assoc. Prof. Dr. Volkan Patoğlu and Assoc. Prof. Dr. Esra Erdem Patoğlu at Sabanci University. They consistently steered me in the right the direction whenever they thought I needed it. I would also like to thank my jury members for their useful input.*

*Also, I would like to acknowledge my friends Arsalan Javeed, Damien Jade Duff, and Amir Sultan; my brother Faseeh Ahmad who were with me in my hard times and always encouraged me.*

*Finally, I must express very profound gratitude to my parents, my son Eesa Ahmed, and my partner Zarkhania Javed for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them. Thank you :)*

# Contents

# List of Figures

# List of Tables

## List of Algorithms

# Chapter I

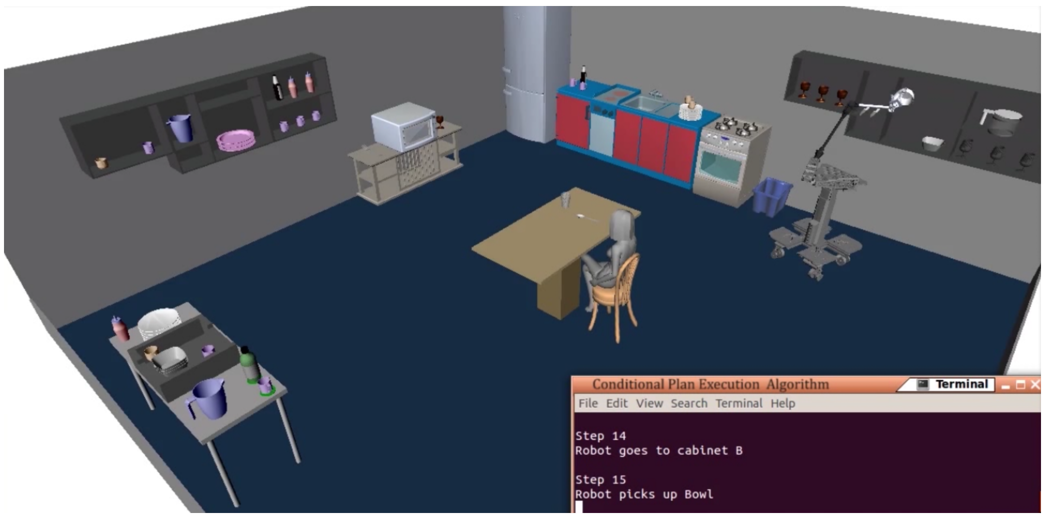
## 1 Introduction

Planning is the art and practice of thinking before acting [1]. Automated planning, sometimes referred simply as planning, is a branch of artificial intelligence that concerns with the realization of strategies or action sequences by intelligent agents to complete some task. Formally, given a world model, an initial world state, and the desired goal conditions, planning problem is finding the (best) course of actions that transform the world from the initial state to a state where the goal conditions hold. In known environments with available models, planning can be done offline. Solutions can be found and evaluated prior to execution. In partial observable environments, however, the strategy often needs to be revised online and the models/policies must be adapted. *Plan execution monitoring* is an example of such approaches that utilizes online sensing with classical planning to compute solutions. Classical planning assumes complete knowledge and full observability about the environment, and its complexity is NP-complete for polynomially bounded plans [2].

As an alternative to plan execution monitoring, *conditional planning* is focused on finding solutions from the initial state to goals, in the presence of incomplete knowledge and sensing actions [3, 4, 5, 6, 7]. Conditional planning considers all contingencies during the planning phase to compute a conditional plan, which is essentially a tree of sensing and actuation actions, where the root represents the initial state, leaves represent goal states, and each branch of the tree from the root to a leaf represents a possible sequence of actions to reach a goal state. Computing conditional plan is an intractable problem; for polynomially bounded plans with partial observability, it is PSPACE-complete [8].

Conditional planning computes valid plans in the presence of incomplete knowledge and partial observability, however, computation of valid conditional plans does not guarantee successful plan execution in real-life robotic applications. In robotic applications, the computed conditional plans have to be executed by robots that are capable of navigation and manipulation tasks. This necessitates integration of low-level feasibility checks (such as collision, force closure, and reachability checks) into high-level planning domains. The integration of low-level feasibility checks into high-level planning techniques comes under hybrid planning framework [9, 10, 11, 12, 13, 14, 15, 16, 17, 6, 7, 18, 19] and ensures that the computed plan is not just valid, but feasible as well. This, however, is a challenging problem, since feasibility checks are performed over continuous spaces of robotic configurations whereas high-level planning is done over discrete representations of the world.

In this dissertation, we present a formal framework that utilizes conditional and hybrid planning approaches to handle problems involving planning under incomplete knowledge and partial observability. In Section 1.1, we introduce the problem and its challenges. Section 1.2 provides a brief introduction to our framework. In Section 1.3, we discuss the main contributions of this dissertation, while Section 1.4 presents dissertation overview.


Conditional Plan Execution Algorithm
Terminal
File Edit View Search Terminal Help

```
Step 14
Robot goes to cabinet B

Step 15
Robot picks up Bowl
```

Figure 1.1: A service robot setting up kitchen table.

## 1.1 Problem statement

In order to present the problem of planning under incomplete knowledge and partial observability, let us consider a complex service robotic scenario where a mobile manipulation robot needs to set up the kitchen table as shown in Figure 1.1. The robot can navigate around the environment, picking and placing objects as required. Kitchenware, such as mugs, spoons, knives, plates may be found in cabinets or may be left on other flat surfaces, such as countertops or shelves. Also, there is a kitchen table, where the robot needs to place proper kitchenware complying with table setting etiquette.

In order to complete the task of setting up the kitchen table, the robot needs to compute a sequence of manipulation and navigation actions that lead to a state where the table is set. However, in this scenarios, the robot does not have full observability of environment, for example, the location of the bowl may be unknown, the cleanliness of fork may not be known or even the robot may be unaware of the type of the food for which it needs to set

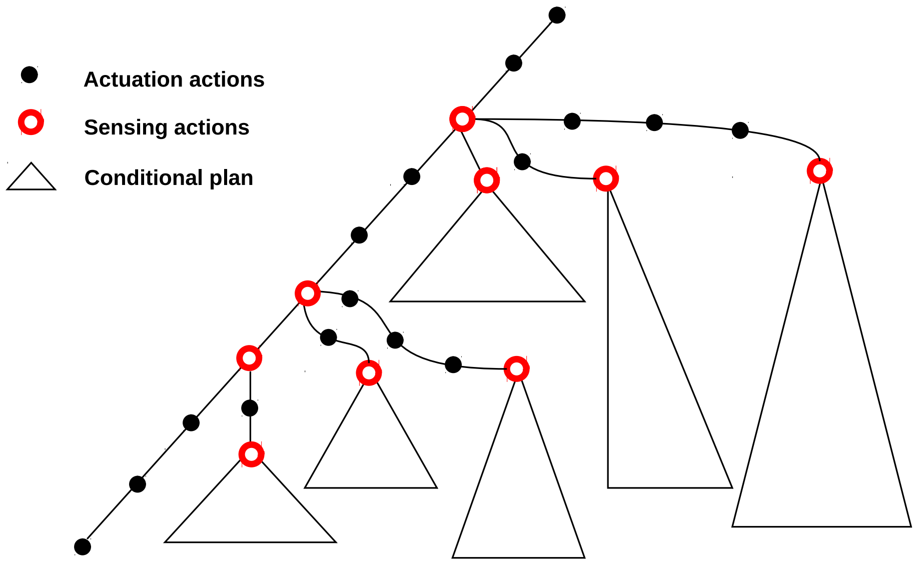


Figure 1.2: A sample model of a conditional plan.

up the kitchen table. The robot needs to plan under incomplete knowledge about the initial state and thus conventional classical planning techniques cannot be used to solve this problem. Therefore, in order to solve such a task, we need to look beyond classical planning approaches and provide solutions that can handle incomplete knowledge and partial observability.

Furthermore, the proposed solutions are to be executed by the robot. This has its own challenges, for example, the proposed solution must make sure that the robot does not collide with the environment or it grasps the objects properly so they do not fall during manipulation. Thus, the solutions should be valid as well as feasible with respect to robots, making them utilizable in real-world applications.

## 1.2 Overview of our approach

Our approach extends hybrid planning beyond classical planning, to conditional planning, which allows for dealing with incomplete knowledge due to partial observability at the time of planning. In conditional planning, sensing actions are considered as a part of planning [3, 4, 5, 6, 7]; according to the possible outcomes of sensing actions, different conditional plans are computed. Therefore, a conditional plan looks like a tree of actions, where branching occurs at vertices that characterize *sensing actions*; other vertices characterize *actuation actions* as shown in Figure 1.2. For instance, in the example above, the possibility of a plate being dirty can be considered as part of conditional planning and a sensing action to check the cleanliness of the plate may be computed as a part of the conditional plan. While executing such a conditional plan, according to the outcome of the sensing action (e.g., computed using a perception algorithm), if the plate is detected to be clean then the robot puts it on the table; otherwise, the robot first cleans the plate and then puts it on the table. No replanning is needed for this contingency since the robot plans for the sensing actions as required and knows what to do with each possible outcome of a sensing action. Moreover, we integrate low-level feasible checks as executability conditions of actions to prevent infeasible actions from being part of a computed conditional plan. This makes computed plans feasible for robots to be executed in real-world applications.

## 1.3 Dissertation contributions

The summary of the main contributions of this dissertation are listed below:

- A parallel hybrid conditional planning framework:

  1. We have provided formal definitions of sensing and actuation actions to formalize service robotic domains.
  2. We have proposed a novel algorithm for parallel computation of hybrid conditional plans for service robotic domains.
  3. We have implemented the planner HCPlan that provides the realization of the hybrid conditional planning algorithm.
  4. We have implemented a parallel version of HCPlan that utilizes parallel computation of branches to compute conditional plans faster.

- Extensions of the planner, HCPlan:

  1. We have extended the conditional planning algorithm to enable the computation of partial conditional plans as an anytime algorithm.
  2. We have implemented HCPlan-Anytime, the anytime version of the planner HCPlan.
  3. We have extended the conditional planning towards a reactive approach to computing partial conditional plans.
  4. We have implemented HCPlan-Reactive, the reactive version of the planner HCPlan.

- Service robotics benchmarks for planning under incomplete knowledge and partial observability:
    1. We have introduced benchmark domains for real-world service robotic applications involving manipulation and navigation tasks under incomplete knowledge and partial observability.
    2. We have developed planning instances for the benchmark domains. Also, we provide a mechanism to create new instances.
    3. We have implemented feasibility checks for the benchmark domains.
- Experimental evaluation of HCPLAN, HCPLAN-ANYTIME, and HCPLAN-REACTIVE:
    1. We have completed a comprehensive experimental evaluation of HCPLAN using the benchmark domains.
    2. We have implemented a plan execution monitoring algorithm and compared its performance with HCPLAN for the benchmark domains.
    3. We have provided a comparison between an alternate parallel hybrid conditional planner HCP-ASP for the benchmark domains.
    4. We have completed an experimental evaluation of HCPLAN-ANYTIME for the benchmark domains along with a comparison of the results with HCPLAN.
    5. We have completed an experimental evaluation of HCPLAN-REACTIVE for the benchmark domains along with a comparison of the results with HCPLAN and plan execution monitoring.

## 1.4 Overview of dissertation

- Chapter 2 provides a literature review about the state of the art approaches that plan under incomplete knowledge and partial observability.
- Chapter 3 presents a brief overview of action description language $\mathcal{C}+$, the input language of our hybrid conditional planning framework.
- Chapter 4 describes our parallel hybrid conditional planning framework.
- Chapter 5 describes anytime and reactive parallel hybrid conditional planners as extensions to our planner, HCPlan.
- Chapter 6 describes service robotic benchmarks for planning under incomplete knowledge and partial observability.
- Chapter 7 discusses the experimental evaluation of our hybrid conditional planning framework for the benchmark domains along with a comparison to the other state of the art approaches.
- Chapter 8 provides dynamic simulation and physical execution of plans computed with our approach for real-world service robotic scenarios.
- Chapter 9 provides concluding remarks about the dissertation along with ongoing work.

# Chapter II

## 2 Literature Review

This dissertation focuses on planning under incomplete knowledge and partial observability. In the literature, we can find attempts to solve the problem using plan execution monitoring, policy generation, and conditional planning. In Section 2.2, we discuss the attempts in the literature to solve the problem. Furthermore, while dealing with robotics applications, integration of feasibility checks are important to ensure plan feasibility. Integration of these checks in task planning come under the name of hybrid planning and Section 2.3 describes recent work about these approaches.

### 2.1 Plan execution monitoring

Planning is a model-based approach to action selection where different types of models are used to make precise the different types of agents, environments, and controllers [20, 21]. Classical planning is the simplest form of planning, concerned with the achievement of goals in a deterministic environment where the initial state is completely known [22]. Classical planning assumes the model of the world to be finite, discrete and deterministic.

In order to operate in real-world applications where the environment is dynamic and unpredictable, the assumptions about the deterministic model of the world have to be modified. Furthermore, in such dynamic and unpredictable environment, the agents need the ability to detect if the execution has proceeded as planned and when it does not, use that information to come up with some new strategy to complete the task. An execution monitoring system allows the robot to identify such failures, classify them and update the robot plan, such that it can recover from such failures. Bjäreland [23] defines an execution monitoring system as:

*"Execution monitoring is an agent's process of identifying discrepancies between observations of the actual world and the predictions and expectations derived from its representation of the world, classifying such discrepancies, and recovering from them."*

Plan execution monitoring combines classical planning with execution monitoring approaches to plan, execute, and monitor the plans in real-world applications. Plan execution monitoring utilizes online sensing to help the agents recover from failures that may occur during the execution of the plan. These failures may occur due to partial observability and incomplete knowledge about the environment.

### Working of an execution monitoring system

A plan execution monitoring approach consists of numerous sub-modules and Figure 2.1 shows the structure of a basic approach.

- Model describes the abstract image of the world that can be used for planning.
- Observations are taken in response to active sensing requests or be delivered to the passive agent at a certain frequency. Typically, these

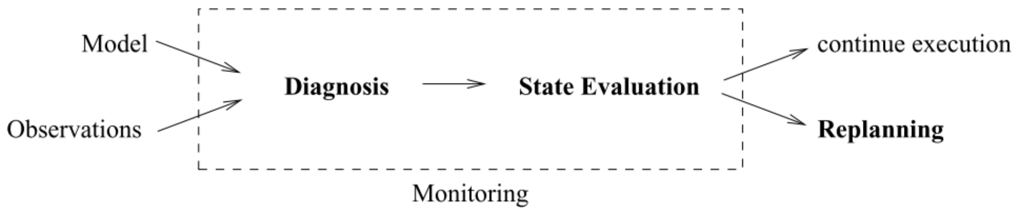


Figure 2.1: A simple plan execution monitor system

observations do not reveal the complete state of the world.

- Diagnosis is the task of estimating the actual state of the environment and determining a sequence of events that produced it, based on the model. State estimation is typically focused on detecting the discrepancies between what was predicted by the model and what was actually observed in the real world.

- State evaluation determines whether in the estimated state, the validity and/or optimality of the current plan is preserved or not. It evaluates the relevance of the discovered and diagnosed discrepancy between the prediction and the estimated state, to decide whether any kind of replanning is required or advisable.

SHAKEY was one of the earliest robotic systems that used planning based on STRIPS language [24] and execution monitoring system PLANEX[25] in order to execute plans on a real-world environment. Since then a lot of plan execution monitoring approaches have been developed and applied to domestic service robotics systems [26, 27, 28, 29, 30, 31, 32, 33, 34]. Furthermore, plan execution monitoring is a broad research topic and the reader is encouraged to read [35, 36] for detailed surveys on the topic for robotic applications.

## 2.2 Planning under incomplete knowledge

Work on planning under incomplete knowledge and partial observability can be divided into three main categories: plan execution monitoring, policy generation, and conditional planning. We discuss each one of them in the following sections.

### 2.2.1 Policy generations

Partially observable markov decision processes (POMDPs) have been studied to generate policies for solving planning under partial observability [37, 38, 39, 40]. Policy generation focuses on maximizing some reward functions and generates a policy (condition-action pairs) over a finite or an infinite horizon. The complexity of policy generation is undecidable [41, 42]. Unlike POMDPs, conditional planning focuses on the computation of plans (tree of actions) that are guaranteed to reach goal states. Conditional plans are more suitable for execution by an intelligent agent since they provide the complete sequence of actions from initial state to goal states along with steps where sensing should be performed. Moreover, the agent does not have to maintain its belief states in a compiled form and this is why sometimes policies are transformed into conditional plans [43].

### 2.2.2 Conditional planning

Conditional planning allows us to compute plans in the presence of uncertainties that arise due to partial observability and lack of complete knowledge at the time of planning. In conditional planning, *sensing actions* are considered as part of planning [3, 4, 5, 6, 7], and therefore, conditional plan can be viewed as a tree of (deterministic) *actuation actions* and (non-deterministic) *sensing actions.* For instance, in kitchen table setting scenario introduced in Section 1.1, the possibility of a plate being dirty can be considered as part of conditional planning and a sensing action to check the cleanliness of the plate may be computed as a part of the conditional plan.

Computing conditional plans is one of the hardest planning problems [44, 45]. Indeed plan existence for conditional planning is $2-EXP$-complete [46, 47]. Even for polynomially bounded plans with limited number of sensing actions, the complexity of the problem is $\Sigma_2^P$-complete [8]. Despite such a high complexity associated with their computation, we still see a variety of work on conditional planning that has led to some *online conditional planners*, such as CLG [48], K-Planner [49], SDR [50], HCP [51] and CPOR [52], and *offline conditional planners,* such as Contingent-FF [53], POND [54], PKS [55, 56], CLG (offline version), ASCP [57], DNFct [58], PO-PRP [43], HCPLAN [6] and HCP-ASP [7].

Online conditional planners use online sensing to compute plans, therefore they do not need to handle a potentially exponential number of contingencies during the planning phase. This typically results in less computation and planning time compared to offline approaches. However, since computed plans are not complete with respect to contingencies, they may suffer from failures to reach the goal state. Offline conditional plan, on the other hand, constructs a plan which is complete with respect to contingencies considered during planning phase [6, 7]. Such plans can be represented as trees where special vertices are included for different outcomes of sensing and each branch represents a possible way of reaching a goal state from the incomplete initial state under different outcomes of sensing. The work on offline conditional planning can further be divided into two groups: *Search-based* *offline conditional planning approaches* view the conditional planning problem as a non-deterministic search problem in belief space and build planners (e.g., as in Contingent-FF, PKS, POND, HCP) utilizing search algorithms (e.g., forward search, heuristic search) to compute solutions. *Compilation-based offline conditional planning approaches* compile the conditional planning problem into many planning problems in state space and utilize conformant or classical planning approaches to compute solutions (e.g., as in CLG, DNFct, PO-PRP, HCP-ASP). In that sense, our planner HCPlan is offline compilation-based; we utilize action-language based classical planning framework CCalc [59] to solve conditional planning problems.

For a better comparison, let us give some more details about these related offline compilation-based approaches. The main idea behind most of the compilation-based algorithms [48, 60, 22] is to compute parts of branches (i.e., sequences of actuation actions between pairs of closest sensing actions) by using classical planners; and then combine them into a tree. Firstly, these algorithms need to decide on the order of the sensing actions along the branches; various sorts of heuristics are used to decide for the next sensing action. Then, for every two closest sensing actions $a$ and $b$ on a branch, these algorithms try to compute a sequence of actuation actions using a classical planner. For that, they need to specify the planning problem: what is the initial state? what are the goals? To overcome these difficulties, the related approaches introduce preconditions for each sensing action; in this way, the goals can be specified in terms of the preconditions of the sensing action $b$. As for the initial state, these approaches consider all possible initial states and try to find a conformant plan by transforming the conformant planning problem into a classical planning problem. Recall that conformant planning considers incomplete initial state and no observability, and its aim is to find an action sequence that reaches the goal for every possible initial state [61]. Therefore, the size of the generated classical planning problem obtained from these approaches and the time required to compute a solution is very large. Furthermore, a solution may not exist for the conformant planning problem.

Different from most of the offline compilation-based planners (except HCP-ASP), our approach is: *parallel* in sense that it computes different branches of conditional plans simultaneously, *hybrid* in the spirit of [10, 11, 12, 13, 14]; motivated by robotics applications, our approach embeds feasibility checks (e.g., collision checks, reachability checks, graspability checks) into conditional planning. Moreover, it models multi-valued sensing actions as non-deterministic actions (so ordering of sensing actions and solving conformant planning problems are not needed), it uses non-monotonic *default* constructs to represent non-occurrences of sensing actions and it allows for concurrency of actuation actions. Figure 2.1 compares our planner to different conditional planning approaches in the literature.

HCPlan has similarities with HCP-ASP: they both compute hybrid conditional plans, utilize parallel computation of branches, implement reuse of plans to speed up plan computation, use non-monotonic default constructs to represent non-occurrences of sensing actions and model sensing actions as non-deterministic actions. Both, do not need to decide on the order of sensing actions in advance. On the other hand, HCP-ASP utilizes answer set programming (ASP) [62] with Clingo [63], while HCPlan utilizes CCalc [59] with action language $\mathcal{C}+$ [64] and SAT solvers. The output of HCP-ASP is directed acyclic graphs and/or trees whereas, HCPlan computes trees. Furthermore, HCPlan has a novel extension where it computes partial hybrid

Table 2.1: A comparison between state of the art conditional planners.

| Name | Approach | Hybrid | Parallel | Mode | Language (variant) |
|---|---|---|---|---|---|
| HCP | search-based | no | no | online | PDDL |
| CPOR | search-based | no | no | online | PDDL |
| Contingent-FF | search-based | no | no | offline | STRIPS |
| POND | search-based | no | no | offline | PPDDL |
| PKS | search-based | yes | no | offline | STRIPS |
| ASCP | search-based | no | no | offline | $\mathcal{A}_K^c$/ASP |
| DNFct | compilation-based | no | no | offline | PDDL |
| CLG | compilation-based | no | no | offline/online | PDDL |
| K-Planner | compilation-based | no | no | online | STRIPS |
| SDR | compilation-based | no | no | online | PDDL |
| PO-PRP | compilation-based | no | no | offline | PDDL/STRIPS |
| HCP-ASP | compilation-based | yes | yes | offline | ASP |
| HCPlan | **compilation-based** | **yes** | **yes** | **offline/online** | $\mathcal{C}+$ |

conditional plans based on anytime and reactive (semi-online) planning.

## 2.3 Hybrid planning approaches

Hybrid planning involves the integration of high-level task planning techniques with low-level planning approaches like motion planning, geometric planning etc [65, 66, 67, 68, 69, 70, 71]. Hybrid planning is studied a lot in literature especially for planning systems involving robots since computation of valid and feasible plan is required for smooth execution. Recent work on hybrid planning can be divided into three main categories as follow:

1. Modifications/introductions of search algorithms for motion/task planning [67, 72, 12, 14, 73, 13].

2. Integration in formal methods [11, 74].

3. Modification in representation of domains [75, 10, 9, 17, 7].

Our hybrid conditional framework is similar to the last group since we integrate feasibility checks in action descriptions via external atoms (in the spirit of semantic attachments in theorem proving [76]) without having to modify the classical planners or motion planners. The reader is encouraged to read the recent studies [14, 17] that describe surveying and empirically analyzing some of these approaches.

## 2.4 Novelties in our approach

We propose a novel parallel hybrid conditional planning framework for service robotic applications. Our approach is offline and compilation-based, similar to [7, 48, 60, 22], where parts of branches (i.e., sequences of actuation actions between pairs of closest sensing actions) are computed using classical planners, and combined into a tree. From the perspective of modeling, our approach is different from other offline compilation-based approaches except [7] in following ways:

1. Our approach capitalizes on a non-monotonic logical representation, where defaults are used to describe occurrences (and non-occurrences) of sensing actions, eliminating the need to decide on the order of sensing actions in advance (before conditional planning) or separate their computation from that of the actuation actions. While computing a hybrid conditional plan, our approach also plans for the sensing actions required to reach the goal.

2. Our approach represents sensing actions as non-deterministic actions, eliminating the need for conformant planning.

3. Our conditional planner is hybrid, that is, low-level feasibility checks (e.g., the existence of collision-free trajectories, reachability checks,

graspability checks) are integrated into action descriptions to ensure that the computed plans are physically executable.

4. Our conditional planner is parallel and utilizes parallel computation of branches to compute plans faster.

5. Our approach utilizes re-usability of previously computed branches to reduce computational work and planning time.

# Chapter III

## 3 Action Descriptions Language $\mathcal{C}+$

CCalc [59] is a knowledge representation and automated reasoning system that utilizes SAT solvers to compute plans for a planning system. The idea of CCalc is to represent domain description in the form of causal laws in action language $\mathcal{C}+$ [64] along with a planning problem and compute an action plan that satisfies the goal state.

### 3.1 Language of $\mathcal{C}+$

We start with a set of symbols, called *constants*; each constant $e$ is associated with a non-empty finite set $Dom(e)$ of symbols. The constants are divided into two types: *fluent constants* and *action constants*. An *atom* is an expression of the form $e = v$ where $e$ is a constant and $v \in Dom(e)$. We use $e$ (respectively, $\neg e$) instead of $e = true$ (respectively, $e = false$). A $formula$ is a propositional combination of atoms. A *fluent formula* (respectively, an *action formula*) is a formula such that all constants appearing in it are fluent (respectively, action) constants.

**Causal laws**

The action description language $\mathcal{C}+$ consists of three kinds of expressions (called *causal laws*):

1. *static laws* of the form:

   $$\textbf{caused } F \textbf{ if } G$$

   where $F$ and $G$ are fluent formulas.

2. *action dynamic laws* of the form:

   $$\textbf{caused } F \textbf{ if } G$$

   where $F$ is an action formula and $G$ is a formula.

3. *fluent dynamic laws* of the form:

   $$\textbf{caused } F \textbf{ if } G \textbf{ after } H$$

   where $F$ and $G$ are fluent formulas, and $H$ is a formula.

Static laws represent causal dependencies between fluents in the same state while action dynamic laws express causal dependencies between concurrently executed actions. Fluent dynamic laws are the most important element of the language, because they can be used to describe direct effects of actions. Moreover, an *action description* is represented by a set of causal laws.

**Commonsense law of inertia**

We can represent the commonsense law of inertia for a simple fluent literal $F$ with the expression:

$$\textbf{inertial } F$$

which stands for fluent dynamic law:

$$\textbf{caused } F \textbf{ if } F \textbf{ after } F$$

**Exogeneity of actions**

We can express that an action fluent $F$ is exogenous (i.e., the causes of the occurrence or non-occurrence of the action $F$ is not given in the action domain description) by:

$$\textbf{exogenous } F$$

which stands for

$$\textbf{caused } F \textbf{ if } F$$

$$\textbf{caused } \neg F \textbf{ if } \neg F$$

**State constraints**

We can constraint the occurence of a fluent $F$ by the expression:

$$\textbf{constraint } \neg F$$

which stands for:

$$\textbf{caused } \bot \textbf{ if } F$$

**Defaults**

We can express the occurrences (and non-occurrences) of a fluent $F$ at each state by the expression:

$$\textbf{default } F$$

which stands for:

$$\textbf{caused } F \textbf{ if } F$$

**Preconditions of actions**

Preconditions $G$ of an action fluent $F$ can be expressed as follow:

$$\textbf{nonexecutable } F \textbf{ if } \neg G$$

which stands for:

$$\textbf{caused } \bot \textbf{ if } \top \textbf{ after } F \land \neg G$$

## Effects of deterministic and non-deterministic actions

Direct effects $G$ of a deterministic action fluent $F$ can be expressed as follow:

$$F \textbf{ causes } G$$

which stands for:

$$\textbf{caused } G \textbf{ if } \top \textbf{ after } F$$

Direct effects $G$ of a non-deterministic action fluent $F$ can be expressed as follow:

$$F \textbf{ may cause } G$$

which stands for:

$$\textbf{caused } G \textbf{ if } G \textbf{ after } F$$

## Conditional effects of actions

Effects of an action can be “conditional” i.e. they may be caused by executing the action in some states, but not in others. Conditional effects $G$ of an action fluent $F$ under the conditions $H$ can be expressed as follow:

$$F \textbf{ causes } G \textbf{ if } H$$

which stands for:

$$\textbf{caused } G \textbf{ if } \top \textbf{ after } F \wedge H$$

The meaning of an action description can be described by a *transition system*—a directed graph whose nodes are characterized by the values of fluents and whose edges correspond to the actions that are executed. Table 3.1 summaries causal laws in action description language $\mathcal{C}+$.

Table 3.1: Causal laws and some useful abbreviations in the action description language $\mathcal{C}+$.

Static causal laws: **caused** $F$ **if** $G$

Dynamic causal laws: **caused** $F$ **if** $G$ **after** $H$

| Descriptions | Abbreviations | Causal Laws |
|---|---|---|
| Inertia holds for a property $F$ | **inertial** $F$ | **caused** $F$ **if** $F$ **after** $F$ |
| Exogeneity of actions an action $F$ | **exogenous** $F$ | **caused** $F$ **if** $F$ & **caused** $\neg F$ **if** $\neg F$ |
| No state satisfies property $F$ | **constraint** $\neg F$ | **caused** $\bot$ **if** $F$ |
| By default, property $F$ holds at every state | **default** $F$ | **caused** $F$ **if** $F$ |
| Action $F$ has preconditions $G$ | **nonexecutable** $F$ **if** $\neg G$ | **caused** $\bot$ **if** $\top$ **after** $F \wedge \neg G$ |
| Action $F$ has a direct effect $G$ | $F$ **causes** $G$ | **caused** $G$ **if** $\top$ **after** $F$ |
| Action $F$ has a direct effect $G$ under conditions $H$ | $F$ **causes** $G$ **if** $H$ | **caused** $G$ **if** $\top$ **after** $F \wedge H$ |
| Sensing action $F$ a has direct effect $G$ | $F$ **may cause** $G$ | **caused** $G$ **if** $G$ **after** $F$ |
| Sensing action $F$ has a direct effect $G$ under conditions $H$ | $F$ **may cause** $G$ **if** $H$ | **caused** $G$ **if** $G$ **after** $F \wedge H$ |

# Chapter IV

## 4 Hybrid Conditional Planning Framework

In this Chapter, we discuss our planning framework to solve the problem of planning under incomplete knowledge and partial observability. Motivated by service robotics applications where planning needs to be done under partial observability, we introduce a generic hybrid conditional planning approach. Our framework includes a parallel offline compilation-based hybrid conditional planner that requires domain description to be expressed in the language of $\mathcal{C}+$.

In Section 4.1, we describe the mathematical definition of a hybrid conditional plan. Section 4.2 provides formalization of actuation and sensing actions in action description language $\mathcal{C}+$. Section 4.3 discusses integration of feasibility checks in $\mathcal{C}+$. Section 4.4 discuss the planning problem to compute a hybrid sequential plan which serves as a single branch for hybrid conditional plans. Section 4.5 describes our algorithm to generate hybrid conditional plans and we conclude the chapter by discussing the novelties of our framework in Section 4.6.

## 4.1 Hybrid conditional plan

A hybrid conditional plan can be characterized as a labeled directed tree whose vertices, edges, labels are defined as follows. A sample tree that characterizes a hybrid conditional plan is presented in Figure 4.1.

### Vertices

The set $V = V_a \cup V_s$ of vertices consists of two types of vertices. The vertices in $V_a$ characterize hybrid actuation actions (e.g., the robot's move, pick, place, clean actions integrated with feasibility checks); these actions "change the state of the world" when they are executed (e.g., the location/status of the robot/object changes). The vertices in $V_s$ characterize sensing actions or information gathering actions in general (e.g., checking whether a plate is clean or not, asking whether the user wants soup or pizza, checking the location of an object); these actions do not change the state of the world, but only "mental states of the robot" when they are executed (e.g., the robot learns and knows that the plate is clean, and what the user wants). We call the vertices in $V_a$ as actuation vertices and the vertices in $V_s$ as sensing vertices. The leaves of the tree are in $V_a$.

### Edges

The set $E$ of edges between vertices in $V$ characterizes the order of actions: an edge $(x, y)$ expresses that the action denoted by the vertex $x$ is to be executed before the action denoted by $y$. Each vertex in $V_a$ has at most one outgoing edge based on the assumption that the actuation actions are deterministic. Each vertex in $V_s$ has at least two outgoing edges. Each sensing action may

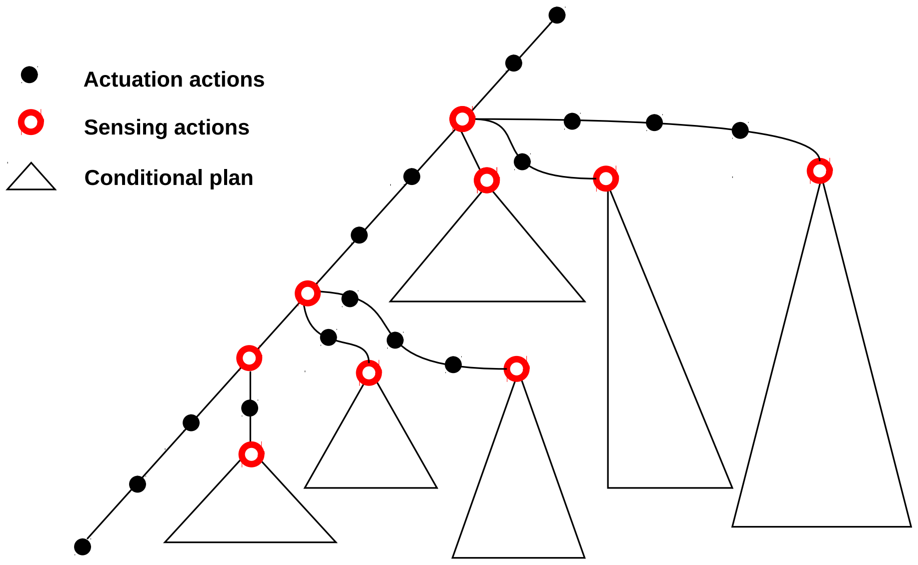


Figure 4.1: A sample hybrid conditional plan.

lead to different outcomes/observations (e.g., checking the cleanliness of a plate may lead to the observation that the plate is dirty or clean). Then, depending on the observation, each edge from a vertex in $V_s$ may lead to a different actuation action (e.g., if the plate is observed to be clean then the robot places it on the table; otherwise, it washes the plate).

## Labels

Let us denote by $E_s$ the set of outgoing edges from vertices in $V_s$. Then a labeling function maps every edge $(x, y)$ in $E_s$ by a possible outcome of the sensing action characterized by the vertex $x$.

## 4.2 Formal definition of actuation and sensing actions

Consider a robotic action domain, with a set of actuation actions and a set of sensing actions, where few properties are known while others are not known to the robot. For instance, in the kitchen table setting example, introduced in Section 1.1, the robot knows its location, whereas the locations of some objects in the kitchen and whether they are clean or dirty may be unknown to the robot. We describe the actuation and sensing actions by formulas in a non-monotonic formalism framework. The objective of the framework is to compute a sequential plan for a planning problem that contains both sorts of actions to reach the goal. In addition to the validity of the plan, the robotic actions need to be feasible to ensure smooth plan execution in real-world scenarios. We formalize both sorts of actions in action description language $\mathcal{C}+$ [64], where actions and change are described by causal laws.

### 4.2.1 Describing actuation actions

We assume that actuation actions are deterministic i.e. the outcome of an actuation action is known while planning. The preconditions and effects of the actuation actions can be described by formulas in $\mathcal{C}+$, as shown in Table 3.1. For instance, the effect of action $pick\_up(O1, M1)$ representing picking up an object $O1$ with a manipulator $M1$ can be described by the following formula:

$$\mathit{pick\_up}(\mathit{O1}, \mathit{M1}) \textbf{ causes } \mathit{objAt}(\mathit{O1}) = \mathit{M1}$$

which expresses that, after the robot picks up the object $O1$ with the manipulator $M1$, the new location of the object $O1$ becomes $M1$ which refers to robot manipulator hand. A precondition of this action, that the robot has to be near the location of the object $O1$ to grasp it, can be expressed by the formula:

$$\begin{array}{l}\textbf{nonexecutable } \mathit{pick\_up}(\mathit{O1}, \mathit{M1}) \\ \quad \textbf{if } \mathit{robAt} = \mathit{L1}, \mathit{objAt}(\mathit{O1}) = \mathit{L2}, \mathit{L1} \neq \mathit{L2}\end{array}$$

External computations (e.g., feasibility checks) can be embedded as executability conditions of actuation actions using external atoms as discussed in Section 4.3. Moreover, it is possible to model actuation actions as non-deterministic actions. This can be done by removing the assumption about their outcomes being deterministic and modeling them as non-deterministic actions similar to sensing actions discussed in Section 4.2.2.

### 4.2.2 Describing sensing actions

Now, let us consider some properties of the domain that are not fully observable. For instance, dirtiness/cleanliness of all objects may not be fully observable; so the robot is unaware of them. To learn about the cleanliness of an object, the robot has to inspect the object; but this may be possible only during execution of the conditional plan when the robot can navigate near it and manipulate the object. Likewise, the robot may not know the location of every object. To learn about the location of an object, it may need to search for it by looking at the possible locations of the object, but this is also possible only during the execution of the plan. Similarly, when the robot is about to set up the table for lunch, it may not know whether the person wants to have soup or pizza etc. Therefore, in order to learn about the person's wishes, it has to ask the person, once again during the execution of the plan. All the examples above are sensing actions that change the knowledge of the robot, but not the state of the world. We provide a formal way to describe these sensing actions using expressive of action language $\mathcal{C}+$. We represent the effect of sensing action as non-deterministic by using *may cause* construct provided by $\mathcal{C}+$. The preconditions and effects of a sensing action about checking the cleanliness of an object can be formalized as:

$$\begin{array}{l}\textbf{nonexecutable } \textit{check\_is\_clean}(\textit{T1}) \\ \quad \textbf{if } \textit{isClean}(\textit{T1}) \neq \textit{unkown} \\ \textit{check\_is\_clean}(\textit{T1}) \textbf{ may cause } \textit{isClean}(\textit{T1}) = \textit{C1} \\ \quad \textbf{if } \textit{isClean}(\textit{T1}) = \textit{unknown}\end{array}$$

We introduce a new construct *determines* in $\mathcal{C}+$ to formally represent sensing actions as non-deterministic action:

$$A \textbf{ determines } F \textbf{ if } G$$

where $A$ is a sensing action, $F$ is fluent whose value is *unknown* and can be determined by sensing action $A$, and $G$ are the preconditions of $A$. This construct stands for a following set of formulas:

$$\begin{array}{l}\textbf{nonexecutable } A \textbf{ if } \neg G \\ A \textbf{ may cause } F = v \ \textbf{ if } G\end{array}$$

where $v$ is a valid outcome of fluent $F$. Therefore, above sensing action for checking cleanliness of an object can be represented as follow:

$$\begin{array}{l}\textit{check\_is\_clean}(\textit{T1}) \textbf{ determines } \textit{isClean}(\textit{T1}) = \textit{C1} \\ \quad \textbf{if } \textit{isClean}(\textit{T1}) = \textit{unkown}\end{array}$$

and by execution of this sensing action $check\_is\_clean(T1)$ its outcome $C1$, either *yes* or *no*, is determined.

## 4.3 Feasibility checks integration

Hybrid planning approaches, where task planning is integrated with low-level feasibility checks [9, 10, 11, 12, 13, 14] ensure plan feasibility along with computation of valid plans. Planning framework of CCalc allows integration of external computations like feasibility checks in the formulas of the language of $\mathcal{C}+$. In order to describe the working and integration of these feasibility checks, let us consider an example where a bi-manual service robot needs to navigate around a kitchen environment and manipulate kitchenware. Suppose the computed plan involves picking up a bowl as shown in Figure 4.2. During the execution of this manipulation action, the robot needs to make sure that it does not collide with the environment and cause harm to itself or environment and grasps the bowl properly so it does not fall during execution etc. These situations are very common in robotics environments and handling such cases is critical to ensure feasible execution of plans. The idea is to compute a plan that only has feasible actions and ensuring this will result in the computed plan being valid and feasible in real-world applications.

CCalc provides a construct *where* to allow integration of external computations like feasibility checks in action domain descriptions. We integrate feasibility checks as preconditions of actions. For example, in example of picking up a bowl considered above, a precondition for $pick\_up$ action can

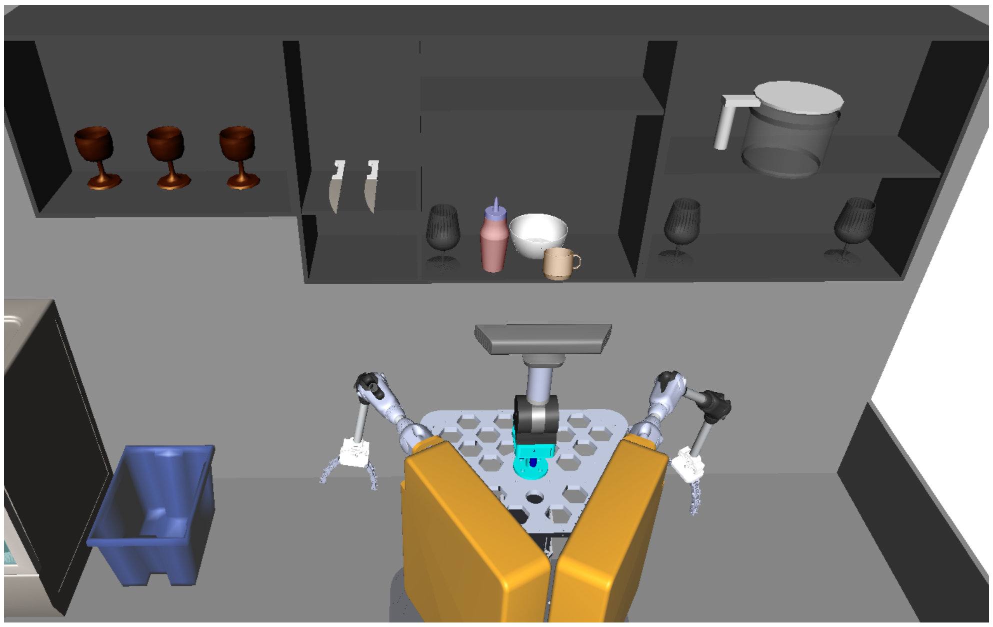


Figure 4.2: A bi-manual robots wants to pick up a bowl with one of its manipulator.

be added as:

$$\begin{aligned}&\textbf{nonexecutable } \textit{pick\_up}(\textit{T1}, \textit{M1})\\ &\quad \textbf{if } \textit{objAt}(\textit{T1}) = \textit{L1}, \textit{robAt} = \textit{L1}\\ &\quad \textbf{where } \textit{unPickable}(\textit{T1}, \textit{M1}, \textit{L1})\end{aligned}$$

where $T1$ is an object to grasp, $M1$ represents a robot manipulator, $L1$ is location of object, and *unPickable* is an external atom, which represents the outcome of some external computations (i.e. feasibility check for picking up an object in this case).

In order to perform these external computations (i.e. feasibility checks), we first model the environment and robot in simulation environment of OpenRAVE [77]. OpenRAVE provides an environment for testing, developing, and deploying motion planning algorithms in real-world robotics applications. The OpenRAVE environment can generate some useful databases for a robot and Table 4.1 gives a brief overview of these databases. The robot first checks for candidate grasps for an object (e.g bowl in this case), according to robot gripper and geometry of the object, using *grasping* database. Next, these candidate grasps are checked for collisions with the environment and remaining feasible grasps are noted. As the robot is mobile and needs to navigate while performing manipulation tasks, we need to find some suitable robot base configuration so that it can reach the object and utilize one of the feasible grasps for manipulation purposes. Therefore, some candidate collision-free robot base configurations are sampled and for each of these configurations, each of the feasible grasps is checked for a successful grasping of the bowl using a robotic manipulator. This is done by using *inversekinematics*, *convexdecomposition*, *inversereachability* and *kinematicreachability* databases provide by the OpenRAVE simulation environment. It is important to note that during this whole process, the robot needs to perform collision checks, reachability checks, and graspability checks. A successful solution consists of a valid feasible grasp, a robot base configuration, and a trajectory for a robot manipulator. These solutions are saved so that they can be utilized during plan execution by the robot. If a solution cannot be found in the given amount of time, the action is termed as infeasible (although a solution may still exist it was not found due to probabilistically complete nature of sample-based motion planners), and the outcome of these external computation returns "False". Thus, the robot knows that preconditions for that action were not met (i.e. the action is not feasible). Similarly, other objects are also checked for feasibility and the outcomes of these external computations are saved in a lookup table to be used by the robot later. It is important to mention here that all these computations are performed prior

Table 4.1: A summary of robot databases in OpenRAVE.

| Database | Description |
|---|---|
| convexdecomposition | Gives convex decomposition of link geometry of the robot. |
| grasping | Simulate grasping of objects and computing force closure metrics. |
| inversekinematics | Manages compiled inverse kinematics solutions for robots using ikfast. |
| inversereachability | Gives inverse reachability space of manipulators. |
| kinematicreachability | Gives 6D kinematic reachability space of a robot's manipulators. |

to the planning phase and robot utilizes them in the planning framework to compute plans that include feasible actions only and thus, ensure plan feasibility in real-world applications.

In order to emphasize the importance of these feasibility checks, let us consider a conditional plan with and without them. Since each branch of a conditional plan depicts a possible execution of actuation/sensing actions to reach a goal, it is essential that these actions are checked against relevant feasibility constraints in real-world applications. In robotics applications, these constraints are required for collision-free navigation and reachable/graspable manipulation, as depicted in Figure 4.3 and Figure 4.4 with two conditional plans computed for a robotics scenario, where two bi-manual mobile manipulators are responsible for setting up a kitchen table. It is quite obvious that the plans generated are different from each other. Also, it is important to note that none of the branches of the non-hybrid plan (Figure 4.3) is executable in real world and therefore whole conditional plan is infeasible, since the actuation actions (denoted red) are not feasible; whereas every branch of the hybrid plan (Figure 4.4) is feasible in the real world.

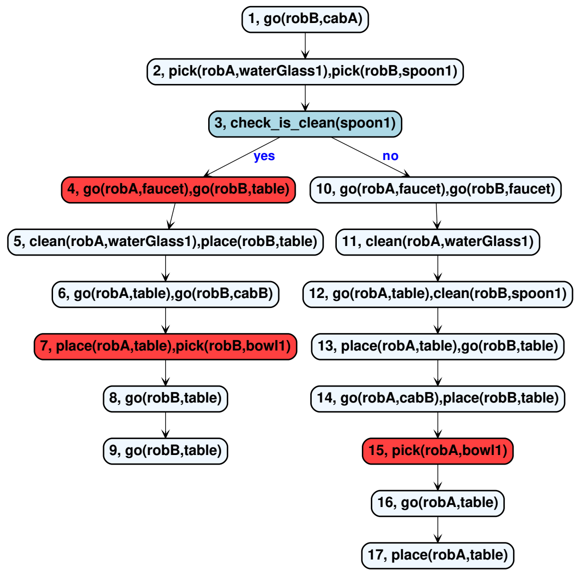

1, go(robB,cabA)
2, pick(robA,waterGlass1),pick(robB,spoon1)
3, check_is_clean(spoon1)
yes
no
4, go(robA,faucet),go(robB,table)
10, go(robA,faucet),go(robB,faucet)
5, clean(robA,waterGlass1),place(robB,table)
11, clean(robA,waterGlass1)
6, go(robA,table),go(robB,cabB)
12, go(robA,table),clean(robB,spoon1)
7, place(robA,table),pick(robB,bowl1)
13, place(robA,table),go(robB,table)
8, go(robB,table)
14, go(robA,cabB),place(robB,table)
9, go(robB,table)
15, pick(robA,bowl1)
16, go(robA,table)
17, place(robA,table)

Figure 4.3: An infeasible hybrid conditional plans generated without feasibility checks integration. Red boxes show infeasible actions.

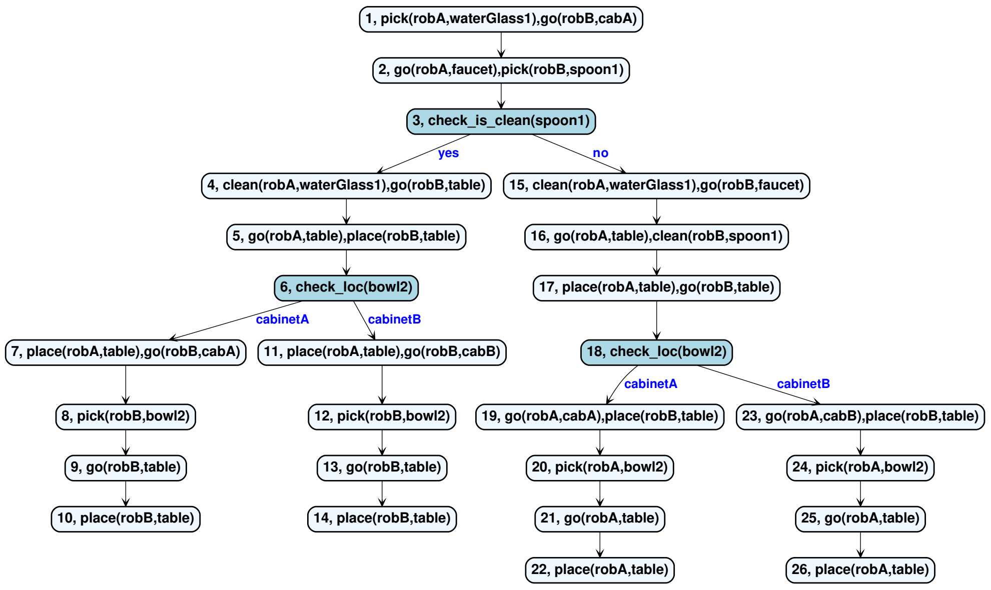

1, pick(robA,waterGlass1),go(robB,cabA)
2, go(robA,faucet),pick(robB,spoon1)
3, check_is_clean(spoon1)
yes
no
4, clean(robA,waterGlass1),go(robB,table)
15, clean(robA,waterGlass1),go(robB,faucet)
5, go(robA,table),place(robB,table)
16, go(robA,table),clean(robB,spoon1)
6, check_loc(bowl2)
17, place(robA,table),go(robB,table)
cabinetA
cabinetB
7, place(robA,table),go(robB,cabA)
11, place(robA,table),go(robB,cabB)
18, check_loc(bowl2)
cabinetA
cabinetB
8, pick(robB,bowl2)
12, pick(robB,bowl2)
19, go(robA,cabA),place(robB,table)
23, go(robA,cabB),place(robB,table)
9, go(robB,table)
13, go(robB,table)
20, pick(robA,bowl2)
24, pick(robA,bowl2)
10, place(robB,table)
14, place(robB,table)
21, go(robA,table)
25, go(robA,table)
22, place(robA,table)
26, place(robA,table)

Figure 4.4: A feasible hybrid conditional plans generated with feasibility checks integration.

## 4.4 Planning problem

Once the domain description containing all the actuation actions along with the integration of feasibility checks using external atoms and sensing actions have been formalized, we define a planning problem describing initial state and goal states, and ask the planner for a plan of length $k$. The initial state is a conjunction of fluents that may be known or unknown during planning. The keyword *unknown* is used to represent fluents which are not known. For example, in planning problem below, the location of plate and cleanliness of water glass is not known to the robot at the initial state.

$$\begin{aligned}
&\mathit{robAt} = \mathit{table}\ \textbf{holds at}\ 0 \wedge\\
&\mathit{objAt}(\mathit{waterGlass}) = \mathit{shelfA}\ \textbf{holds at}\ 0 \wedge\\
&\mathit{objAt}(\mathit{plate}) = \mathit{unknown}\ \textbf{holds at}\ 0 \wedge\\
&\mathit{isClean}(\mathit{plate}) = \mathit{yes}\ \textbf{holds at}\ 0 \wedge\\
&\mathit{isClean}(\mathit{waterGlass}) = \mathit{unknown}\ \textbf{holds at}\ 0 \wedge \ldots \wedge\\
&\mathit{tableSet}\ \textbf{holds at}\ k.
\end{aligned}$$

Using the planning framework of CCalc, we compute a hybrid sequential plan of length $k$, which consists of a sequence of actuation and sensing actions that reaches a goal. By iterating over $k = 0, 1, \ldots, \mathit{maxstep}$, CCalc guarantees to find the plan with the shortest length. Along with the computation of plan, it also reports plan history which contains state information at each step.

## 4.5 Computation of hybrid conditional plans

Once we have domain description and a planning problem, we can compute a hybrid sequential plan, that represents an action plan of actuation actions and sensing action to reach a goal state, as discussed in Section 4.4. Now, we discuss a conditional planning algorithm that can perform these computations in parallel to compute a hybrid conditional plan. A hybrid conditional plan is represented by a tree (each branch represents a hybrid sequential plan) where each branch from the root to leaves represents a unique way of reaching goals under different contingencies. Algorithm 1, 2, 3 and 4 summarizes our parallel conditional planning approach and in order to understand the notation used in algorithms, Table 4.2 describes the input symbols used in them.

**Algorithm 1** HYBRID_CONDITIONAL_PLAN($\mathcal{D}, \mathcal{P}, S$)

```
Input: See Table 4.2 for descriptions of input symbols above.
Output: A conditional plan, specified by its root root, if one exists; otherwise, failure.
   // Compute a hybrid sequential plan $P$ of length $n$ with history $H$.
1: $exists\_P, P, H \leftarrow$ HYBRID_SEQUENTIAL_PLAN($\mathcal{D}, \mathcal{P}$)
2: if not $exists\_P$ then
3:     return failure
   // Construct an initial branch $B$ of the tree from the root $root$ to leaf (represented by $P[0]$ and $P[n]$ respectively).
4: $root \leftarrow$ CREATE_BRANCH($P, H, S$)
   // Add new sub-branches to initial branch of the tree, in parallel.
5: $root \leftarrow$ TRAVERSE&GROW_TREE($\mathcal{D}, \mathcal{P}, root, H, S, \mathcal{K}$)
6: return $root$
```

Our algorithm first computes a hybrid sequential plan by assigning nondeterministic outcomes for sensing actions and this plan serves as the initial branch for the hybrid conditional plan. Next, for all the sensing actions

Table 4.2: Input symbols used in algorithms.

| Symbols | Description |
|---|---|
| $\mathcal{D}$ | Robotic action domain description |
| $\mathcal{P}$ | Planning problem with an initial state and goals |
| $S$ | Set of all possible sensing actions in $\mathcal{D}$, each sensing action $s_A$ tupled with a set $O_{s_A}$ |
| $O_{s_A}$ | Set of all possible outcomes of a sensing action $s_A$ |
| $S_p$ | Set of all possible sensing actions in $\mathcal{D}$, each sensing action $s_A$ tupled with a set $P_{s_A}$ |
| $P_{s_A}$ | Set of tuples $\langle o, p \rangle$ where $o$ is outcome while $p$ represents probability for each outcome of a sensing action $s_A$ |
| $P$ | Hybrid sequential plan of length $n$ ($P[i]$ denotes the $i$'th action) |
| $H$ | History $H$ of hybrid sequential plan $P$ that contains information about each state ($H[i]$ represents information about state when the action $P[i]$ occurs) |
| $B$ | Branch or sub-branch of a hybrid conditional plan |
| $I$ | Initial state of a planning problem $\mathcal{P}$ |
| $root$ | Root of a plan |
| $\mathcal{K}$ | Lookup table with each entry of form $\langle node, o, plan \rangle$ where $node$ is a sensing node, $o$ is outcome of $node$ and $plan$ represents root node of computed/saved sub-branch $B$ |
| $\mathcal{Q}$ | Queue of tasks in which each task is given as a tuple $\langle node, o \rangle$ where $node$ is a sensing node and $o$ represents outcome of $node$ for which new plan needs to be computed |
| $\mathcal{Q}_{\mathcal{P}}$ | Priority queue of tasks in which each task is given as a tuple $(p, \langle node, o \rangle)$ where $node$ is a sensing node, $o$ represents outcome of $node$ for which new sub-branch $B$ needs to be computed and $p$ represents probability of the $root$ of $B$ |
| $\%T_C$ | Percentage that represents current computation level of a hybrid conditional plan using HCPlan-Anytime |
| $\%T_D$ | Percentage that represents threshold level for computation of a hybrid conditional plan using HCPlan-Anytime |
| $D_{th}$ | is threshold for depth until which all sensing nodes will be expanded from the current node in each itteration of planning during computation of a hybrid conditional plan using HCPlan-Reactive |
| $B_{exec}$ | is the branch executed by the robot in computation of a partial hybrid conditional plan using HCPlan-Reactive |

**Algorithm 2** CREATE_BRANCH$(P, H, S)$

**Input:** See Table 4.2 for descriptions of input symbols above.

**Output:** A branch or sub-branch $B$ of the hybrid conditional plan denoted by its root node *root*.

```
 1: for $i = 0 \dots n-1$ do
 2:     initialize $node$                          ▷ create a node for $P[i]$
 3:     if $i \neq n-1$ then
 4:         initialize $child$                     ▷ create a node for $P[i+1]$
 5:         $node.children.append(child)$          ▷ link parent to child
 6:     if $P[i] \in S$ then                       ▷ is node a sensing action?
 7:         $node.sensing = True$
 8:     if $node.parent.sensing = True$ then       ▷ is parent a sensing action?
 9:         $s_A = node.parent$
10:         $o_s \leftarrow$ current outcome of $s_A$ in $H[i] \cap O_{s_A}$
11:         $node.edge\_label = o_s$               ▷ label the edge
12:     if $i == 0$ then                           ▷ is node root?
13:         $root = node$
14:     $node.depth = node.parent.depth + 1$
15:     $node.state = H[i]$
16: return $root$
```

**Algorithm 3** TRAVERSE&GROW_TREE($\mathcal{D}, \mathcal{P}, root, H, S, \mathcal{K}$)

**Input:** See Table 4.2 for descriptions of input symbols above.

**Output:** A conditional plan in the form of a tree, specified by its root *root*.

```
 1: initialize $\mathcal{Q}$                                    ▷ create a queue of tasks
 2: $\mathcal{Q} \leftarrow$ UPDATE_QUEUE($root, \mathcal{Q}, \mathcal{K}$)                    ▷ update queue
 3: while not $\mathcal{Q}$.empty do
 4:     do in parallel for $\langle node, o\rangle \leftarrow dequeue(\mathcal{Q})$
 5:         $I = node.state$                              ▷ get state information
 6:         $I' \leftarrow$ modify $I$ according to new outcome $o$
 7:         $\mathcal{P}' \leftarrow$ modify $\mathcal{P}$ with new initial state $I'$
 8:         $exists\_P, P, H \leftarrow$ HYBRID_SEQUENTIAL_PLAN($\mathcal{D}, \mathcal{P}'$)
 9:         if $exists\_P$ then
10:             $new\_child \leftarrow$ CREATE_BRANCH($P, H, S$)
11:             $new\_child.edge\_label = o$
12:             $\mathcal{Q} \leftarrow$ UPDATE_QUEUE($new\_child, \mathcal{Q}, \mathcal{K}$)
13:             $node.children.append(new\_child)$
14:             Add ($\langle node, o, new\_child\rangle$) to $\mathcal{K}$
15: return $root$
```

**Algorithm 4** UPDATE_QUEUE($root, \mathcal{Q}, \mathcal{K}$)

**Input:** See Table 4.2 for descriptions of input symbols above.

**Output:** Updated task queue $\mathcal{Q}$.

```
 1: $node = root$
 2: while $node.children$ do
 3:     if $node.sensing = True$ then                ▷ if node is a sensing action?
 4:         $s_A = node.name$
 5:         $o_A = node.children.edge\_label$     ▷ get outcome of sensing action
 6:         for $o = O_{s_A} \setminus \{o_A\}$ do                    ▷ for all other outcomes
 7:             if $\langle node, o\rangle \notin \mathcal{K}$ then                    ▷ if not in lookup table
 8:                 $enqueue(\mathcal{Q}, \langle node, o\rangle)$              ▷ Update $\mathcal{Q}$ with new task
 9:     $node = node.children$
10: return $\mathcal{Q}$
```

in this branch, the other valid possible outcomes of each sensing action are identified and appended as tuples in a task queue to be computed. Now, for each of these tasks, new hybrid sequential plans are computed in parallel and appended as branches/sub-branches to the initial branch of the hybrid conditional plan with proper labels. Each of these branches is further checked for new tasks. The process continues until we get a hybrid conditional plan which is complete with respect to all the contingencies that were considered during its computation. The computed hybrid conditional plan is represented as a tree of sensing and actuation actions.

In the end, if the computed hybrid conditional plan has a maximum branching factor $b$ and the maximum depth $d$, the conditional plan has at most $b^d$ leaves. Therefore, our hybrid conditional planner calls the hybrid sequential planner CCalc at most $b^d$ times. The planning framework is generic and applicable to any action domain description involving actuation and sensing actions.

## 4.6 Novelties in our framework

The novelties of our hybrid conditional planning framework are due to how we address the problems taking advantage of the expressive formalisms. Instead of the PDDL-based classical planners [78, 79, 80] used in the related work, we use an action language based planner CCalc and capitalize on the expressiveness of this planner.

### Defaults to represent occurrences of sensing actions

One can express the defaults in the input language of CCalc since it is based on non-monotonic logic. Defaults are useful, for instance, when the agents do not have complete information about where the objects are exactly but have some commonsense knowledge about their whereabouts. For instance, a robot may not know where every single book in the house is but it may know that "by default, books are in the bookcase." If later the robot finds a book on the table, then this observation does not cause an inconsistency with its knowledge base. Instead, this observation is treated as an exception to what is expected by default. Note that such exceptions cannot be handled in formalisms/planners based on monotonic logic, such as PDDL.

We utilize defaults for conditional planning by explicitly formalizing that "by default, no sensing is done." In this way, the computation of a plan includes sensing actions as needed. If the robot, later on, performs a relevant sensing action in the plan, then the robot knows that the "sensing is performed" and no inconsistency is caused. In this way, the robot does not have to decide in advance the order of sensing actions and computes a single plan that consists of actuation and sensing actions.

### Modeling sensing actions with non-deterministic effects

Sensing actions have non-deterministic outcomes and it is difficult to formalize non-deterministic actions in many planning languages. Due to this difficulty, instead of formalizing sensing actions explicitly as non-deterministic actions, the related work computes conformant plans of actuation actions between every two closest sensing actions on a branch of a conditional plan.

We model sensing actions as non-deterministic exogenous actions in the input language of CCalc. We also model preconditions of sensing actions (e.g., a robot can check whether a plate is clean or dirty only if it is holding the plate close by), and explicitly represent what is unknown (e.g., it is unknown that the plate is clean). When a possible outcome of a sensing action is non-deterministically determined, what is unknown becomes known to the robot. In this way, branches of a conditional plan (i.e., sequences of actuation actions and sensing actions) from an (incomplete) state to a goal state can be computed using CCalc; and the planner does not have to solve conformant planning problems to construct a conditional plan.

### Hybrid planning

Unlike the other conditional planning approaches in literature except [7, 56], our approach is hybrid in the spirit of [9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]: motivated by robotics applications, our approach embeds feasibility checks (e.g., collision checks, reachability checks, graspability checks) into conditional planning. We explicitly embed feasibility checks into descriptions of actuation/sensing actions, thanks to external atoms. In this way, infeasible conditional plans (i.e., conditional plans where some actuation actions cannot be physically executed according to the feasibility checks) are eliminated as early as possible.

### Parallel computation

Our conditional planning algorithm incrementally builds a conditional plan by computing its branches and attaching the sister branches in parallel. None of the other conditional planning approaches except [7] computes conditional plan in a parallel manner.

### Plan Re-use

Our conditional planning algorithm saves each solution of a query to the CCalc planner in a database and utilizes plan re-use to reduce computational work and planning time.

# Chapter V

# 5 Anytime and Reactive Hybrid Conditional Planning

In Chapter 4, we discussed the hybrid conditional planning framework and how it can be used to compute hybrid conditional plans. In this chapter, we go further and discuss the variations of HCPlan. Hybrid conditional plans are computed offline and can typically take considerable computational time even with the utilization of parallel processing of branches. In Section 5.1, we introduce a probabilistic framework for hybrid conditional planning, HCPlan-Anytime. In Section 5.2, we introduce HCPlan-Reactive which is a semi-online variation of HCPlan that utilizes beneficial features from plan execution monitoring.

## 5.1 Anytime hybrid conditional planning

The planner HCPlan introduced in Chapter 4 uses conditional planning with the integration of feasibility checks to generate hybrid conditional plans (trees) of sensing and actuation actions. Hybrid conditional plans consider all possible contingencies at the initial state and computed plans that are complete with respect to the contingencies considered during planning [6, 7]. In applications where the number of contingencies becomes really large, computation of such complete hybrid conditional plans can take a considerable amount of time. One solution to this problem can be to compute partial conditional plans that consider not all, but a limited number of contingencies for plan computation. However, this should be done in a way that the most relevant contingencies are kept while contingencies which are least probable are removed. Therefore, by removing computation of less probable branches, the planning time needed for computation of plan can be reduced. It is important to note that plans computed in this way consider a reduced number of offline contingencies and in case of a contingency not considered during planning, some kind of recovery mechanism such as execution monitoring has to be implemented. The idea is to reduced offline planning time for computation of conditional plans and in case of plan failure during execution, they can be addressed with online re-planning.

We introduce a novel hybrid conditional planner HCPlan-Anytime that computes partial hybrid conditional plans based on the idea discussed above. Another useful feature of the planner is that it can be interrupted at any given time, and it will still return a valid partial conditional plan that had been computed until then. This can be very beneficial for some service robotic applications, where plan execution and planning can be performed simultaneously and failures during the execution can be monitored with re-planning.

*Anytime algorithms* are defined as algorithms that return some answer for any allocation of computation time, and are expected to return better answers when given more time [81, 82, 83]. In that spirit, HCPlan-Anytime is anytime. Figure 5.1 shows generation of a partial hybrid conditional plan computed using HCPlan-Anytime.

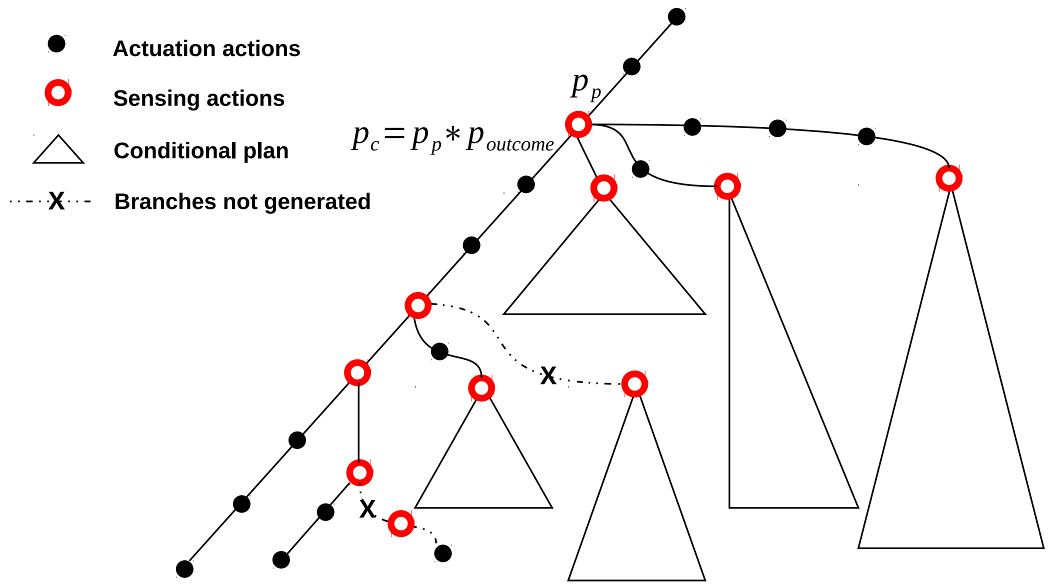


Figure 5.1: Generation of a partial hybrid conditional plan using HCPlan-Anytime.

**Algorithm 5** HYBRID_CONDITIONAL_PLAN_ANYTIME($\mathcal{D}, \mathcal{P}, S_P, \%T_D$)

**Input:** See Table 4.2 for descriptions of input symbols above.

**Output:** A partial conditional plan, specified by its root *root*, if one exists; otherwise, failure.

```
   // Compute a hybrid sequential plan $P$ of length $n$ with history $H$.
1: $exists\_P, P, H \leftarrow$ HYBRID_SEQUENTIAL_PLAN($\mathcal{D}, \mathcal{P}$)
2: if not $exists\_P$ then
3:     return failure
   // Construct an initial branch $B$ of the tree from the root $root$ to leaf
   (represented by $P[0]$ and $P[n]$ respectively).
4: $root \leftarrow$ CREATE_BRANCH_ANYTIME($P, H, S_P$)
   // Add new sub-branches to initial branch of the tree, in parallel.
5: $root \leftarrow$ GROW_TREE_ANYTIME($\mathcal{D}, \mathcal{P}, root, S_p, \mathcal{K}, \%T_D$)
6: return $root$
```

**Algorithm 6** CREATE_BRANCH_ANYTIME$(P, H, S_P)$

**Input:** See Table 4.2 for descriptions of input symbols above.

**Output:** A branch or sub-branch $B$ of the hybrid conditional plan denoted by its root node *root*.

```
 1: for $i = 0 \dots n-1$ do
 2:     initialize $node$                                        ▷ create a node for $P[i]$
 3:     if $i \neq n-1$ then
 4:         initialize $child$                                   ▷ create a node for $P[i+1]$
 5:         $node.children.append(child)$                        ▷ link parent to child
 6:         $node.probability = node.parent.probability$
 7:     if $P[i] \in S_P$ then                                   ▷ is node a sensing action?
 8:         $node.sensing = True$
 9:     if $node.parent.sensing = True$ then                     ▷ is parent a sensing action?
10:         $s_A = node.parent$
11:         $o_s \leftarrow$ current outcome of $s_A$ in $H[i] \cap P_{s_A}$
12:         $p_{o_s} \leftarrow$ probability of $o_s$ in $P_{s_A}$          ▷ get probability of outcome
13:         $node.edge\_label = o_s$                              ▷ label the edge
14:         $node.probability = node.parent.probability * p_{o_s}$    ▷ update rule
15:     if $i == 0$ then                                         ▷ is node root?
16:         $root = node$
17:     $node.depth = node.parent.depth + 1$
18:     $node.state = H[i]$
19: return $root$
```

**Algorithm 7** GROW_TREE_ANYTIME($\mathcal{D}, \mathcal{P}, root, S_p, \mathcal{K}, \%T_D$)

**Input:** See Table 4.2 for descriptions of input symbols above.

**Output:** A conditional plan in the form of a tree, specified by its root *root*.

```
 1: $\%T_C = root.get\_leaf().probability$                    ▷ initialize $\%T_C$
 2: initialize $\mathcal{Q}_\mathcal{P}$                      ▷ create a priority queue of tasks
 3: $\mathcal{Q}_\mathcal{P} \leftarrow$ UPDATE_QUEUE_ANYTIME($root, \mathcal{Q}_\mathcal{P}, \mathcal{K}$)
 4: while not $\mathcal{Q}_\mathcal{P}$.empty do
 5:     do in parallel for $(p, \langle node, o \rangle) \leftarrow dequeue(\mathcal{Q}_\mathcal{P})$
 6:         $I = node.state$                                 ▷ get state information
 7:         $I' \leftarrow$ modify $I$ according to new outcome $o$
 8:         $\mathcal{P}' \leftarrow$ modify $\mathcal{P}$ with new initial state $I'$
 9:         $exists\_P, P, H \leftarrow$ HYBRID_SEQUENTIAL_PLAN($\mathcal{D}, \mathcal{P}'$)
10:         if $exists\_P$ then
11:             Set probability of $P[0] = p$
12:             $new\_child \leftarrow$ CREATE_BRANCH_ANYTIME($P, H, S_p$)
13:             $new\_child.edge\_label = o$
14:             Add $\langle node, o, new\_child \rangle$ to $\mathcal{K}$
15:             $\mathcal{Q}_\mathcal{P} \leftarrow$ UPDATE_QUEUE_ANYTIME($new\_child, \mathcal{Q}_\mathcal{P}, \mathcal{K}$)
16:             $node.children.append(new\_child)$
17:             $\%T_C$ += $new\_child.get\_leaf().probability$    ▷ update $\%T_C$
18:         if $\%T_C \geq \%T_D$ then                      ▷ is %tree desired reached?
19:             return $root$
20:         if $interruption == True$ then                  ▷ if interruption occurs?
21:             return $root$
22: return $root$
```

**Algorithm 8** UPDATE_QUEUE_ANYTIME($root, \mathcal{Q}_{\mathcal{P}}, \mathcal{K}$)

```
Input: See Table 4.2 for descriptions of input symbols above.
Output: Updated task priority queue $\mathcal{Q}_{\mathcal{P}}$.
1:  $node = root$
2:  while $node.children$ do
3:      if $node.sensing = True$ then                  ▷ if node is a sensing action?
4:          $s_A = node.name$
5:          $o_A = node.children.edge\_label$          ▷ get outcome of sensing action
6:          $p_A = node.probability$                   ▷ get probability of node
7:          for $o = O_{s_A} \setminus \{o_A\}$ do      ▷ for all other outcomes
8:              $p_o \leftarrow$ probability of $o$ in $P_{s_A}$   ▷ get probability of outcome
9:              $p = p_A * p_o$                         ▷ update probability
10:             if $\langle node, o \rangle \notin \mathcal{K}$ then   ▷ if not in lookup table
11:                 $enqueue(\mathcal{Q}_{\mathcal{P}},(p, \langle node, o \rangle))$   ▷ Update $\mathcal{Q}_{\mathcal{P}}$ with new task
12:     $node = node.children$
13: return $\mathcal{Q}_{\mathcal{P}}$
```

Algorithms 5, 6, 7, and 8 describe the working of HCPlan-Anytime to compute partial hybrid conditional plans. It utilizes probabilities in addition to outcomes of sensing actions in order to compute partial plans. Normally, the probability of a parent and child is same, except in case of a parent being a sensing action where following probability rule is applied:

$$p_{child} = p_{parent} * p_{outcome}$$

where $p_{child}$ is the probability of child, $p_{parent}$ is the probability of parent, and $p_{outcome}$ is the probability for the current outcome of sensing action that parent represents. A priority queue is utilized to give priority to the computation of more probable branches. $\%T_D$ represents an upper bound for the percentage of partial plan to be computed while $\%T_C$ represents actual percentage of plan computed. As the planner is anytime, it can be interrupted at any instance and will give a valid partial plan computed until that time.

Note that if outcomes of all sensing actions are equiprobable and a complete plan is computed using HCPlan-Anytime, the resultant hybrid conditional plan is similar to that obtained from HCPlan.

## 5.2 Reactive hybrid conditional planning

Online conditional planners use online sensing to compute plans, therefore they do not need to handle a potentially exponential number of contingencies during the planning phase. This typically results in less computation and planning time compared to offline conditional planning approaches. However, since a limited number of contingencies are considered, they may suffer from failures to reach the goal state. Offline conditional plan, on the other hand, constructs a plan which is complete with respect to contingencies considered during planning phase [6, 7]. These computations of conditional plans can, however, take a significant amount of time and plan execution cannot start until the plans have been computed.

We propose a novel semi-online extension to our hybrid conditional planner HCPlan that computes partial conditional plans until a predefined depth. We call the new semi-online hybrid conditional planner HCPlan-Reactive. The planner takes care of not only planning but also the execution of plans. The execution of a plan can start as soon as the first partial plan is computed and failures during execution can be addressed with re-planning. This approach reduces the offline planning time by computing partial plans and therefore, execution can start earlier. The computation of a partial plan with HCPlan-Reactive is shown in Figure 5.2.

Algorithms 9, 10, 11, 12, and 13 describes the generation and execution of partial plans using hybrid conditional reactive planner HCPlan-

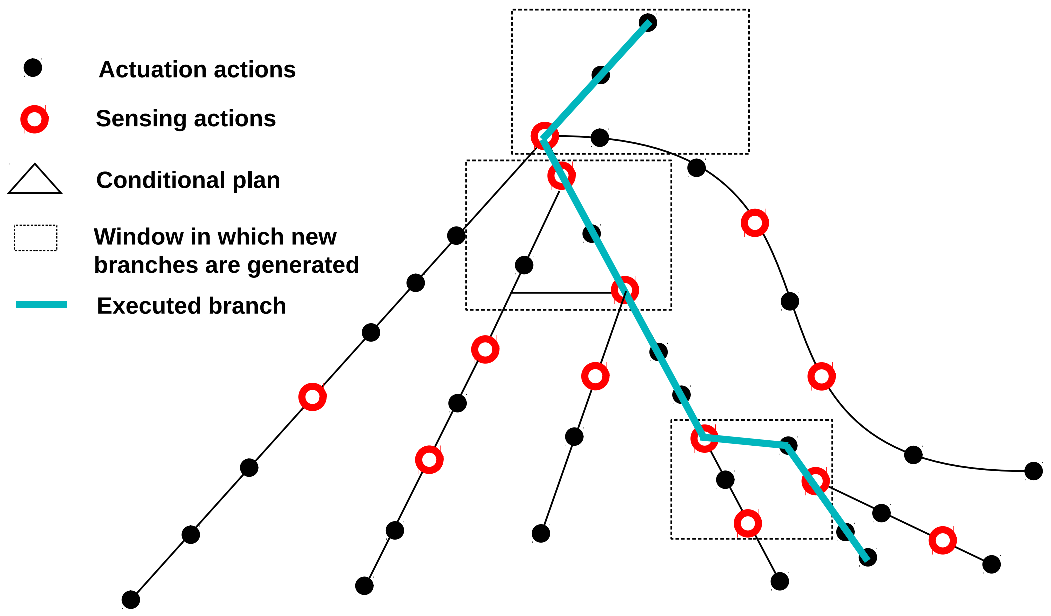


Figure 5.2: Generation of a partial hybrid conditional plan using HCPlan-Reactive.

Reactive. Note that HCPlan-Reactive is a semi-online hybrid parallel conditional planner that resembles HCPlan and plan execution monitoring. If the reactive depth threshold for plan computation is set to a number equal or more than the length of maximum branch plan computed with HCPlan, than HCPlan-Reactive behaves like a complete offline planner similar to HCPlan and the tree computed will be similar to the one generated by HCPlan. On the other hand, if the reactive depth threshold is set to one, the planner tends to behave more like an online planner and resembles the plan execution monitoring approach. However, the plan will still be a tree of sensing and actuation actions unlike the one returned by plan execution monitoring.

**Algorithm 9** PLAN&EXECUTE_HCPLAN_REACTIVE($\mathcal{D}, \mathcal{P}, S, D_{th}$)

```
Input: See Table 4.2 for descriptions of input symbols above.
Output: A partial conditional plan, specified by its root $root$ and branch
    executed by the robot in plan $B_{exec}$.
    // A partial conditional plan expanded till reactive depth $D_{th}$, specified
    by its root $root$.
1: $root \leftarrow$ HYBRID_CONDITIONAL_PLAN_REACTIVE($\mathcal{D}, \mathcal{P}, S, D_{th}$)
    // execute the plan and in case of unknown outcome of sensing, re-plan.
2: $root, B_{exec} \leftarrow$ EXECUTE_PLAN_REACTIVE($root$)
3: return $root, B_{exec}$
```

**Algorithm 10** HYBRID_CONDITIONAL_PLAN_REACTIVE($\mathcal{D}, \mathcal{P}, S, D_{th}$)

```
Input: See Table 4.2 for descriptions of input symbols above.
Output: A partial conditional plan, specified by its root $root$, if one exists;
    otherwise, failure.
    // Compute a hybrid sequential plan $P$ of length $n$ with history $H$.
1: $exists\_P, P, H \leftarrow$ HYBRID_SEQUENTIAL_PLAN($\mathcal{D}, \mathcal{P}$)
2: if not $exists\_P$ then
3:     return failure
    // Construct an initial branch $B$ of the tree from the root $root$ to leaf
    (represented by $P[0]$ and $P[n]$ respectively).
4: $root \leftarrow$ CREATE_BRANCH($P, H, S$)
    // Add new sub-branches to initial branch of the tree, in parallel.
5: $root \leftarrow$ GROW_TREE_REACTIVE($\mathcal{D}, \mathcal{P}, root, S, \mathcal{K}, D_{th}$)
6: return $root$
```

**Algorithm 11** GROW_TREE_REACTIVE($\mathcal{D}, \mathcal{P}, root, S, \mathcal{K}, D_{th}$)

**Input:** See Table 4.2 for descriptions of input symbols above.

**Output:** A conditional plan in the form of a tree, specified by its root *root*.

```
 1: initialize $\mathcal{Q}$                                        ▷ create a queue of tasks
 2: $\mathcal{Q} \leftarrow$ UPDATE_QUEUE_REACTIVE($root, \mathcal{Q}, \mathcal{K}, D_{th}$)       ▷ update queue
 3: while not $\mathcal{Q}$.empty do
 4:     do in parallel for $\langle node, o\rangle \leftarrow dequeue(\mathcal{Q})$
 5:         $I = node.state$                                    ▷ get state information
 6:         $I' \leftarrow$ modify $I$ according to new outcome $o$
 7:         $\mathcal{P}' \leftarrow$ modify $\mathcal{P}$ with new initial state $I'$
 8:         $exists\_P, P, H \leftarrow$ HYBRID_SEQUENTIAL_PLAN($\mathcal{D}, \mathcal{P}'$)
 9:         if $exists\_P$ then
10:             $new\_child \leftarrow$ CREATE_BRANCH($P, H, S$)
11:             $new\_child.edge\_label = o$
12:             $\mathcal{Q} \leftarrow$ UPDATE_QUEUE_REACTIVE($new\_child, \mathcal{Q}, \mathcal{K}, D_{th}$)
13:             $node.children.append(new\_child)$
14:             Add ($\langle node, o, new\_child\rangle$) to $\mathcal{K}$
15: return $root$
```

**Algorithm 12** UPDATE_QUEUE_REACTIVE($root, \mathcal{Q}, \mathcal{K}, D_{th}$)

**Input:** See Table 4.2 for descriptions of input symbols above.

**Output:** Updated task queue $\mathcal{Q}$.

```
 1: $node = root$
 2: while $node.children$ & $node.depth \leq D_{th}$ do          ▷ check till $D_{th}$ only
 3:     if $node.sensing = True$ then                 ▷ if node is a sensing action?
 4:         $s_A = node.name$
 5:         $o_A = node.children.edge\_label$    ▷ get outcome of sensing action
 6:         for $o = O_{s_A} \setminus \{o_A\}$ do                      ▷ for all other outcomes
 7:             if $\langle node, o\rangle \notin \mathcal{K}$ then                     ▷ if not in lookup table
 8:                 $enqueue(\mathcal{Q}, \langle node, o\rangle)$              ▷ update $\mathcal{Q}$ with new task
 9:     $node = node.children$
10: return $\mathcal{Q}$
```

**Algorithm 13** EXECUTE_PLAN_REACTIVE($root$)

**Input:** See Table 4.2 for descriptions of input symbols above.

**Output:** A partial conditional plan, specified by its root $root$ and branch executed by the robot in plan $B_{exec}$.

```
 1: $node = root$
 2: while $node.children$ do
 3:     if not $node.sensing$ then                         ▷ is an actuation action?
 4:         EXECUTE_ACTION($node$)
 5:     else                                               ▷ is node a sensing action?
 6:         $o_A \leftarrow$ SENSE_OUTCOME($node$)
 7:         if $o_A \notin node.children.edge\_labels$ then            ▷ is unknown?
 8:             $D_{th}$ += $node.depth$                    ▷ modify reactive depth $D_{th}$
 9:             $node \leftarrow$ GROW_TREE_REACTIVE($\mathcal{D}, \mathcal{P}, node, S, \mathcal{K}, D_{th}$)
10:             $node = node.get\_child(o_A)$          ▷ get child with outcome $o_A$
11:         else
12:             $node = node.get\_child(o_A)$
13:         EXECUTE_ACTION($node$)
14: $B_{exec} = node.get\_path()$                              ▷ get executed branch
15: return $root, B_{exec}$
```

# Chapter VI

## 6 Benchmark Domains

In order to demonstrate the feasibility of our approach, we consider three complex service robotics domains that involve manipulation and navigation tasks. In Section 6.1, we consider a mobile manipulation benchmark, followed by a navigation and manipulation benchmark, in Section 6.2 and Section 6.3 respectively, that serve as the platform to evaluate our hybrid conditional framework and compare it with other approaches in the literature.

### 6.1 Mobile manipulation benchmark

In a mobile manipulation benchmark, we consider a service robot that can navigate around the environment performing manipulation tasks. To demonstrate such a scenario, we consider a bi-manual mobile manipulator that is responsible for setting up a kitchen table, as depicted in Figure 6.1. The mobile manipulator can navigate around the kitchen to pick up and place objects as long as collision-free trajectories exist. Kitchenware, such as mugs, spoons, knives, plates may be found in the cabinets or may be left on other flat surfaces, such as the countertops or shelves. In the kitchen, there also exists a faucet to clean the kitchenware as required. Finally, there is a kitchen table, where the proper kitchenware must be placed on to comply with table setting etiquette.

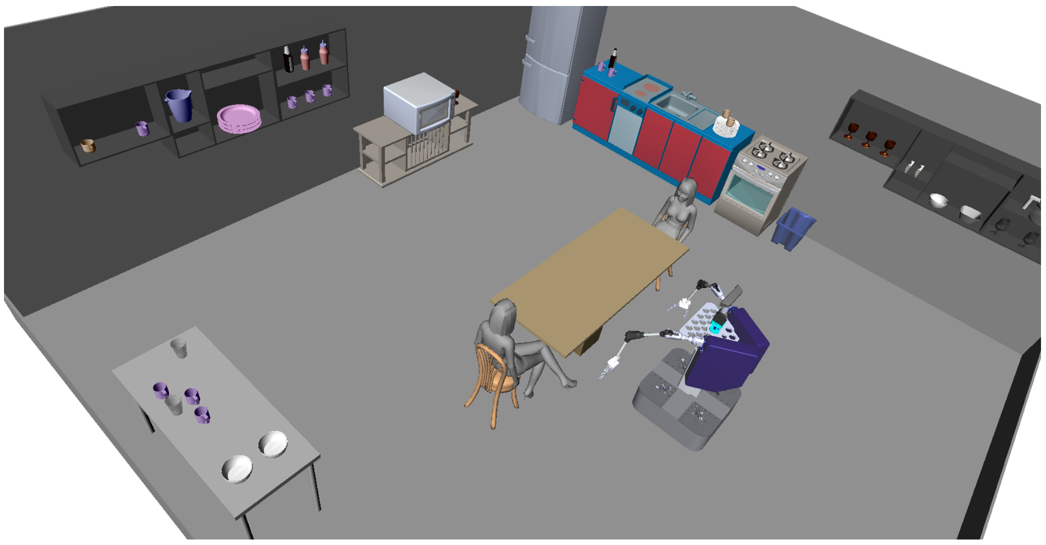


Figure 6.1: An image showing a kitchen environment where a mobile manipulation is trying to set up the kitchen table.

For the kitchen table set up benchmark, four actuation actions are considered in the domain: *goto*, *pick_up*, *place_on* and *clean*. Note that, in hybrid planning, the feasibility of these actions needs to be checked. A probabilistic motion planner (based on OMPL [84]) is used to implement the precondition of *goto* action, while reachability, graspability and collision checks (based on OpenRAVE [77]) are implemented as the preconditions of *pick_up* and *place_on* actions.

Note that the environment is not completely observable during the planning. Three types of possible sources of uncertainties are considered in this domain. First, the person might have different food preferences (e.g., soup, pizza, salad), which can only be revealed when directly communicated with the user during the plan execution; that is, this information is not available for the planning ahead of time. This uncertainty directly affects the plan, as the kitchenware to be placed on table varies based on the type of the meal

Table 6.1: Kitchen table setting benchmark domain description.

| **Fluents** | **Description** |
|---|---|
| *foodIs* | represents the food requested |
| *robotAt* | represents a robot location |
| *objectAt* | represents an object location |
| *isClean* | represents if an object is clean or not |

| **Sensing actions** | **Description** |
|---|---|
| *check_food_type* | checks the type of food required |
| *check_loc* | checks the location of an object |
| *check_is_clean* | checks if an object is clean or not |

| **Actuation actions** | **Description** | **Feasibility checks** |
|---|---|---|
| *goto* | robot navigates to a location | collision |
| *pick_up* | robot picks object in hand | collision, reachability, graspability |
| *place_on* | robot places object at a location | collision, reachability |
| *clean* | robot cleans object in hand | collision |

(e.g., spoon and bowl are required for having soup, while they are irrelevant for eating pizza). Second, the locations of the kitchenware are uncertain and might not be known by the robot during the planning phase. These locations can be reliably gathered only if the robot actively searches for these objects when it needs to use them. Third, the cleanliness/dirtiness of the objects may not be known for sure.

Along these lines, three sensing actions for information gathering are considered in the domain: *check_food_type*, *check_loc* and *check_is_clean*. The first action *check_food_type* is used to determine the type of food the user desires. The sensing action *check_loc* is utilized to resolve the uncertainty of the locations of kitchenware. Finally, *check_is_clean* is introduced to determine the cleanliness of kitchenware. Table 6.1 shows the kitchen table setting benchmark domain description.

In order to perform an experimental evaluation of this benchmark domain, we created 25 benchmark instances by varying the initial setting for possible uncertainty about the environment. In Table 6.1, we show that the initial state consists of four types of fluents. $foodIs$ represent the requested food and maybe unknown at the initial state. We consider three possible values for this fluent. $robotAt$ represent the robot location and is known at all the states. $objectAt$ represent the location of objects out of five possible locations and $isClean$ is a boolean fluent that represent the objects cleanliness. Both of these fluents may be unknown at the initial state. Along with these fluent, we consider 14 objects in the kitchen table setting domain description. Therefore, an initial state in the worst-case, can have a maximum of 29 unknown fluents (14 for $objectAt$, 14 for $isClean$ and 1 for $foodIs$).

We introduce a parameter $\%Deg_{incomplete}$ that describe the degree of incomplete knowledge about the initial state during the planning phase. It describes the percentage of unknown fluents in the instances compared to the worst-case instance where all the 29 fluents are unknown. The benchmark instances are arranged in ascending order of $\%Deg_{incomplete}$. It is important to note that the same value of $\%Deg_{incomplete}$ for the instances does not mean that these instances have the same initial state since different fluents may be unknown.

## 6.2 Navigation benchmark

In order to demonstrate a navigation based benchmark domain, we consider mobile service robots in an office floor environment as shown in Figure 6.2. The environment consists of locations such as offices, a storeroom, a common room, and a kitchen. People can be working in their offices, watching TV in the common room, having a snack in the kitchen or anywhere else on the office floor. We consider a couple of mobile robots that can navigate around the office floor, asking and serving drinks to the people. The robots can interact with the people in order to serve them drinks. Moreover, the person can simply request nothing to drink in which case the robot does not need to serve anything to the person. Each robot has a couple of slots to hold the beverages (i.e each robot can hold a maximum of two units of beverages). This condition is necessary to ensure smooth navigation of the robot while holding the beverages. The robots do not have any manipulators and are dependent upon the humans to mount and unmount the beverages to/from it. The robots share the same knowledge among them, that is, if one robot identifies the location of a person, the others also know this automatically or similarly if one robot takes an order, the other robots know it as well and can serve the ordered beverage to the specific person. Finally, the goal of this benchmark domain is that the robots should navigate around the environment, identifying the location of all the people, ask them for the drinks they require and finally serve them with the requested drinks until all people have been served.

For this benchmark, five fluents are considered in the domain: *robotAt*, *personAt*, *drinkInHand*, *requestedDrink* and *isServed.* In order to perform navigation tasks, the robot has an actuation ac-

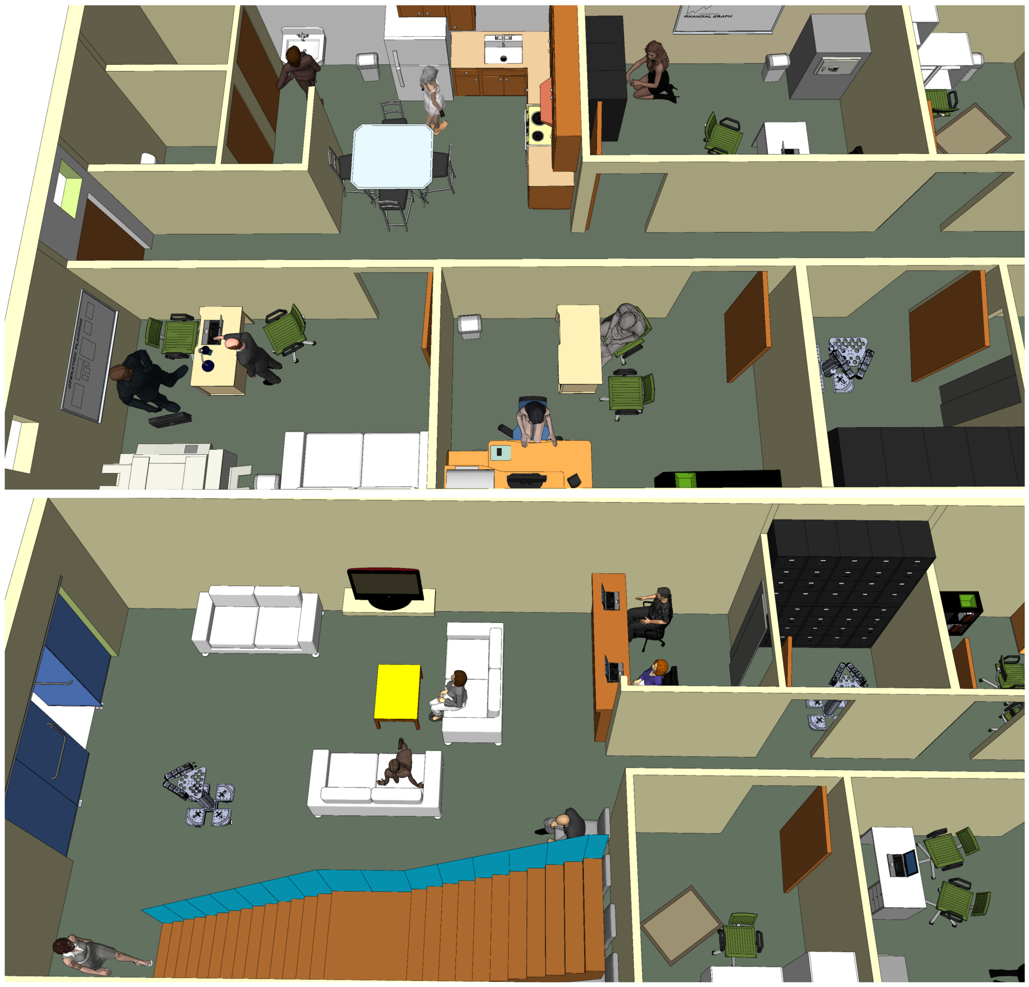


Figure 6.2: An image showing an office floor environment where a couple of mobile service robots are trying to serve beverages to the people.

tion *goto*. Along with this actuation action, we also consider two sensing actions: *check_requested_drink* and *check_if_person_at*. *check_requested_drink* asks the person about the type of drink a person wants. As preconditions for this sensing action, we consider that the robot should know the location of the person and the robot has to be near the person to perform this sensing action. The outcome of this sensing action is either *none* in which case the robot considers that the person has been served or the person may ask for a specific drink,such as cola or water. In the latter case, the robot needs to fetch the drink and fulfill the order requested by the person to complete the goal. *check_if_person_at* is about looking around in the vicinity of the robot's location and identify if a specific person is present near or not. As a precondition for this sensing action, the robot should be located at the checking location. The outcome of this sensing action is either *yes* or *no*. Table 6.2 describes the office beverages serving domain description. In order to perform an experimental evaluation of this benchmark domain, we created 15 benchmark instances by varying the initial setting for possible uncertainty about the environment.

## 6.3 Manipulation benchmark

In order to demonstrate a manipulation based benchmark domain, we consider a safety critical benchmark domain in the form of a typical laboratory environment as shown in Figure 6.3. The environment consists of a laboratory table and some laboratory equipment, such as beakers, flasks, and measuring cylinders. People come to the laboratory setup to perform some experiments. There is a stationary bi-manual manipulator in the environment whose purpose is to clean the environment after the person has com-

Table 6.2: Office beverages serving benchmark domain description.

| Fluents | Description |
|---|---|
| *robotAt* | represents the location of a robot |
| *personAt* | represents the location of a person |
| *drinkInHand* | represents the drink in hand |
| *requestedDrink* | represents the requested drink |
| *isServed* | represents if a person is served or not |

| Sensing actions | Description |
|---|---|
| *check_requested_drink* | checks the type of drink requested |
| *check_if_person_at* | checks if a person is at the location |

| Actuation actions | Description | Feasibility checks |
|---|---|---|
| *goto* | robot navigates to a location | collision |

pleted the experiments. A tray is present in the environment where the dirty objects are kept for cleaning purposes and a basket where the broken objects can be discarded. The goal of the robot is to place all the dirty objects in the tray, all the broken objects in the basket, and all the remaining clean and unbroken objects on the table to their respective shelves so that the laboratory table environment is well organized.

For this benchmark, three fluents are considered in the domain: *objectAt*, *isClean*, and *isBroken*. In order to perform manipulation tasks, the robot can perform two actuation actions *pick_up* and *place_on*. We consider three sensing actions: *check_loc*, *check_is_clean*, and *check_is_broken*. *check_loc* allows the robot to check for the location of an object while *check_is_clean* and *check_is_broken* allow the robot to check if an object is clean or if it is broken respectively. Table 6.3 describes the office beverages serving domain description. In order to perform an experimental evaluation of this benchmark domain, we created 15 benchmark instances by varying the initial setting for possible uncertainty about the environment.

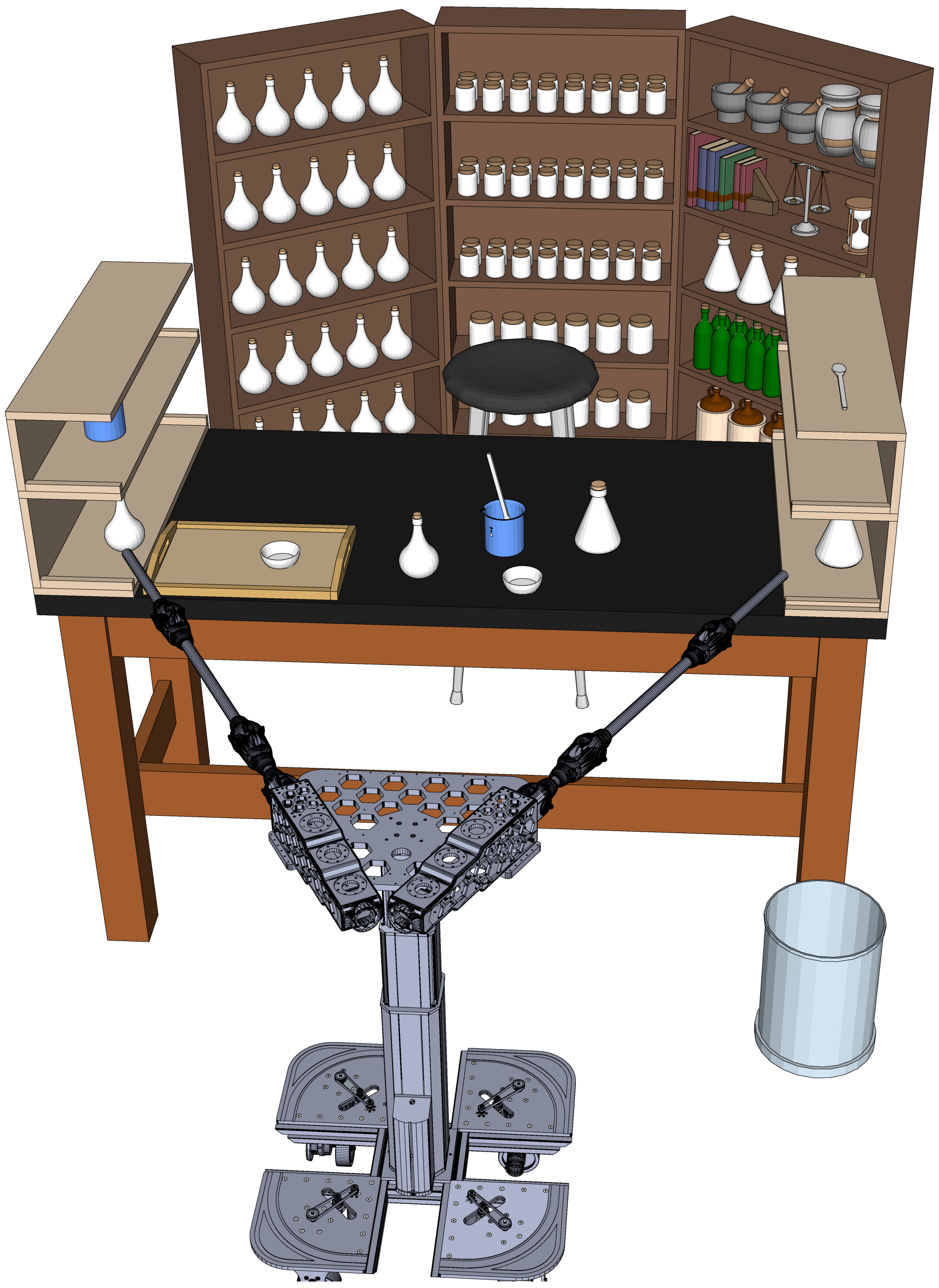


Figure 6.3: An image showing a laboratory cleaning environment where a service robot is cleaning the laboratory table.

Table 6.3: Laboratory table cleaning benchmark domain description.

| Fluents | Description |
|---|---|
| $objectAt$ | represents the location of an object |
| $isClean$ | represents if an object is clean or not |
| $isBroken$ | represents if an object is broken or not |

| Sensing actions | Description |
|---|---|
| $check\_loc$ | checks the location of an object |
| $check\_is\_clean$ | checks if an object is clean or not |
| $check\_is\_broken$ | checks if an object is broken or not |

| Actuation actions | Description | Feasibility checks |
|---|---|---|
| $pick\_up$ | robot picks an object in hand | collision, reachability, graspability |
| $place\_on$ | robot places an object at a location | collision, reachability |

# Chapter VII

## 7 Experimental Evaluation

In Section 7.1, we describe our experimental setup followed by experimental evaluation of our framework for the benchmark domains in Section 7.2. Section 7.3 and Section 7.4 discusses the use of parallelism and plan re-use respectively, to compute the plans faster in our framework. Section 7.5 discusses the importance of feasibility checks integration in the benchmark domains. Section 7.6 and Section 7.7 compare HCPlan to plan execution monitoring and HCP-ASP, respectively. Section 7.8 compares the results between HCPlan and HCPlan-Anytime for the kitchen table setting benchmark domain. Section 7.9 and Section 7.10 provide a comparison among HCPlan, HCPlan-Reactive and plan execution monitoring using the kitchen table setting benchmark.

### 7.1 Experimentation setup

All experiments are conducted on a PC workstation running Ubuntu 14.04 on 16 2.4GHz Intel E5-2665 CPU cores with 64GB memory. The feasibility checks are pre-computed and cached in a hash table for further use. Our HCPlan uses CCalc along with MiniSAT 2.2.0 as SAT solver and Python 2.7.

## 7.2 HCPlan results for benchmark domains

We compute hybrid conditional plans for all the benchmark domains and report tree parameters in Tables 7.1, 7.2, and 7.3 as follow: the maximum length of a branch from the root to a leaf $LB_{max}$ (i.e., the maximum length of a hybrid sequential plan that can be executed by the robot) and the number of actuation actions $A$ and the number of sensing actions $S$ in that branch, the maximum branching factor $BF_{max}$ (i.e., the maximum number of sensory outcomes), the total number of nodes in the tree $N$ (i.e., the size of the tree), the total number of decision nodes $D_s$ (i.e. sensing actions) and the total number of leaves in the tree $LF$ (i.e., the number of different hybrid sequential plans from an initial state to a goal state). Along with these tree parameter, we also compute the planning time $T$ in seconds, parallel speedups $PS_{16}$ for the level of parallelism "16" and degree of incomplete knowledge about initial state $Deg_{incomplete}$ for each benchmark instance.

The results for all the domains are similar, therefore we further discuss the results with the help of a kitchen table setting benchmark, since it is more generic than the other two domains. We expect the size of computed plans to increase as the degree of uncertainty in initial state increases and this is what we observe in Table 7.1. The plans for all benchmark instances are complete with respect to the contingencies that were considered during the planning phase.

In order to explain the results, let us consider Instance 24. The number of nodes in the generated plan is 4869 and it consists of 776 leaf nodes. The branch with maximum length consists of 30 actions (23 actuation and 7 sensing) which mean that if the plan is executed by the robot, it has to perform a maximum of 23 actuation actions to reach the goal under the

Table 7.1: HCPlan results for the kitchen table setting benchmark with feasibility checks integrated.

| $Inst.$ | $\%Deg_{incomplete}$ | $LB_{max}(A+S)$ | $BF_{max}$ | $N$ | $D_S$ | $LF$ | $T$ [sec] | $PS_{16}$ |
|---|---|---|---|---|---|---|---|---|
| 1 | 3.45 | 14 (13+1) | 3 | 32 | 1 | 3 | 12.56 | 1.37 |
| 2 | 6.9 | 14 (13+1) | 3 | 36 | 2 | 4 | 12.45 | 1.58 |
| 3 | 10.34 | 14 (13+1) | 3 | 36 | 2 | 4 | 15.51 | 1.45 |
| 4 | 10.34 | 14 (13+1) | 5 | 50 | 2 | 7 | 11.34 | 2.26 |
| 5 | 10.34 | 17 (15+2) | 5 | 84 | 3 | 8 | 35.75 | 1.71 |
| 6 | 13.79 | 19 (16+3) | 5 | 109 | 6 | 11 | 29.82 | 1.91 |
| 7 | 17.24 | 18 (16+2) | 3 | 64 | 5 | 7 | 22.67 | 1.64 |
| 8 | 20.69 | 16 (15+1) | 3 | 45 | 3 | 5 | 34.2 | 1.21 |
| 9 | 20.69 | 21 (17+4) | 5 | 309 | 25 | 42 | 24.95 | 5.43 |
| 10 | 24.14 | 19 (16+3) | 5 | 100 | 11 | 16 | 37.7 | 1.84 |
| 11 | 24.14 | 21 (17+4) | 5 | 177 | 13 | 24 | 26.29 | 3.28 |
| 12 | 24.14 | 20 (18+2) | 5 | 254 | 12 | 26 | 77.07 | 2.37 |
| 13 | 27.59 | 22 (19+3) | 5 | 119 | 11 | 16 | 35.04 | 2.19 |
| 14 | 31.03 | 23 (19+4) | 5 | 123 | 13 | 18 | 72.78 | 1.54 |
| 15 | 34.48 | 23 (18+5) | 5 | 191 | 22 | 27 | 79.32 | 1.84 |
| 16 | 34.48 | 23 (20+3) | 5 | 282 | 17 | 34 | 152.24 | 1.67 |
| 17 | 37.93 | 21 (18+3) | 5 | 309 | 24 | 50 | 40.36 | 4.15 |
| 18 | 37.93 | 21 (18+3) | 5 | 530 | 50 | 76 | 31.47 | 8.12 |
| 19 | 37.93 | 28 (23+5) | 5 | 863 | 96 | 137 | 196 | 2.9 |
| 20 | 41.38 | 23 (20+3) | 5 | 1325 | 135 | 206 | 129.89 | 5.55 |
| 21 | 41.38 | 27 (21+6) | 5 | 1281 | 133 | 201 | 238.57 | 3.37 |
| 22 | 44.83 | 26 (19+7) | 5 | 1928 | 162 | 320 | 89.02 | 10.25 |
| 23 | 44.83 | 28 (20+8) | 5 | 2138 | 222 | 308 | 265.71 | 5.15 |
| 24 | 48.28 | 30 (23+7) | 5 | 4869 | 558 | 776 | 269.14 | 9.11 |
| 25 | 48.28 | 31 (23+8) | 5 | 7392 | 719 | 1129 | 362.51 | 10.29 |

Table 7.2: HCPlan results for the office beverages serving benchmark with feasibility checks integrated.

| $Inst.$ | $\%Deg_{incomplete}$ | $LB_{max}(A+S)$ | $BF_{max}$ | $N$ | $D_S$ | $LF$ | $T$ [sec] |
|---|---|---|---|---|---|---|---|
| 1 | 10 | 12 (9+3) | 2 | 14 | 3 | 3 | 117.84 |
| 2 | 10 | 11 (10+1) | 6 | 36 | 1 | 6 | 21.41 |
| 3 | 20 | 16 (11+5) | 6 | 105 | 8 | 24 | 109.88 |
| 4 | 20 | 25 (18+7) | 6 | 152 | 37 | 36 | 240.87 |
| 5 | 20 | 12 (10+2) | 6 | 231 | 7 | 36 | 49.64 |
| 6 | 20 | 19 (12+7) | 2 | 46 | 14 | 10 | 491.28 |
| 7 | 30 | 18 (14+4) | 6 | 604 | 29 | 108 | 217.22 |
| 8 | 30 | 19 (14+5) | 6 | 578 | 103 | 108 | 482.05 |
| 9 | 30 | 22 (14+8) | 6 | 372 | 102 | 72 | 546.5 |
| 10 | 40 | 36 (24+12) | 6 | 4267 | 789 | 900 | 2525.97 |
| 11 | 40 | 25 (17+8) | 6 | 1784 | 313 | 324 | 1392.57 |
| 12 | 50 | 37 (22+15) | 6 | 9999 | 2809 | 2160 | 9276.31 |
| 13 | 50 | 34 (19+15) | 6 | 10980 | 2929 | 2592 | 9925.3 |
| 14 | 50 | 34 (20+14) | 6 | 8730 | 2130 | 2268 | 4760.07 |
| 15 | 50 | 29 (18+11) | 6 | 15539 | 1052 | 3240 | 3809.43 |

Table 7.3: HCPlan results for the laboratory cleaning benchmark with feasibility checks integrated.

| $Inst.$ | $\%Deg_{incomplete}$ | $LB_{max}(A+S)$ | $BF_{max}$ | $N$ | $D_S$ | $LF$ | $T$ [sec] |
|---|---|---|---|---|---|---|---|
| 1 | 7.41 | 11 (9+2) | 2 | 34 | 3 | 4 | 7.01 |
| 2 | 7.41 | 11 (9+2) | 5 | 83 | 6 | 10 | 6.06 |
| 3 | 7.41 | 12 (10+2) | 5 | 77 | 6 | 10 | 7.39 |
| 4 | 11.11 | 14 (11+3) | 5 | 75 | 7 | 20 | 7.58 |
| 5 | 11.11 | 15 (12+3) | 4 | 147 | 13 | 16 | 28.97 |
| 6 | 14.81 | 14 (10+4) | 4 | 284 | 25 | 64 | 45.06 |
| 7 | 14.81 | 15 (11+4) | 4 | 277 | 27 | 32 | 62.29 |
| 8 | 14.81 | 15 (11+4) | 5 | 923 | 63 | 250 | 31.57 |
| 9 | 18.52 | 14 (9+5) | 4 | 482 | 95 | 128 | 47.73 |
| 10 | 18.52 | 17 (12+5) | 4 | 1715 | 197 | 256 | 353.4 |
| 11 | 22.22 | 15 (9+6) | 4 | 1417 | 193 | 256 | 117.54 |
| 12 | 22.22 | 17 (11+6) | 5 | 1716 | 153 | 320 | 108.79 |
| 13 | 22.22 | 19 (13+6) | 5 | 3210 | 265 | 640 | 286.07 |
| 14 | 25.93 | 19 (12+7) | 5 | 6144 | 634 | 1280 | 767.07 |
| 15 | 29.63 | 20 (12+8) | 4 | 4128 | 673 | 1024 | 1052.74 |

incomplete initial state. It took about 269.14 seconds to generate the plan. This means that it computes 776 possible ways to reach the goal state with each possible way being computed in 0.35 seconds.

The planning time is expected to increase as the degree of uncertainty in the initial state increases since the size of the generated plan will increase. This is what we generally observe, but in instances where the degree of uncertainty in the initial state is the same, it does not take similar planning time to generate the plans. For example, in Instances 22 and Instance 23 the planning time is 89.02 seconds and 265.71 seconds respectively even with a similar number of leaves. This is due to the different number of parallel speedups. Even though the level of parallelism is 16 for both instances, the parallel speedups are different which results in different planning times.

## 7.3 Parallel computation of branches

In order to understand the effect of parallel computation of branches for our hybrid conditional planning framework, we repeated the generation of plans using the benchmark instances for our kitchen table setting domain with different values for the level of parallelism. In Table 7.4, we report sequential planning time $T_{seq}$ (planning time using a single core) in seconds, the parallel speedups $PS_x$, where x level of parallelism along with the number of leaves $LF$ for each benchmark instance.

We observe that an increase in the level of parallelism results in a faster generation of the plans. For example, in Instance 25 more than an hour is needed to generate the plan using a single core compared to 362.51 seconds it takes when the level of parallelism is set to 16, which shows a parallel speedup of 10.29. We also observe that parallel speedup is more prominent

Table 7.4: Effect of parallel computation of branches for HCPlan for kitchen table setting benchmark.

| $Inst.$ | $LF$ | $T_{seq}$ [sec] | $PS_2$ | $PS_4$ | $PS_8$ | $PS_{16}$ |
|---|---|---|---|---|---|---|
| 1 | 3 | 17.25 | 1.31 | 1.3 | 1.34 | 1.37 |
| 2 | 4 | 19.64 | 1.58 | 1.61 | 1.51 | 1.58 |
| 3 | 4 | 22.56 | 1.44 | 1.45 | 1.4 | 1.45 |
| 4 | 7 | 25.65 | 1.72 | 2.3 | 2.19 | 2.26 |
| 5 | 8 | 61.01 | 1.5 | 1.69 | 1.71 | 1.71 |
| 6 | 11 | 56.98 | 1.39 | 1.71 | 1.9 | 1.91 |
| 7 | 7 | 37.1 | 1.62 | 1.59 | 1.61 | 1.64 |
| 8 | 5 | 41.4 | 1.22 | 1.2 | 1.21 | 1.21 |
| 9 | 42 | 135.44 | 1.76 | 3.22 | 4.89 | 5.43 |
| 10 | 16 | 69.31 | 1.55 | 1.72 | 1.84 | 1.84 |
| 11 | 24 | 86.3 | 1.75 | 2.54 | 3.19 | 3.28 |
| 12 | 26 | 182.76 | 1.67 | 2.16 | 2.34 | 2.37 |
| 13 | 16 | 76.58 | 1.86 | 2.07 | 2.16 | 2.19 |
| 14 | 18 | 111.86 | 1.43 | 1.58 | 1.55 | 1.54 |
| 15 | 27 | 145.7 | 1.53 | 1.77 | 1.82 | 1.84 |
| 16 | 34 | 254.39 | 1.36 | 1.59 | 1.62 | 1.67 |
| 17 | 50 | 167.35 | 1.84 | 2.9 | 3.86 | 4.15 |
| 18 | 76 | 255.51 | 1.88 | 3.78 | 6.16 | 8.12 |
| 19 | 137 | 569.33 | 1.58 | 2.19 | 2.66 | 2.9 |
| 20 | 206 | 721.03 | 1.85 | 3.1 | 4.65 | 5.55 |
| 21 | 201 | 803.15 | 1.67 | 2.51 | 3.06 | 3.37 |
| 22 | 320 | 912.04 | 1.95 | 3.95 | 6.84 | 10.25 |
| 23 | 308 | 1367.63 | 1.81 | 3 | 4.2 | 5.15 |
| 24 | 776 | 2451.39 | 1.91 | 3.78 | 6.52 | 9.11 |
| 25 | 1129 | 3730.72 | 2.01 | 4.08 | 7.09 | 10.29 |

in benchmark instances where plans generated have more leaves, as more calls to planner have to be made.

In general, generation of hybrid conditional plans is computationally expensive, our planner utilizes the parallel computation of branches to reduce planning time. Our hybrid conditional planner is scalable and harvests the computation power of the running machine to effectively reduce the generation time of plans.

## 7.4 Re-use of saved branches

In Table 7.5, we evaluate the effectiveness of re-usage of the computed branches for our hybrid conditional planning framework with respect to computation and time for the kitchen table setting benchmark. We report the number of query calls to the planner with and without re-usage of the computed branches, $Q_{orig}$ and $Q_{reusage}$ respectively, along with the percentage of the query call reduced using re-use of the computed branches $\%Q_{reduced}$. We also report the planning time in seconds with and without for each benchmark instance, $T_{orig}$ and $T_{orig}$, respectively. Finally, the percentage of planning time improved $\%T_{improved}$ by the re-use of computed branches is also reported.

For the initial benchmark instances, the number of saved branches is small but as the instances increase, more branches are computed and saved, therefore we observe that the effectiveness of re-usage of the computed branches becomes more significant. For example, for Instance 24, re-use of the computed branches reduces the percentage of query calls to planner by 22.29% by taking 19.14% of less planning time. Our framework improves on computation time of plans by reducing the number of query calls to the planner.

Table 7.5: Statistics for re-use of the computed branches in the kitchen table setting benchmark.

| $Inst.$ | $Q_{orig}$ | $Q_{reusage}$ | $\%Q_{reduced}$ | $T_{orig}$ [sec] | $T_{reusage}$ [sec] | $\%T_{improved}$ |
|---|---|---|---|---|---|---|
| 01 | 3 | 3 | 0.00 | 12.56 | 12.56 | 0.00 |
| 02 | 4 | 4 | 0.00 | 12.45 | 12.45 | 0.00 |
| 03 | 4 | 4 | 0.00 | 15.51 | 15.51 | 0.00 |
| 04 | 7 | 7 | 0.00 | 11.34 | 11.34 | 0.00 |
| 05 | 8 | 7 | 12.50 | 35.75 | 34.74 | 2.83 |
| 06 | 11 | 10 | 9.09 | 29.82 | 29.3 | 1.74 |
| 07 | 7 | 7 | 0.00 | 22.67 | 22.67 | 0.00 |
| 08 | 5 | 5 | 0.00 | 34.2 | 34.2 | 0.00 |
| 09 | 42 | 38 | 9.52 | 24.95 | 24.52 | 1.72 |
| 10 | 16 | 14 | 12.50 | 37.7 | 35.97 | 4.59 |
| 11 | 24 | 22 | 8.33 | 26.29 | 26.29 | 0.00 |
| 12 | 26 | 23 | 11.54 | 77.07 | 76.04 | 1.34 |
| 13 | 16 | 14 | 12.50 | 35.04 | 33.83 | 3.45 |
| 14 | 18 | 15 | 16.67 | 72.78 | 71.84 | 1.29 |
| 15 | 27 | 23 | 14.81 | 79.32 | 77.85 | 1.85 |
| 16 | 34 | 28 | 17.65 | 152.24 | 150.23 | 1.32 |
| 17 | 50 | 41 | 18.00 | 40.36 | 37.24 | 7.73 |
| 18 | 76 | 64 | 15.79 | 31.47 | 28.02 | 10.96 |
| 19 | 137 | 126 | 8.03 | 196 | 191.79 | 2.15 |
| 20 | 206 | 187 | 9.22 | 129.89 | 120.76 | 7.03 |
| 21 | 201 | 166 | 17.41 | 238.57 | 227.45 | 4.66 |
| 22 | 320 | 261 | 18.44 | 89.02 | 80.06 | 10.07 |
| 23 | 308 | 235 | 23.70 | 265.71 | 252.29 | 5.05 |
| 24 | 776 | 603 | 22.29 | 269.14 | 217.62 | 19.14 |
| 25 | 1129 | 851 | 24.62 | 362.51 | 310.37 | 14.38 |

## 7.5 Integration of feasibility checks

Feasibility checks are integrated into task planning to improve the feasibility of computed plans during the execution phase. Based on this assumption, we integrated feasibility checks in our hybrid conditional planning framework. Computation of feasibility checks is a computationally expensive task and their computation needs to be justified in the overall framework.

We evaluate the effectiveness of feasibility checks integration for the hybrid conditional planning framework using the kitchen table setting benchmark domain. We compute plans with and without feasibility checks integration for first 15 instances of the benchmark domain. The computed plans are understandably different where one approach utilizes feasibility checks integration during the planning phase to make the plans more feasible during the execution phase, while the other does not.

We want to evaluate the successful execution of plans under all contingencies. For every instance, we separate all possible branches, where each branch shows a unique possible way of setting up the kitchen table under different initial settings. Each branch is then simulated in OpenRAVE [77] dynamic simulation environment by checking for executability and continuity of all the actions. We retry to re-instantiate geometric ramifications for each failed action up to 10 maximum times. If however, geometric ramifications fail after these many attempts, then that branch is labeled as infeasible for execution. It is important to note that we use non-deterministic sampling based probabilistically complete motion planners, therefore failure to find a motion plan does not necessarily imply that the branch is infeasible.

We perform the simulation process for both types of plans i.e. with and without feasibility checks integration. Once all the branches have been

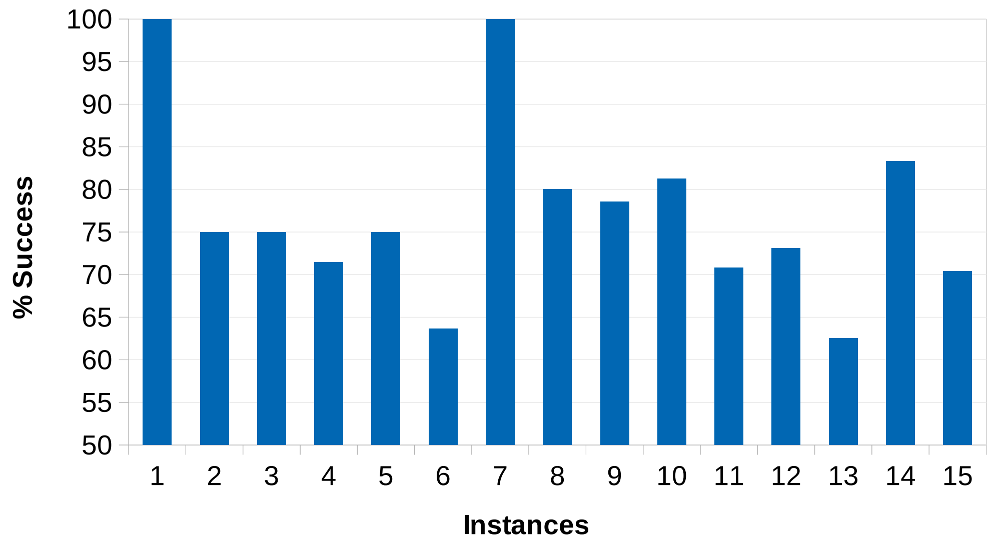


Figure 7.1: Plot showing the percentage of successful execution of branches without feasibility checks integration for the kitchen table setting benchmark.

simulated, the percentage success of plan execution $\%S$ is computed for plans, by comparing the feasible branches with respect to all the branches in the tree.

All the branches of plans computed with feasibility checks integration for the kitchen table setting benchmark have been evaluated to be successful; however, this was not the case for the plans computed without them. Figure 7.1 shows the percentage of successful execution of the branches for the kitchen table setting benchmark instances without feasibility checks integration. We see that the integration of feasibility checks for service robotics domains, such as the kitchen table setting benchmark are really important since the success of execution of plans without them is significantly lower (for example, in Instances 6 and 13 the success rate is slightly higher than 60%).

As mentioned before, pre-computation of feasibility checks is computationally expensive; however, the results in Figure 7.1 justifies their use for service robotic domains as it significantly improves the feasibility of computed plans for real-life executions. It is important to point out here, while the integration of feasibility checks improves the feasibility of execution of the plans, in general, successful plan execution cannot be guaranteed. This may be due to the incomplete nature of the feasibility check algorithms. Therefore, to ensure successful plan execution, a plan execution monitoring module is still useful.

## 7.6 Comparison between plan execution monitoring and HCPlan

Hybrid conditional planning is performed in an offline manner and it plans for contingencies during the planning phase. An alternative approach to hybrid conditional planning is plan execution monitoring, where some assumptions are made for these contingencies and a classical planner along with an execution monitor is utilized to compute as well as execute the plan. The basic plan execution monitoring works as follow: (1) Make assumptions about uncertainties in initial state, (2) computes a hybrid classical plan using a classical planner, that consists of sequence of actuation actions to reach goal for modified initial state, (3) execute the plan and monitor the observable fluents for discrepancies, (4) re-plan if a discrepancy is detected between the expected and the observed values of these fluents and (5) execute and monitor until the goal is reached or failure to reach the goal occurs.

Since classical planners require complete information about the initial state, in our experiments evaluation, we deterministically assign legitimate values (i.e., outcomes of relevant sensing actions) to the fluents that are not known (due to incomplete knowledge about initial state) non-deterministically. Due to this non-deterministic assignment, each instance is solved 5 times, and the averages along with maximum values are reported in results. A parameter that effects plan execution monitoring framework is, how often the observable fluents are sensed. The more frequent sensing is performed typically the better it is, but real-world scenarios may have some limitations about how often sensing is feasible. We report the results where sensing is performed at intervals of every 3 and 5 time steps. Tables 7.6 and 7.7 present the comparison of plans computed for kitchen table setting benchmark using plan execution monitoring with sensing at every 3 and 5 steps, respectively along with hybrid conditional plans. For plan execution monitoring: the total plan length to reach the goal $L$, the number of re-planning attempts $R$, and the planning time $T$ (offline and online) in seconds are reported. As the experiment is performed 5 times, the maximum $(.)_{max}$ and the average $(.)_{av}$ values are listed in the tables. For hybrid conditional planning framework: the maximum length of a branch from the root to a leaf $LB_{max}$ along with the number of actuation $A$ and sensing $S$ actions in that branch, the total number of leaves in the tree $LF$ and the planning time (offline) in seconds are reported. It is important to note that $A$ in $LB_{max}$ represents the upper bound for the number of actuation actions that will be performed while execution of hybrid conditional plans.

Hybrid conditional planning framework spends more time on planning compared to plan execution monitoring. This is understandable since it computes plans for all possible contingencies. For example, in Table 7.7 for Instance 24 hybrid conditional planning takes 269.14 seconds to compute the whole conditional plan compared on average 90.59 seconds (189.01 seconds in the worst case) for plan execution monitoring with sensing at every 5 steps.

Table 7.6: Comparison between the computed plans using plan execution monitoring with sensing after every '3' steps and hybrid conditional planning for the kitchen table setting benchmark.

| | Plan Execution Monitoring | | | HCPLAN | | |
|---|---|---|---|---|---|---|
| $Inst.$ | $L_{max}(L_{av})$ | $R_{max}(R_{av})$ | $T_{max}(T_{avg})$ [sec] | $LB_{max}(A+S)$ | $LF$ | $T$ [sec] |
| 01 | 16 (14.8) | 1 (0.6) | 10.29 (8.8) | 14 (13+1) | 3 | 12.56 |
| 02 | 19 (17.2) | 2 (1.4) | 15.44 (13.49) | 14 (13+1) | 4 | 12.45 |
| 03 | 22 (18.4) | 3 (1.8) | 25.35 (16.03) | 14 (13+1) | 4 | 15.51 |
| 04 | 22 (20.2) | 3 (2.4) | 27.72 (19.56) | 14 (13+1) | 7 | 11.34 |
| 05 | 24 (19.8) | 3 (1.6) | 54.45 (30.81) | 17 (15+2) | 8 | 35.75 |
| 06 | 23 (21.8) | 3 (2.6) | 35.93 (27.32) | 19 (16+3) | 11 | 29.82 |
| 07 | 30 (25.4) | 5 (3.6) | 63.39 (43.87) | 18 (16+2) | 7 | 22.67 |
| 08 | 30 (26.8) | 5 (4) | 83.97 (64.54) | 16 (15+1) | 5 | 34.2 |
| 09 | 26 (22.8) | 4 (3.2) | 34.36 (27.97) | 21 (17+4) | 42 | 24.95 |
| 10 | 30 (28.6) | 5 (4.6) | 74.87 (50.09) | 19 (16+3) | 16 | 37.7 |
| 11 | 32 (29.6) | 6 (5.2) | 57.85 (37.19) | 21 (17+4) | 24 | 26.29 |
| 12 | 30 (27) | 5 (4) | 77.36 (52.73) | 20 (18+2) | 26 | 77.07 |
| 13 | 32 (29.6) | 6 (5.2) | 90.43 (53.87) | 22 (19+3) | 16 | 35.04 |
| 14 | 36 (30.6) | 7 (5.2) | 75.04 (54.39) | 23 (19+4) | 18 | 72.78 |
| 15 | 38 (32) | 8 (6) | 90.11 (52.16) | 23 (18+5) | 27 | 79.32 |
| 16 | 36 (32.4) | 7 (5.8) | 277.76 (142.21) | 23 (20+3) | 34 | 152.24 |
| 17 | 40 (34) | 9 (7) | 80.12 (48.61) | 21 (18+3) | 50 | 40.36 |
| 18 | 40 (37.6) | 9 (8.2) | 73.52 (58.56) | 21 (18+3) | 76 | 31.47 |
| 19 | 38 (34) | 8 (6.4) | 149.77 (94.31) | 28 (23+5) | 137 | 196 |
| 20 | 39 (37.6) | 8 (7.6) | 163.27 (86.32) | 23 (20+3) | 206 | 129.89 |
| 21 | 41 (37) | 9 (7.4) | 140.69 (103.5) | 27 (21+6) | 201 | 238.57 |
| 22 | 43 (35.8) | 10 (7.6) | 87.63 (68.95) | 26 (19+7) | 320 | 89.02 |
| 23 | 44 (41.8) | 10 (9.2) | 177.57 (94.23) | 28 (20+8) | 308 | 265.71 |
| 24 | 48 (39.4) | 11 (8.2) | 113.31 (74.68) | 30 (23+7) | 776 | 269.14 |
| 25 | 51 (43.6) | 12 (9.8) | 150.24 (103.68) | 31 (23+8) | 1129 | 362.51 |

Table 7.7: Comparison between the computed plans using plan execution monitoring with sensing after every '5' steps and hybrid conditional planning for the kitchen table setting benchmark.

| | Plan Execution Monitoring | | | HCPLAN | | |
|---|---|---|---|---|---|---|
| $Inst.$ | $L_{max}(L_{av})$ | $R_{max}(R_{av})$ | $T_{max}(T_{avg})$ [sec] | $LB_{max}(A+S)$ | $LF$ | $T$ [sec] |
| 01 | 18 (15) | 1 (0.4) | 10.41 (8.26) | 14 (13+1) | 3 | 12.56 |
| 02 | 23 (17) | 2 (0.8) | 14.16 (10.19) | 14 (13+1) | 4 | 12.45 |
| 03 | 28 (20) | 3 (1.4) | 27.43 (15.26) | 14 (13+1) | 4 | 15.51 |
| 04 | 23 (21) | 2 (1.6) | 21.41 (17.53) | 14 (13+1) | 7 | 11.34 |
| 05 | 25 (23) | 2 (1.6) | 48.38 (33.66) | 17 (15+2) | 8 | 35.75 |
| 06 | 34 (28) | 4 (2.8) | 46.48 (30.35) | 19 (16+3) | 11 | 29.82 |
| 07 | 40 (33) | 5 (3.6) | 62.34 (37.87) | 18 (16+2) | 7 | 22.67 |
| 08 | 39 (33.8) | 5 (3.8) | 83.65 (70.57) | 16 (15+1) | 5 | 34.2 |
| 09 | 33 (31) | 4 (3.6) | 31.88 (25.37) | 21 (17+4) | 42 | 24.95 |
| 10 | 45 (39) | 6 (4.8) | 94.97 (64.56) | 19 (16+3) | 16 | 37.7 |
| 11 | 39 (34) | 5 (3.8) | 89.75 (58.05) | 21 (17+4) | 24 | 26.29 |
| 12 | 50 (38.8) | 5 (4.2) | 240.66 (112.08) | 20 (18+2) | 26 | 77.07 |
| 13 | 54 (40) | 8 (5.2) | 74.19 (42.05) | 22 (19+3) | 16 | 35.04 |
| 14 | 54 (44.8) | 8 (6) | 92.96 (67.19) | 23 (19+4) | 18 | 72.78 |
| 15 | 54 (43) | 8 (5.8) | 69.98 (48.77) | 23 (18+5) | 27 | 79.32 |
| 16 | 55 (43.8) | 7 (5.6) | 74.44 (55.83) | 23 (20+3) | 34 | 152.24 |
| 17 | 58 (50) | 8 (7) | 67.71 (56.25) | 21 (18+3) | 50 | 40.36 |
| 18 | 58 (44) | 9 (6.2) | 83.4 (52.74) | 21 (18+3) | 76 | 31.47 |
| 19 | 60 (52) | 8 (7) | 163.35 (84.92) | 28 (23+5) | 137 | 196 |
| 20 | 65 (50) | 9 (6.8) | 174.09 (109.24) | 23 (20+3) | 206 | 129.89 |
| 21 | 65 (55) | 9 (7.8) | 137.63 (99.83) | 27 (21+6) | 201 | 238.57 |
| 22 | 64 (55.2) | 10 (8.4) | 80.29 (60.95) | 26 (19+7) | 320 | 89.02 |
| 23 | 60 (54.2) | 9 (7.8) | 123.07 (88.25) | 28 (20+8) | 308 | 265.71 |
| 24 | 80 (68.8) | 12 (10.6) | 189.01 (90.59) | 30 (23+7) | 776 | 269.14 |
| 25 | 59 (56) | 9 (8.4) | 234.59 (127.63) | 31 (23+8) | 1129 | 362.51 |

However, it is important to note that, hybrid conditional planning framework generates a tree with 776 leaves; therefore, it just takes 0.35 seconds to generate a hybrid sequential plan (which represents one possible way to reach the goal from an initial state with incomplete knowledge). Also, hybrid conditional plans are computed offline and do not require re-planning whereas, plan execution monitoring had to pause the execution and re-plan on average 10.6 times (12 times in the worst case).

We also note that the total number of actuation actions to be executed are typically more for plan execution monitoring compared to hybrid conditional plans. For example, in Instance 24 if robot utilizes plan execution monitoring framework than it needs to execute on average 68.8 (and a maximum of 80) robotic actuation actions, whereas with hybrid conditional planning it just needs to perform a maximum of 23 robotic actuation actions to reach goal under any contingency. The number of actuation actions is critical especially in robotics domains where execution of physical actions takes precious time and resources. It is important to mention here that, $A$ represents the upper bound for the number of actuation actions for an instance, therefore typically the number of actuation actions will be lower than this value. Furthermore, hybrid conditional planning framework also provides the exact location for sensing to be performed during the execution.

## 7.7 Comparison between HCP-ASP and HCPlan

In order to compare our planner with another hybrid conditional planner HCP-ASP, we choose to use the kitchen table setting benchmark domain. The input language of HCPlan is $\mathcal{C}+$ [64], while it is answer set programming (ASP) [62] for HCP-ASP. Therefore, in order to compute the plans for kitchen table setting benchmark using HCP-ASP, the translation of the benchmark domain from $\mathcal{C}+$ into ASP is performed using a formal transformation. The translation soundness is guaranteed by the module theorem [85].

We computed the plans with HCPlan and HCP-ASP for all the 25 benchmark instances of the kitchen table setting domain. In Table 7.8, we report the minimum length of hybrid sequential plan $L_{min}$, the maximum length of hybrid sequential plan $L_{max}$, the average branching factor $BF_{avg}$, the number of nodes in the tree $N$, the number of sensing nodes in the tree $D_S$, the number of leaves of the tree $L$ and the computation time in seconds $T$ for all the plans computed trees by both planners.

HCP-ASP and HCPlan are based on similar algorithms: they start by computing a shortest branch (i.e., a sequence of actuation actions and sensing actions) to reach a goal state, and then traverse this branch to identify the sensing actions and to compute branches for alternate outcomes of all sensing actions; in this way, they compute the complete hybrid conditional plan. While the branches computed initially are optimal in plan length, in general they may be different. Furthermore, even though all plans computed from sensing nodes to goal states have minimum plan lengths, the plans from the root node to a goal state are not necessarily the shortest. Optimal plan length for each branch may be computed if all branches in the plans are computed from the root node to the goal states; however, this process is

Table 7.8: Comparison between HCP-ASP and HCPLAN using the plans generated for the kitchen table setting benchmark domain.

| | HCP-ASP | | | | | | | HCPLAN | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| *Inst.* | $L_{min}$ | $L_{max}$ | $BF_{avg}$ | $N$ | $D_S$ | $L$ | $T$ [sec] | $L_{min}$ | $L_{max}$ | $BF_{avg}$ | $N$ | $D_S$ | $L$ | $T$ [sec] |
| 1 | 9 | 14 | 3 | 32 | 1 | 3 | 10.56 | 9 | 14 | 3 | 32 | 1 | 3 | 12.56 |
| 2 | 10 | 14 | 2.5 | 37 | 2 | 4 | 14.21 | 10 | 14 | 2.5 | 36 | 2 | 4 | 12.45 |
| 3 | 10 | 14 | 2.5 | 37 | 2 | 4 | 18.81 | 10 | 14 | 2.5 | 36 | 2 | 4 | 15.51 |
| 4 | 8 | 14 | 4 | 54 | 2 | 7 | 12.52 | 8 | 14 | 4 | 50 | 2 | 7 | 11.34 |
| 5 | 10 | 16 | 2.5 | 40 | 2 | 4 | 71.27 | 10 | 17 | 3.33 | 84 | 3 | 8 | 35.75 |
| 6 | 11 | 18 | 2.2 | 59 | 5 | 7 | 38.79 | 11 | 19 | 2.67 | 109 | 6 | 11 | 29.82 |
| 7 | 11 | 18 | 2.2 | 56 | 5 | 7 | 51.47 | 11 | 18 | 2.2 | 64 | 5 | 7 | 22.67 |
| 8 | 11 | 20 | 2.17 | 63 | 6 | 8 | 118.05 | 11 | 16 | 2.33 | 45 | 3 | 5 | 34.2 |
| 9 | 11 | 23 | 2.48 | 336 | 27 | 41 | 67.49 | 11 | 21 | 2.64 | 309 | 25 | 42 | 24.95 |
| 10 | 11 | 16 | 2.33 | 138 | 12 | 17 | 81.95 | 11 | 19 | 2.36 | 100 | 11 | 16 | 37.7 |
| 11 | 11 | 21 | 3.86 | 118 | 7 | 21 | 25.11 | 11 | 21 | 2.77 | 177 | 13 | 24 | 26.29 |
| 12 | 11 | 21 | 2.39 | 234 | 18 | 26 | 735.26 | 11 | 20 | 3.08 | 254 | 12 | 26 | 77.07 |
| 13 | 11 | 16 | 2.5 | 83 | 8 | 13 | 106.69 | 11 | 22 | 2.36 | 119 | 11 | 16 | 35.04 |
| 14 | 11 | 20 | 2.36 | 122 | 11 | 16 | 252.42 | 11 | 23 | 2.31 | 123 | 13 | 18 | 72.78 |
| 15 | 11 | 24 | 2.29 | 199 | 24 | 32 | 170.16 | 11 | 23 | 2.18 | 191 | 22 | 27 | 79.32 |
| 16 | 11 | 22 | 2.41 | 159 | 17 | 25 | 589.47 | 11 | 23 | 2.94 | 282 | 17 | 34 | 152.24 |
| 17 | 8 | 22 | 2.33 | 100 | 12 | 17 | 133.74 | 8 | 21 | 3.04 | 309 | 24 | 50 | 40.36 |
| 18 | 9 | 21 | 3.33 | 197 | 12 | 29 | 27.58 | 9 | 21 | 2.5 | 530 | 50 | 76 | 31.47 |
| 19 | 9 | 25 | 2.55 | 878 | 84 | 131 | 889.16 | 9 | 28 | 2.42 | 863 | 96 | 137 | 196 |
| 20 | 12 | 23 | 2.51 | 1603 | 174 | 263 | 368.18 | 12 | 23 | 2.52 | 1325 | 135 | 206 | 129.89 |
| 21 | 9 | 24 | 2.65 | 864 | 84 | 140 | 800.26 | 9 | 27 | 2.5 | 1281 | 133 | 201 | 238.57 |
| 22 | 8 | 28 | 2.69 | 2945 | 262 | 444 | 267.96 | 8 | 26 | 2.97 | 1928 | 162 | 320 | 89.02 |
| 23 | 9 | 31 | 2.41 | 1541 | 163 | 231 | 816.17 | 9 | 28 | 2.38 | 2138 | 222 | 308 | 265.71 |
| 24 | 8 | 32 | 2.47 | 3900 | 435 | 641 | 622.96 | 8 | 30 | 2.39 | 4869 | 558 | 776 | 269.14 |
| 25 | 12 | 33 | 2.56 | 7366 | 764 | 1192 | 142.05 | 12 | 31 | 2.57 | 7392 | 719 | 1129 | 362.51 |

computationally expensive. Moreover, combining these branches into a single hybrid conditional plan will be a difficult task.

Results in Table 7.8 show that the minimum length of branch $L_{min}$ for both planners is similar. However, while the first computed branch is guaranteed to have the same plan length, the plans computed by HCP-ASP and HCPlan, in general, are not the same (i.e the plans may have different sensing and actuation actions, and the order of actions may differ). Similarly, the future branches may differ and this results in trees with different sizes and lengths. Although the plans computed by HCPlan and HCP-ASP may be different, they consider all possible contingencies starting from the same state and the robot can follow any one of the branches to reach a goal state.

Even though the resulting trees are similar, the computation times differ for HCP-ASP and HCPlan. Indeed even though both planners are based on similar algorithms for hybrid conditional planning, their input languages and solvers differ. In particular, HCP-ASP relies on ASP solvers (GrinGo [86] for grounding and Clasp [87] for solving) while HCPlan utilizes CCalc [59] (SWI-Prolog [88] for grounding and SAT solvers such as MiniSAT [89] for solving). Along these lines, our experimental evaluations indicate that CCalc with MiniSAT is more efficient in computing plans for service robotic domains tested.

## 7.8 Comparison between HCPlan and HCPlan-Anytime

HCPlan is a compilation-based hybrid conditional planner that utilized parallel computation of branches to computes plans that are complete for the contingencies considered during the planning phase. HCPlan-Anytime utilizes a similar framework and extends it to compute partial plans that contain most probable contingencies. HCPlan-Anytime trades off completeness for better computation time by neglecting generation of least probable branches. This trade-off between planning time and completeness of plans can be justified in real-world robotics applications where plans are needed to be computed and executed in real-time. Moreover, HCPlan-Anytime is an anytime planner, that is, it can be stopped at any moment during the planning phase and will still provide a valid partial hybrid conditional plan that includes all the most probable branches that have been computed until that moment. Since HCPlan-Anytime computes partial plans, we expect the tree sizes of the plans computed with HCPlan-Anytime to be smaller compared to ones computed with HCPlan.

Tables 7.9 and 7.10 show the comparison between complete plans computed with HCPlan versus 70% and 95% of partial plans computed with HCPlan-Anytime for the kitchen table setting benchmark instances. We report the average branching factor $BF_{avg}$ (i.e., the average number of sensory outcomes), the total number $N$ of nodes in the tree (i.e., the size of the tree), the total number $LF$ of leaves in the tree (i.e., the number of different hybrid sequential plans from an initial state to a goal state) and the planning time spend on tree, $T$ in seconds. We also report $\%T_x$, that represents the percentage of planning time taken in the computation of partial plan with the $x$ amount by HCPlan-Anytime compared to the planning time of complete plan computed with HCPlan.

The results comply with our expectations. As HCPlan-Anytime computes partial plans by not computing least probable branches, the average branching factor is lower than complete plans computed with HCPlan. This results in lesser number of leaves and smaller size of the tree. Moreover, the planning time reduces as well. For example, in Table 7.9, Instance 25 $BF_{avg}$ for complete plan computed with HCPlan is 2.57 compared to 1.82 for 70% partial plan computed with HCPlan-Anytime. The planning time for computation of plans falls from 362.51 seconds to 188.66 seconds which represents a 47.96% decrease in planning time to compute the plan. Figure 7.2 shows a partial hybrid conditional plan for an instance of kitchen table setting benchmark computed by HCPlan-Anytime where $x$ is 70.

Table 7.9: Comparison between the complete plans computed with HCPlan and 70% of the partial plans computed with HCPlan-Anytime for the kitchen table setting benchmark.

| | HCPlan | | | | HCPlan-Anytime | | | | |
|---|---|---|---|---|---|---|---|---|---|
| *Scen* | $BF_{avg}$ | $N$ | $LF$ | $T$ [sec] | $BF_{avg}$ | $N$ | $LF$ | $T$ [sec] | $\%T_{70}$ |
| 01 | 3 | 32 | 3 | 12.56 | 3 | 32 | 3 | 12.81 | 0 |
| 02 | 2.5 | 36 | 4 | 12.45 | 2 | 33 | 3 | 12.28 | 1.37 |
| 03 | 2.5 | 36 | 4 | 15.51 | 2 | 33 | 3 | 15.41 | 0.64 |
| 04 | 4 | 50 | 7 | 11.34 | 2 | 30 | 3 | 10.87 | 4.14 |
| 05 | 3.33 | 84 | 8 | 35.75 | 2 | 41 | 4 | 30.23 | 15.44 |
| 06 | 2.67 | 109 | 11 | 29.82 | 1.67 | 57 | 5 | 24.81 | 16.8 |
| 07 | 2.2 | 64 | 7 | 22.67 | 1.6 | 44 | 4 | 18.44 | 18.66 |
| 08 | 2.33 | 45 | 5 | 34.2 | 2 | 41 | 4 | 31.61 | 7.57 |
| 09 | 2.64 | 309 | 42 | 24.95 | 1.86 | 157 | 20 | 21.29 | 14.67 |
| 10 | 2.36 | 100 | 16 | 37.7 | 1.82 | 75 | 10 | 35.28 | 6.42 |
| 11 | 2.77 | 177 | 24 | 26.29 | 2.18 | 110 | 14 | 22.93 | 12.78 |
| 12 | 3.08 | 254 | 26 | 77.07 | 2.3 | 150 | 14 | 65.8 | 14.62 |
| 13 | 2.36 | 119 | 16 | 35.04 | 1.89 | 89 | 9 | 32.74 | 6.56 |
| 14 | 2.31 | 123 | 18 | 72.78 | 1.75 | 77 | 10 | 69.21 | 4.91 |
| 15 | 2.18 | 191 | 27 | 79.32 | 1.74 | 130 | 15 | 65.98 | 16.82 |
| 16 | 2.94 | 282 | 34 | 152.24 | 2.3 | 131 | 14 | 143.36 | 5.83 |
| 17 | 3.04 | 309 | 50 | 40.36 | 1.76 | 126 | 17 | 33.97 | 15.83 |
| 18 | 2.5 | 530 | 76 | 31.47 | 1.78 | 150 | 15 | 20.75 | 34.06 |
| 19 | 2.42 | 863 | 137 | 196 | 1.71 | 403 | 56 | 187.04 | 4.57 |
| 20 | 2.52 | 1325 | 206 | 129.89 | 1.79 | 413 | 57 | 111.77 | 13.95 |
| 21 | 2.5 | 1281 | 201 | 238.57 | 1.75 | 565 | 75 | 218.12 | 8.57 |
| 22 | 2.97 | 1928 | 320 | 89.02 | 2.24 | 1076 | 171 | 75.1 | 15.64 |
| 23 | 2.38 | 2138 | 308 | 265.71 | 1.82 | 942 | 116 | 228.92 | 13.85 |
| 24 | 2.39 | 4869 | 776 | 269.14 | 1.69 | 1510 | 205 | 160.33 | 40.43 |
| 25 | 2.57 | 7392 | 1129 | 362.51 | 1.82 | 2333 | 292 | 188.66 | 47.96 |

Table 7.10: Comparison between the complete plans computed with HCPLAN and 95% of the partial plans computed with HCPLAN-ANYTIME for the kitchen table setting benchmark.

| | HCPLAN | | | | HCPLAN-ANYTIME | | | | |
|---|---|---|---|---|---|---|---|---|---|
| *Scen* | $BF_{avg}$ | $N$ | $LF$ | $T$ [sec] | $BF_{avg}$ | $N$ | $LF$ | $T$ [sec] | $\%T_{95}$ |
| 01 | 3 | 32 | 3 | 12.56 | 3 | 32 | 3 | 12.41 | 1.19 |
| 02 | 2.5 | 36 | 4 | 12.45 | 2.5 | 36 | 4 | 12.79 | 0 |
| 03 | 2.5 | 36 | 4 | 15.51 | 2.5 | 36 | 4 | 15.55 | 0 |
| 04 | 4 | 50 | 7 | 11.34 | 3 | 40 | 5 | 11 | 3 |
| 05 | 3.33 | 84 | 8 | 35.75 | 3 | 74 | 7 | 35.5 | 0.7 |
| 06 | 2.67 | 109 | 11 | 29.82 | 2.5 | 99 | 10 | 29.85 | 0 |
| 07 | 2.2 | 64 | 7 | 22.67 | 2.2 | 64 | 7 | 22.55 | 0.53 |
| 08 | 2.33 | 45 | 5 | 34.2 | 2.33 | 45 | 5 | 33.69 | 1.49 |
| 09 | 2.64 | 309 | 42 | 24.95 | 2.52 | 289 | 39 | 24.64 | 1.24 |
| 10 | 2.36 | 100 | 16 | 37.7 | 2.36 | 100 | 16 | 38.21 | 0 |
| 11 | 2.77 | 177 | 24 | 26.29 | 2.69 | 173 | 23 | 25.86 | 1.64 |
| 12 | 3.08 | 254 | 26 | 77.07 | 2.83 | 226 | 23 | 72.95 | 5.35 |
| 13 | 2.36 | 119 | 16 | 35.04 | 2.27 | 115 | 15 | 35.31 | 0 |
| 14 | 2.31 | 123 | 18 | 72.78 | 2.23 | 119 | 17 | 71.95 | 1.14 |
| 15 | 2.18 | 191 | 27 | 79.32 | 2.05 | 177 | 24 | 78.33 | 1.25 |
| 16 | 2.94 | 282 | 34 | 152.24 | 2.62 | 191 | 22 | 147.33 | 3.23 |
| 17 | 3.04 | 309 | 50 | 40.36 | 2.65 | 249 | 39 | 37.7 | 6.59 |
| 18 | 2.5 | 530 | 76 | 31.47 | 2 | 323 | 39 | 28.27 | 10.17 |
| 19 | 2.42 | 863 | 137 | 196 | 2.18 | 700 | 108 | 194.69 | 0.67 |
| 20 | 2.52 | 1325 | 206 | 129.89 | 1.95 | 647 | 91 | 111.26 | 14.34 |
| 21 | 2.5 | 1281 | 201 | 238.57 | 2.29 | 1086 | 162 | 229.13 | 3.96 |
| 22 | 2.97 | 1928 | 320 | 89.02 | 2.79 | 1727 | 282 | 84.37 | 5.22 |
| 23 | 2.38 | 2138 | 308 | 265.71 | 2.06 | 1572 | 208 | 242.49 | 8.74 |
| 24 | 2.39 | 4869 | 776 | 269.14 | 2.19 | 3581 | 553 | 246.13 | 8.55 |
| 25 | 2.57 | 7392 | 1129 | 362.51 | 2.1 | 4665 | 648 | 263.79 | 27.23 |

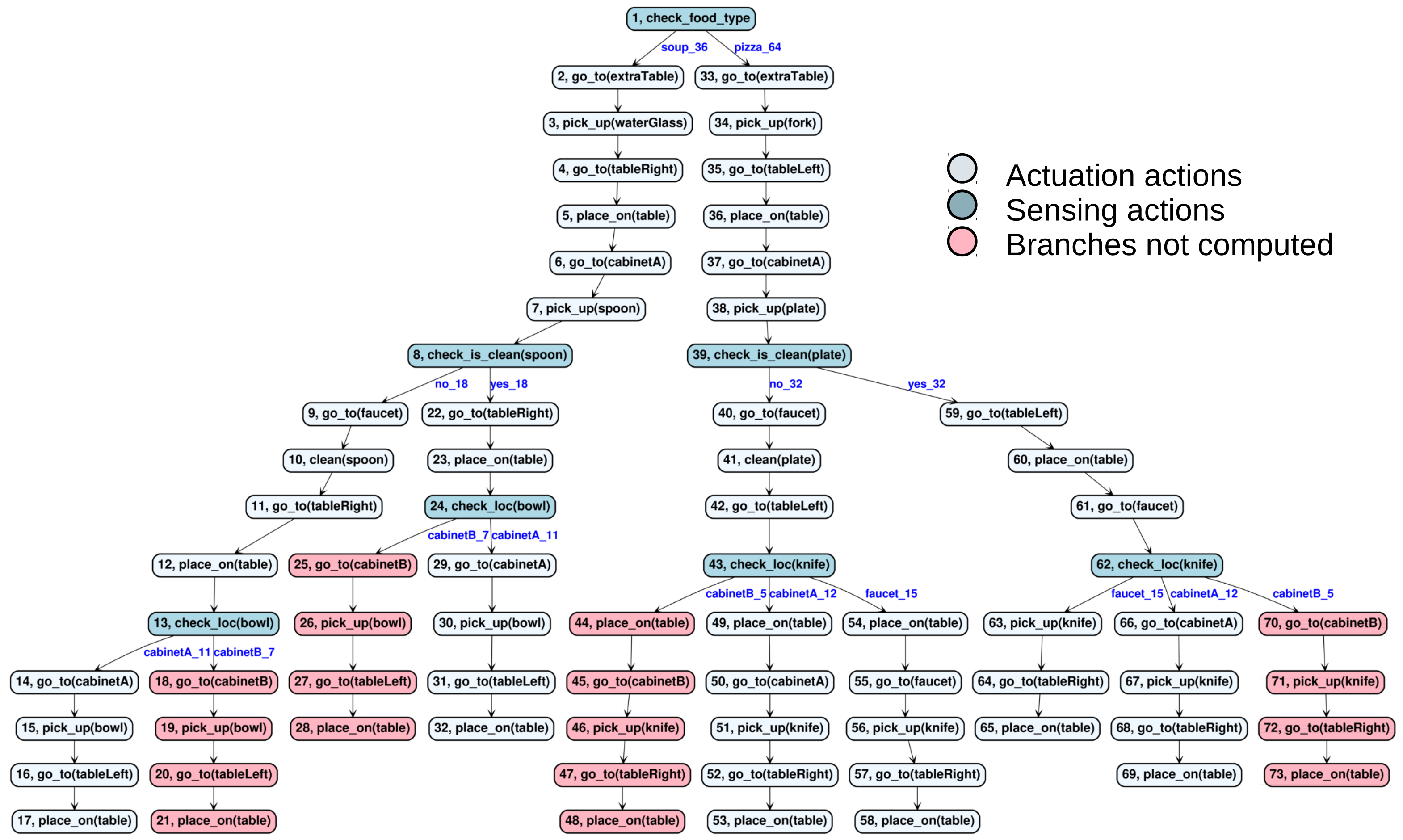


Figure 7.2: An anytime hybrid conditional partial plan computed for an instance of the kitchen table setting benchmark where $x$ is ‘70’.

## 7.9 Comparison between HCPlan and HCPlan-Reactive

HCPlan-Reactive extends hybrid conditional planning framework and utilizes guidance from online sensing to compute partial plans by expanding portions that are more relevant during the execution phase. Since HCPlan-Reactive expands a relevant portion of tree guided by online sensing instead of blind expansion as done in offline plans; we call it a reactive planner. Furthermore, since HCPlan-Reactive computes partial plans as compared to HCPlan, it results in a significantly smaller size for plans.

Tables 7.11 and 7.12 show the comparison between complete plans computed with HCPlan and partial plans computed with HCPlan-Reactive for kitchen table setting benchmark instances with a reactive depth threshold $D_{th}$ of '3' and '5', respectively. We report the computed tree parameters: the total number $N$ of nodes in the tree (i.e., the size of the tree) and the total number $LF$ of leaves in the tree (i.e., the number of different hybrid sequential plans from an initial state to a goal state). We also report the planning time spend on the tree, $T$ in seconds.

We observe smaller trees in case of HCPlan-Reactive compared to HCPlan which is understandable as former computes partial plans compared to the complete plans in case of HCPlan. For example, in Table 7.12, in Instance 25 HCPlan computes a tree with 7392 nodes with 1129 leaves in 362.51 seconds whereas, HCPlan-Reactive takes just 80 seconds to solve the same instance and computes 92 leaves instead. This shows that only 8.15% of the complete plan was computed and this partial plan computation reduces the planning time by more than 450%. HCPlan-Reactive takes some online planning time to reduce the load on offline computation

Table 7.11: Comparison between the complete plans computed with HCPlan and the partial plans computed using HCPlan-Reactive with reactive depth threshold $D_{th}$ of ‘3’, for the kitchen table setting benchmark.

| | HCPlan | | | HCPlan-Reactive | | |
|---|---|---|---|---|---|---|
| *Scen* | $N$ | $LF$ | $T$ [sec] | $N$ | $LF$ | $T$ [sec] |
| 1 | 32 | 3 | 12.56 | 32 | 3 | 12.79 |
| 2 | 36 | 4 | 12.45 | 33 | 3 | 12.48 |
| 3 | 36 | 4 | 15.51 | 33 | 3 | 14.94 |
| 4 | 50 | 7 | 11.34 | 30 | 3 | 11.39 |
| 5 | 84 | 8 | 35.75 | 41 | 4 | 32.74 |
| 6 | 109 | 11 | 29.82 | 51 | 5 | 24.04 |
| 7 | 64 | 7 | 22.67 | 35 | 3 | 18.82 |
| 8 | 45 | 5 | 34.2 | 41 | 4 | 34.24 |
| 9 | 309 | 42 | 24.95 | 171 | 20 | 19.51 |
| 10 | 100 | 16 | 37.7 | 42 | 4 | 35.58 |
| 11 | 177 | 24 | 26.29 | 85 | 11 | 25.01 |
| 12 | 254 | 26 | 77.07 | 132 | 11 | 70.22 |
| 13 | 119 | 16 | 35.04 | 75 | 7 | 30.44 |
| 14 | 123 | 18 | 72.78 | 51 | 5 | 68.33 |
| 15 | 191 | 27 | 79.32 | 74 | 8 | 64.31 |
| 16 | 282 | 34 | 152.24 | 150 | 15 | 145.87 |
| 17 | 309 | 50 | 40.36 | 76 | 10 | 33.2 |
| 18 | 530 | 76 | 31.47 | 143 | 17 | 21.71 |
| 19 | 863 | 137 | 196 | 199 | 26 | 173.78 |
| 20 | 1325 | 206 | 129.89 | 117 | 15 | 87.14 |
| 21 | 1281 | 201 | 238.57 | 142 | 13 | 187.16 |
| 22 | 1928 | 320 | 89.02 | 119 | 18 | 25.27 |
| 23 | 2138 | 308 | 265.71 | 163 | 19 | 162.19 |
| 24 | 4869 | 776 | 269.14 | 165 | 19 | 96.6 |
| 25 | 7392 | 1129 | 362.51 | 489 | 51 | 93.77 |

Table 7.12: Comparison between the complete plans computed with HCPlan and the partial plans computed using HCPlan-Reactive with reactive depth threshold $D_{th}$ of '5', for the kitchen table setting benchmark.

| | HCPlan | | | HCPlan-Reactive | | |
|---|---|---|---|---|---|---|
| $Scen$ | $N$ | $LF$ | $T$ [sec] | $N$ | $LF$ | $T$ [sec] |
| 1 | 32 | 3 | 12.56 | 32 | 3 | 12.78 |
| 2 | 36 | 4 | 12.45 | 33 | 3 | 12.75 |
| 3 | 36 | 4 | 15.51 | 33 | 3 | 15.02 |
| 4 | 50 | 7 | 11.34 | 30 | 3 | 11.24 |
| 5 | 84 | 8 | 35.75 | 41 | 4 | 32.57 |
| 6 | 109 | 11 | 29.82 | 57 | 5 | 24.21 |
| 7 | 64 | 7 | 22.67 | 44 | 4 | 18.23 |
| 8 | 45 | 5 | 34.2 | 41 | 4 | 33.39 |
| 9 | 309 | 42 | 24.95 | 114 | 13 | 18.57 |
| 10 | 100 | 16 | 37.7 | 42 | 4 | 35.04 |
| 11 | 177 | 24 | 26.29 | 112 | 14 | 24.77 |
| 12 | 254 | 26 | 77.07 | 219 | 19 | 77.59 |
| 13 | 119 | 16 | 35.04 | 85 | 8 | 30.77 |
| 14 | 123 | 18 | 72.78 | 68 | 7 | 75.22 |
| 15 | 191 | 27 | 79.32 | 97 | 10 | 68.98 |
| 16 | 282 | 34 | 152.24 | 127 | 12 | 144.56 |
| 17 | 309 | 50 | 40.36 | 94 | 12 | 38.51 |
| 18 | 530 | 76 | 31.47 | 249 | 30 | 22.08 |
| 19 | 863 | 137 | 196 | 318 | 41 | 185.98 |
| 20 | 1325 | 206 | 129.89 | 470 | 49 | 101.03 |
| 21 | 1281 | 201 | 238.57 | 264 | 26 | 215.74 |
| 22 | 1928 | 320 | 89.02 | 171 | 27 | 30.96 |
| 23 | 2138 | 308 | 265.71 | 316 | 31 | 203.45 |
| 24 | 4869 | 776 | 269.14 | 574 | 66 | 112.33 |
| 25 | 7392 | 1129 | 362.51 | 913 | 92 | 80 |

significantly. Figure 7.2 shows a partial hybrid conditional plan for a sample kitchen table setting benchmark instance computed by HCPlan-Reactive where reactive depth threshold $D_{th}$ is set to '5'.

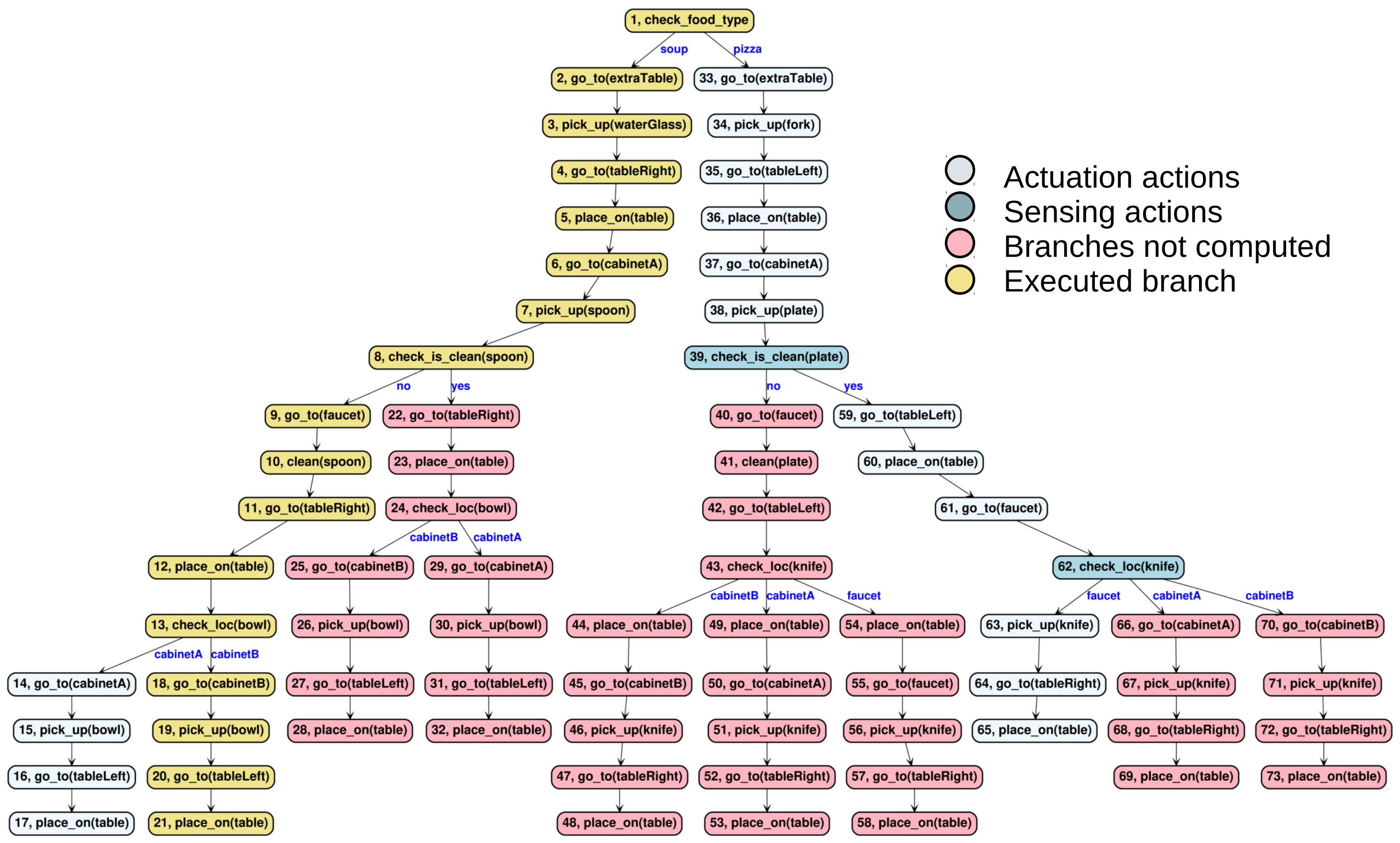


Figure 7.3: A reactive hybrid conditional partial plan computed for an instance of the kitchen table setting benchmark where reactive depth threshold $D_{th}$ is set to '5'.

## 7.10 Comparison between plan execution monitoring and HCPlan-Reactive

Plan execution monitoring and HCPlan-Reactive not only compute the plan to reach the goal state but also return the executed branch/plan followed by the robot. Therefore, comparing the executed plans is possible. Tables 7.11 and 7.12 show the comparison between the plans executed by plan execution monitoring and the branch executed by HCPlan-Reactive with sensing step/reactive depth threshold $D_{th}$ of '3' and '5', respectively, to reach the goal for the kitchen table setting benchmark instances. For HCPlan-Reactive, we report the length of executed branch $B_{exec}$ along with the number of sensing actions $S$ and actuation action $A$ in it, re-planning attempts $R$ and planning time $T$ in seconds. For plan execution monitoring, the computation time of the plan $T$ in seconds, the number $R$ of re-planning attempts, and the total plan length $L$ to reach the goal are reported. Since, each instance of benchmark domains is solver using plan execution monitoring '5' times, the maximum $(.)_{max}$ and the average $(.)_{av}$ values are listed in the table.

Tables 7.13 and 7.14 show that the plans obtained from HCPlan even have shorter plan length compared to plan execution monitoring. Therefore, HCPlan-Reactive is more suited to generate the plans in service robotics application than plan execution monitoring, where the actions performed by robots are expensive and have a cost associated with them. Moreover, re-planning for new plans represent halt in execution and we typically want re-planning time to be as minimum as possible. This is especially true in service robotics applications as re-planning represents an idle time for robots. The number of re-planning attempts is significantly less for HCPlan-Reactive

Table 7.13: Comparison between plan execution monitoring and HCPlan-Reactive with the sensing step/adaptive depth threshold of '3' for the kitchen table setting domain.

| | Plan Execution Monitoring | | | HCPlan-Reactive | | |
|---|---|---|---|---|---|---|
| *Scen* | $L_{max}(L_{av})$ | $R_{max}(R_{av})$ | $T_{max}(T_{avg})$ [sec] | $B_{exec}(A+S)$ | $R$ | $T$ [sec] |
| 01 | 16 (14.8) | 1 (0.6) | 10.29 (8.8) | 11 (10+1) | 1 | 12.79 |
| 02 | 19 (17.2) | 2 (1.4) | 15.44 (13.49) | 11 (10+1) | 1 | 12.48 |
| 03 | 22 (18.4) | 3 (1.8) | 25.35 (16.03) | 11 (10+1) | 1 | 14.94 |
| 04 | 22 (20.2) | 3 (2.4) | 27.72 (19.56) | 14 (13+1) | 1 | 11.39 |
| 05 | 24 (19.8) | 3 (1.6) | 54.45 (30.81) | 13 (11+2) | 2 | 32.74 |
| 06 | 23 (21.8) | 3 (2.6) | 35.93 (27.32) | 14 (11+3) | 2 | 24.04 |
| 07 | 30 (25.4) | 5 (3.6) | 63.39 (43.87) | 11 (10+1) | 1 | 18.82 |
| 08 | 30 (26.8) | 5 (4) | 83.97 (64.54) | 12 (10+2) | 2 | 34.24 |
| 09 | 26 (22.8) | 4 (3.2) | 34.36 (27.97) | 15 (11+4) | 2 | 19.51 |
| 10 | 30 (28.6) | 5 (4.6) | 74.87 (50.09) | 15 (13+2) | 2 | 35.58 |
| 11 | 32 (29.6) | 6 (5.2) | 57.85 (37.19) | 18 (15+3) | 3 | 25.01 |
| 12 | 30 (27) | 5 (4) | 77.36 (52.73) | 19 (17+2) | 2 | 70.22 |
| 13 | 32 (29.6) | 6 (5.2) | 90.43 (53.87) | 12 (10+2) | 1 | 30.44 |
| 14 | 36 (30.6) | 7 (5.2) | 75.04 (54.39) | 23 (19+4) | 3 | 68.33 |
| 15 | 38 (32) | 8 (6) | 90.11 (52.16) | 16 (13+3) | 3 | 64.31 |
| 16 | 36 (32.4) | 7 (5.8) | 277.76 (142.21) | 15 (13+2) | 2 | 145.87 |
| 17 | 40 (34) | 9 (7) | 80.12 (48.61) | 21 (18+3) | 3 | 33.2 |
| 18 | 40 (37.6) | 9 (8.2) | 73.52 (58.56) | 17 (13+4) | 3 | 21.71 |
| 19 | 38 (34) | 8 (6.4) | 149.77 (94.31) | 16 (11+5) | 2 | 173.78 |
| 20 | 39 (37.6) | 8 (7.6) | 163.27 (86.32) | 18 (14+4) | 3 | 87.14 |
| 21 | 41 (37) | 9 (7.4) | 140.69 (103.5) | 22 (17+5) | 3 | 187.16 |
| 22 | 43 (35.8) | 10 (7.6) | 87.63 (68.95) | 18 (13+5) | 3 | 25.27 |
| 23 | 44 (41.8) | 10 (9.2) | 177.57 (94.23) | 22 (17+5) | 4 | 162.19 |
| 24 | 48 (39.4) | 11 (8.2) | 113.31 (74.68) | 17 (12+5) | 3 | 96.6 |
| 25 | 51 (43.6) | 12 (9.8) | 150.24 (103.68) | 25 (19+6) | 4 | 93.77 |

Table 7.14: Comparison between plan execution monitoring and HCPlan-Reactive with the sensing step/adaptive depth threshold of '5' for the kitchen table setting domain.

| | Plan Execution Monitoring | | | HCPlan-Reactive | | |
|---|---|---|---|---|---|---|
| $Scen$ | $L_{max}(L_{av})$ | $R_{max}(R_{av})$ | $T_{max}(T_{avg})$ [sec] | $B_{exec}(A+S)$ | $R$ | $T$ [sec] |
| 01 | 18 (15) | 1 (0.4) | 10.41 (8.26) | 11 (10+1) | 1 | 12.78 |
| 02 | 23 (17) | 2 (0.8) | 14.16 (10.19) | 11 (10+1) | 1 | 12.75 |
| 03 | 28 (20) | 3 (1.4) | 27.43 (15.26) | 11 (10+1) | 1 | 15.02 |
| 04 | 23 (21) | 2 (1.6) | 21.41 (17.53) | 14 (13+1) | 1 | 11.24 |
| 05 | 25 (23) | 2 (1.6) | 48.38 (33.66) | 13 (11+2) | 2 | 32.57 |
| 06 | 34 (28) | 4 (2.8) | 46.48 (30.35) | 14 (11+3) | 1 | 24.21 |
| 07 | 40 (33) | 5 (3.6) | 62.34 (37.87) | 11 (10+1) | 1 | 18.23 |
| 08 | 39 (33.8) | 5 (3.8) | 83.65 (70.57) | 12 (10+2) | 2 | 33.39 |
| 09 | 33 (31) | 4 (3.6) | 31.88 (25.37) | 15 (11+4) | 2 | 18.57 |
| 10 | 45 (39) | 6 (4.8) | 94.97 (64.56) | 15 (13+2) | 2 | 35.04 |
| 11 | 39 (34) | 5 (3.8) | 89.75 (58.05) | 18 (15+3) | 2 | 24.77 |
| 12 | 50 (38.8) | 5 (4.2) | 240.66 (112.08) | 19 (17+2) | 2 | 77.59 |
| 13 | 54 (40) | 8 (5.2) | 74.19 (42.05) | 12 (10+2) | 1 | 30.77 |
| 14 | 54 (44.8) | 8 (6) | 92.96 (67.19) | 23 (19+4) | 3 | 75.22 |
| 15 | 54 (43) | 8 (5.8) | 69.98 (48.77) | 16 (13+3) | 2 | 68.98 |
| 16 | 55 (43.8) | 7 (5.6) | 74.44 (55.83) | 15 (13+2) | 1 | 144.56 |
| 17 | 58 (50) | 8 (7) | 67.71 (56.25) | 21 (18+3) | 2 | 38.51 |
| 18 | 58 (44) | 9 (6.2) | 83.4 (52.74) | 17 (13+4) | 2 | 22.08 |
| 19 | 60 (52) | 8 (7) | 163.35 (84.92) | 16 (11+5) | 2 | 185.98 |
| 20 | 65 (50) | 9 (6.8) | 174.09 (109.24) | 18 (14+4) | 2 | 101.03 |
| 21 | 65 (55) | 9 (7.8) | 137.63 (99.83) | 22 (17+5) | 3 | 215.74 |
| 22 | 64 (55.2) | 10 (8.4) | 80.29 (60.95) | 18 (13+5) | 3 | 30.96 |
| 23 | 60 (54.2) | 9 (7.8) | 123.07 (88.25) | 22 (17+5) | 3 | 203.45 |
| 24 | 80 (68.8) | 12 (10.6) | 189.01 (90.59) | 17 (12+5) | 3 | 112.33 |
| 25 | 59 (56) | 9 (8.4) | 234.59 (127.63) | 25 (19+6) | 3 | 80 |

than plan execution monitoring, in fact, it does not exceed '4' in any of the benchmark instance. For example, in  Instance 25 of Table 7.13, HCPlan-Reactive has to re-plan '4' times while plan execution monitoring has to perform re-planning on average 9.8 times (a maximum of 12 in the worst case).

# Chapter VIII

## 8 Execution of Hybrid Conditional Plans

In Chapter 4, we have discussed the computation of hybrid conditional plans for service robotic domains. In this chapter, we discuss the execution of these plans. We show dynamic and physical execution of such plans in complex mobile manipulation domain, like the kitchen table setting benchmark that was introduced in Section 6.1. Section 8.1 shows the dynamic simulation of a hybrid conditional plan with a bi-manual mobile manipulator robot CoCoA while, Section 8.2 shows physical execution of a plan with a bi-manual stationary manipulator Baxter for the kitchen table setting benchmark domain.

### 8.1 Dynamic simulation

In order to demonstrate the dynamic simulation of a hybrid conditional plan computed for a kitchen table setting benchmark instance, we use OPEN-RAVE [77] simulation environment. We integrate feasibility checks like graspability, reachability, and collision checks during the planning phase in order to ensure plan executability. The hybrid conditional plan generated by HC-PLAN for an instance of kitchen table setting scenario is shown in Figure 8.1.

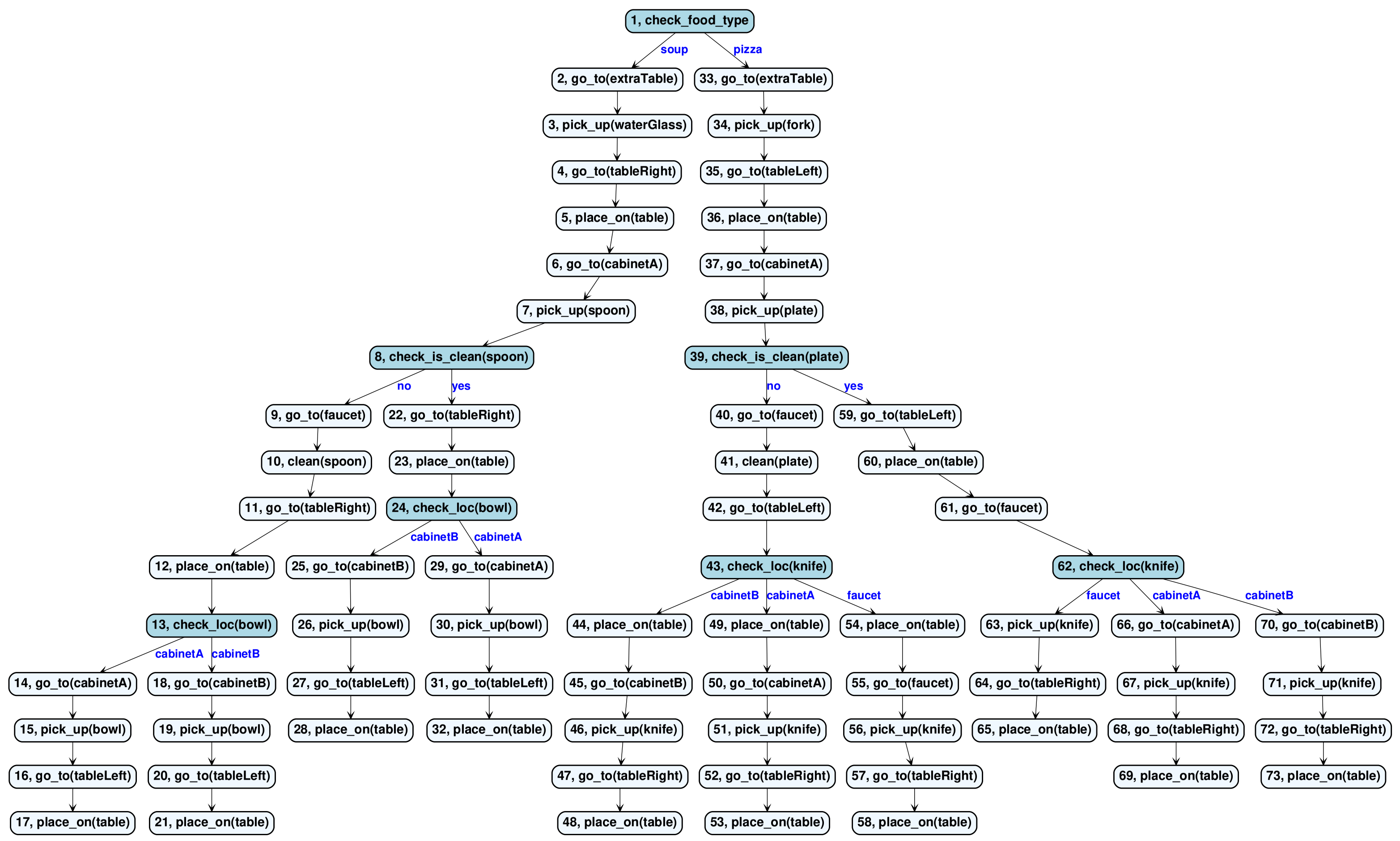
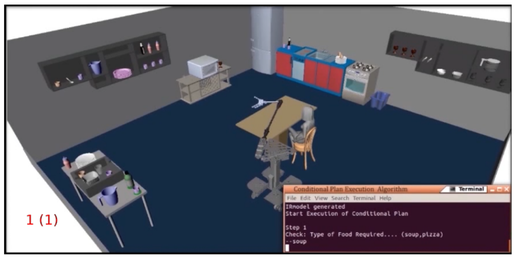


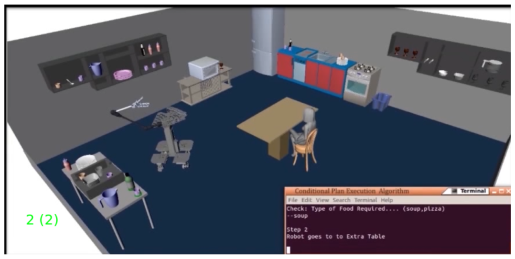
Figure 8.1: A hybrid conditional plan for an instance of kitchen table setting benchmark instance using a single arm mobile manipulator robot.

In the simulation, the robot executes the plan from the root node and depending upon the outcome of sensing action branch ‘2’ of the plan in Figure 8.1 is executed. Snapshots showing the dynamic simulation using a single arm mobile manipulator robot, CoCoA is shown in Figure 8.2. Each snapshot is labeled as $x(y)$, where $x$ is the action number in the branch and $y$ represents the actual node number in the plan. Moreover, the green color represents an actuation action while red represents a sensing action. Figure 8.2 shows the dynamic simulation of branch ‘2’ of the hybrid conditional plan (shown in Figure 8.1) of the kitchen table setting scenario by the CoCoA robot: (a) robot inquires the person about the type of food and the person answers soup (b) robot goes to the extra table (c) robot picks up the water glass (d) robot goes to right side of the table (e) robot places the water glass on the table (f) robot goes to the cabinet A (g) robot picks up the spoon (h) robot checks cleanliness of the spoon and finds that it is not clean (i) robot goes to the faucet (j) robot cleans the spoon (k) robot goes to right side of the table (l) robot places the spoon on the table (m) robot looks for the bowl and finds it at the cabinet B (n) robot goes to the cabinet B (o) robot picks up the bowl (p) robot goes to right side of the table (q) robot places the bowl on the table (r) kitchen table is set for soup.


1 (1)
Conditional Plan Execution Algorithm
Terminal
File Edit View Search Terminal Help

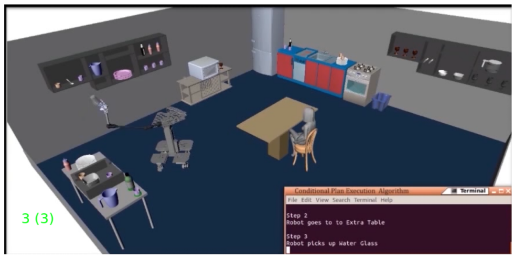
```
IRmodel generated
Start Execution of Conditional Plan

Step 1
Check: Type of Food Required.... (soup,pizza)
--soup
```


2 (2)
Conditional Plan Execution Algorithm
Terminal
File Edit View Search Terminal Help

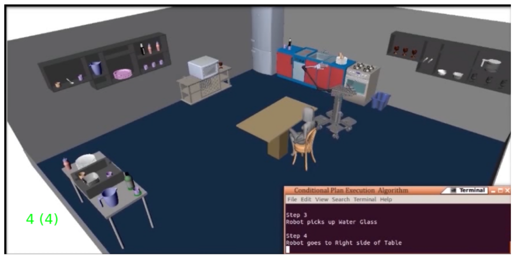
```
Check: Type of Food Required.... (soup,pizza)
--soup

Step 2
Robot goes to to Extra Table
```


3 (3)
Conditional Plan Execution Algorithm
Terminal
File Edit View Search Terminal Help

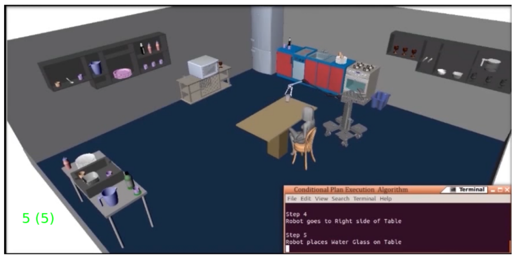
```
Step 2
Robot goes to to Extra Table

Step 3
Robot picks up Water Glass
```


4 (4)
Conditional Plan Execution Algorithm
Terminal
File Edit View Search Terminal Help

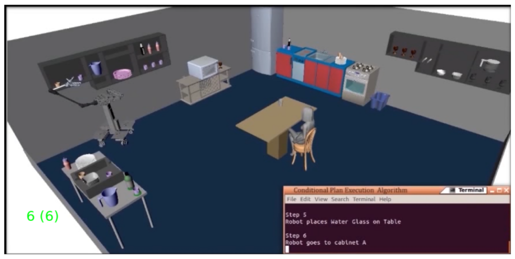
```
Step 3
Robot picks up Water Glass

Step 4
Robot goes to Right side of Table
```


5 (5)
Conditional Plan Execution Algorithm
Terminal
File Edit View Search Terminal Help

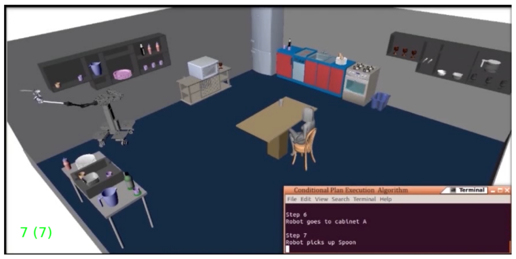
```
Step 4
Robot goes to Right side of Table

Step 5
Robot places Water Glass on Table
```


6 (6)
Conditional Plan Execution Algorithm
Terminal
File Edit View Search Terminal Help

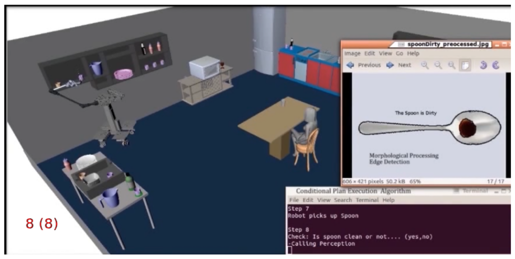
```
Step 5
Robot places Water Glass on Table

Step 6
Robot goes to cabinet A
```


7 (7)
Conditional Plan Execution Algorithm
Terminal
File Edit View Search Terminal Help

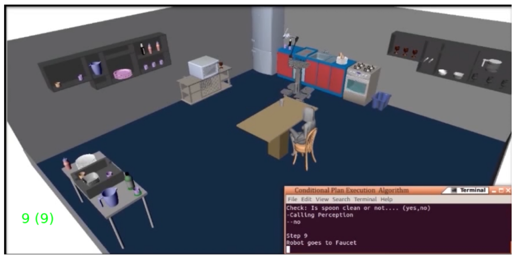
```
Step 6
Robot goes to cabinet A

Step 7
Robot picks up Spoon
```


8 (8)
spoonDirty_preocessed.jpg
Image Edit View Go Help
Previous
Next
The Spoon is Dirty
Morphological Processing
Edge Detection
606 × 421 pixels 50.2 kB 65%
17 / 17
Conditional Plan Execution Algorithm
Terminal
File Edit View Search Terminal Help

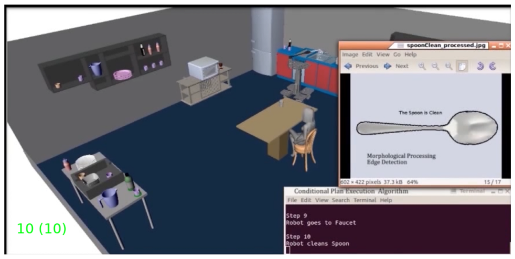
```
Step 7
Robot picks up Spoon

Step 8
Check: Is spoon clean or not.... (yes,no)
-Calling Perception
```


9 (9)
Conditional Plan Execution Algorithm
Terminal
File Edit View Search Terminal Help

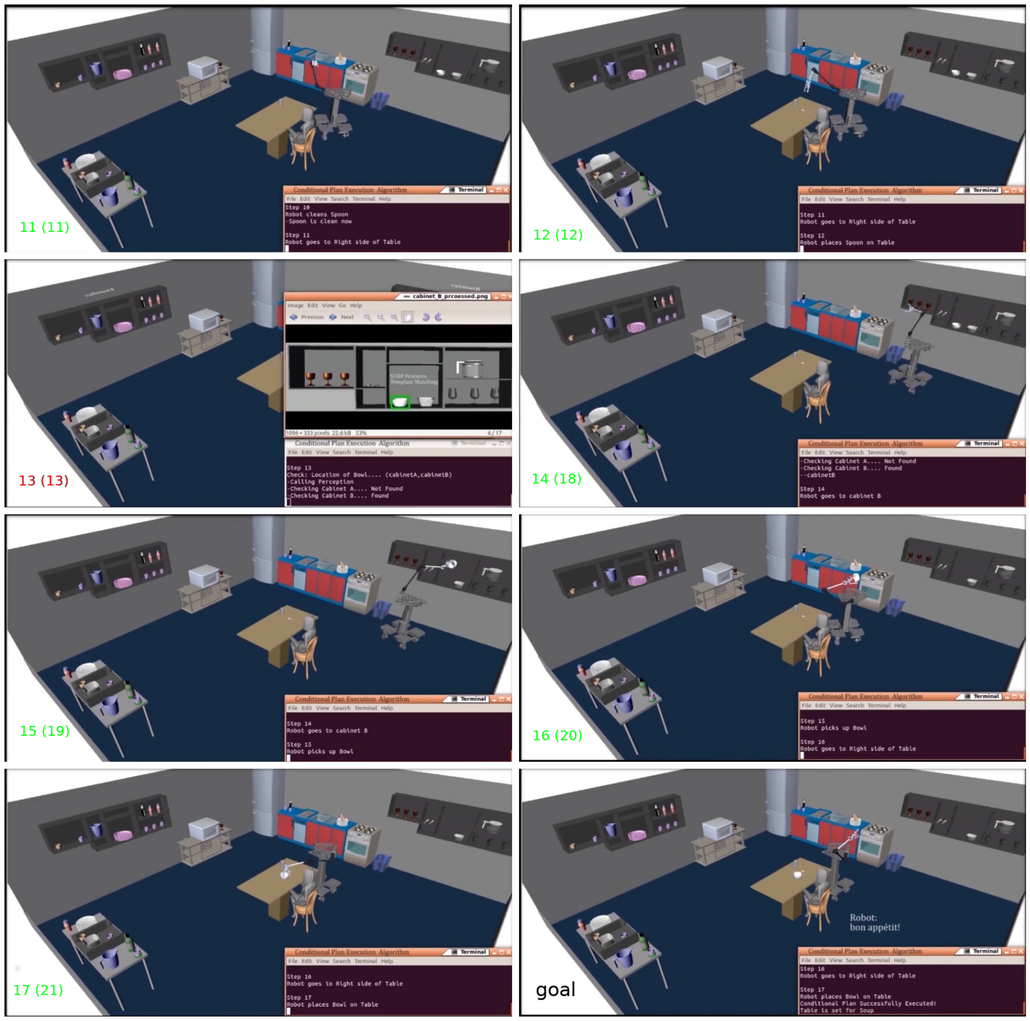
```
Check: Is spoon clean or not.... (yes,no)
-Calling Perception
--no

Step 9
Robot goes to Faucet
```



```
Step 9
Robot goes to Faucet

Step 10
Robot cleans Spoon
```



```
Step 10
Robot cleans Spoon
-Spoon is clean now

Step 11
Robot goes to Right side of Table
```

```
Step 11
Robot goes to Right side of Table

Step 12
Robot places Spoon on Table
```

```
Step 13
Check: Location of Bowl.... (cabinetA,cabinetB)
-Calling Perception
-Checking Cabinet A.... Not Found
-Checking Cabinet B.... Found
```

```
-Checking Cabinet A.... Not Found
-Checking Cabinet B.... Found
--cabinetB

Step 14
Robot goes to cabinet B
```

```
Step 14
Robot goes to cabinet B

Step 15
Robot picks up Bowl
```

```
Step 15
Robot picks up Bowl

Step 16
Robot goes to Right side of Table
```

```
Step 16
Robot goes to Right side of Table

Step 17
Robot places Bowl on Table
```

```
Step 16
Robot goes to Right side of Table

Step 17
Robot places Bowl on Table
Conditional Plan Successfully Executed!
Table is set for Soup
```

Figure 8.2: Dynamic simulation showing the robot setting up the kitchen table for ‘soup’.

## 8.2 Physical execution

In order to demonstrate the physical execution of a hybrid conditional plan on kitchen table setting benchmark introduced in Section 6.1 we use a bi-manual mobile manipulator robot, Baxter. We use ROS [90] and OMPL [84] to generate and execute motion plans for Baxter robot. The hybrid conditional plan generated by HCPlan for an instance of kitchen table setting scenario for the Baxter is shown in Figure 8.3. The robot executes the plan from the root node and depending upon the outcome of sensing action different branches can be executed.

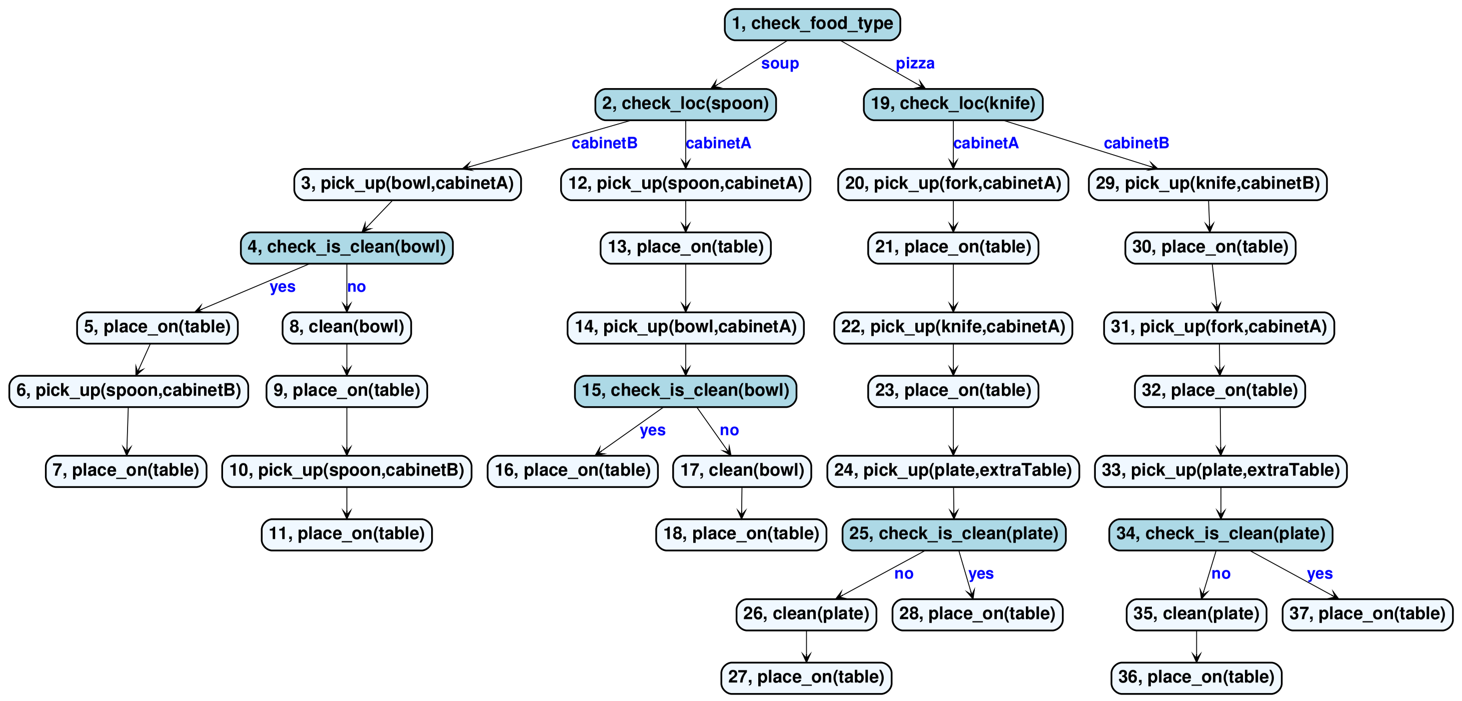


Figure 8.3: A hybrid conditional plan for an instance of the kitchen table setting benchmark instance using a bi-manual manipulator robot.

Figure 8.4 show snapshots of execution of ‘1’ branch of hybrid conditional plan shown in Figure 8.3 by Baxter robot: (a) robot inquires the person about the type of food and the person answers soup (b) robot looks for the spoon and finds it at the cabinet B (c) robot picks up the bowl from the cabinet A (d) robot checks cleanliness of the bowl and finds it is clean (e) robot places the bowl on the table (f) robot picks the spoon from the cabinet B (g) robot places the spoon on the table (h) kitchen table is set for soup.

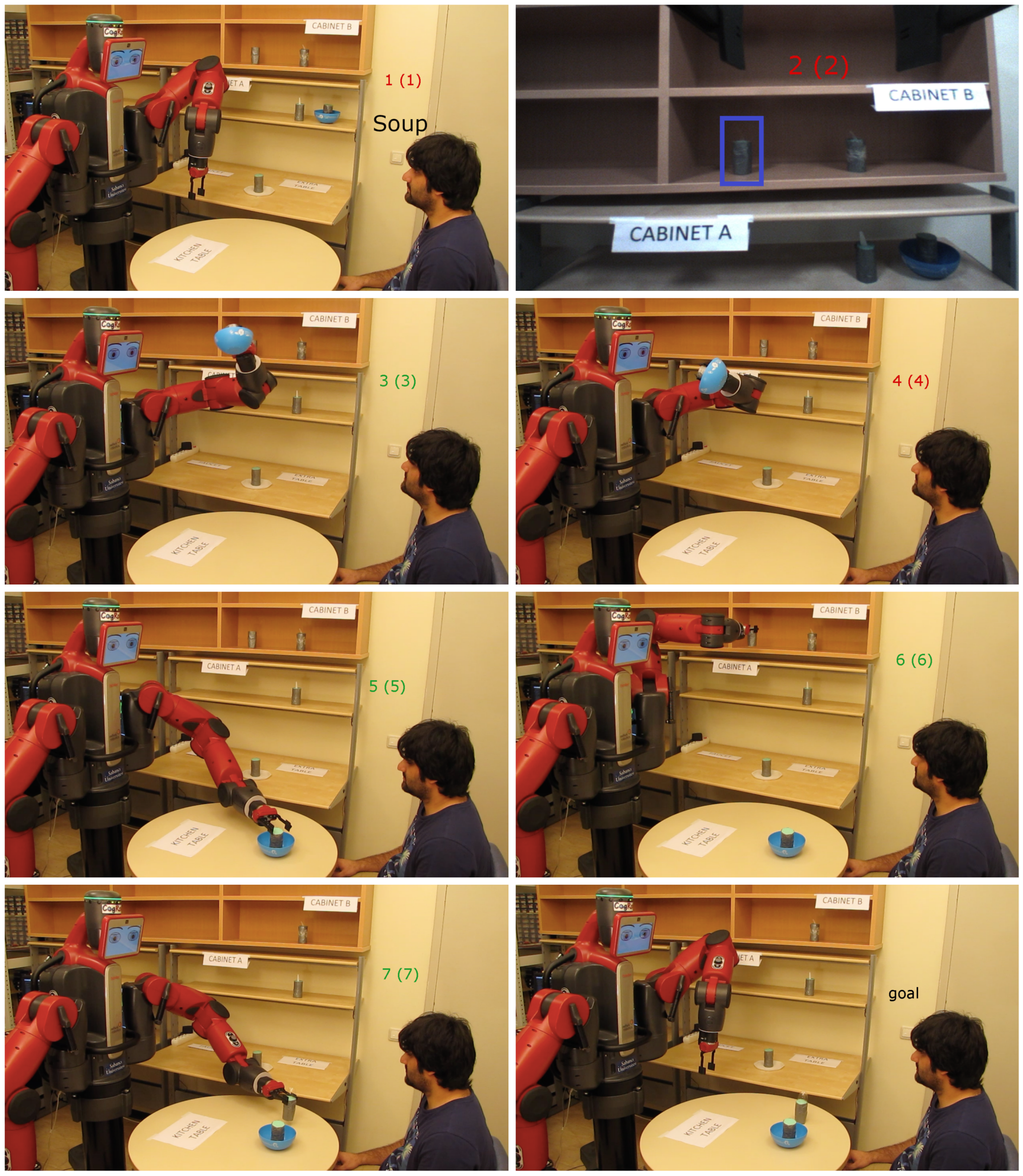


Figure 8.4: Physical execution showing the robot setting up the kitchen table for ‘soup’.

Figure 8.4 show snapshots of execution of '6' branch of hybrid conditional plan shown in Figure 8.3 by Baxter robot: (a) robot inquires person about the type of food and the person answers pizza (b) robot looks for the fork and finds it at the cabinet A (c) robot picks up the knife from the cabinet A (d) robot places the knife on the table (e) robot picks the fork from the cabinet A (f) robot places the fork on the table (g) robot picks the plate from the extra table (h) robot checks cleanliness of the plate and finds that it is clean (i) robot places the plate on the table (j) kitchen table is set for pizza.

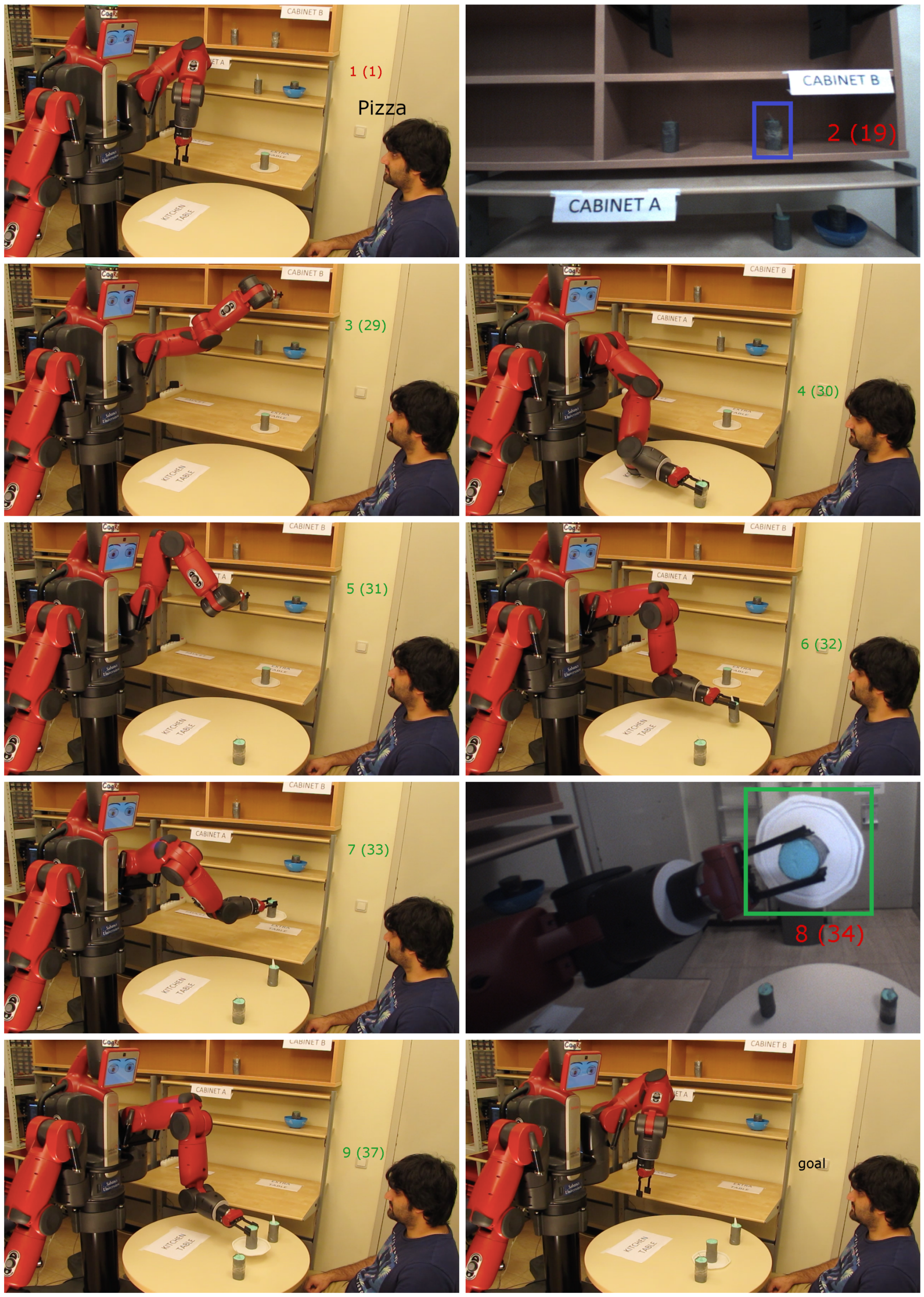


Figure 8.5: Physical execution showing the robot setting up the kitchen table for ‘pizza’.

# Chapter XI

## 9 Conclusions

We have introduced a novel hybrid conditional planning framework that extends hybrid sequential planning with non-deterministic sensing actions and utilizes this extension to compute the branches of a hybrid conditional plan. The planner HCPlan is a compilation based hybrid conditional planner that utilizes parallel computation of branches to speed up the computation task. We have provided formal definitions of sensing and actuation actions using action language $\mathcal{C}+$ and introduced a mechanism to develop service robotic benchmark for planning under incomplete knowledge and partial observability. We have developed real-world benchmarks that involve manipulation and navigation tasks by service robots.

We have evaluated the benchmarks using our hybrid conditional planning framework and show that our approach is complete with respect to contingencies considered during the planning phase. We evaluate the effectiveness of parallel computation of branches and re-usability of the branches and showed them as a useful extension to our planning framework. We have demonstrated the importance of integration of feasibility checks in hybrid conditional planning framework, especially for service robotics domains, by providing an experimental evaluation of the successful execution of plans.

Furthermore, we have compared our framework to other planning under uncertainty approaches like the plan execution monitoring approach using a benchmark domain and showed that our approach significantly outperforms this alternative in terms of plan lengths. In that sense, computing a hybrid conditional plan in advance may be more preferable for applications where execution of the robotic actions is costly and/or where re-planning is not desired during the execution phase. We also have compared HCPlan to another state of the art hybrid conditional planner HCP-ASP, by translation of the kitchen table setting benchmark domain to the language of ASP and showed that both approaches are complete with respect to the contingencies considered during the planning phase.

We have introduced anytime and reactive versions of our hybrid conditional planning framework by implementing HCPlan-Anytime and HCPlan-Reactive. These extensions compute partial hybrid conditional plans that are less computationally demanding, making them useful in real-world applications where the robots can perform the planning and execution simultaneously.

We have provided dynamic simulations and physical executions of several instances of the kitchen table setting benchmark using a service robot. These demonstrations show that our hybrid conditional planning framework computes the plans that are feasible to be utilized in real-world applications. Physical executions and dynamic simulations of the hybrid conditional plans generated by our framework are available at `http://cogrobo.sabanciuniv.edu/?p=1126`.

# References

[1] P. Haslum, "Admissible heuristics for automated planning by," 2006.

[2] K. Erol, D. S. Nau, and V. S. Subrahmanian, "Complexity, decidability and undecidability results for domain-independent planning," *Artif. Intell.*, vol. 76, no. 1-2, pp. 75–88, 1995. [Online]. Available: https://doi.org/10.1016/0004-3702(94)00080-K

[3] D. H. D. Warren, "Generating conditional plans and programs," in *AISB (ECAI)*, 1976, pp. 344–354.

[4] M. A. Peot and D. E. Smith, "Conditional nonlinear planning," in *Proceedings of the First International Conference on Artificial Intelligence Planning Systems.* San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1992, pp. 189–197. [Online]. Available: http://dl.acm.org/citation.cfm?id=139492.139518

[5] L. Pryor and G. Collins, "Planning for contingencies: A decision-based approach," *J. Artif. Int. Res.*, vol. 4, no. 1, pp. 287–339, May 1996. [Online]. Available: http://dl.acm.org/citation.cfm?id=1622737.1622749

[6] A. Nouman, I. F. Yalciner, E. Erdem, and V. Patoglu, "Experimental evaluation of hybrid conditional planning for service robotics," in *International Symposium on Experimental Robotics, ISER 2016, Tokyo, Japan, October 3-6, 2016.*, 2016, pp. 692–702. [Online]. Available: https://doi.org/10.1007/978-3-319-50115-4_60

[7] I. F. Yalciner, A. Nouman, V. Patoglu, and E. Erdem, "Hybrid

conditional planning using answer set programming," *TPLP*, vol. 17, no. 5-6, pp. 1027–1047, 2017. [Online]. Available: https://doi.org/10.1017/S1471068417000321

[8] C. Baral, V. Kreinovich, and R. Trejo, "Computational complexity of planning and approximate planning in presence of incompleteness," in *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, IJCAI 99, Stockholm, Sweden, July 31 - August 6, 1999. 2 Volumes, 1450 pages*, 1999, pp. 948–955. [Online]. Available: http://ijcai.org/Proceedings/99-2/Papers/040A.pdf

[9] 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 *IEEE International Conference on Robotics and Automation, ICRA 2011, Shanghai, China, 9-13 May 2011*, 2011, pp. 4575–4581. [Online]. Available: https://doi.org/10.1109/ICRA.2011.5980160

[10] A. Hertle, C. Dornhege, T. Keller, and B. Nebel, "Planning with semantic attachments: An object-oriented view," in *ECAI 2012 - 20th European Conference on Artificial Intelligence. Including Prestigious Applications of Artificial Intelligence (PAIS-2012) System Demonstrations Track, Montpellier, France, August 27-31 , 2012*, 2012, pp. 402–407. [Online]. Available: https://doi.org/10.3233/978-1-61499-098-7-402

[11] E. Plaku, "Planning in discrete and continuous spaces: From LTL tasks to robot motions," in *Advances in Autonomous Robotics - Joint Proceedings of the 13th Annual TAROS Conference and*

*the 15th Annual FIRA RoboWorld Congress, Bristol, UK, August 20-23, 2012*, 2012, pp. 331–342. [Online]. Available: https://doi.org/10.1007/978-3-642-32527-4_30

[12] L. P. Kaelbling and T. Lozano-Pérez, "Integrated task and motion planning in belief space," *I. J. Robotics Res.*, vol. 32, no. 9-10, pp. 1194–1227, 2013. [Online]. Available: https://doi.org/10.1177/0278364913484072

[13] S. Srivastava, E. Fang, L. Riano, R. Chitnis, S. J. Russell, and P. Abbeel, "Combined task and motion planning through an extensible planner-independent interface layer," in *2014 IEEE International Conference on Robotics and Automation, ICRA 2014, Hong Kong, China, May 31 - June 7, 2014*, 2014, pp. 639–646. [Online]. Available: https://doi.org/10.1109/ICRA.2014.6906922

[14] F. Lagriffoul, D. Dimitrov, J. Bidot, A. Saffiotti, and L. Karlsson, "Efficiently combining task and motion planning using geometric constraints," *I. J. Robotics Res.*, vol. 33, no. 14, pp. 1726–1747, 2014. [Online]. Available: https://doi.org/10.1177/0278364914545811

[15] D. Hadfield-Menell, E. Groshev, R. Chitnis, and P. Abbeel, "Modular task and motion planning in belief space," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2015, Hamburg, Germany, September 28 - October 2, 2015*, 2015, pp. 4991–4998. [Online]. Available: https://doi.org/10.1109/IROS.2015.7354079

[16] F. Lagriffoul and B. Andres, "Combining task and motion planning: A culprit detection problem," *I. J. Robotics Res.*, vol. 35, no. 8,

pp. 890–927, 2016. [Online]. Available: https://doi.org/10.1177/0278364915619022

[17] E. Erdem, V. Patoglu, and P. Schüller, "A systematic analysis of levels of integration between high-level task planning and low-level feasibility checks," *AI Commun.*, vol. 29, no. 2, pp. 319–349, 2016. [Online]. Available: https://doi.org/10.3233/AIC-150697

[18] J. Bidot, L. Karlsson, F. Lagriffoul, and A. Saffiotti, "Geometric backtracking for combined task and motion planning in robotic systems," *Artif. Intell.*, vol. 247, pp. 229–265, 2017. [Online]. Available: https://doi.org/10.1016/j.artint.2015.03.005

[19] F. Lagriffoul, N. T. Dantam, C. Garrett, A. Akbari, S. Srivastava, and L. E. Kavraki, "Platform-independent benchmarks for task and motion planning," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 3765–3772, 2018. [Online]. Available: https://doi.org/10.1109/LRA.2018.2856701

[20] S. J. Russell and P. Norvig, *Artificial Intelligence - A Modern Approach (3. internat. ed.)*. Pearson Education, 2010. [Online]. Available: http://vig.pearsoned.com/store/product/1,1207,store-12521_isbn-0136042597,00.html

[21] M. Ghallab, D. S. Nau, and P. Traverso, *Automated planning - theory and practice*. Elsevier, 2004.

[22] H. Geffner, "Non-classical planning with a classical planner: The power of transformations," in *Logics in Artificial Intelligence - 14th European Conference, JELIA 2014, Funchal, Madeira, Portugal, September*

*24-26, 2014. Proceedings*, 2014, pp. 33–47. [Online]. Available: https://doi.org/10.1007/978-3-319-11558-0_3

[23] M. Bjäreland, "Model-based execution monitoring," in *Linköping Studies in Science and Technology, Dissertation No 688, available at http://www. ida. liu. se/labs/kplab/people/marbj*, 2001.

[24] R. Fikes and N. J. Nilsson, "STRIPS: A new approach to the application of theorem proving to problem solving," *Artif. Intell.*, vol. 2, no. 3/4, pp. 189–208, 1971. [Online]. Available: https://doi.org/10.1016/0004-3702(71)90010-5

[25] R. Fikes, P. E. Hart, and N. J. Nilsson, "Learning and executing generalized robot plans," *Artif. Intell.*, vol. 3, no. 1-3, pp. 251–288, 1972. [Online]. Available: https://doi.org/10.1016/0004-3702(72)90051-3

[26] M. P. Georgeff and A. L. Lansky, "Reactive reasoning and planning," in *Proceedings of the 6th National Conference on Artificial Intelligence. Seattle, WA, USA, July 1987.*, 1987, pp. 677–682. [Online]. Available: http://www.aaai.org/Library/AAAI/1987/aaai87-121.php

[27] K. Chen, F. Yang, and X. Chen, "Planning with task-oriented knowledge acquisition for a service robot," in *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, 2016, pp. 812–818. [Online]. Available: http://www.ijcai.org/Abstract/16/120

[28] R. Janssen, E. van Meijl, D. D. Marco, R. van de Molengraft, and M. Steinbuch, "Integrating planning and execution for ROS enabled service robots using hierarchical action representations," in

*16th International Conference on Advanced Robotics, ICAR 2013, 25-29 November 2013, Montevideo, Uruguay*, 2013, pp. 1–7. [Online]. Available: https://doi.org/10.1109/ICAR.2013.6766556

[29] A. Küstenmacher, N. Akhtar, P. Plöger, and G. Lakemeyer, "Towards robust task execution for domestic service robots," *Journal of Intelligent and Robotic Systems*, vol. 76, no. 1, pp. 5–33, 2014. [Online]. Available: https://doi.org/10.1007/s10846-013-0005-6

[30] E. Shpieva and I. Awaad, "Integrating task planning, execution and monitoring for a domestic service robot," *it - Information Technology*, vol. 57, no. 2, pp. 112–121, 2015. [Online]. Available: https://doi.org/10.1515/itit-2014-1064

[31] E. Erdem, E. Aker, and V. Patoglu, "Answer set programming for collaborative housekeeping robotics: representation, reasoning, and execution," *Intelligent Service Robotics*, vol. 5, no. 4, pp. 275–291, 2012. [Online]. Available: https://doi.org/10.1007/s11370-012-0119-x

[32] M. Gianni, P. Papadakis, F. Pirri, M. Liu, F. Pomerleau, F. Colas, K. Zimmermann, T. Svoboda, T. Petricek, G. M. Kruijff, H. Khambhaita, and H. Zender, "A unified framework for planning and execution-monitoring of mobile robots," in *Automated Action Planning for Autonomous Mobile Robots, Papers from the 2011 AAAI Workshop, San Francisco, California, USA, August 7, 2011*, 2011. [Online]. Available: http://www.aaai.org/ocs/index.php/WS/AAAIW11/paper/view/3852

[33] E. Erdem, V. Patoglu, and Z. G. Saribatur, "Integrating hybrid

diagnostic reasoning in plan execution monitoring for cognitive factories with multiple robots," in *IEEE International Conference on Robotics and Automation, ICRA 2015, Seattle, WA, USA, 26-30 May, 2015*, 2015, pp. 2007–2013. [Online]. Available: https://doi.org/10.1109/ICRA.2015.7139461

[34] J. P. Mendoza, M. M. Veloso, and R. G. Simmons, "Plan execution monitoring through detection of unmet expectations about action outcomes," in *IEEE International Conference on Robotics and Automation, ICRA 2015, Seattle, WA, USA, 26-30 May, 2015*, 2015, pp. 3247–3252. [Online]. Available: https://doi.org/10.1109/ICRA.2015.7139646

[35] C. Fritz, "Execution monitoring – a survey – fall 2005," 2005.

[36] O. Pettersson, "Execution monitoring in robotics: A survey," *Robotics and Autonomous Systems*, vol. 53, no. 2, pp. 73–88, 2005. [Online]. Available: https://doi.org/10.1016/j.robot.2005.09.004

[37] E. J. Sondik, "The optimal control of partially observable markov processes," Ph.D. dissertation, Stanford University, 1971.

[38] ——, "The optimal control of partially observable markov processes over the infinite horizon: Discounted costs," *Operations Research*, vol. 26, no. 2, pp. 282–304, 1978. [Online]. Available: https://doi.org/10.1287/opre.26.2.282

[39] R. D. Smallwood and E. J. Sondik, "The optimal control of partially observable markov processes over a finite horizon," *Operations*

*Research*, vol. 21, no. 5, pp. 1071–1088, 1973. [Online]. Available: https://doi.org/10.1287/opre.21.5.1071

[40] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," *Artif. Intell.*, vol. 101, no. 1-2, pp. 99–134, 1998. [Online]. Available: https://doi.org/10.1016/S0004-3702(98)00023-X

[41] O. Madani, S. Hanks, and A. Condon, "On the undecidability of probabilistic planning and infinite-horizon partially observable markov decision problems," in *Proceedings of the Sixteenth National Conference on Artificial Intelligence and Eleventh Conference on Innovative Applications of Artificial Intelligence, July 18-22, 1999, Orlando, Florida, USA.*, 1999, pp. 541–548. [Online]. Available: http://www.aaai.org/Library/AAAI/1999/aaai99-077.php

[42] ——, "On the undecidability of probabilistic planning and related stochastic optimization problems," *Artif. Intell.*, vol. 147, no. 1-2, pp. 5–34, 2003. [Online]. Available: https://doi.org/10.1016/S0004-3702(02)00378-8

[43] C. Muise, V. Belle, and S. A. McIlraith, "Computing contingent plans via fully observable non-deterministic planning," in *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, ser. AAAI'14. AAAI Press, 2014, pp. 2322–2329. [Online]. Available: http://dl.acm.org/citation.cfm?id=2892753.2892874

[44] P. Haslum and P. Jonsson, "Some results on the complexity of planning with incomplete information," in *Recent Advances in AI*

*Planning, 5th European Conference on Planning, ECP'99, Durham, UK, September 8-10, 1999, Proceedings*, 1999, pp. 308–318. [Online]. Available: https://doi.org/10.1007/10720246_24

[45] H. Turner, "Polynomial-length planning spans the polynomial hierarchy," in *Logics in Artificial Intelligence, European Conference, JELIA 2002, Cosenza, Italy, September, 23-26, Proceedings*, 2002, pp. 111–124. [Online]. Available: https://doi.org/10.1007/3-540-45757-7_10

[46] J. Rintanen, "Complexity of planning with partial observability," in *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling (ICAPS 2004), June 3-7 2004, Whistler, British Columbia, Canada*, 2004, pp. 345–354. [Online]. Available: http://www.aaai.org/Library/ICAPS/2004/icaps04-041.php

[47] C. Baral, V. Kreinovich, and R. Trejo, "Computational complexity of planning and approximate planning in the presence of incompleteness," *Artif. Intell.*, vol. 122, no. 1-2, pp. 241–267, 2000. [Online]. Available: https://doi.org/10.1016/S0004-3702(00)00043-6

[48] A. Albore, H. Palacios, and H. Geffner, "A translation-based approach to contingent planning," in *Proceedings of the 21st International Jont Conference on Artifical Intelligence*, ser. IJCAI'09. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2009, pp. 1623–1628. [Online]. Available: http://dl.acm.org/citation.cfm?id=1661445.1661706

[49] B. Bonet and H. Geffner, "Planning under partial observability by classical replanning: Theory and experiments," in *IJCAI*

*2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, 2011, pp. 1936–1941. [Online]. Available: https://doi.org/10.5591/978-1-57735-516-8/IJCAI11-324

[50] R. I. Brafman and G. Shani, "Replanning in domains with partial information and sensing actions," *J. Artif. Intell. Res.*, vol. 45, pp. 565–600, 2012. [Online]. Available: https://doi.org/10.1613/jair.3711

[51] S. Maliah, R. I. Brafman, E. Karpas, and G. Shani, "Partially observable online contingent planning using landmark heuristics," in *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling, ICAPS 2014, Portsmouth, New Hampshire, USA, June 21-26, 2014*, 2014. [Online]. Available: http://www.aaai.org/ocs/index.php/ICAPS/ICAPS14/paper/view/7913

[52] R. Komarnitsky and G. Shani, "Computing contingent plans using online replanning," in *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA.*, 2016, pp. 3159–3165. [Online]. Available: http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12379

[53] J. Hoffmann and R. I. Brafman, "Contingent planning via heuristic forward search witn implicit belief states," in *Proceedings of the Fifteenth International Conference on Automated Planning and Scheduling (ICAPS 2005), June 5-10 2005, Monterey, California, USA*, 2005, pp. 71–80. [Online]. Available: http://www.aaai.org/Library/ICAPS/2005/icaps05-008.php

[54] D. Bryce, S. Kambhampati, and D. E. Smith, "Planning graph heuristics for belief space search," *J. Artif. Intell. Res.*, vol. 26, pp. 35–99, 2006. [Online]. Available: https://doi.org/10.1613/jair.1869

[55] R. P. A. Petrick and F. Bacchus, "A knowledge-based approach to planning with incomplete information and sensing," in *Proceedings of the Sixth International Conference on Artificial Intelligence Planning Systems, April 23-27, 2002, Toulouse, France*, 2002, pp. 212–222. [Online]. Available: http://www.aaai.org/Library/AIPS/2002/aips02-022.php

[56] ——, "Extending the knowledge-based approach to planning with incomplete information and sensing," in *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling (ICAPS 2004), June 3-7 2004, Whistler, British Columbia, Canada*, 2004, pp. 2–11. [Online]. Available: http://www.aaai.org/Library/ICAPS/2004/icaps04-005.php

[57] P. H. Tu, T. C. Son, and C. Baral, "Reasoning and planning with sensing actions, incomplete information, and static causal laws using answer set programming," *TPLP*, vol. 7, no. 4, pp. 377–450, 2007. [Online]. Available: https://doi.org/10.1017/S1471068406002948

[58] S. T. To, T. C. Son, and E. Pontelli, "Contingent planning as and/or forward search with disjunctive representation," in *Proceedings of the Twenty-First International Conference on International Conference on Automated Planning and Scheduling*, ser. ICAPS'11. AAAI Press, 2011, pp. 258–265. [Online]. Available: http://dl.acm.org/citation.cfm?id=3038485.3038519

[59] N. McCain and H. Turner, "Causal theories of action and change," in *Proceedings of the Fourteenth National Conference on Artificial Intelligence and Ninth Innovative Applications of Artificial Intelligence Conference, AAAI 97, IAAI 97, July 27-31, 1997, Providence, Rhode Island, USA.*, 1997, pp. 460–465. [Online]. Available: http://www.aaai.org/Library/AAAI/1997/aaai97-071.php

[60] B. Bonet and H. Geffner, "Flexible and scalable partially observable planning with linear translations," in *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada.*, 2014, pp. 2235–2241. [Online]. Available: http://www.aaai.org/ocs/index.php/AAAI/AAAI14/paper/view/8642

[61] R. P. Goldman and M. S. Boddy, "Expressive planning and explicit knowledge," in *Proceedings of the Third International Conference on Artificial Intelligence Planning Systems, Edinburgh, Scotland, May 29-31, 1996*, 1996, pp. 110–117. [Online]. Available: http://www.aaai.org/Library/AIPS/1996/aips96-014.php

[62] G. Brewka, T. Eiter, and M. Truszczynski, "Answer set programming: An introduction to the special issue," *AI Magazine*, vol. 37, no. 3, pp. 5–6, 2016. [Online]. Available: http://www.aaai.org/ojs/index.php/aimagazine/article/view/2669

[63] M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub, "Clingo = ASP + control: Preliminary report," *CoRR*, vol. abs/1405.3694, 2014. [Online]. Available: http://arxiv.org/abs/1405.3694

[64] E. Giunchiglia, J. Lee, V. Lifschitz, N. McCain, and H. Turner,

"Nonmonotonic causal theories," *Artificial Intelligence*, vol. 153, no. 1, pp. 49 – 104, 2004, logical Formalizations and Commonsense Reasoning. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S000437020300167X

[65] S. Cambon, R. Alami, and F. Gravot, "A hybrid approach to intricate motion, manipulation and task planning," *I. J. Robotics Res.*, vol. 28, no. 1, pp. 104–126, 2009. [Online]. Available: https://doi.org/10.1177/0278364908097884

[66] F. Gravot, A. Haneda, K. Okada, and M. Inaba, "Cooking for humanoid robot, a task that needs symbolic and geometric reasonings," in *Proceedings of the 2006 IEEE International Conference on Robotics and Automation, ICRA 2006, May 15-19, 2006, Orlando, Florida, USA*, 2006, pp. 462–467. [Online]. Available: https://doi.org/10.1109/ROBOT.2006.1641754

[67] F. Gravot, S. Cambon, and R. Alami, "asymov: A planner that deals with intricate symbolic and geometric problems," in *Robotics Research, The Eleventh International Symposium, ISRR, October 19-22, 2003, Siena, Italy*, 2003, pp. 100–110. [Online]. Available: https://doi.org/10.1007/11008941_11

[68] R. Lallement, "Symbolic and geometric planning for teams of robots and humans. (planification symbolique et géométrique pour des équipes de robots et d'humains)," Ph.D. dissertation, INSA Toulouse, France, 2016. [Online]. Available: https://tel.archives-ouvertes.fr/tel-01534323

[69] M. Colledanchise, D. Almeida, and P. Ögren, "Towards blended reactive

planning and acting using behavior trees," *CoRR*, vol. abs/1611.00230, 2016. [Online]. Available: http://arxiv.org/abs/1611.00230

[70] L. P. Kaelbling and T. Lozano-Pérez, "Hierarchical task and motion planning in the now," in *IEEE International Conference on Robotics and Automation, ICRA 2011, Shanghai, China, 9-13 May 2011*, 2011, pp. 1470–1477. [Online]. Available: https://doi.org/10.1109/ICRA.2011.5980391

[71] S. Stock, "Hierarchical hybrid planning for mobile robots," *KI*, vol. 31, no. 4, pp. 373–376, 2017. [Online]. Available: https://doi.org/10.1007/s13218-017-0507-7

[72] K. Hauser and J.-C. Latombe, "Integrating task and prm motion planning : Dealing with many infeasible motion planning queries," 2009.

[73] A. Gaschler, R. P. A. Petrick, M. Giuliani, M. Rickert, and A. Knoll, "KVP: A knowledge of volumes approach to robot task planning," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems, Tokyo, Japan, November 3-7, 2013*, 2013, pp. 202–208. [Online]. Available: https://doi.org/10.1109/IROS.2013.6696354

[74] N. T. Dantam, Z. K. Kingston, S. Chaudhuri, and L. E. Kavraki, "Incremental task and motion planning: A constraint-based approach," in *Robotics: Science and Systems XII, University of Michigan, Ann Arbor, Michigan, USA, June 18 - June 22, 2016*, 2016. [Online]. Available: http://www.roboticsproceedings.org/rss12/p02.html

[75] 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 *Logic Programming and Nonmonotonic Reasoning, 10th International Conference, LPNMR 2009, Potsdam, Germany, September 14-18, 2009. Proceedings*, 2009, pp. 342–354. [Online]. Available: https://doi.org/10.1007/978-3-642-04238-6_29

[76] R. W. Weyhrauch, "Prolegomena to a theory of mechanized formal reasoning," *Artif. Intell.*, vol. 13, no. 1-2, pp. 133–170, 1980. [Online]. Available: https://doi.org/10.1016/0004-3702(80)90015-6

[77] R. Diankov, "Automated construction of robotic manipulation programs," Ph.D. dissertation, Carnegie Mellon University, Robotics Institute, August 2010.

[78] M. Fox and D. Long, "PDDL2.1: an extension to PDDL for expressing temporal planning domains," *J. Artif. Intell. Res.*, vol. 20, pp. 61–124, 2003. [Online]. Available: https://doi.org/10.1613/jair.1129

[79] D. V. McDermott, "The 1998 AI planning systems competition," *AI Magazine*, vol. 21, no. 2, pp. 35–55, 2000. [Online]. Available: http://www.aaai.org/ojs/index.php/aimagazine/article/view/1506

[80] M. Ghallab, C. Knoblock, D. Wilkins, A. Barrett, D. Christianson, M. Friedman, C. Kwok, K. Golden, S. Penberthy, D. Smith, Y. Sun, and D. Weld, "Pddl - the planning domain definition language," 08 1998.

[81] T. L. Dean and M. S. Boddy, "An analysis of time-dependent planning," in *Proceedings of the 7th National Conference on Artificial Intelligence, St. Paul, MN, USA, August 21-26, 1988.*, 1988, pp. 49–54. [Online]. Available: http://www.aaai.org/Library/AAAI/1988/aaai88-009.php

[82] M. S. Boddy and T. L. Dean, "Solving time-dependent planning problems," in *Proceedings of the 11th International Joint Conference on Artificial Intelligence. Detroit, MI, USA, August 1989*, 1989, pp. 979–984. [Online]. Available: http://ijcai.org/Proceedings/89-2/Papers/021.pdf

[83] M. S. Boddy, "Anytime problem solving using dynamic programming," in *Proceedings of the 9th National Conference on Artificial Intelligence, Anaheim, CA, USA, July 14-19, 1991, Volume 2.*, 1991, pp. 738–743. [Online]. Available: http://www.aaai.org/Library/AAAI/1991/aaai91-115.php

[84] I. A. Sucan, M. Moll, and L. E. Kavraki, "The open motion planning library," *IEEE Robot. Automat. Mag.*, vol. 19, no. 4, pp. 72–82, 2012. [Online]. Available: https://doi.org/10.1109/MRA.2012.2205651

[85] J. Babb and J. Lee, "Cplus 2asp: Computing action language ${\cal C}$ + in answer set programming," in *Logic Programming and Nonmonotonic Reasoning, 12th International Conference, LPNMR 2013, Corunna, Spain, September 15-19, 2013. Proceedings*, 2013, pp. 122–134. [Online]. Available: https://doi.org/10.1007/978-3-642-40564-8_13

[86] M. Gebser, T. Schaub, and S. Thiele, "Gringo : A new grounder for answer set programming," in *Logic Programming and Nonmonotonic Reasoning, 9th International Conference, LPNMR 2007, Tempe, AZ, USA, May 15-17, 2007, Proceedings*, 2007, pp. 266–271. [Online]. Available: https://doi.org/10.1007/978-3-540-72200-7_24

[87] M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub, "*clasp*

: A conflict-driven answer set solver," in *Logic Programming and Nonmonotonic Reasoning, 9th International Conference, LPNMR 2007, Tempe, AZ, USA, May 15-17, 2007, Proceedings*, 2007, pp. 260–265. [Online]. Available: https://doi.org/10.1007/978-3-540-72200-7_23

[88] J. Wielemaker, T. Schrijvers, M. Triska, and T. Lager, "Swi-prolog," *TPLP*, vol. 12, no. 1-2, pp. 67–96, 2012. [Online]. Available: https://doi.org/10.1017/S1471068411000494

[89] N. Eén and N. Sörensson, "An extensible sat-solver," in *Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003. Santa Margherita Ligure, Italy, May 5-8, 2003 Selected Revised Papers*, 2003, pp. 502–518. [Online]. Available: https://doi.org/10.1007/978-3-540-24605-3_37

[90] M. Quigley, K. Conley, B. P Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y Ng, "Ros: an open-source robot operating system," vol. 3, 01 2009.