Constraint-based Methods for Human-aware Planning

Örebro Studies in Technology 72

Uwe Köckemann

Constraint-based Methods for Human-aware Planning

Supervisors: **Federico Pecora**
Lars Karlsson

Title: Constraint-based Methods for Human-aware Planning

Publisher: Örebro University, 2016
www.publications.oru.se

Printer: Örebro University, Repro 09/2016

ISSN 1650-8580
ISBN 978-91-7529-159-8

Abstract

Uwe Köckemann (2016): Constraint-based Methods for Human-aware Planning. Örebro Studies in Technology 72.

As more robots and sensors are deployed in work and home environments, there is a growing need for these devices to act with some degree of autonomy to fulfill their purpose. Automated planning can be used to synthesize plans of action that achieve this. The main challenge addressed in this thesis is to consider how the automated planning problem changes when considered in the context of environments that are populated by humans. Humans have their own plans, and automatically generated plans should not interfere with these. We refer to this as social acceptability. Opportunities for proactive behavior often arise during execution. The planner should be able to identify these opportunities and proactively plan accordingly. Both social acceptability and proactivity require the planner to identify relevant situations from available information. We refer to this capability as context-awareness, and it may require complex inferences based on observed human activities. Finally, planning may have to consider cooperation with humans to reach common goals or to enable robots and humans to support one another.

This thesis analyzes the requirements that emerge from human-aware planning — what it takes to make automated planning socially acceptable, proactive, context aware, and to make it support cooperation with humans. We formally state the human-aware planning problem, and propose a planning and execution framework for human-aware planning that is based on constraint reasoning and flaw-resolution techniques, and which fulfills the identified requirements. This approach is modular and extendable: new types of constraints can be added and solvers can be exchanged and re-arranged. This allows us to address the identified requirements for human-aware planning. In particular, we introduce Interaction Constraints (ICs) for this purpose, and propose patterns of Ics for social acceptability, proactivity, and context-awareness. We also consider cooperative plans in which certain actions are assigned to humans and the implications that this has. We evaluate the proposed methods and patterns on a series of use cases, as well as a variety of domains including a real-world robotic system. We evaluate the proposed methods and patterns on a series of use cases, as well as a variety of domains including a real-world robotic system. introduce Interaction Constraints (ICs) for this purpose, and propose patterns of ICs for social acceptability, proactivity, and context-awareness. We also consider cooperative plans in which certain actions are assigned to humans and the implications that this has. We evaluate the proposed methods and patterns on a series of use cases, as well as a variety of domains including a real-world robotic system.

Keywords: Task Planning, Constraint-based Planning, Human-aware Planning

Uwe Köckemann, School of Science and Technology
Örebro University, SE-701 82 Örebro, Sweden, uwe.kockemann@oru.se

Acknowledgements

```
;; I'm not one for big words so here you have some questionable code:
(thanks
  (to (and Lars Federico))
  (for (and lots-of-fun great-cooperation)) )
(special—thanks (to Alessandro) (for lucia-winterschool-2014) )
(thanks (to Amy) (for (employing (and Jenny me))) )
(thanks (to Rachid-Alami) (for reviewing-my-thesis) )
(thanks
  (to all@AASS)
  (for (and
    (making-a-great-place-to (and (work enjoy-life)))
    (games-of (and innebandy football))
    (nights-of (and movies boardgames party)) ) ) )
(geeky—thanks
  (to (and Fabien Marcello Tomek))
  (for (and the-foundation surprises helping-lester cthulhu-at-sea)) )
(thanks
  (to (and Lia Mathias))
  (for (and (lots—of (and games late-night-whisky)) all-the-creep)))
(thanks
  (to (and Marjan Fabien Lia))
  (for keeping-plants-alive))
(defun f (n) (+ n (f (+ n 1))))
(* (f 1)
  (thanks
    (to Jenny)
    (for (and
      being-awesome being-the-best support
      games-of-heroes-of-the-storm coming-to-sweden so-much-more )) ) )
(* (seconds-since 1983 07 07)
  (thanks
    (to parents)
    (for life-time-of-support) ) )
(signed (choose—one Sincerely Best Cheers) Uwe )
```

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1
Introduction

More and more robotic and sensor technology is deployed in home and work environments. This opens up a world of possibilities to automatically provide support and information to its users. We expect these robotic and sensor devices to take care of daily chores, but at the same time we don't want them to interfere with human activities. Any system that is to tackle this challenge necessarily involves a variety of what we refer to as *"types of knowledge"*: It has to reason about the consequences of its actions to reach its goals, it has to consider time and resources, interactions between intended actions and human activities and possibly negative impacts on user experience.

As a simple example consider a vacuum cleaning robot. This robot is tasked with cleaning all rooms in a house and it can plan ahead in order to solve this task. It has to do so without disturbing the inhabitants of the house. It uses information provided by a system for activity recognition to create a plan that respects these constraints and executes the plan while adapting to dynamically observed human activities. Throughout this thesis we refer to the decision-making component of such a robot as the *Human-aware Planner and Executive (HaPlEx)*.

In this thesis we address human-aware planning, by which we mean the problem of determining how to reach a set of goals by using a set of available actions that have to be executed in an environment that is shared with non-controllable agents such as humans. Human presence leads to many challenges for a planner. It may constrain what actions can be performed, or when and where they can be performed and under what circumstances. One of the main challenges here is that a plan should not interfere with human activities.

Example 1.1. *Imagine a household with a newborn child and a robot to take care of some daily chores. After a great deal of convincing from the parents' side, the child finally falls asleep. Two minutes later the robot decides to start vacuuming in the same room.*

This example points to the need to reason about interaction with humans during execution as well as during planning in order to ensure that plans are *socially acceptable*. If a "human-aware" plan is executed without regard for new information, a situation like the one described in Example 1.1 cannot be avoided. If the plan is flexible enough the vacuuming action could be delayed dynamically (assuming the *HaPlEx* is capable of detecting the need for the delay). Besides avoiding negative interactions such as the one in the above example, in many cases we can identify possibilities for positive interaction, or *proactivity*.

Example 1.2. *There is only one person at home and the stove is turned on. The person gets ready to leave the house at which point the robot will approach the person and notify them about the fact that the stove is turned on.*

Proactivity means that the *HaPlEx* should, to some extent, be capable of inferring itself which goals to achieve. These goals can be to provide information that a user might be interested in (as in the above example) or to support them in their activities.

Example 1.3. *If someone gets up at night to go to the bathroom the light should be turned on. If that person is grandpa, robotic assistance should be provided if possible.*

The last two examples feature proactivity but they also show a need for *context-awareness*. Identifying that the last person is leaving the house is an example of a context which may not be directly apparent from the input data of a planner and require inference capabilities to be detected. Context-awareness can also be used to add additional information that may be relevant to human-awareness.

Example 1.4. *A member of a family is cooking. The HaPlEx infers that after cooking is finished all members of the family will eat in the dining room. The HaPlEx can use this information to avoid tasks that would interfere with the eating activity. It can also plan under the assumption that during this time every member of the family will be in the dining room and will not risk disturbing someone when cleaning the bedrooms.*

These inferences may involve a variety of knowledge types. Frequently, temporal relations between human activities and actions of the *HaPlEx* are important. In Example 1.1 the problem is not that the robot executes a vacuuming action but that it temporally coincides with the sleeping activity of the child. Often there are also dependencies on preferences or properties of humans. In Example 1.3 the proactive goal for robot assistance should only be added if the person getting up is grandpa. There may be situations in which the *HaPlEx* has to make decisions that have a negative impact on user experience. In such cases we would like to be able to reason about this in terms of a *social cost* that can be used as a metric to compare the degree of "human-awareness" of alternative courses of action.

1.1 Research Question & Contributions

The human-aware planning problem is not yet well defined and has complex ramifications in terms of both representation and reasoning. To address it in a systematic way is not a trivial task. In summary, the methodology we have chosen in this thesis, consists of the following: an initial statement of assumptions; an identification of first functional and then technical requirements, with the former made to some extent testable by the inclusion of use cases; a presentation of the approach itself; and evaluations of requirement fulfillment and efficiency of the approach in terms of modeling capability (based on use cases) and computational performance, and a demonstration of real-world deployment.

The main contributions of this thesis are as follows. We perform a requirement analysis for human-aware planning (Chapter 2). These requirements will be divided into *functional requirements*, *technical requirements*, and *external requirements*. Functional requirements describe the basic capabilities of a human-aware planning system. Technical requirements are used to describe the capabilities of a planner that aims to fulfill the functional requirements. External requirements are closely related to the subject of this thesis but out of its scope. As part of the functional requirements we formulate a series of use cases[1] whose realization can be used to indicate that the functional requirements are achieved. These use cases are our main way to validate that our approach meets the stated requirements. We present a modular and extendable constraint-based planning and execution approach to the realize *HaPlEx* that fulfills many of the identified requirements (Chapter 4). It uses flaw-resolution to integrate a variety of types of constraints and is capable of solving the provided use cases. It offers integration with the *Robot Operating System (ROS)*[2]. We provide some formal properties of the approach and a template for a complexity analysis. We also provide three approaches for pruning in search spaces that result from combining many different constraint types. To model human-awareness for planning and execution we use *Interaction Constraints (ICs)*. In light of the presented requirement analysis we also provide and analyze patterns of *ICs* for human-awareness (Chapter 5). We evaluate the human-aware capabilities of the presented approach by solving a series of use cases covering some of the requirements. We also compare to possible solutions in the Planning Domain Definition Language (PDDL) [103], which is the standard used by the International Planning Competition (IPC)[3]. Finally, we demonstrate the approach on a robotic system, provide a benchmark for *ICs*, and include results from two domains that test scalability for social acceptability, proactivity, and context-

[1] We use this term to describe situations characterizing the interactions between the *HaPlEx* and the human(s) in the environment.

[2] http://www.ros.org/

[3] `http://www.icaps-conference.org/index.php/Main/Competitions`

awareness (Chapter 6). The following is a summary of the contributions of this thesis.

- We provide a requirement analysis for human-aware planning
  - Functional requirements (goals, social acceptability, proactivity, context awareness, planning for cooperation, modeling capabilities)
  - Technical requirements
- We propose a constraint-based planning approach based on the technical requirements
  - It is extendable and modular
  - Includes variety of constraint types
  - Can reason about interaction constraints for human-aware planning
  - Provides ROS interface to access sensor data and execution capabilities
  - Supports reactor-based execution of external processes
  - We analyze formal properties of the approach
  - We include different methods of pruning for constraint-based planning
- The proposed planning approach is then used to realize the functional requirements of human-aware planning
  - Making plans socially acceptable and maintaining social acceptability during execution
  - Recognizes opportunities for proactiviy and adapts the planning problem accordingly
  - Context-awareness to extend the range of situations that the human-aware planner can take into account
  - Planning for cooperation with humans in the environment

1.2 Setting the Stage

Let us begin by introducing all the aspects we consider independent of concrete applications. Consider Figure 1.1. The stage itself is *the environment*. The environment could be a house, an apartment or a workplace that is frequented by *humans* who live or work in it, and who engage in various *activities* over time. Some activities may tend to be followed by other activities, and some activities may be complex and consist of several simpler activities. In addition, humans may have preferences that vary from person to person or from day to day for the same person.

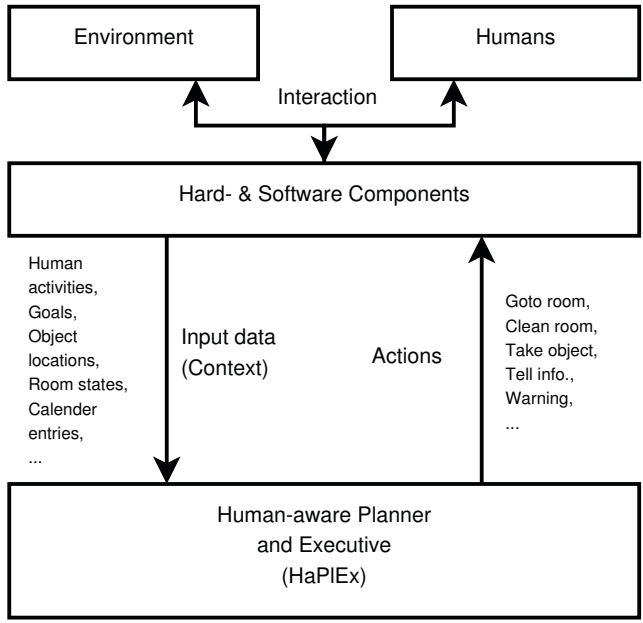

Figure 1.1: Setting the stage for human-aware planning and execution.

There is a *set of hardware & software components* deployed in the environment. Hardware may include sensors and actuators in a smart home, robots, smart phones, tablets or personal computers. Software components are often used to translate sensor data into symbolic form or translate symbolic commands into robot control actions.

Hardware and software components provide input to and execute commands of the *Human-aware Planner and Executive (HaPlEx)*. The abilities of these components are typically different from the abilities of the humans. The *HaPlEx* plans for sets of goals, dispatches and monitors the execution of the generated plans. Goals can be known in advance but are often extended during runtime by requests of humans or triggered by specific situations. In general we assume that the *HaPlEx* plans and executes continuously.

This scenario implies various challenges that will be translated into requirements in the next chapter. Human activities and *HaPlEx* actions occur over intervals of time and may be related in various ways since they take place in the same environment. We already saw a series of examples of positive and negative interactions in this section. During planning and execution, the *HaPlEx* has to be informed about and take into account these interactions and how they relate to the humans' preferences.

In many situations it cannot be assumed that the *HaPlEx* is fully informed about future human activities (timing, whether they occur) during planning. Rather, information about activities will often become known to the *HaPlEx* online, through sensing and inference. Making these inferences is also part of the planning process/computational task, as we want our system to be contextualized and proactive. This also makes it desirable for the *HaPlEx* to create plans that are somewhat flexible and modifiable. In addition, some goals might be added based on observed human activities, and if the latter may be unknown at planning time, the former may be too. And even if activities are known in advance, it may still be more convenient not to explicitly state each related goal, but instead infer them from these activities.

1.3 Outline

The rest of this thesis is organized as follows.

Chapter 2 analyzes the functional and technical requirements of human-aware planning as well as some external requirements.

Chapter 3 provides a survey of work in planning wrt. the technical requirements outlined in the previous section. In the same way human-aware planning is surveyed wrt. the functional requirements.

Chapter 4 describes a constraint-based planning framework that satisfies the technical requirements.

Chapter 5 tackles the functional requirements by using the constraint-based planner outlined in Chapter 4. It also provides solutions to the presented use cases.

Chapter 6 summarizes a series of experiments that demonstrate how efficient the resulting human-aware planning approach tackles the given requirements.

Chapter 7 summarizes the thesis, discusses to which degree requirements have been addressed successfully and points out open problems and future work.

1.4 Publications

Parts of this thesis have been published in a number of papers, available at http://aass.oru.se.

- U. Köckemann, F. Pecora, and L. Karlsson. Towards planning with very expressive languages via problem decomposition into multiple CSPs. In *Proceedings of COPLAS 2012: ICAPS Workshop on Constraint Satisfaction Techniques for Planning and Scheduling Problems*, 2012.

- U. Köckemann, F. Pecora, and L. Karlsson. Expressive planning through constraints. In *Proceedings of the 12th Scandinavian Conference on Artificial Intelligence (SCAI)*, 2013.
- U. Köckemann, F. Pecora, and L. Karlsson. Grandpa Hates Robots — Interaction Constraints for Planning in Inhabited Environments. In *Proceedings of the 28th Conference on Artifical Intelligence (AAAI)*, 2014.
- U. Köckemann, F. Pecora, and L. Karlsson. Inferring context and goals for online human-aware planning. In *Proceedings of the IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, 2015.

Chapter 2
Requirements for Human-Aware Planning

In this chapter we will derive a series of functional requirements for human-aware planning that are addressed by the approach presented in this thesis. From these functional requirements we will derive a set of technical requirements in a top-down manner. All these requirements will provide the basis for the discussion of related work in Chapter 3 where we analyze existing work in planning wrt. the technical requirements, as well as existing work in human-aware planning wrt. the functional requirements. In Chapter 4 we formulate a constraint-based planning framework that fulfills the technical requirements. This will be used in Chapter 5 to tackle the functional requirements. The list of requirements presented in the following is obviously not exhaustive and we will look into some related requirements that lie outside the scope of our work in Section 2.3. The presented list of requirements is only a first attempt at capturing the challenges of human-aware planning. It is most likely missing several aspects and should be completed in the future based on experience gained from use. For four of the presented functional requirements we present a series of use cases that we will use to show how we can move from a natural language specification of human-aware features to a domain definition. The use cases also provide a first test of the *HaPlEx*'s capability to fulfill the specified requirements.

2.1 Functional Requirements

The first and obvious requirement for the *HaPlEx* is that it must be capable of determining how to act in order to reach given goals by using available actions (FR_{GI}). Goals can be distinguished by when they become known to the *HaPlEx*. They may be known from the start, provided by humans during execution, or appear in specific situations as opportunities to aid humans. We do not provide

use cases for this requirement since it is fulfilled by design (as shown later in Section 4.4.8 on resolving open goals).

The second requirement is *social acceptability* (FR_{SA}) [172]. Certain combinations of human activities and robot actions (i.e., *situations*) result in negative interactions, and FR_{SA} means that the *HaPlEx* should avoid such negative interactions while planning and executing. What constitutes a negative interaction may depend on individual human preferences. We can distinguish between situations that can be avoided by planning around them, and situations where this cannot be done. We can also distinguish between situations that, while not optimal, may be acceptable if there is no way to avoid them, and those that are unacceptable. We use the following four use cases to measure that our approach is capable of social acceptability (FR_{SA}). Each of these use cases assumes that the *HaPlEx* is deployed in a household environment where a robot has goals involving vacuum cleaning different rooms. The use cases specify constraints on how these goals may be achieved.

- **Use Case SA-1:** "No robots are allowed in the office."
- **Use Case SA-2:** "Don't vacuum while the child is resting."
- **Use Case SA-3:** "Don't vacuum while the child is resting or pay a social cost of 10."
- **Use Case SA-4:** "Don't vacuum while the child is resting or pay a social cost of 5 or 10 for a small or large overlap respectively."

The third requirement is *proactivity* (FR_{PA}): the *HaPlEx* should be able to detect opportunities to help humans and to adopt new goals and plans in order to provide this help. In Example 1.4 the *HaPlEx* could infer that the eating activity will occur after cooking. It could further infer that eating requires the table to be set, which could be planned for proactively. We can already see similarities between proactivity and social acceptability: The *HaPlEx* has to identify situations and then react accordingly. We will solve this problem with *Interaction Constraints* (*ICs*) that allow us to describe and reason about complex situations and how the *HaPlEx* should react to these situations. As for social acceptability, we can distinguish between situations in which the *HaPlEx must* be proactive and ones where it may ignore the opportunity to be proactive. The complexity of proactive behavior may also vary. While setting a table may lead to a complex planning task for a robot [164], much simpler scenarios are possible. For instance, if someone gets up at night the *HaPlEx* should turn on the light. We use the following three use cases to measure that our approach is capable of proactivity (FR_{PA}). These use cases assume that the *HaPlEx* is deployed in an assisted living household environment. Each use cases contains a description of an opportunity for proactivty and how the *HaPlEx* should behave in case the opportunity presents itself.

- **Use Case PA-1:** "If human gets out of bed at night, turn on the light."
- **Use Case PA-2:** "If human gets out of bed at night, turn on the light. Turn the light off 2 minutes after the human is back in bed."
- **Use Case PA-3:** "If human gets out of bed at night, turn on the light. Turn the light off 2 minutes after the human is back in bed. If the person is the grandfather, also provide a robotic assistant."

The fourth requirement is *context awareness* (FR_{CA}). For many applications, the directly observed human activities and sensor events only describe the surface of the actual situation. FR_{CA} means that the *HaPlEx* must be able to infer more complex activities or situations than those that are directly observable. FR_{CA} supports the social acceptability (FR_{SA}) and proactivity (FR_{PA}) requirements. FR_{SA}, for instance, may require that there are no robots in the kitchen while someone is eating there. Contextual information can be used to infer that when someone is cooking the whole family will eat after the cooking activity is finished. To support FR_{PA} it is often useful to infer a goal from an observed sequence of human actions, or sensor traces thereof. If some person puts on their shoes and takes their jacket, the *HaPlEx* could infer that they are about to leave the house and proactively remind them to switch off the stove (as in Example 1.2). We use the following four use cases to measure that our approach is capable of context-awareness (FR_{CA}). The following use cases assume a household environment and some constraints for social acceptability (e.g., do not vacuum while humans are cooking) or proactivity (e.g., set up the table for dinner) that rely on the inferred context. Each use case describes a situation and what can be deduced from observing that situation.

- **Use Case CA-1:** "If the stove is on while a person is in the kitchen, that person is cooking."
- **Use Case CA-2:** "If the stove is on while a sensor indicates human presence in the kitchen, an unknown family member is cooking."
- **Use Case CA-3:** "If someone puts on his/her shoes and then puts on his/her jacket he/she will leave the house next."
- **Use Case CA-4:** "If there is someone cooking in the evening there will be a 40 minute dinner activity 10 minutes after cooking finishes."

The fifth requirement, *planning for cooperation* (FR_{PC}) means that the plans computed by the *HaPlEx* may involve actions executed by humans, and vice versa. This becomes necessary when humans and the *HaPlEx* have shared goals, or when they have different abilities and the goals of either party depend on subgoals that can only be achieved by the other. In many robotic domains we can identify tasks that can only be executed by either a human or a robot

(or other system component) [197]. A robot may be capable of heavy lifting but lack the skills for fine-grained manipulation. We use the following three use cases to measure that our approach is capable of planning for cooperation (FR_{PC}). These use cases assume that robot and human have to work together on a common goal and describe actions that involve humans alone or in cooperation with a robot. This may be necessary in scenarios where the *HaPlEx* requires actions that can (or should) only be executed by a human. The last use cases handles contingencies that can arise from operators that rely on human participation.

- **Use Case PC-1:** "Ask a human to go somewhere."
- **Use Case PC-2:** "Ask a human to give an object to a robot."
- **Use Case PC-3:** "If an expected human action takes too long to start, issue a reminder."

The sixth requirement, called FR_{KR}, is for engineers and/or users to be able to provide knowledge to the *HaPlEx* about human preferences and activities. This is entailed by three of the previous requirements (FR_{SA}, FR_{PA} and FR_{CA}). In all three one needs to describe situations and ways to react to given situations. In the case of social acceptability (FR_{SA}), one needs to describe unwanted situations and (when applicable) ways to avoid them. For proactivity (FR_{PA}), one needs to describe situations that provide opportunities for the *HaPlEx* to help and how it should behave in such situations. For context inference (FR_{CA}), one needs to describe situations from which the *HaPlEx* can infer additional information. Finally, we need to model human actions in order to include them in the planning process which poses some challenges compared to modeling actions for controllable components: we may need to inform humans about the requested action and such an action must be modeled in a way that does not require its direct execution or a fixed execution time. All previously described use cases can be seen as use cases for FR_{KR} since they test the capability of our approach to create convenient models from natural language descriptions.

The functional requirements illustrated in this section lead to a set of technical requirements. Domain and problem formulations have to include a variety of types of knowledge to adequately capture complex interactions with and support of humans in the environment. These types include (but are not limited to) causal knowledge in order to reach goals (see FR_{GI}), temporal knowledge to relate human activities and plans of the *HaPlEx* in time (for FR_{SA}, FR_{PA} and FR_{CA}), assigning costs to situations (for FR_{SA}), and handling static background knowledge (e.g., for preferences of individual humans). A *HaPlEx* must be capable of representing and reasoning about these types of knowledge, a technical requirement we denote TR_{Know}. There may also be additional types required for specific domains, such as spatial information or ontologies. The latter implies

the next requirement: the *HaPlEx* must be extendable with new types of knowledge (TR_{Extend}). The achievement of requirements TR_{Know} and TR_{Extend} is, in turn, facilitated by a system that is modular wrt. representation formalisms and reasoning algorithms (TR_{Modul}). If we want to extend an instance of a *HaPlEx* with n reasoners for different types of knowledge to use $n + 1$ reasoners it should not be required to significantly change the existing n reasoners to add the new one or to adapt the new one to the existing n. Modularity also means that whatever concrete approach is chosen for representing and/or reasoning about a specific type of knowledge, it can be easily exchanged. This in turn allows the *HaPlEx* to benefit directly from the state of the art since outdated reasoning approaches can be replaced by their successors.

Since the *HaPlEx* includes an executive it must have online planning capabilities to be able to continuously satisfy human-awareness during execution (TR_{Online}). For this it must operate in a closed loop with activity recognition [46] and similar types of event detection [185].

Another important aspect of human-aware planning is dealing with uncertainty about what human activities will occur and when and where (TR_{Uncert}). In Example 1.1, for instance, it is hard to predict or to assume any regular schedule for the child. The *HaPlEx* has to make sure that it does not interfere with the sleeping activity regardless of when it occurs. Uncertainty also implies that plans should be flexible (TR_{Flex}). Recall Example 1.4 in the previous chapter. Since the *HaPlEx* may not know the duration of cooking it is not clear when eating will start. Thus any action of the *HaPlEx* that has to start after eating (e.g., cleaning the dining room) needs a flexible start time.

2.2 Technical Requirements

The functional requirements illustrated in the previous section lead to a set of technical requirements. Domain and problem formulations have to include a variety of types of knowledge to adequately capture complex interactions with and support of humans in the environment. These types include (but are not limited to) causal knowledge in order to reach goals (see FR_{Gl}), temporal knowledge to relate human activities and plans of the *HaPlEx* in time (for FR_{SA}, FR_{PA} and FR_{CA}), assigning costs to situations (for FR_{SA}), and handling static background knowledge (e.g., for preferences of individual humans). A *HaPlEx* must be capable of reasoning about these types of knowledge, a technical requirement we denote TR_{Know}. There may also be additional types required for specific domains, such as spatial information or ontologies. The latter implies the next requirement: the *HaPlEx* must be extendable with new types of knowledge (TR_{Extend}). The combination of TR_{Know} and TR_{Extend} in turn implies a requirement for modularity of representation formalisms and reasoning algorithms (TR_{Modul}). If we want to extend an instance of a *HaPlEx* with n reasoners for different types of knowledge to use $n + 1$ reasoners it should not be required to significantly change the existing n reasoners to add the new one

or to adapt the new one to the existing π. Modularity also means that whatever concrete approach is chosen for representing and/or reasoning about a specific type of knowledge, it can be easily exchanged. This in turn allows the *HaPlEx* to benefit directly from the state of the art since outdated reasoning approaches can be replaced by their successors.

Since the *HaPlEx* includes an executive it must have online planning capabilities to be able to continuously satisfy human-awareness during execution (TR_{Online}). For this it must operate in a closed loop with activity recognition [46] and similar types of event detection [185].

Another important aspect of human-aware planning is dealing with uncertainty about what human activities will occur and when and where (TR_{Uncert}). In Example 1.1, for instance, it is hard to predict or to assume any regular schedule for the child. The *HaPlEx* has to make sure that it does not interfere with the sleeping activity regardless of when it occurs. Uncertainty also implies that plans should be flexible (TR_{Flex}). Recall Example 1.4 in the previous chapter. Since the *HaPlEx* may not know the duration of cooking it is not clear when eating will start. Thus any action of the *HaPlEx* that has to start after eating (e.g., cleaning the dining room) needs a flexible start time.

2.3 External Requirements

In this section we provide an overview of requirements and fields of research that are closely related to the *HaPlEx* but are outside the scope of this thesis. The *HaPlEx* requires Activity Recognition [95, 125, 200, 224, 185] to recognize human activities from sensor data. The usage of sensor data distinguishes activity recognition from context-awareness (FR_{CA}) which will often use recognized activities as an input. Along the same lines, we consider as tangential to our work plan recognition [203, 192], which aims to find a set of goals to explain given a sequence of actions. This can also be seen as a form of context-awareness. Human-computer interaction [124] is required in any scenario that includes interaction between the *HaPlEx* and its users (e.g., via keyboard, phone, voice. or gestures). It can be used to input goals, new constraints (permanent or for a limited time) or as part of cooperation. Informing users about the *HaPlEx*'s plans may reduce the number of conflicts and in many cases asking for permission to execute specific actions or apologizing for inconvenient situations could lead to a better user experience. If we consider long-term deployment of the *HaPlEx* it will have to adapt to new people in the environment. Learning approaches can be employed to update models of humans (e.g., their preferences) over time [143]. Mixed-initiative planning [83, 6, 96, 202] may be required to involve human operators in the creation of plans. This increases the human control over the behavior of the *HaPlEx*. Humans being aware and involved in the planning process may also increase the chances of successfully executing those plans. Plan repair [88, 110] is required to adapt complex plans rather than re-planning which may increase the performance of the *HaPlEx*

significantly. When considering autonomous systems deployed in human environments their motions for movement and manipulation are also required to be human-aware [205, 141, 72].

We do not explicitly consider any of these aspects, e.g., by including them into our use cases. At the same time we attempted to design the *HaPlEx* in such a way that it can interact with or be extended with solutions to these external requirements (TR_{Extend}, TR_{Modul}).

FR_{Gl}	*HaPlEx* must be capable of reaching a set of *Goals* given a model of the environment and actions it can execute to change the environment.
FR_{SA}	*HaPlEx* must produce plans that only interact in a *Socially Acceptable* manner with the activities of humans.
FR_{PA}	*HaPlEx* must be *ProActive* by detecting when opportunities to provide help arise and to adopt new goals for that purpose.
FR_{CA}	*HaPlEx* must identify complex situations or activities that arise from combinations of human activities and actions of the *HaPlEx Context Awareness*. Once identified these situations may be relevant to FR_{SA} and FR_{PA}
FR_{PC}	*HaPlEx* must be capable of *Planning for Cooperation* with humans in case of overlapping goals or need for human abilities.
FR_{KR}	*HaPlEx* must support users and/or engineers in providing *Knowledge* about, for instance, preferences and activities.
TR_{Know}	*HaPlEx* must be capable of dealing with a variety of *knowledge* types to adequately represent all aspects of human-aware planning domains.
TR_{Extend}	*HaPlEx* should be *extendable* with new types of knowledge to allow modeling and solving of domain specific aspects.
TR_{Modul}	To support many types of knowledge in an extendable manner, the *HaPlEx* must reason about them in a *modular* way.
TR_{Online}	Human-awareness must be maintained during execution. Consequently the *HaPlEx* must work in a closed loop with systems for activity recognition and be able to deal with causal interference of observed human activities with its plans. Social acceptability (FR_{SA}) needs to be maintained *online* during during execution.
TR_{Uncert}	*HaPlEx* must be robust when dealing with *uncertainty* about time, place and type of human activities.
TR_{Flex}	*HaPlEx* must be capable of producing plans that are *flexible* wrt., e.g., the durations of human-activities.

Table 2.1: Requirements for human-aware planning and resulting technical requirements. FR = Functional Requirement, TR = Technical Requirement.

Chapter 3
Related Work

In this chapter we provide a survey of existing work in planning and analyze its suitability for human-aware planning with respect to the technical requirements established in the previous chapter. We then discuss related work more specifically in human-aware planning.

In the previous chapter, we have identified the following functional requirement for the human-aware planner and executive (*HaPlEx*). It needs to be capable of reaching goals (FR_{Gl}) given a set of available actions. Plans should be socially acceptable for humans in the environment and respect their preferences where possible (FR_{SA}). This entails that we need the ability to put hard and soft constraints on relations between the plan and human activities. Hard constraints are those that must be satisfied while soft constraints are optional and may be ignored even if that lowers the quality of a solution. Another important aspect is proactivity (FR_{PA}), which can range from simple actions that can be executed at any time to requiring a complex change of the current plan. Context awareness is required to deal with the interaction between a plan and human activities (FR_{CA}). The results of this inference may become important since it can provide crucial input to other aspects of human-aware planning. Finally the *HaPlEx* needs the capacity to plan for cooperation with humans (FR_{PC}). To fulfill these requirements we need a way to represent and reason about the above mentioned functional requirements (FR_{KR}). From these functional requirements we derived a set of technical requirements. The *HaPlEx* is required to reason about different types of knowledge (TR_{Know}) in an extendable and modular way (TR_{Extend}, TR_{Modul}). It requires flexible temporal reasoning (TR_{Flex}),online capabilities (TR_{Online}) and must account for uncertainty wrt. human activities (TR_{Uncert}). As pointed out before there are some related requirement such as human-computer interaction and mixed-initiative planning that we will discuss shortly towards the end of this chapter.

The aim of this chapter is to analyze the suitability of existing planning approaches for human-aware planning and to show that there exists no approach that provides a solutions to all the requirements listed above that can

be applied during planning as well as when facing situations encountered during execution. We will first cover research on automated planning in light of the technical requirements (Section 3.1). Then we proceed to analyze existing work on (or related to) human-aware planning (Section 3.2).

3.1 Automated Planning

This section gives a survey of existing work in planning and analyzes its capabilities wrt. the technical requirements of human aware planning that were established in Chapter 1. Obviously automated planning achieves FR_{GI} but we also need to consider other requirements stated in Chapter 1.

3.1.1 State-Space Planning

In state-space planning [84, 105] the state of the environment and of agents is modeled by a set of state-variable assignments. The following introduction is a short version of the one found in Ghallab et al. [105, Ch. 2] with some renaming/rearranging of symbols to fit the notation used in this thesis. A state-variable assignment has the form

$$x \leftarrow v$$

where $x \in X$ is a state-variable and $v \in D_x$ is an assigned value. X is the set of all state-variables and D_x is the domain of variable x. For propositional planning system it is assumed that $D_x = \{True, False\}$ for all x. The state variable x has the form

$$(p\ t_1\ \ldots\ t_k)$$

where p is a symbol representing some property or relation and the terms t_i are variables or constants representing objects that this relation applies to. A state-variable is ground when all its terms are constants.

A state $s = \{(x \leftarrow v) | x \in X\}$ is a set of state-variable assignments for the set of all variables in X. A planning operator

$$o = (Name_o, \mathcal{P}_o, \mathcal{E}_o)$$

consists of a name $Name_o$, the set of preconditions $\mathcal{P}_o$, and the set of set of effects $\mathcal{E}_o$. The preconditions must hold in a state to allow applying the operator. The effects describe how a state changes when the operator is applies. Operators and state are ground if all their elements are ground. A ground operator is called an action. Applying an action a to a state s yields a successor state s' given by the state transition function

$$s' = \gamma(s, a) = \{(x \leftarrow c) | x \in X\}$$

where c is taken from $(x \leftarrow c) \in \mathcal{E}_a$ if possible or else from $(x = c) \in s$. A state-variable planning problem is a triple

$$\Pi = (s_0, g, A)$$

consisting of an initial state s_0, a goal g (describing a partial state) and a set of ground actions A. A solution to this problem is a plan $\pi = \langle a_1, \ldots, a_n \rangle$ leading to a sequence of states $\langle s_0, s_1, \ldots, s_n \rangle$ by applying the transition function such that $s_i = \gamma(s_{i-1}, a_i)$ and for all i a_i is applicable to s_{i-1} and s_n reaches all goals ($g \subseteq s_n$).

This is the classical approach to planning and the most well studied one. Complexity results can be found, e.g., in [9, 10, 105]. There exists a wide variety of heuristics [230, 113, 118, 121, 119, 195] for forward planning which explores the state-space starting from the initial state s_0 and repeatedly applies actions until a goal state is reached. Heuristic forward planners have shown impressive performance, e.g., in the International Planning Competition (IPC) [1]. It entails, however, many assumptions such as implicit time, sequential plans, static environments and that planning is done offline [105]. Not all approaches rely on heuristics to handle the complexity of state-space planning. TLPLAN [8], for instance, uses temporal logic formulas to add control knowledge to reduce the search space of the planner. This is an example of a domain-dependent approach, i.e., an approach that includes search control knowledge as part of its domain model.

Considering the assumptions mentioned above it is clear that a pure state-space planning approach is not sufficient for human-aware planning. Adding other types of knowledge (TR_{Know}), such as temporal knowledge, typically requires to change the problem formulation (enriching the state-space) or to compile knowledge into actions and state-space. This idea is used frequently for extensions of state-space planning. We refrain from this as much as possible in favor of modularity (TR_{Modul}). We will see, however, that while state-space planning cannot provide a complete solution on its own, it can be used effectively to contribute an important aspect of the solution: reaching goals.

3.1.2 SAT-, CSP-, ASP-based Planning

There exist approaches that use the same problem formulation as state-space planning but convert the problem into some other, well studied, problem formulation and solve the resulting problem with efficient algorithms for this other formulation. Examples of such problem formulations include Boolean satisfiability (SAT), Constraint Satisfaction Problems (CSP) and Answer Set Programming (ASP). Usually, the resulting problem is to find a solution containing a limited number of n actions (bounded planning problem). If no solution can be found for the limited horizon of n actions, a new problem is generated to find a solution with $n + 1$ actions. The rules that are used to convert the planning

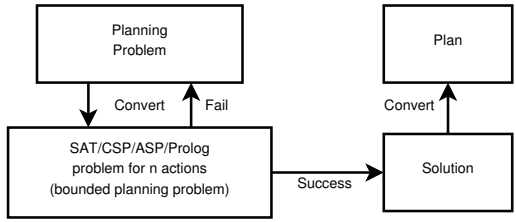

Figure 3.1: The planning problem is converted to a limited horizon SAT/CSP/Prolog/ASP problem. If a solution for the converted problem with a horizon of n actions is found it is converted to a plan. Otherwise a problem for $n + 1$ actions is created.

problem into a limited horizon problem of a different formalism play a crucial role in all of these approaches. Figure 3.1 illustrates the approach. Another important crucial point here is to decide when to stop trying to find a solution. The planning graph (see below) can be used to decide the maximum plan length that needs to be considered (see, e.g., [105, Ch. 6]).

SatPlan [135] uses a set of encoding rules to translate a planning problem into a series of SAT problems for a growing number of actions. Since the original work on SatPlan there have been a lot of advancements in how to convert classical planning problems to SAT problems [134, 136, 137, 138]. Similar approaches utilize Constraint Processing [14, 15, 16, 225, 128, 130, 162], Prolog [232, 27] or Answer Set Programming [159].

All of these approaches are well suited to include other types of knowledge. A CSP formulation of a planning problem, for instance, can be extended with constraints describing spatial aspects of the domain. This approach of "merging" many knowledge types into one problem lacks modularity. Each type of knowledge has to be carefully tied in with the rest and often a good understanding of the according formalism, as well as the way the underlying solvers work is required to do this properly.

SAT/CSP based approaches offer generalization possibilities such as clause learning and have been used to realize solution extraction for GraphPlan [137]. They depend heavily, however, on the rules that are used to create the bounded planning problem (e.g., there are many ways to convert planning operators to SAT clauses).

Using any of these approaches as the sole way to express every type of knowledge violates the modularity requirement (TR_{Modul}) and would make it hard to incorporate approaches that are better suited to include knowledge representation formalisms that are better suited to represent specific human preferences and other requirements. Temporal reasoning, for instance, could be made unnecessarily hard. Simple temporal problems can be solved in polynomial time [66], but it is not clear if this property would remain mixing a simple temporal problem into a SAT encoding of a planning domain. It is also not clear

how these approaches would scale when adding a variety of types of knowledge that is required for human-aware planning directly into the underlying representation. In addition, it might be difficult to maintain the problem formulation when adding dynamic events such as human activities during execution.

All of these approaches have a very desired property wrt. human-aware planning: Once we formulated the planning problem as a limited horizon problem of SAT/CSP/ASP more information can be added (TR_{Know}) by extending the limited horizon problem with additional constraints. This has been shown, e.g., in the work of Erdem et al. [81] who use ASP to integrate planning with spatio-temporal reasoning. Kavuluri et at. [139] used an optimal CSP approach to include hard and soft constraints on plans (and plan trajectories). Tsamardinos et al. [223] use CSPs as the underlying idea to incorporate multiple types of knowledge. Usually these approaches use time-steps at which values are assigned or actions executed for temporal reasoning. In the technical requirements we established that a more flexible notion of time is desirable (TR_{Flex}). In addition, the resulting problems and solutions may be hard to maintain during execution (TR_{Online}). If more actions are required due to human activities the problem has to be solved from scratch.

3.1.3 Graph Plan

GraphPlan [23] uses a data structure called the planning graph to solve the classical planning problem. It first performs an expansion step (of the planning graph) to analyze the reachability of goals. Then it uses an extraction step to get a solution based on the planning graph. The planning graph alternates a propositional layer and an action layer. The propositional layer collects all effects that can be provided by actions applicable to the previous propositional layer (even if they cannot be reached at the same time). The action layer contains all actions that can be applied to the previous propositional layer. In addition, each layer requires mutual exclusion relations between actions with conflicting preconditions and propositions that can not be reached simultaneously. When this graph is expanded it will eventually contain all reachable propositions. If at this point it does not contain all goal propositions the goal is not reachable. When one propositional layer contains all goals a solution extraction is attempted by selecting non-mutual exclusive actions on each layer such that all remaining open goals are reached. Preconditions of selected actions are added as open goals and have to be reached by previous layers until the initial state is reached that has to provide all remaining open goals (i.e., the resulting plan is applicable to the initial state). If no solution can be found in this way the planning graph is expanded one step further and solution extraction is attempted again. The planning graph can be used to inject non-causal constraints into the planning problem. Its mutual exclusion constraints, e.g., can be enriched by temporal constraints to facilitate temporal reasoning and resource scheduling during planning as was done by several approaches listed in Section 3.1.6.

3.1.4 Plan-space Planning

Plan-Space Planning [167, 186, 182, 238] uses the same action models as state-space planning but searches in the space of partial plans. Again we provide a short formal introduction based on Ghallab et al. [105, Ch. 5]. A partial plan

$$\pi = (A, \prec, B, L)$$

consists of a set of operators A, a set of ordering constraints between operators $\prec$, variable binding constraints B and a set of causal links between actions L. Each partial plan represents a set of sequential ground plans. Operators in A can be partially instantiated. A causal link $a_i \xrightarrow{p} a_j$ means that action a_i provides proposition p to satisfy a precondition of action a_j. Partial plans are refined by adding actions to A, ordering constraints to $\prec$, binding constraints to B or causal links to L.

Algorithms for plan-space planning are based on flaw-resolution. A partial plan is a solution when it has no flaws. There are two types of flaws: open goals and threats. Open goal flaws can be resolved by adding a causal link from the effect of a new or existing action to the goal (i.e., the link marks the effect as an achiever of that goal). In addition, an open goal resolver may adds some variable assignments to match the effect to the open goal. A threat on a causal link $a_i \xrightarrow{p} a_j$ is an action a_k that has an effect $\neg p$. Threats are resolved by ordering a_k to be before a_i or after a_j. The initial partial plan (i.e., the root node of the search) contains two special operators: the initial state operator only contains the initial state as effects. The goal operator contains only the goal in its preconditions.

This approach avoids unnecessary or premature commitments that are found in state-space planning. The order of actions is irrelevant in many cases. To reach a goal we may not be required to fully specify the operator that achieves it. Consider two robots moving two distinct objects to reach a goal. If the locations are independent it does not matter in which order we apply these actions. Furthermore, for each individual object we do not need to specify (at first) which robot moves it. This idea is known as least commitment planning [233].

Kambhampati and Srivastava [129] propose a refinement based approach to integrate state-space and plan-space planning. Their framework, UCP, uses a state-space planner to propose refinements for a plan-space planner. This is an interesting example of how refinement search (or flaw-resolution) can be used to integrate different types of reasoners. In the above paper the integrated approaches are both planning approaches. We will see more examples in the following sections where similar approaches have been used to integrate planning with time and resource scheduling.

As we will see in Chapter 4, the idea of flaw-resolution can be extended to satisfy FR_{KR} and TR_{Know} in a modular way (TR_{Modul}) and will be used as the basis of our approach to constraint-based planning. Using plan-space planning

to reach open goals can have several advantages. First, it is easily extended with additional types of knowledge (see, e.g., Section 3.1.6 on temporal planning). Second, the idea of least commitment can provide a lot of potential for pruning inconsistencies early on. If we find a partial plan to be inconsistent we can safely prune every refinement of this plan.

3.1.5 Hierarchical Task Networks

Hierarchical Task Network (HTN) planning [82, 180, 179] augments the classical state-space approach (Section 3.1.1) with a notion of tasks that can be reduced to simpler tasks by the means of *methods*). The aim of an HTN planner is to reduce all tasks to operators/actions. The following short introduction is based on Ghallab et al. [105, Ch. 11]. A task network $w = (\mathrm{U}, \mathcal{C})$ contains a set of task nodes and constraints $\mathcal{C}$ on the order in which these tasks have to be achieved. The set of constraints $\mathcal{C}$ may include precedence, just before/after or between constraints and has a lot of room for generalizations. Tasks are divided into primitive and non-primitive ones. Primitive tasks can be accomplished by operators if their preconditions are satisfied. A non-primitive task can be accomplished by a *method*

$$\mathrm{m} = (\mathit{Name}_\mathrm{m}, \mathit{Task}_\mathrm{m}, \mathit{Subtasks}_\mathrm{m}, \mathcal{C}_\mathrm{m})$$

with a name Name_m, the task it accomplishes Task_m and a dependence on a set of subtasks $\mathit{Subtasks}_\mathrm{m}$ and constraints $\mathcal{C}_\mathrm{m}$. $\mathit{Subtasks}_\mathrm{m}$ and $\mathcal{C}_\mathrm{m}$ together constitute a task network. Applying a method m to achieve a task u in task network $w = (\mathrm{U}, \mathcal{C})$ yields a new task network

$$\delta(w, \mathrm{u}, \mathrm{m}) = ((\mathrm{U} - \{\mathrm{u}\}) \cup \mathit{Subtasks}_\mathrm{m}, \mathcal{C} \cup \mathcal{C}_\mathrm{m}).$$

A task network is said to be primitive if it contains no non-primitive tasks (e.g., after all non-primitive tasks have been decomposed into primitive ones). An HTN planning problem

$$\Pi = (\mathrm{s}_0, w, \mathcal{O}, \mathrm{M})$$

consists of an initial state s_0, initial task network w, a set of operators $\mathcal{O}$ and a set of methods M. A solution is found by reaching a primitive task network which contains only operators and extracting a plan from it that satisfies all constraints.

HTN models are domain-dependent since methods can be used as recipes providing solutions for parts of the problem (thus guiding the search) that could be difficult to find in a domain independent planning setting. Methods can also be used to describe preferable solutions so they could have benefits wrt. social acceptability (FR_{SA}). The constraint sets of task networks are easily extendable with other types of constraints such as temporal constraints (TR_{Know}, TR_{Flex}). Castillo et al. [33, 34], for instance, extended the HTN planner SIADEX [63]

with temporal reasoning. Around the same time a similar approach was presented by Yorke-Smith [237].

Although the solution presented in the following chapters of this thesis exploits heuristic planning to reach open goals, it would be relatively easy to use an HTN approach such as SHOP2 [179] in the framework presented in Chapter 4. Exploiting the modeling capabilities of HTN approaches may lead to better solutions in some cases than relying on planning heuristics that (potentially) add unnecessary actions. The HTN approach has been used for human-aware planning [3, 155] (the contribution to human-aware planning is discussed in Section 3.2.1). HTN planning has also been extended to reason about diverse types of knowledge for a robot application by Stock et al. [212]. FAPE [78] integrates of planning and acting by a combination of HTN, PSP, timelines, and resources. Actions are refined into skills and plan repair is used when actions fail.

3.1.6 Temporal Planning

Combining planning with reasoning about time plays an important role in human-aware planning. Many of the functional requirements (i.e., FR_{SA}, FR_{PA} and FR_{CA}) stated in Chapter 1 entail reasoning about temporal relations between actions chosen by the *HaPlEx* and human activities. Human activities have to be considered during planning (in case they are scheduled) and execution. Both cases may cause plans to be delayed or even require replanning to assert human-awareness. We need the capability to consider human activities and the means to relate them to the actions in a plan in a flexible way (TR_{Flex}) to be able to constrain plans to be socially acceptable and proactive.

Time is often introduced into planning domains by attaching durations to operators (see, e.g., PDDL 2.1 [89]). Preconditions and effects are related to the action itself by stating when they have to be valid wrt. the operator. Preconditions may have to be valid before or during the operator can be applied and effects are added during or after the operator's execution interval. The total time a plan requires to reach all goals (i.e., the makespan) becomes interesting and is often subject to optimization in temporal planning. Actions no longer have to be executed in a strict sequence since concurrency becomes possible. This also allows to reach goals that require concurrency, for instance, when an action can only be applied during a window of opportunity opened up by some other action [105, Ch. 14]. Time can also be used to constrain (until) when certain goals have to be achieved and to introduce future events into the planning problem (also called timed-initial literals [89]). The temporal problem that needs to be solved by temporal planners is often a *Simple Temporal Problem (STP)*. STPs use bounds on the start- and end-times of temporal intervals. Temporal constraints are propagated by reducing these bounds until a solution is found or some interval does not have a possible start- or end-time. This does not require backtracking and thus STPs can be solved in polynomial time

via inference [187]. When resources or future events are involved the problem becomes more difficult and often scheduling approaches are used to solve it.

Temporal planners based on state-space planning often limit the start times of actions to a small set, called the decision epochs [68, 112, 7]. Cushing [62] pointed out that using decision epochs becomes problematic for problems that require concurrency. In a similar way decision epochs may lack flexibility for human-aware planning where we often have to delay plan execution to account for human activities observed during execution or while waiting for humans in a cooperative scenario. If the start times of subsequent actions are fixed, delays can lead to unnecessary inconsistencies.

The timing of a plan may depend on the timing of human activities that is usually not known while planning. This was one of the reasons for the technical requirement for flexible temporal reasoning (TR_{Flex}). To avoid frequent replanning we need flexibility in human-aware plans so that they can be delayed during execution. Temporal planners that fix start times for actions (and do not re-introduce flexibility somehow later) are thus of limited use to human-aware planning.

We will now discuss approaches to temporal planning and their relation to the stated technical requirements (see Chapter 2). Most of the work discussed here is built on the basic planning techniques discussed in the previous sections.

The first type of approach we distinguish here combines plan-space planning (see Section 3.1.4) with temporal information. Early work in this direction includes O-Plan [61] and its successor O-Plan2 [76, 214]. These approaches use the idea of flaw-resolution and exploit simple temporal problems that can be solved in polynomial time [231]. O-Plan2 was extended to handle sharable resources. Laborie and Ghallab [104, 151, 152] present IxTeT which integrates planning and scheduling by using plan-space planning. IxTeT is also based on flaw-resolution and treats open goals and potential resource over-usages in the same way. As we will see in Chapter 4 we generalize this idea to create a constraint-based planner that supports any type of constraint that fulfills certain assumptions.

TLplan [8] is an approach that uses temporal logic [80] with modalities such as “always”, “eventually” or “never” on propositions to guide the search for plans. It inspired TALplan [69] which uses temporal action logics (first-order logic with explicit time and durative actions). TALplan was extended to support concurrency and resources [150]. These approaches are interesting for social acceptability since temporal logic could be used to express some constraints we are interested in (e.g., states that are never allowed to manifest). The approach to human-aware planning presented by Cirillo [46] is based on PTLplan [131], a conditional and probabilistic extension of TLplan. Since social acceptability (FR_{SA}), proactivity (FR_{PA}) and context-awareness (FR_{CA}) depend on complex relations between temporal intervals at different times, this type of temporal representation is not sufficient for fulfilling those requirements. It does, however, provide some useful features to deal with uncertainty.

Many approaches to temporal planning are based on GraphPlan [146, 209, 210, 207, 97, 161, 101, 102, 41]. They could be interesting since the planning graph is convenient to introduce additional information into the problem. In this way we could, e.g., use mutex relations to enforce social acceptability. However, these approaches are often limited by the layered structure of the temporal planning graph as pointed out by Cushing [62]. In fact it seems as if research on temporal planning with graph plan lost traction after Cushing's paper. Many approaches still use the planning graph for heuristic computations. Maris and Regnier [165] overcome the problem pointed out by Cushing by ignoring mutual exclusion relations in the planning graph and making them the concern of temporal reasoning.

Haslum and Geffner [112] presented a heuristic approach which uses decision epochs and supports time and resources and optimizes the time to reach all goals (also called the *makespan*). Haslum [111] discusses HPS^*_a and TPS, two heuristic temporal planning approaches using decision epochs. Sapa [68] is a state-space based temporal planner that uses decision epochs and can handle resources. It uses a combination function to merge heuristic values for costs and for makespan and supports timed-initial literals by using an event queue. Crikey [109, 54, 53] uses heuristic forward search for temporal planning with durative actions and timed initial literals. Crikey3 was later extended to support linear continuous processes by integrating an incremental simple temporal problem solver with a linear program solver [55, 56]. Another extension is POPF [57] which uses the idea of least (or late) commitment to delay timing and ordering decisions when possible. All of these approaches support durative actions and external events by compiling them away. Durative actions are compiled into start and end actions. Timed initial literals are compiled into dummy actions. However, this way of dealing with external events seems problematic if the events have to be temporally related to the actions chosen by the planner. If we require human-awareness of a plan wrt. external events (i.e., human activities), the compiled actions would further increase in complexity, which raises the question as to whether this method would scale very well. As pointed out before, to achieve modularity (TR_{Modul}) we prefer to avoid compiling knowledge into actions of a planning problem that is difficult in itself, especially since human-aware planning needs to consider more than planning with durative actions and resources. That being said, the big advantage of the previous approaches is the fact that they use (variants of) planning heuristics that can lead to very good performance (as shown by their success in the international planning competition). Optic [18] is another extension that resulted from this line of work and includes time-dependent costs and an alternative approach to handling timed initial literals by formulating them as a mixed integer problem.

Tsamardinos et al. [223] present an approach to conditional temporal reasoning (TR_{Flex} and TR_{Uncert}). Temporal CSPs are extended with observation nodes and labels to model under which conditions certain branches are exe-

cuted. It would be very interesting to employ this approach to planning for cooperation (FR_{PC}) as it provides both contingencies and flexible timing.

Timeline-based planning approaches [42, 28, 93, 78] move away from the idea of extending the state-space with temporal information and instead plan directly on temporal intervals that describe the changes of variables over time. This type of representation makes it straightforward to relate effects of actions to human activities [93]. They support flexible time (TR_{Flex}), are extendable and easy to integrate with execution frameworks (TR_{Online}) such as T-REX [168, 169]. Explicit intervals require the planner to establish causal links to decide which effect is to achieve each precondition (recall Section 3.1.4 on plan-space planning).

The approach by Chien et al. [42] integrates planning with execution and focuses on iterative plan repair when failures are detected (relevant for TR_{Online}). Bresina et al. [28] focuses on time-line based mixed-initiative approach. Fratini, Pecora and Cesta [93] present OMPS which plans by time-line completion which is based on flaw-resolution and can accommodate custom constraints (TR_{Extend}) but no sound formal theory is provided. Magrelli and Pecora [163] use OMPS to integrate action-based planning, time and resources. Frank and Jónsson [92] propose constraint-based attribute and interval planning implemented in the constraint-based planning framework EUROPA. The resulting planner uses configuration rules that are capable of disjunctive preconditions and conditional effects. With respect to the technical requirements pointed out in Chapter 2, a timeline based approach seems suitable for supporting TR_{Flex} and TR_{Online}. However, since the types of constraints we require for human-awareness may vary (TR_{Know}, TR_{Extend}, TR_{Modul}) from domain to domain we propose a generalization of timeline-based planning in the next chapter.

Schattenberg [201] proposes a flaw-resolution based model to integrate partial-order planning and hierarchical task network planning with scheduling. We will use this idea to integrate many different types of knowledge (TR_{Know}) in a convenient way to fulfill the requirements for extendability and modularity (TR_{Extend}, TR_{Modul}).

Dvořák and Barták present FILUTA [77] which integrates planning with time and resource scheduling. FILUTA uses state-variable based plan-space planning with an underlying simple temporal network (STN). Resource scheduling is handled as a disjunctive temporal problem on top of an underlying STN. This is similar to the idea of meta-CSP reasoning [36, 164] which is discussed in some detail in Section 3.1.10 and is closely related to the approach we will use to integrate different reasoning approaches in Chapter 4 to satisfy the technical requirements of human-aware planning.

We chose a timeline-based approach for easy integration with execution and its flexibility that allows adaption to dynamic events (TR_{Online}, TR_{Uncert}). In addition, timeline-based approaches provide a base for reasoning about the relations between human activities and the plans of the *HaPlEx*. We will exploit the extendability (TR_{Extend}) of partial-order planning approaches which

will allow to integrate a variety of knowledge types needed for human-aware planning (TR_{Know}) in a modular way (TR_{Modul}). At the same time we would like to be able to exploit forward planning heuristics. As none of the presented approaches satisfies all our requirements, we will propose an integration of many of the underlying ideas in the next chapter.

3.1.7 Planning under Uncertainty

Handling uncertainty is useful for many aspects of planning. States, for instance, or context may not be known completely or may only be partially observable [199] or the outcomes of actions may be non-deterministic [22]. In this work we are mainly interested in uncertainty about human activities (TR_{Uncert}). Since we cannot assume to know all human activities in advance uncertainty plays a major part in human-aware planning. We may not know, for instance, who will perform which activity when and where or if an activity will occur at all. One way to include uncertainty in the operator model introduced in Section 3.1.1 is to change the effects of an operator to a set:

$$o = (Name, \mathcal{P}, \{p_1 : \mathcal{E}_1, \ldots, p_n : \mathcal{E}_n\}) \tag{3.1}$$

This operator pattern considers n alternative sets of effects and has an annotated probability p_i for each possible set of effects $\mathcal{E}_i$. Typically, uncertainty in planning is addressed in one of three ways. Conformant planning approaches [208] attempt to find a plan that works regardless of the true values of non-deterministic variables (i.e., regardless of which set of effects $\mathcal{E}_i$ from Equation 3.1 will take place). Contingent or conditional planning approaches [208, 131, 234] produce a course of action for each possibility (or outcome of actions). Conditional plans contain actions whose execution depends on which $\mathcal{E}_i$ is realized. Often these approaches produce policies [25] rather than plans, where a policy is a mapping from the state-space to actions that provide a guideline on how to act in each state, rather than a sequence of actions. The third alternative is to ignore uncertainty (or plan with the most likely outcomes) and deal with situations when they manifest during execution, e.g., through monitoring and replanning.

There are advantages and disadvantages with all three approaches. Conformant planning can ignore the combinations of possible assignments to all uncertain variables but it may produce unnecessarily complicated plans in order to do so or fail to find a plan. Contingent planners provide solutions that are prepared for each possibility, but the size of the plan is combinatorial in the number of alternative outcomes. Deferring these problems until execution and relying on replanning or plan repair can be convenient when we know that a problem can be fixed as soon as we know what will happen. It may, however, be problematic if some branches lead to a dead-end that would have been

avoidable by the other two approaches. Of course, combinations of the three approaches are possible as well.

When planning is extended with other types of knowledge, such as temporal knowledge, the possibilities to introduce uncertainty increase further. Temporal constraints, for instance, might have unknown (and uncontrollable) durations [229, 228, 173]. There might also be a distribution over the type of temporal constraint. But even if we ignore such complications the introduction of human activities leads to a large amount of possible events when considering uncertain start and end times and random humans, locations and activity classes. A fully conformant approach is quickly ruled out even in simple cases. Suppose we have a robot that is supposed to clean a bathroom and a constraint that forbids to clean the bathroom while a human is using it. Since at any time in the future this could be the case, in a conformant plan a robot will never be allowed to enter the room. A similar argument can be made for a fully conditional approach, since the number of branches becomes large quickly even for simple cases. The best way to deal with completely unforeseen activities seems to be online planning. We will, however, in this thesis see how we can handle partially specified activities in a conformant way. More specifically, we will handle cases in which we know the time of an activity but may have incomplete information about who will execute it, where it is executed and which concrete activity it will be. In these cases conditional planning approaches could become feasible as well.

Planning with external events was introduced by Blythe [24]. It accounts for dynamic environments that change due to events not in control of the planner. In Blythe's paper external events have a form similar to the operator introduced earlier:

$$(Name, \mathcal{P}, \mathcal{E}, \mathrm{p})$$

If the preconditions $\mathcal{P}$ hold in a state then the external events $\mathcal{E}$ may occur with probability p. This could allow to model human activities that are likely to occur in specific states and consider these activities during planning. It misses, however, a notion of time that would allow to constrain actions to take place before or after possible human activities.

The approach by Gerevini et al. [102] uses plan-space planning with linear action graphs (a variant of the planning graph used in GraphPlan). A disjunctive temporal problem [211, 222] is added to this linear action graph and external events are handled via scheduling.

Existing approaches in this area could be used to address scenarios in which human activities threaten the preconditions of robot actions or their goals. They could provide plans that work despite dynamic events introduced by humans in the environment. They usually do not account for preferences that are required for social acceptability (FR_{SA}) or opportunities that arise from external events as in the requirement for proactivity (FR_{PA}).

3.1.8 Other Planning Extensions

All the previously discussed planning extensions address specific issues such as time or uncertainty. Along these lines there exists similar work that covers, e.g., planning with continuous variables [160] that we will not discuss here.

According to the technical requirements stated in Chapter 2 human-aware planning requires the integration of different types of knowledge (TR_{Know}) which, in turn, suggests a modular solution (TR_{Modul}). In this section we will look at some existing work that suggests more general ways to integrate different types of knowledge with planning.

PDDL axioms [215] allow to imply facts from states. It was shown how compiling axioms into actions causes significantly larger domain formulations and plans, thus adding complexity to the planning problem. This way of adding information is appealing since it allows to model complex aspects of a domain without using operators and could be used to achieve social acceptability (FR_{SA}). PDDL axioms are of limited use to human-aware planning since they do not support time and derived facts are not allowed to be used as effects of actions (i.e., they cannot be used as goals). Our approach, Interaction Constraints (*ICs*, introduced in Chapter 4), can be seen as a more general version of domain axioms that does support a variety of constraint types and may even add new goals to achieve proactivity (FR_{PA}).

Semantic attachments [74, 75] allow planners to query external black-box procedures to evaluate whether certain preconditions hold and to assert effects through different theories (e.g., linear programs). This way of using external procedures limits the information we can get to assignments of state-variables. Semantic attachments cannot access states before or after the current state and are thus not well suited to provide a general reasoner for the interaction between humans and the *HaPlEx*. They only operate on the current state and, as pointed out before, relations between human activities and actions of the *HaPlEx* almost always require reasoning about time.

Geffner [99] proposes functional STRIPS, an approach that allows for more compact domain models by using interpreted predicates and functions [199, Ch. 8]. The usage of first-order logic allows functional STRIPS to express complex computations and to refer to objects via function symbols. Recently, Francès and Geffner [91] investigated heuristics for a sub-set of functional STRIPS that are based on a generalization of the relaxed planning graph (called constrained relaxed planning graph) and use constraint propagation to calculate heuristic values that are more informed than the ones resulting from the relaxed planning graph.

Planning Modulo Theories (PMTs) [108] are based on SAT Modulo Theories [13] to provide extensions for planning. The basic idea of PMTs is to use state-variable based planning with a generalization of the possible values of a state-variable to a theory T, where T can be, for instance, set theory, the theory of integers or real numbers, Boolean (as in classical propositional planning)

or finite domains as in state-variable planning (as detailed in Section 3.1.1). Theories are evaluated to determine whether an action can be applied or not. Information exchange between theories is possible as well as they may use other theories (e.g., set cardinality is an integer). States are changed by applying actions. Theoretical properties and further development of the approach were left as future work.

3.1.9 Planning for Robotics & Other Applications

Planners for real-world applications such as robotics often has to deal with requirements similar to the ones in human-aware planning. Even if they do not consider environments with humans they usually have a requirement for online planning and execution (TR_{Online}) , temporal flexibility (TR_{Flex}) and domain-specific knowledge (TR_{Extend}). In addition, robot agents are frequently used in human-aware planning scenarios. For this reason we will summarize some works that combine planning and robotics and some other planning applications. This will also show that the stated requirements, while derived from human-aware planning, overlap with the requirements of many other real-world applications. Consequently, a planning system that fulfills these requirements should be useful for applications other than human-aware planning.

The ability to encode procedural knowledge with methods made HTN-based planners (Section 3.1.5) the choice for many applications. Nau et al. [178] survey a number of applications of the HTN planners SHOP and SHOP2 by government, university and industry. The list includes two applications to unmanned air vehicles (UAVs). At least one of these uses HTN in conjunction with temporal reasoning. Chien et al. [43] discuss the integration of HTN planning with scheduling, thermal constraints and maneuver constraints in a satellite domain to produce schedules for science and engineering missions.

We discussed IxTeT in the context of temporal planning (Section 3.1.6). Lemai and Ingrand [156] extend IxTeT for execution and replanning in robotic systems with IxTeT-EXEC. Their approach considers new flaws during execution and allows for newly added goals and changing capacities of resources. Handling changes in the planning problem during execution is essential for human-aware planning since new goals may arise dynamically during plan execution requiring a human-aware system to adapt and take opportunities or re-plan (TR_{Online}).

Temporal planning was also used by McGann et al. [168, 169], who integrate temporal planning with probabilistic state estimation and the executive T-REX to use unmanned underwater vehicles for ocean exploration. Doherty, Kvarnström and Heintz [70] show the integration of a temporal logic based planner into execution and monitoring of an unmanned aircraft system. Bresina et al. [28] present MAGPEN (Mixed-initiative Activity Plan GENerator) for mars exploration rovers. The approach aids humans in generating daily

plans for mars exploration that need to consider temporal, resource and safety constraints while optimizing the amount of exploration that is done.

Temporal constraints and resources were integrated with planning in the work by Rabenau et al. [189] who show the benefits of using the RAXEM tool for uplink planning which significantly reduces the human workload of plan creation.

A Prolog-based approach was used by Barták and Zhou [17] to solve the Petrobras domain [123, 226]. This domain requires optimization of time and costs for shipping cargo between oil platforms. Other approaches for the same domain were compared [221].

An early approach to planning for space applications, RAX-PS, by Jonsson et al. [126] uses flaw-resolution to refine an initial plan into a working solution. This approach maintains timelines for every variable used by the planner to manage concurrent activities.

Several solutions to the liner shipping fleet repositioning problem were proposed by Tierney et al. [216]. In this problem vessels are moved between for different services. One of their solutions uses the temporal planner POPF [52], another one is based on mixed-integer programs and finally a new method for temporal optimization planning is introduced. POPF produces sub-optimal results, while the other two approaches are optimal. Task planning was combined with geometric reasoning by Gaschler et al. [98]. Their work uses reasoning about volumes for robotic domains. Volumes are used to bridge the gap between continuous path planning and symbolic reasoning. All of these are interesting examples of how timeline-based plans are executed (TR_{Online}) and how different types of knowledge are integrated (TR_{Know}).

Geometric and temporal reasoning for service robots was integrated with planning and execution by Erdem et al. [81]. They use answer-set programming as an underlying approach to integrate planning with execution monitoring and other types of reasoning (recall Section 3.1.2). Their work is a good example of a more general way of integrating different types of knowledge (TR_{Know}).

Task and motion planning were integrated by Cambon et al. [31] by extending state-space planning to keep configurations and configuration candidates that corresponds to a given state. The validity of a state depends on the existence of a configuration which is proven by a motion planner. Their work is based on previous work by Gavrot et al. [107].

The planning framework EUROPA [92] was used by Reddy et al. [194] to plan solar array operation on the International Space Station (ISS). They point out that no existing planning approach can be used out of the box. Non-linear and continuous constraints need to be mapped to a discrete space and domain-specific constraints need to be made available.

Task and motion planning were integrated for robotic manipulation by Bidot et al. [19] and Lagriffoul et al. [154, 153]. The idea is to use task planning to generate a series of manipulation actions and motion planning to find a way to execute them. If problems arise during motion planning the search needs to

backtrack in the geometric space, since the same action may be feasible when using a different motion. Geometric backtracking is a difficult problem that is tackled by using planning heuristics and geometric constraint propagation to reduce the involved search spaces. Work on hybrid task and motion planning is interesting as it shows how combining different types of knowledge (TR_{Know}) is useful for applications other than human-aware planning.

Temporal, spatial and resource reasoning was integrated with planning by Mansouri and Pecora [164]. Their work considers a robotic domain to set a table and involves metric and qualitative constraints, all of which are integrated using the Meta-CSP framework[1]. The ideas of the meta-CSP approach are closely related to the way in which we integrate different types of constraint in Chapter 4. We discuss the meta-CSP approach in more detail in Section 3.1.10 on hybrid reasoning techniques.

All of these approaches underscore the need for extendability (TR_{Extend}) and for a diversity of knowledge types (TR_{Know}). They also provide valuable insight into issues regarding online planning and execution (TR_{Online}), and are often concerned with plan repair, uncertainty and replanning (TR_{Uncert}). Practical applications often integrate multiple types of knowledge (TR_{Know}). We have seen examples for the integration of different combinations of planning, temporal reasoning, spatial and geometric reasoning and execution. This reinforces our technical requirements for extendability (TR_{Extend}) and modularity (TR_{Modul}) to be able to deal with a variety of real-world domains. Some of the presented approaches are based on flaw-resolution which is used frequently to integrate different types of knowledge. Work using meta-CSP reasoning goes in a similar direction. All of this points to the fact that flaw-resolution is a flexible way to integrate many types of knowledge. We take this one step further in Chapter 4 by providing an approach that supports arbitrary types of constraints as long as each type of constraint fulfills certain assumptions.

3.1.10 Hybrid Reasoning

We have established the requirement to combine multiple types of knowledge as required by a domain (TR_{Know}, TR_{Extend}) to deal with the requirements of human-aware planning. This makes it interesting to look into research outside of the planning area that contributes in this direction. Many of the previously discussed approaches can be considered as hybrid reasoning approaches. In this section we focus on hybrid reasoning research that does not consider task planning.

Meta-CSP reasoning [164, 67], builds on Constraint Satisfaction Problems (CSPs). A meta-CSP is modeled as a hierarchy of meta problems whose flaws are resolved by adding constraints to problems lower in the hierarchy. At the bottom of that hierarchy lie one or more regular CSPs. A reusable resource

[1] An API for the integration of multiple reasoners (`metacsp.org`).

scheduling problem, for instance, can be solved as a meta problem on top of a temporal problem [36]. Moffitt and Pollack [171] use a meta-CSP approach for optimal rectangle packing. Di Rocco et al. [67] apply meta-CSPs to the adaptation of plans during execution via configuration planning. Pecora et al. [185] and Cirillo et al. [50] used the idea of meta-CSP reasoning to connect motion planning of individual vehicles with timing decisions to coordinate a set of vehicles. Mansouri and Pecora [164] integrate spatial, temporal and resource aspects of a domain via meta-CSP reasoning. It is interesting to note that all of these approaches are motivated by practical concerns but use approaches that can be generalized.

Satisfiability modulo theories [13] extend SAT with first-order theories that can be used to attach additional types of knowledge. As mentioned before, this approach has been used for planning as well [108]. Apart from the adaption to planning this could be an interesting way to extend the types of knowledge used by a SAT-based planner by providing, for instance, a more elaborate theory of time.

Mösenlechner and Beetz [174] propose a combination of symbolic and simulation-based temporal projection for robotic manipulation. The idea is to avoid the overhead of full simulation, while using its benefits when they are needed.

Wolfman and Weld [235] extend SAT with metric constraints for resource planning. The resulting approach LPSAT combines the RELSAT solver with the Simplex solver CASSOWARY and incorporates clause learning and backjumping into this combination. Clause learning and backjumping based on multiple types of knowledge is an interesting problem for constraint-based planning. We provide some initial considerations on this issue later in Chapter 4.

Combining many types of knowledge is of interest for many applications that do not require planning. Looking at approaches such as the Meta-CSP reasoning we can see flaw-resolution at work outside of work that is directly concerned with planning. This is another indication that flaw-resolution can be used to provide a solution to the functional requirements of human-aware planning.

3.2 Human-aware Planning

In this section we will analyze existing work on human-aware planning wrt. the functional requirements discussed in Chapter 1. Let us reiterate the functional requirements. The *HaPlEx* needs to be able to achieve goals (FR_{Gl}) in a socially acceptable way (FR_{SA}). Social acceptability needs to be provided by every plan and has to asserted throughout plan execution (TR_{Online}). The *HaPlEx* has to be proactive (FR_{PA}) to take opportunities to support humans in the environment. It has to be context-aware (FR_{CA}) to facilitate social acceptability (FR_{SA}) and proactivity (FR_{PA}). Finally, the *HaPlEx* needs to be capable of planning for cooperation between humans and robots for cases in which goals overlap or

the goals of one party depend on subgoals that can only be achieved by the other (FR_{PC}).

3.2.1 Existing Work on Human-Aware Planning

The seminal work by Alami et al. [3] proposes an architecture for human-aware cognitive robotics that includes a human-aware task planner (HATP) that was discussed in more detail by Alami et al. [4] and later Montreuil et al. [172] The approach addresses social constraints (or rules) (FR_{SA}) that consider costs and utilities, states undesirable for humans, desirable and undesirable sequences of actions and protocols for human-robot interaction to accommodate cultural conventions. Humans are modeled by the actions they can perform. To create plans it uses Hierarchical Task Network (HTN) planning [179]. The work by Lallement et al. [155] is a more recent continuation of the work of this group that extends HTN planning with a notion of agents and supports the features of its predecessor. In these works human-awareness is considered mainly by constraining plans and using action costs and utilities.

The approach presented in this thesis uses Interaction Constraints (*ICs*) as a more general form of human-aware constraints that can be decoupled from action models (TR_{Modul}). Sequences of actions may be unproblematic by themselves, but their effects could interact in a negative way with human activities under certain conditions (e.g., temporal intersections). Using *ICs* we can attribute costs or utilities not just to single actions but also to more complex situations. Human-awareness is considered mainly by constraining plans and using action costs and utilities. Costs/utilities may not only apply to actions but have to be attributed to complex situations that are not directly related to actions or only partially depended on them. For modeling cooperation with humans (FR_{PC}) we take a similar approach as presented in the above works by utilizing operators that involve humans.

Automated planning for remote human-robot teamwork was addressed by Narayanan et al. [176, 177]. They propose a peer-to-peer teaming approach that makes use of the SAPA planner and compare it to supervised teaming in which the robot is used as a mobile tool. Their work studies the differences between these two approaches and evaluates them on an urban search and rescue domain in terms of subjective performance (e.g., mental workload, situation awareness). Their conclusion is that peer-to-peer teaming and planning aid in reducing the cognitive workload for humans on short term tasks.

Beliefs about humans are integrated with plan recognition by Talamadupula et al. [213]. Their work maps beliefs into a planning problem and uses the result of plan recognition for human-robot coordination (FR_{PC}). Chakraborti et al. [39, 40] combine the belief model of Talamadupula et al. [213] with integer programming to solve a shared resource problem to minimize the conflicts between possible human plans and robot plans (FR_{SA}). Goal recognition is handled by the method proposed by Ramírez and Geffner [193] and poten-

tial human plans are predicted by using a state-space planner [113]. Zhang et al. [239] deal with capability modeling in multi-agent planning. This is relevant when planning for cooperation (FR_{PC}) since models of human capabilities may be incomplete when the system is deployed and often have to be adapted over time. Planning for serendipity [37] modifies robot plans in order to support known human-plans (or goals), e.g., by providing an object that a human may otherwise have to fetch. Planning for serendipity as described in the paper cited above lies in the intersection between planning for cooperation (FR_{PC}) and proactivity (FR_{PA}).

A theory of mind approach was used by Hiatt et al. [114] to model the beliefs, desires and intentions of human agents to create explanations of unexpected human behavior which in turn allows a robot to react accordingly (FR_{PA}). This is not plan recognition but the purpose is similar in that it could provide information of value to the *HaPlEx*.

In a discussion about human-machine teamwork Christoffersen and Woods [44] argue that such teams often succeeded due to human-adaptability and that despite this teams failures were often blamed on human error. They point out that to facilitate better team play between human and machine, automated agents need to be observable (or understandable) and directable (i.e., their behavior can be adapted online based on human input).

Proactive robot behavior (FR_{PA}) for working with humans was studied by Pandey et al. [183]. Two use cases were proposed. In the first one the robot reaches out with its manipulator while asking the human to hand over an object. In the second one the robot will proactively suggest a place to put an object when the human has to make it accessible to the robot. The authors' aim is to show that proactivity is preferable for the humans involved and causes less confusion than a non proactive solution.

The POMDP-based system presented by Hoey and Grzés [115] aims to aid people with cognitive disabilities with a set of tasks (FR_{PA}). To achieve this their work aims at generating fitting prompts and cues given a partially observable state space. In this regard their work also seems relevant for context-awareness (FR_{CA}) and to some degree planning for cooperation (FR_{PC}), since the aim of the system is that the human successfully executes their plan.

Anticipatory action selection was proposed by Hoffmann and Breazeal [117, 116] to improve human-robot team performance when compared to purely reactive behavior during execution. Their approach uses a first-order Markov model to predict future states to allow action selection based on these. A workbench assembly domain is used for evaluation and their approach is compared to a purely reactive one on three suggested metrics for team fluency (time between human and robot actions, time in concurrent motion human idle time). They use fixed sequences of actions for the robot to choose from. While this is not a human-aware planning paper the proposed metrics could be of interest to evaluate the quality of cooperative plans and their execution (FR_{PC}).

Cooperation for ride-sharing was discussed by Kamar and Horvitz [127]. They present a multi-agent approach to cooperation with shared plans in a ride-sharing domain that accounts for preferences and self-interested agents. Their work is related to planning for cooperation (FR_{PC}) although we consider more the causal planning aspect than the optimization of a shared plan between multiple agents.

Collaborative planning, human-aware navigation and manipulation was covered by Kirsch, et al. [142]. They summarize some requirements for human-robot collaboration (FR_{PC}): Robot actions should be predictable and understandable by humans. Furthermore robots should adapt to the preferences of individual humans and act in a safe way. Robots should be able to distinguish the adaptability of humans from human errors. They suggest collaborative planning by introducing collaborative actions and an evaluation based on the criteria above. To achieve this they use a transformational planning approach [175] that modifies reactive plans. Projection is used to simulate navigation and manipulation aspects of a plan.

Context-awareness (FR_{CA}) and proactivity (FR_{PA}) are considered in SAM [185]. SAM is an architecture that uses temporal reasoning and rule-based decisions to realize activity recognition and control decisions for proactive human assistance. These rules are the predecessors of *ICs*, which are not limited in the constraint types of their conditions and resolvers. Social constraints (FR_{SA}) [220] were introduced into SAM to allow a planner to consider social norms and conveniences. A notion of social context is introduced to determine robot behavior in specific social situations. Integrating their work with interaction constraints for context-awareness (to detect social context) and social acceptability (to adapt the plan based on the context) is an interesting direction for future work.

The conditional human-aware robot task planner [47, 48] is based on the probabilistic planning approach PTLplan [131]. It is capable of creating socially acceptable (FR_{SA}) plans that are conditional on alternative human agendas (TR_{Uncert}). It also considers control rules based on temporal logic to speed up search (see TLplan [8]) and partially observable states. The same representation is also used for specifying interaction constraints for social acceptability. This approach was used in a real-world experiment [49]. The PhD thesis by Cirillo [46] discusses both the human-aware task planner and rule-based proactive behaviors offered by SAM.

Human-aware planning for robotics was considered by Tipaldi and Arras [218]. Their approach aims to create a probabilistic spatio-temporal model of human activities that is then used in two problems for robots in crowded environments: the maximum encounter problem and the minimum interference coverage problem [219]. The planning problem is solved via temporal Markov decision processes. They also covered some problems related to human-aware planning for robotics, such as people detection [217] and Vasquez, Okal and Arras' work on navigation in crowded environments [227].

In a discussion about agent models of humans and robots, Coradeschi and Saffiotti [58] argue for a view in which a robotic system, the environment and humans are modeled as agents with different degrees of controllability and observability. From the robotics point of view they argue for a shift from multi-purpose robots to smart environments with robotic components. We take a similar view regarding the controllability of robots and humans in planning for cooperation FR_{PC}. While we create plans that involve human participation, we can not directly execute the corresponding actions. Instead the *HaPlEx* has to ask the human and be robust wrt. their response.

An example of human-robot cooperation was presented by Rosenthal et al. [197, 198]. In their work a robot guides a human visitor through an environment (providing its knowledge of the environment) while relying on the human to open doors, etc. While not focused on planning, their work provides an interesting example of the type of division of competences we expect from a human-aware planner with cooperative capabilities (FR_{PC}).

An opportunistic approach that adapts plans during execution (TR_{Online}) was presented by Kruse and Kirsch [147]. Their approach aims to make robot actions understandable by humans in the environment. This is certainly an important issue for human-aware planning especially when it comes to planning for cooperation between humans and the *HaPlEx* (FR_{PC}). If the system acts in an unnatural way humans may find it hard to cooperate with it.

A policy-based solution to make agents acceptable to humans (FR_{SA}) was presented by Bradshaw et al. [26]. Their notion of policy is clearly distinguished from the usage of the term in planning under uncertainty (recall Section 3.1.7). The approach handles three types of conflicts (all of which are hard constraints that cannot be violated): authorized vs forbidden, forbidden vs required, required vs not required. They also discuss the roles that agents can play in human-agent interaction that range from simply carrying out commands to possibly replacing humans. In between these extremes, the discussed roles cover some requirements that we see for human-aware planning: Achieving human goals ("Carry out Intent") (FR_{GI}), pro-activity ("Offer help as needed") (FR_{PA}). Planning, however, is not considered in their work.

An approach to teamwork with unknown, possibly irrational, teammates was considered by Wu et al. [236]. Their approach considers teammates that may not be aware of the system (FR_{PC}). With respect to human-aware planning this type of approach is interesting to plan for cooperation in cases in which there is no model of human actions and/or the humans in the environment change frequently.

PIKE [158] is an executive for contingent plans that involves both humans and robots (FR_{GI}, TR_{Online}, TR_{Uncert}). The approach uses extended causal links and simple temporal networks under uncertainty to propagate human choices during plan execution to decide their intentions and adapt actions accordingly. Observed human choices prune the space of possible robot actions and force contingent branches to maintain consistency of causal links and temporal con-

straints. The level of adaptation their approach realizes seems very good to execute cooperative plans (FR_{PC}) that include contingencies.

Planning, execution and social norms for social robots were combined by Carlucci et at. [32]. Their work uses a planner based on Answer Set Programming (ASP), an executive based on Petri nets, and models domain-independent social norms as tuples of propositional statements. The first element in the tuple makes the second element mandatory which allows to specify how the system should react when specific states are encountered during planning and execution. Since social norms are domain-independent the planning domain itself does not need to be modified, but a mapping from norms to the planning domain is required.

In summary most of the existing work on human-aware planning focuses on scenarios including robots. While we also use robots in many examples and domains throughout this thesis, our requirement analysis (Chapter 2 and approach (Chapters 4 and 5) do not assume that human-aware planning necessarily includes robots (recall Section 1.2). As a result, we contribute a more general view on the subject than was presented in existing work and we address a range of issues that have not previously been addressed in an integrated manner.

3.2.2 Related Work for External Requirements

In this section we will discuss some work on the related requirements listed in Chapter 1. While we do not address them directly they are interesting to discuss since their solutions can, in many cases directly, benefit the *HaPlEx*.

Activity recognition [125, 200, 95, 224, 196] is concerned with deriving (human) activities from sensor data. Obviously this contributes significantly to human-aware planning since reacting to human activities in a good way it the main concern of human-aware planning. The planning approach presented in the next chapter is directly compatible with any time-line based system for activity recognition. In a similar fashion intention recognition could provide important input to a human-aware planning system. Burghardt and Kirste [30], for instance, use probabilistic models for intention recognition. The way in which we approach online planning and execution (at the end of Chapter 4) allows direct integration of activity recognition.

Mixed-initiative planning [6, 85, 96, 202] includes humans in the planning process. When plans have to be executed in environments with humans or in cooperation with them the *HaPlEx* involving humans in plan creation should make plan execution more likely to succeed. Plan recognition [203, 133, 132] is concerned with deriving a set of goals from observed actions of agents in the environment. Understanding human actions in this way opens up opportunities for supporting their goals and to predict future actions which in turn can lead to improved quality of human-aware plans. Lesh et al. [157] make use of plan recognition to reduce the need for communication in human-computer collab-

oration. More recent work by Ramírez and Geffner [192, 193] use heuristic planning to perform plan recognition.

Human-aware motion planning (aka human-aware navigation/manipulation) [204, 206, 205, 141, 72, 73] deals with the problem of navigating robots or robots manipulating objects in environments with human presence. Even though the domain is a different one, many of its requirement are similar and solutions to one may prove interesting for the other. Kruse et al. [148, 149] work on human-aware navigation and motion planning in confined environments. Graf et al [106] are concerned with safe navigation of robots in environments populated by humans and with safe execution of fetch and carry tasks.

Human-computer or human-robot interaction plays an important role in many human-aware planning applications. Fong et al. [87] present a survey of socially interactive robotic systems. While their survey is not directly related to human-aware planning they also provide a discussion and references to work that is concerned with the way in which humans perceive robots in the environment which can be important for human-robot cooperation (FR_{PC}).

User input could be used to add constraints online possibly for a limited duration (e.g., "Leave me alone for half an hour"). In scenarios that have proactive or cooperative aspects communicating the system's actions to the human can increase the chance of successful plan execution. The work by Pandey et al. [183] provides a nice example of how proactive communication of a robot's intentions benefits user experience by decreasing confusion.

Broz et al. [29] use POMDPs to plan for human-robot interactions in navigation domains. The work by Clodic et al. [51] proposes an architecture for human-robot interaction and cooperation.

Since humans introduce a major source of uncertainty and it is not feasible to know every contingency or plan for all of them plan repair (or adaptation) and replanning [88, 110] become important issues. In this work we use replanning if needed, but if plans are hard to find to begin with plan repair may provide better results especially if only small changes are necessary as new information becomes available during execution. This intuition is also backed by the results of Fox et al. [88]. FAPE [78] is a timeline based planning and acting approach that considers plan repair by removing actions and then resolving flaws via plan-space planning in case an action fails during execution. A similar approach to plan repair would make an interesting extension to the approach described in Chapter 4 which also uses a timeline-based representation.

Another area that is closely related to human-aware planning is multi-agent planning [71, 64]. Humans can be considered as uncontrollable or only partially controllable agents [58] and multi-agent planning considers some interesting aspects such as goal negotiation, plan coordination and decision making for sets of agents with their own preferences. It also considers different levels of dependencies, cooperation and communication capabilities [64]. Ideas of multi-agent planning could be beneficial for human-aware planning. In the present

thesis, however, we consider the *HaPlEx* as a single agent and model humans via operators, background knowledge and their activities. The approach presented in the next chapter could be extended to a multi-agent view in the future but this is out of the scope of this thesis.

Finally, there is the issue of planning with preferences [12]. Here one can distingush between quantitative and qualitative methods. For instance, PDDL3 [100] includes quantitative preferences where one can compute and aggregate numerical preference values for different categories of preferences for a plan[11]. There is also a variety of quantitative approaches, such LPP [20, 21] which is based on temporal logic. In our approach, we utilize a numerical cost function to represent preference values, and we use interaction constraints to represent what situations the preferences apply to.

3.3 Discussion, Open Problems & Contributions

As pointed out in Chapter 1, satisfying the functional requirements entails a number of technical requirements. In the first part of this chapter we gave a comprehensive overview of automated planning and approaches to extend it. While some of these approaches do so partially, there exists no planning framework that fulfills all of the technical requirements for supporting a variety of types of knowledge (TR_{Know}), in an extendable (TR_{Extend}) and modular (TR_{Modul}) way and that supports flexible temporal reasoning (TR_{Flex}) while being able to deal with uncertainty about human activities (TR_{Uncert}). We will use all we learned in the survey presented in this chapter to create a planning framework that is capable of handling a variety of types of knowledge (or types of constraints) sufficient for human-aware planning. More concretely we will use the idea of flaw-resolution to integrate a variety of types of knowledge in a clear and simple way and add a type of constraint (Interaction Constraints *ICs*) that will allow us to fulfill the functional requirements.

There are many approaches that can contribute to the technical requirements of human-aware planning. Choosing the best approach to reach open goals (or perform tasks) is already difficult and the best choice depends on the problem at hand. We would like to avoid commitment where possible and allow any existing approach to be used. At the same time we require a formal model of how decisions are made.

Many approaches that integrate different types of knowledge are based on flaw-resolution. We also found this technique used in planning applications and in hybrid reasoning (e.g., in meta-CSP-based approaches). This points to the fact that flaw-resolution is a flexible way to integrate many types of knowledge. Yet this idea has, to the best of our knowledge, not been generalized. The constraint-based planning approach we present in Chapter 4 is an attempt to do just this in order to fulfill the technical requirements of human-aware planning.

Heuristic state-space planning comes with a wide variety of powerful heuristics that can be expected to work well for many domains. Its integration in more expressive approaches requires some adaptation. Plan-space planning and other flaw-resolution based methods provide an ideal way for the integration into a more expressive description of context. Ideas of least-commitment also could provide benefits in pruning inconsistencies as early as possible, since it means finding them earlier in the search space. However, as can be seen in the results of the International Planning Competition[2], heuristics for plan-space planning do not perform as well as heuristics for state-space planning.

Hierarchical task network planners are easy to integrate with additional types of knowledge [178]. Methods are domain-dependent solution recipes that give the domain designer more control over the solution process rather than requiring knowledge of heuristics to write a planning domain.

In this thesis we employ heuristic state-space planning for causal reasoning and goal achievement (together with a number of other types of solvers) to reach open goals. State-space planning is however not built into the general framework presented in the next chapter, where the corresponding solver could easily be substituted by a solver that uses plan-space or hierarchical task network planning. In fact, even combinations of existing planning solutions are possible and may be reasonable depending on the application. We could, for instance, use heuristic state-space planning to generate initial plans and plan-space planning for plan repair without any difficulty. In the same way we could leave certain aspects of a domain to task decomposition.

From the temporal point of view, timeline-based planners provide a solid basis to put constraints between human activities and effects of a plan. They synthesize timelines, which are temporally flexible (TR_{Flex}) specifications of behavior that can be used for execution monitoring and dispatching by a temporally-aware executive module. The approach we propose in the next chapter provides a combination of all these features and introduces a new type of constraint specifically for human-aware planning that is powerful enough to tackle the functional requirements stated in Chapter 1.

We also argue that our constraint-based planning approach is of interest for planning applications other than human-aware planning, since it is modular and extendable and thus can be tailored around the needs of other applications.

Regarding the functional requirements, there exists some work in the direction of human-aware planning that addresses social acceptability (FR_{SA}) and proactivity (FR_{PA}). There is, however, no approach that allows to model and solve these requirements with a dedicated knowledge representation and reasoning component. We argue that an approach for modeling and reasoning about human-awareness allows to better analyze the complexity of the human-aware planning problem and also provides a means to reason about human-

[2] http://www.icaps-conference.org/index.php/Main/Competitions

related requirements without planning (e.g., during execution). Providing such an approach is one of the major contributions of this thesis.

Chapter 4
A Framework for Constraint-Based Planning

In this chapter we will introduce a planning framework that addresses the technical requirements stated in Chapter 1. It uses flaw-resolution to support a variety of constraint types (TR_{Know}) in an easily extendable (TR_{Extend}) and modular way (TR_{Modul}). Temporal reasoning is flexible (TR_{Flex}) and solutions are easily integrated with systems for activity recognition and execution (TR_{Online}).

Constraint types we consider in this thesis are domain constraints, cost constraints, temporal constraints, reusable resources, open goals, Prolog constraints, set constraints, and Interaction Constraints (*ICs*). *ICs* are a powerful conditional constraint type that is the basis to our solutions to the functional requirements stated in the introduction. Our approaches to reason about *ICs* can deal with partially specified human-activities (by producing a solution that will work for every possibility) and handles dynamically occurring human activities online (TR_{Uncert}).

As mentioned previously our approach is based on flaw-resolution: a *flaw* is created whenever a constraint requires further constraints to be added in order to be satisfied. A set of constraints added for this purpose is referred to as a *resolver*. If there is more than one resolver for a flaw the *HaPlEx* must make a decision among the possible ones. This leads to a search problem over choices of resolvers for all flaws. In order to solve the problem we will have to find a working combination of resolvers for all flaws.

The constraint-based view on planning is easy to integrate with activity recognition, online planning and execution. Online re-planning, in turn, is convenient for handling unforeseen human activities. Equipped with the methods introduced in this chapter we will be able to perform online human-aware planning in a closed loop with activity recognition.

The formulation of constraint-based human-aware planning is a major contribution of this thesis. It provides a generalization of previous approaches

(such as IxTeT [151]). It was created to close the loop between planning and execution/activity recognition for human-aware planning and to incorporate human-aware reasoning capabilities. The result, however, allows the integration of many types of knowledge (or constraints) into planning and can be used for other applications as well.

4.1 Some Notes on the Syntax of the Domain Definition Language

We use a domain definition language based on s-expressions that is similar in some ways to PDDL. All keywords, such as constraint types, are preceded by a colon. Variables are marked by a leading question mark. Constraints of different types are expressed in different ways and we provide a few examples for each type considered in this thesis in the following series of sub-sections. Consider the following examples for a few elements of the language that will appear frequently.

```
;; This is a comment.
(:type c1 c2 c3) ;; List of constraints preceded by their type
c ;; Constant term
?x ;; Variable term
[0 10] ;; Bounds from 0 to 10 (inclusive).
{e1 e2 e3 e4 e5} ;; List of elements.
```

4.2 Representation & Problem Formulation

In this section we will introduce constraint-based planning as flaw-resolution search. Unlike existing work (as discussed in Chapter 3) our formulation of the problem does not depend on the types of constraints that are available. Instead we make a series of assumptions that all types of constraints have to fulfill. After defining problem and solution in this way we discuss all constraint types that are used in this thesis and their relation to the assumptions.

We will first give an overview of the terminology and then provide a detailed introduction for each aspect. *Constraints* are used to restrict (combinations of) assignments to variables from given domains. The most basic type of constraint is the *Statement*. Statements relate state-variable assignments to temporal intervals. They have the form $(\mathcal{I}\ x\ v)$ where $\mathcal{I}$ is a temporal interval, x is the state-variable and v the value. In case of boolean state-variables we may omit v if we assign the value *True*. Statements are our main tool of describing the environment over time. We refer everything the *HaPlEx* knows about the environment and the human in the environment as the *context*. The context is described in the form of *Constraint Databases (CDBs)*.

Definition 4.1 (Constraint Database). *A Constraint Database* Φ *is a finite set of constraints of arbitrary types.*

Type	Flaws?	BK?	Summary of role
statement	yes	-	Connects a state-variable assignment to a temporal interval.
domain	no	-	Limits allowed values for variables.
goal	yes	Operators	Statement that has to be made true by applying actions.
temporal	no	-	Limits possible values for temporal intervals.
resource	yes	-	Constraints availability of resources over time.
cost	no	-	Adds to costs and limits maximum amount of a cost.
prolog	yes	Prolog program	Relations evaluated according to Prolog knowledge base.
set	no	-	Creates sets and enforces set membership constraints.
ic	yes	-	Interaction constraints that limit interaction between humans and robots in the domain. This will be our main tool for human-aware planning.

Table 4.1: Overview of all constraint types used in this thesis. We also indicate if they require flaw-resolution and/or use background knowledge (BK, discussed in detail in Section 4.2.1). Individual constraint types will be introduced in detail in Section 4.4. Note that whether or not a constraint type requires flaw-resolution or background knowledge depends on the underlying approach. For type *temporal*, for instance, we assume a Simple Temporal Problem (STP). In a similar way we allow type *domain* to forbid certain assignments but not to make choices as described in Section 4.4.2.

We use $\Phi[t]$ as the set of all constraints of type t in CDB Φ. Table 4.1 provides an overview of the constraint types that are used in this thesis.

Example 4.1. *We have two statements describing robot locations using intervals I1 and I2 and state-variables* (*at* robot1) *and* (*at* robot2) *with values office and kitchen. The last statement describes a human activity of the father cooking in the kitchen. Note that we also use statements to express activities of humans in the environment.*

```
(:statement ;; constraint type
    (I1 (at robot1) office) ;; robot location
    (I2 (at robot2) kitchen) ;; robot location
    (I3 (activity father kitchen cooking)) ) ;; human activity
```

We will now give an overview of all constraint types and their uses. A more formal introduction of each type will be provided in Section 4.4. *Temporal*

constraints limit temporal intervals. They may put bounds on an interval's release and duration or put constraints on pairs of intervals (e.g., one should end before the other starts). *Prolog constraints* enforce relationships between variables based on queries to a Prolog knowledge base. An example of this is to enforce an adjacency relation between the values of two variables representing locations to restrict possibilities of movement. *Resource constraints* limit the amount of a resource that can be used at the same time. In this way they limit specific intervals and the usage of resources they represent. *Cost constraints* are used to evaluate the quality of a solution (e.g., money spent, fuel used) or to limit the amount of accumulated cost assignments. Costs can also be used to optimize solutions. *Interaction constraints (ICs)* consist of condition constraints and a selection of resolvers. If the condition can be satisfied for a CDB one of the resolvers has to be added to that CDB to satisfy the *IC*. *ICs* are a powerful tool to describe complex situations that require attention. We will use *ICs* to model and reason about human-awareness (FR_{KR}) to solve three of the stated functional requirements (FR_{SA}, FR_{PA} and FR_{CA}) and to model contingencies for planning for cooperation (FR_{PC}).

Example 4.2. *An IC for human-aware planning could be used to model the following situation:*

> *Robots should not vacuum a room while someone is working there.*
> *If this is not possible the plan's quality is lowered.*

The last sentence in the example allows for a violation of the constraint by adding a "social cost" to the plan. Here the condition would be that there is a robot action vacuuming scheduled possibly at the same time and place as a known "human working" activity. This could be resolved by temporal constraints ensuring that vacuuming and working do not overlap. If the overlap cannot be avoided the social cost of the plan can be increased to indicate the negative impact on user experience.

An interesting aspect of *ICs* is that they make use of other constraints. In the above example, temporal constraints are used to describe the situation and a failure to avoid the unwanted situation (disturbing the working person) leads to adding a cost.

We will proceed to provide a general view on the properties of constraints and then devote a subsection providing exact semantics for a selection of important and useful constraint types. If we consider constraints of a specific type we get different, well studied, sub-problems. Temporal constraints pose a Temporal Constraint Satisfaction Problem (TCSP) [66]. Open goals and operators lead to a planning problem [105]. Resource capacities lead to a reusable-resource scheduling problem [36].

Some constraints types require to resolve flaws and thus require search over possible resolvers in order to determine consistency. Other constraint types do

not require to resolve flaws and thus consistency can be determined without search. Resource constraints, for instance, require to resolve potential resource over usages by adding temporal constraints to separate the involved resource usages. In order to prove that a CDB is consistent, we will have to search for a working combination of resolvers for all flaws. This search will be the core of our planning framework.

The constraint-based planning problem is conveniently defined as a search problem. We will now provide an informal outline of this problem. By the end of Section 4.2.1 we will have formal definitions for each of its aspects. The planning problem will be defined as a search over the resolvers of flaws. To properly define any search problem we need to provide definitions for three aspects:

- Nodes of the search space
- The successor function
- Properties of the solution node

Once we have definitions for these three elements, a concrete instance of a search problem requires a root node to start the search from. CDBs constitute the nodes in the search space. Flaws occur whenever a constraint requires to add further constraints (resolvers) in order to be satisfied. Often there are multiple choices of resolvers. Resolving a flaw in a CDB creates a successor in the search space. The depth of the search space is the number of flaws that are encountered when attempting to solve it. It is dynamic since some flaws may only surface after others have been resolved. The branching factor of each expansion of the search space is determined by the number of resolvers of each flaw. A solution node in the search space does not have any flaws and violates no constraints. A dead-end is a node that has a flaw without any resolvers or an unsatisfiable constraint. The concrete structure of the search space depends heavily on the constraint types that are used. We will provide a first attempt at analyzing it in Section 4.5.2. An instance of a planning problem is provided by a root node (i.e., an initial CDB) such as the one shown in Example 4.3, a set of operators, and the background knowledge required by specific constraint types (Prolog constraint, e.g., are evaluated wrt. a Prolog program that is part of the domain definition).

Example 4.3. *Consider the initial context below. It contains three statements describing the location of a robot, the location of a file, and a human activity. Temporal constraints describe release times and durations for some of the intervals of the statements. The activity* (*activity human office* ?*A*) *uses the variable* ?*A that is marked as uncontrollable with two possible values with the two domain constraints. In this way we introduce uncertainty about the activity. The maximum permitted social cost is 100. Finally, there is an open goal that requires the file to be at office1.*

```
(:initial–context
  (:statement
    (s1 (at robot) office1)
    (s3 (at file) office2)
    (a1 (activity human office2 ?A)) )
  (:temporal
    (release s1 [0 0]) (release s2 [0 0]) (release s3 [0 0])
    (release a1 [0 0]) (duration a1 [60 60]) )
  (:cost (<= social 100) )
  (:goal (g1 (at file) office1))
  (:domain
     (in ?A {working sleeping})
     (uncontrollable ?A) ) )
```

4.2.1 Satisfiability, Flaws & Resolvers

This section provides some basic definitions that will allow a formal definition of the search problem outlined in the previous section. We also introduce some basic assumptions on constraint types that will be used later to derive some formal properties. The main advantage of working within explicit assumptions is that our approach will work with any constraint type that fulfills these assumptions.

Let us begin with the most basic element: objects are represented by *terms* that can be constants (e.g., symbolic or integer terms), variables, or complex terms. Complex terms have the form

$$(\mathrm{p}\ \mathrm{t}_1\ \ldots\ \mathrm{t}_n)$$

where p is the name of the term and all t_i are terms. A term is ground when it is a constant or it is complex and contains only ground terms. Each variable has an associated domain provided by its type. This information is handled via domain constraints (discussed in Section 4.4.2).

Given a CDB Φ every constraint c involves a finite number of terms and can thus be mapped to the relational form

$$(\mathrm{r}\ \mathrm{t}_1\ \ldots\ \mathrm{t}_n)$$

where r is the name of the constraint and t_1 through t_n are its terms. We use $Var(\mathrm{c})$ and $Var(\Phi)$ to represent the set of all variable terms occurring in constraint c and CDB Φ, respectively. We say that a constraint $\mathrm{c} = (\mathrm{r}\ \mathrm{t}_1\ \ldots\ \mathrm{t}_n)$ is ground iff t_1 through t_n are ground. We use $Ground(\mathrm{c})$ and $Ground(\Phi)$ to represent the set of all ground terms occurring in constraint c and CDB Φ, respectively.

Let $\mathcal{L}$ be the set of all possible CDBs and $\mathcal{T}^*$ be the set of all constraint types. The function

$$Satisfiable : \mathcal{L} \times 2^{\mathcal{T}^*} \rightarrow \{True, False\}$$

decides whether a CDB can satisfy all constraints, given a set of types. In the same way

$$\mathit{Satisfiable} : \mathcal{T} \times \mathcal{L} \rightarrow \{\mathit{True}, \mathit{False}\}$$

decides whether a CDB can satisfy all constraints of a single type.

A CDB may require to resolve a set of flaws $\mathsf{K} = \{\kappa_1, \ldots, \kappa_n\}$ each in turn requiring us to apply a resolver to fix it. What exactly constitutes a flaw depends on the type of constraint that created it. Thus κ_i are abstract symbolic representations of flaws. We assume each κ contains the information required to provide resolvers (given its constraint type). In case of reusable resources, for instance, κ contains sets of statements that together overuse a resource.

To determine consistency we will need to find a working combination of resolvers for all flaws. Some constraint types never require flaw-resolution. They are either satisfiable or not. Examples 4.4, 4.5 and 4.6 illustrate how these concepts apply to three different constraint types.

Example 4.4. *A goal constraint contains a statement that constitutes an open goal. Each goal constraint creates an open goal flaw in a CDB if the goal has not been reached in that CDB. A resolver for an open goal establishes a link between an existing statement and the goal to achieve it.*

Example 4.5. *A resource constraint creates a resource flaw if there exists a potential over-usage of that resource in a CDB. Resolvers contain temporal constraints that remove the potential over-usage when added to the flawed CDB.*

Example 4.6. *The set of temporal constraints we consider in this thesis have a convex (non-disjunctive) nature. For this reason they do not require to resolve flaws, but are satisfiable if a valid choice for all time-points exists and unsatisfiable otherwise.*

We identify the set of all flaws created by a CDB with a function

$$\mathit{Flaws} : \mathcal{L} \times 2^{\mathcal{T}^*} \rightarrow 2^{\mathsf{K}}$$

and the set of all flaws for a specific constraint type with

$$\mathit{Flaws} : \mathcal{T}^* \times \mathcal{L} \rightarrow 2^{\mathsf{K}}.$$

Flaws are associated with constraint types. Their exact representation and the way in which they are computed depend on their type and will be handled by individual procedures for each type of constraint. The set of all resolvers for all flaws of a CDB is identified with the function

$$\mathit{Resolvers} : \mathcal{L} \times 2^{\mathcal{T}^*} \rightarrow 2^{\mathcal{L}}$$

and the set of all resolvers for a specific flaw with

$$\mathit{Resolvers} : \mathcal{T}^* \times \mathsf{K} \times \mathcal{L} \rightarrow 2^{\mathcal{L}}$$

Each resolver is a CDB Φ_r that represents the change (i.e., which constraints to add) that is required to resolve the flaw. A resolver can contain any type of constraint. Often flaws are resolved by adding constraints of a different type. This in turn may lead to further flaws that need to be resolved. Flaws of a CDB Φ and their resolvers define how the search can branch from Φ. If a flaw has no resolvers for any CDB Φ we consider Φ to be inconsistent.

Example 4.7. *Resource flaws are resolved by adding temporal constraints (see Example 4.5). This could lead to a temporal inconsistency (Example 4.6) or to another resource flaw.*

Resolvers are applied with a transition function

$$\gamma(\Phi_r, \Phi) = \Phi \cup \Phi_r$$

which will be used to create successors of CDBs in the search for a consistent combination of resolvers. The transition function γ is the successor function in the flaw-resolution search space. Any set (or sequence) of resolvers

$$R = \{\Phi_1, \ldots, \Phi_n\}$$

can be combined into a single resolver

$$R^* = \Phi_1 \cup \ldots \cup \Phi_n. \tag{4.1}$$

This works because applying single resolvers is a simple set operation. Note, however that certain flaws may only appear after a subset of resolvers has been applied and that the union of all resolvers may become inconsistent even if applying individual resolvers is consistent. For these reasons our search procedure (see Algorithm 1) resolves one flaw at a time. This allows us to define the application of a set (or sequence) of resolvers R to CDB Φ as

$$\gamma(R, \Phi) = \Phi \cup R^* \tag{4.2}$$

Definition 4.2. (Consistency) *A CDB Φ is consistent wrt. a set of types $\mathcal{T}$ if all constraints are satisfiable and there are no flaws.*

$$\mathit{Consistent}(\Phi, \mathcal{T}) \Leftrightarrow \mathit{Satisfiable}(\Phi, \mathcal{T}) \wedge (\mathit{Flaws}(\Phi, \mathcal{T}) = \emptyset)$$

Definition 4.2 is the formal definition of the goal node in the search space outlined in Section 4.2. Thus it defines what constitutes a solution of the (human-aware) constraint-based planning problem.

Now we define *Satisfiable*, *Flaws* and *Resolvers* by splitting them up into functions for each constraint type t. Let the $\mathcal{T}$ be a set of types of constraints. We can compose the above three functions as follows:

$$Satisfiable(\Phi, \mathcal{T}) \Leftrightarrow \bigwedge_{t \in \mathcal{T}} Satisfiable(t, \Phi) \tag{4.3}$$

$$Flaws(\Phi, \mathcal{T}) \Leftrightarrow \bigcup_{t \in \mathcal{T}} Flaws(t, \Phi)$$

$$Resolvers(\Phi, \mathcal{T}) \Leftrightarrow \bigcup_{t \in \mathcal{T}} \Big(\bigcup_{\kappa \in Flaws(t, \Phi)} Resolvers(t, \kappa, \Phi) \Big) \tag{4.4}$$

Some constraint types do not generate flaws; but one still needs to check that they are satisfiable. For such constraint types we only need to provide $Satisfiable(t, \Phi)$ and assume $Flaws(t, \Phi) = \emptyset$.

Definition 4.3 (Dead-End). *A CDB Φ is considered a dead-end given a set of types $\mathcal{T}$ iff*

$$\begin{aligned} DeadEnd(\Phi, \mathcal{T}) \quad &\Leftrightarrow \quad Satisfiable(\Phi, \mathcal{T}) = False \\ &\vee \quad \exists t \in \mathcal{T}, \kappa \in Flaws(t, \Phi) \text{ s.t. } Resolvers(t, \kappa, \Phi) = \emptyset \end{aligned}$$

This assumes that flaws without resolvers do not become resolvable after other resolvers have been added.

Definition 4.4 (Successor Function). *All possible successors of a node Φ given a set of constraint types $\mathcal{T}$ are provided by Equation 4.4. We use this to provide the successor function (that may depend on the set of operators $\mathcal{O}$ or some background knowledge $\mathcal{B}_i$ which is omitted here):*

$$Successors(\Phi, \mathcal{T}) = \begin{cases} \emptyset & \text{if } DeadEnd(\Phi, \mathcal{T}) \\ \{\gamma(r, \Phi) | r \in Resolvers(\Phi, \mathcal{T})\} & \text{otherwise} \end{cases}$$

We now have all pieces of the search problem outlined in Section 4.2.

Definition 4.5 (Constraint-based Search). *The search space of constraint-based planning given a set of constraint types $\mathcal{T}$ is defined by its nodes, the solution test, and the successor function.*

- *Nodes: Constraint Databases Φ (Definition 4.1)*
- *Solution: Consistent$(\Phi, \mathcal{T})$ (Definition 4.2)*
- *Successor function: Successors$(\Phi, \mathcal{T})$ (Equation 4.4)*

In practice we consider flaw and resolver selection strategies similar to techniques found in the literature on constraint satisfaction [65]. In this way a subset of flaws is selected to be resolved for each node in the search space. In some cases the above functions will depend on specific user-provided knowledge. Operators, for instance, are used to describe how the environment can be changed

to reach goals (see Section 4.4.8 for details). In case of Prolog constraints a Prolog program is used to determine flaws and to decide if a query can be satisfied. We will write *Satisfiable*$(\mathrm{t}, \Phi|\mathcal{B})$, *Flaws*$(\mathrm{t}, \Phi|\mathcal{B})$, *Resolvers*$(\mathrm{t}, \kappa, \Phi|\mathcal{B})$ to indicate the dependence on background knowledge $\mathcal{B}$. Table 4.1 indicates which constraint types require to resolve flaws and their dependence on background knowledge.

4.2.2 Constraint-based Planning Problem & Solution

Now we can define the constraint-based planning problem as an instantiation of the search problem (Definition 4.5) in the previous section:

Definition 4.6 (Constraint-based Planning Problem)**.** *A constraint-based planning problem is a tuple* Π *with*

$$\Pi = (\Phi, \mathcal{O}, \mathcal{B})$$

where Φ *is the initial context (i.e., the root of the search space),* $\mathcal{O}$ *is a set of operators and* $\mathcal{B} = \{\mathcal{B}_1, \ldots, \mathcal{B}_n\}$ *is the set of all required background knowledge (see Table 4.1 for types that require background knowledge).*

Given any constraint-based planning problem $\Pi = (\Phi, \mathcal{O}, \mathcal{B})$ the set of constraint types that Π is concerned with is the set of all constraint types that appear in Φ and each operator $o \in \mathcal{O}$, as well as the constraint types of all possible resolvers for these constraints. Since CDBs are finite by definition the set of all constraint types of relevance to any Π is finite as well. We refer to the set of all constraint types relevant for a problem as *Types*(Π). We used the same notation for individual CDBs in the previous section.

Definition 4.7 (Solution to Constraint-based Planning Problem)**.** *A solution to* Π *is a sequence of resolvers* $\mathrm{R} = \langle \mathrm{R}_1, \ldots, \mathrm{R}_n \rangle$ *added along the path from the root to a solution in the search space (Definition 4.5) such that*

$$\forall \mathrm{R}_i \in \mathrm{R} : \mathrm{R}_i \in \mathit{Resolvers}(\gamma(\{\mathrm{R}_1, \ldots, \mathrm{R}_{i-1}\}, \Phi), \mathit{Types}(\Pi))$$

which means that each R_i *is taken from the set of all legal resolvers given the CDB that results from applying all previous resolvers. The application to* Φ *(Equation 4.4) yields*

$$\Phi' = \gamma(\mathrm{R}, \Phi)$$

with

$$\mathit{Consistent}(\Phi', \mathit{Types}(\Pi)).$$

If a problem Π has a solution given a set of constraint types $\mathcal{T}$, we write *Solvable*$(\Pi, \mathcal{T})$ and occasionally *Solvable*$(\Phi, \mathcal{T})$ if $\mathcal{O}$ and $\mathcal{B}$ are irrelevant. Note, that the above definition limits the types of constraints to the set of relevant types.

4.3 Constraint-based Planning Algorithm

Now we are ready to provide an algorithm that solves the problem defined in the previous section. The outcome of this will be a powerful planning approach that is capable of dealing with a variety of human-aware planning scenarios, online (re-)planning to deal with unforeseen human activities, and closing the loop between activity recognition and planning. In this section we discuss a general approach that is independent of the types of constraints that are used. Section 4.4 details possible solutions for each type of constraint considered in this thesis.

Algorithm 1 shows the general approach we use to solve the constraint-based planning problem. It first uses optional preprocessing procedures for each constraint type (lines 3 and 4). Then it starts the recursive procedure Resolve-All that will test all constraint types in a given order (line 7) for flaws and inconsistencies with TestAndResolve (line 8). Note that detection and resolution of flaws is entirly done in TestAndResolve. If TestAndResolve returns *Failure* Resolve-All fails (line 10). If it returns Φ then Φ is accepted as a solution by type t. If flaws need to be resolved for type t Resolve-All returns a list of resolved CDBs that are tested recursively (lines 12 through 15). If any of these recursive calls returns a CDB then it is a solution (line 15). If none of them works the flaw cannot be resolved and *Failure* is returned (line 16). If no type t requires a change then Φ is returned as a solution (line 17).

This approach closely resembles plan-space planning [167] and meta CSP reasoning [36, 164]. It can be seen as a generalization of the idea of plan-space planning that supports arbitrary types of constraints in a simple common framework. It borrows many ideas from meta CSP approaches, for example in the way in which individual constraint types are handled. However, most meta CSP approaches are concerned with specific problems or specific constraint types. The work presented here can be seen as a generalization of the meta CSP idea to create a novel and very general formulation of the planning problem.

Algorithm 1 is very general but provides a basis to implement concrete approaches by providing the sub-procedure TestAndResolve-t. In most cases we use existing algorithms for finding and selecting flaws and to determine and order resolvers. Another advantage of stating problem and solution in the way we do here is that extensions with arbitrary types of constraints are easily done by providing the definitions as done before and the formal properties provided for decidability, soundness, completeness and complexity will still hold as long as new constraints follow the general assumptions that we made in Section 4.2.

Figure 4.1 shows an example of the composed search for a specific order. It resolves flaws for all constraints in a specific order. When a flaw (or set of flaws) is resolved go back to the first constraint type in line and check again since new constraints of that type may have been added.

Algorithm 1 Constraint-based planning

Require: Φ - a CDB, $\mathcal{O}$ - set of operators, $\mathcal{B}$ - Prolog knowledge base, Θ - an ordering of solvers

```
1: function CB-PLAN(Φ, O, Θ)
2:     global O, Θ, B                          ▷ Available to all sub-procedures
3:     for t ∈ Θ do
4:         PREPROCESS-t(Φ)                     ▷ Optional preprocessing
5:     return RESOLVE-ALL(Φ)
6: function RESOLVE-ALL(Φ)
7:     for t ∈ Θ do
8:         R ← TESTANDRESOLVE-t(Φ[, O][, B])
9:         if R = Failure then
10:            return Failure                  ▷ Φ cannot be fixed
11:        if R ≠ {Φ} then
12:            for Φ' ∈ R do
13:                Φ'' ←RESOLVE-ALL(Φ')
14:                if Φ'' ≠ Failure then
15:                    return Φ''              ▷ Solution from recursive call
16:            return Failure                  ▷ No working resolver
17:    return Φ                                ▷ No flaws: Solution found
```

4.4 Constraint Types & Solvers

Now we will provide details on the constraint types used in this thesis and the approaches we use to solve the sub-problems posed by them. We define each of them by providing the function *Satisfiable* for constraint types that do not require flaw-resolution and the functions *Flaws* and *Resolvers* for the ones that do. Given these functions for each constraint type we have a concrete instance of the search discussed in Section 4.2 which allows us to provide the TESTANDRESOLVE-t sub-procedures required by Algorithm 1. Recall that the three functions work with Constraint Databases (CDBs) Φ, flaws κ and resolvers Φ_r that add constraints to resolve flaws.

4.4.1 Statements

Statements can be seen as a temporal version of the state-variable assignment shown in Section 3.1.1 on state-space planning. A *Statement* s is defined as a triple

$$s = (\mathcal{I}\ x\ v),$$

where $\mathcal{I}$ is a symbolic term representing a flexible temporal interval, x is a state-variable[1] and v is a value. A flexible temporal interval $\mathcal{I}$ consists of Earliest and Latest Start and End Times (EST, LST, EET and LET)

$$\mathcal{I} = [[EST\ LST]\ [EET\ LET]]$$

[1] To avoid confusion with variable terms (called variables most of the time), we will try to consequently stick to the term "state-variable" when talking about the x of a statement.

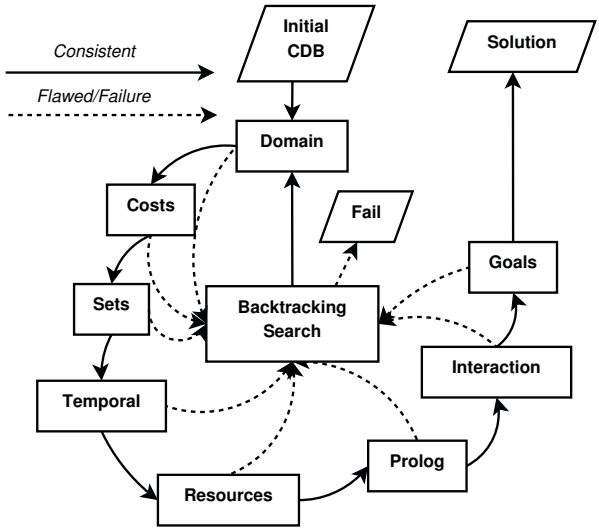

Figure 4.1: Flow of the composed search of the constraint-based planner.

during which the statement holds[2]. In case we want to refer to a specific time point of an interval $\mathcal{I}$ we will use $(EST\ \mathcal{I})$, $(LST\ \mathcal{I})$, $(EET\ \mathcal{I})$ and $(LET\ \mathcal{I})$. The state-variable x has the form $(p\ t_1\ \ldots\ t_n)$, where p is the name of the state-variable and all t_i are its argument terms. If x has no arguments we write p instead of (p). The term v is the assigned value. To refer to individual parts of (sets of) statement(s) we use the functions $Int(s)$, $SV(s)$ and $Val(s)$ for intervals, state-variables and values in turn.

A statement is ground when $\mathcal{I}$, x and v are ground. The function $Var(s)$ represents the set of all variable terms occurring in statement s. We also use $Int(\Phi)$, $SV(\Phi)$ and $Val(\Phi)$ on CDBs. Statements are constraints that establish relations between temporal intervals and state-variable assignments. As such they can lead to flaws if two statements have different values assigned potentially at the same time. This is a scheduling problem as it will be described in Section 4.4.5 and will be treated and solved in exactly in the same way.

Example 4.8. *Two statements describing the locations of a robot and a human. We use a domain definition language in which we precede constraint specifications with their type (in this case* statement*).*

[2]Note that this is not intended to express uncertainty but rather temporal flexibility. Flexible intervals may, however, have advantages when facing uncertainty during execution since their durations can be adjusted (e.g., the start time of an action could be delayed in case a previous action took more time than expected).

```
(:statement
    (I1 (at robot) kitchen)
    (I2 (at human) kitchen) )
```

4.4.2 Domain Constraints

The domains of variables and their assignments are defined by *domain* constraints. These constraints handle variable bindings by performing substitutions. Domain constraints are also used to specify types and their domains and the signatures of state-variables. They can also be used to restrict assignments of variables to subsets of their domain or to replace a variable by an object that was not yet used in a CDB. The last case is useful if we need an object to identify, for instance, a set that was created while planning.

A substitution

$$\sigma = \{x_1/v_1, \ldots, x_n/v_n\}$$

is a constraint that replaces every occurrence of variables x_i in a CDB with term v_i. We use substitutions in the same way on individual constraints. Substituting the same variable with different values or with a value outside of its domain of allowed values leads to an inconsistency.

Each variable x has an associated domain $D[x]$ of constant terms that can be assigned to it. By default we assume we can choose freely among all available values in $D[x]$. Since in practice many variables have the same domain we define domains as types and assign types to variables. Every variable has a type and this type has an associated domain of possible values. Domain constraints are used to describe this information and force substitutions of variables to follow type restrictions. We use (*enum* t $\{v_1 \ \ldots \ v_n\}$) to define a symbolic domain for type t and (*int* t [*min max*]) to define an integer domain for type t with a range between *min* and *max*.

Variables can be restricted further with two domain restriction constraints (*in* x $\{v_1 \ldots v_n\}$) and (*not-in* x $\{v_1 \ldots v_n\}$).

Example 4.9. *Defining six types* robot, human, location, agent, ID, *and* dist. *The type* agent *contains all objects in the domain of* robot *and* human. *Types* ID *and* dist *are integers that range from 0 to 1000 and from 0 to 100, respectively. The signature* sig *defines state-variable* at *with a single argument which is of type* agent *and a value domain which is of type* location. *The second signature defines a state-variable distance that depends on two locations and has a value of type* dist. *Other state-variables may be specified with more arguments. Finally, we mark the variable* ?*Activity as uncontrollable to make sure it cannot be chosen by any solver. The variable* ?X *will be assigned a term from the domain of* ID *that has not been used anywhere in the given CDB.*

```
(:domain
    (enum robot { rob1 })
    (enum human { human1 })
    (enum location { kitchen livingroom bathroom bedroom })
    (enum agent { robot human })
    (int ID [0 1000])
    (int dist [0 100])
    (sig (at agent) location)
    (sig (distance location location) dist)
    (uncontrollable { ?Activity })
    (new-object ?X ID) )
```

Given all substitutions $\sigma \in \Phi$ of a CDB and let $D_\Phi[x]$ be the set of possible values after combining all restrictions for variable x in Φ, we define

$$\begin{aligned} \mathit{Satisfiable}(\mathit{domain}, \Phi) \quad \Leftrightarrow \quad & \forall x \in \mathit{Var}(\Phi) \,:\, D_\Phi[x] \neq \emptyset \\ \wedge \quad & \forall \sigma \in \Phi, x/v \in \sigma : v \in D_\Phi[x] \\ \wedge \quad & \forall \sigma_a \in \Phi, x_a/v_a \in \sigma_a, \sigma_b \in \Phi, x_b/v_b \in \sigma_b : \\ & x_a = x_b \Rightarrow v_a = v_b \end{aligned}$$

The three parts can be summarized as follows. All variables must have non-empty domains. Each assignment of a variable has to be from the domain of allowed values $D_\Phi[x]$ and only one assignment is allowed per variable. Since this constraint type just restricts the set of possible values for single variables it does not require to impose choices via flaw-resolution. It is rather used to validate or reject the variable assignments decided by other solvers.

There may be variables whose values cannot be chosen (i.e., they are uncontrollable by solvers). These uncontrollable variables force a consistent solution to work for every possible value. We mark a variable *x* as uncontrollable with the domain constraint (*uncontrollable* x). In Example 4.3 variable ?*A* is uncontrollable: it represents an activity that is not known at planning time. Hence, any solution to the problem has to work for every possible choice (*reading* and *working* in the example). While we classify (*uncontrollable* x) as a domain constraint, its consequences have to be addressed by other solvers. If x is a flexible temporal interval it becomes the concern of the temporal reasoner to account for uncontrollability.

Domain constraints are evaluated by taking the domain of a variable and removing all values that are not permitted due to domain constraints. The effort of doing this for all variables in Φ is bound by $O(|\mathit{Var}(\Phi)||\Phi[\mathit{domain}]| \max_i |D_i|)$.

To test if the set of all substitutions S is consistent we can test individual substitutions against each other. The effort of doing this can be bound by $O(|S|^2 \max_{\sigma \in S} |\sigma|)$. This can be improved by using hash maps to create a lookup for each variable value pair. If variables are substituted by other variables we have to check for cyclic substitutions. This can be done efficiently via cycle detection in a graph representing all substitutions.

The evaluation of the domain constraint (*uncontrollable* x) is left to individual reasoners. Note that here we just use domain constraints to verify the variable assignments made by other reasoners and no choices are made by this solver.

4.4.3 Temporal Constraints

A temporal context for statements is established by *temporal constraints*. Temporal constraints can impose, for instance, flexible durations, release times or precedence constraints. We express temporal constraints via quantified Allen's interval relations [5, 170] between statements in CDBs. For convenience, we add disjunctions of conceptually neighboring constraints such as *during-or-equals* [94]. We also add the unary temporal constraints *release*, *deadline*, *duration* and *at*. Table 4.2 contains the full list of available temporal constraints together with their definitions.

Example 4.10. *Below we see two examples of statements. One models the robot location as kitchen for the interval* I1 *and the second one models an* activity cooking *by* human *in the* kitchen *during interval* I2. *The first temporal constraint sets a duration for* I1 *between 20 and 30 time units and the second one states that interval* I1 *starts at least 10 time units after interval* I2.

```
(:statement
    (I1 (at robot) kitchen)
    (I2 (activity human cooking kitchen)) )
(:temporal
    (duration I1 [20 30])
    (after I1 I2 [10 inf]) )
```

For every interval $\mathcal{I}$, all time points $(EST\ \mathcal{I})$, $(LST\ \mathcal{I})$, $(EET\ \mathcal{I})$ and $(LET\ \mathcal{I})$ lie in the interval $[\mathsf{T}_{min}, \mathsf{T}_{max}]$. T_{max} is also called the temporal horizon of the planning problem. The special constraint

$$(\textit{planning-interval}\ [\mathsf{T}_{min}\ \mathsf{T}_{max}])$$

allows to set T_{min} and T_{max}. By default T_{max} is infinite (represented by the term *inf* when using temporal constraints).

In addition to temporal constraints we support a set of temporal queries. Queries can be used to compute information based on a CDB. The intersection query $(\textit{intersect}\ \{\mathcal{I}_1\ \ldots\ \mathcal{I}_n\})$ is true iff the intervals $\mathcal{I}_i$ intersect. Intersections can be calculated in different ways. Here we chose the *earliest time assumption* [36] which is commonly used in the scheduling literature. Under this assumption a set of intervals $S = \{\mathcal{I}_1, \ldots, \mathcal{I}_n\}$ intersect iff

$$(\textit{intersect}\ S) \Leftrightarrow \min_{\mathcal{I} \in S}(EET\ \mathcal{I}) > \max_{\mathcal{I} \in S}(EST\ \mathcal{I}). \tag{4.5}$$

Table 4.3 lists a few more temporal queries that can be used to infer properties of propagated temporal intervals. Unlike constraints they will not be enforced via propagation but just evaluated to be true or false.

The consistency of the set of temporal constraints in Table 4.2 can be tested by solving a *Simple Temporal Problem (STP)* [66]. A STP is represented by a distance graph between time-points. Initially this distance graph ensures that the Earliest and Latest Start and End Times satisfy $(EST\ \mathcal{I}) \leqslant (LST\ \mathcal{I})$, $(EST\ \mathcal{I}) \leqslant (EET\ \mathcal{I})$, $(EET\ \mathcal{I}) \leqslant (LET\ \mathcal{I})$ and $(LST\ \mathcal{I}) \leqslant (LET\ \mathcal{I})$ for all intervals $\mathcal{I}$. It is then updated for every constraint according to Table 4.2. A constraint is satisfiable if the distance graph does not contain a negative cycle after the update. The approach we use for TestAndResolve-*temporal* solves the STP with an incremental version of the all-pairs-shortest-path algorithm [86] which we extended with a way to remember previous states of the distance matrix. This allows the solver to revert to these states without propagating from scratch. This is useful for quickly testing a series of resolvers that lead to temporal inconsistencies. Incremental path consistency requires $O(|\Phi[\textit{statement}]|^2|\Phi[\textit{temporal}]|)$ time. Alternatively, Floyd-Warshal's All-Pairs-Shortest-Path algorithm requires $O(|\Phi[\textit{statement}]|^3)$ time [86]. Instead of propagating from scratch after each inconsistency (which requires cubic time) the STP solver jumps back to a stored copy of the distance graph and adds new constraints from there. This realizes a trade-off between memory usage and computational efficiency. Queries do not require propagation and the ones used in this paper can be tested efficiently using the propagated distance graph. Note that more recent approaches for solving the STP [187] could easily be substituted due to the modularity of our approach.

4.4.4 Prolog Constraints

Prolog constraints are useful to compactly express relations between objects that do not change over time. We use them to describe features of the environment, such as, room adjacency or features and preferences of humans in the environment, such as, associations between rooms and offices to specific people. Prolog constraints pose requirements and relations between objects. They are evaluated wrt. a user-provided Prolog program (i.e., background knowledge) $\mathcal{B}$. A set of Prolog constraints $\mathcal{Q}$ is answered by posting a query to a Prolog interpreter that provides a procedure

$$\textsc{PrologQuery}(\mathcal{Q},\mathcal{B}) = \begin{cases} \{\sigma_1,\ldots,\sigma_n\} & \text{if } \mathcal{Q} \text{ can be satisfied given } \mathcal{B} \\ \emptyset & \text{otherwise} \end{cases}$$

which answers a query $\mathcal{Q}$ by all possible consistent substitutions σ_i. It returns an empty set in case the query fails since there is no consistent substitution. If all Prolog constraints are satisfied without substitution PrologQuery returns exactly one empty substitution. The semantics of Prolog constraints are defined, e.g., by Bratko [27].

Relation	$\lvert B\rvert$	Description
$(\textit{release}\ \mathcal{I}\ B_1)$	1	$(ST\ \mathcal{I}) \in B_1$
$(\textit{deadline}\ \mathcal{I}\ B_1)$	1	$(ET\ \mathcal{I}) \in B_1$
$(\textit{at}\ \mathcal{I}\ B_1\ B_2)$	2	$(\textit{release}\ \mathcal{I}\ B_1) \wedge (\textit{deadline}\ \mathcal{I}\ B_2)$
$(\textit{duration}\ \mathcal{I}\ B_1)$	1	$((ET\ \mathcal{I}) - (ST\ \mathcal{I})) \in B_1$
$(\textit{equals}\ \mathcal{I}_1\ \mathcal{I}_2)$	0	$(ST\ \mathcal{I}_1) = (ST\ \mathcal{I}_2) \wedge (ET\ \mathcal{I}_1) = (ET\ \mathcal{I}_2)$
$(\textit{before}\ \mathcal{I}_1\ \mathcal{I}_2\ B_1)$	1	$((ST\ \mathcal{I}_2) - (ET\ \mathcal{I}_1)) \in B_1 \wedge B_1^l > 0$
$(\textit{after}\ \mathcal{I}_1\ \mathcal{I}_2\ B_1)$	1	$((ST\ \mathcal{I}_1) - (ET\ \mathcal{I}_2)) \in B_1 \wedge B_1^l > 0$
$(\textit{meets}\ \mathcal{I}_1\ \mathcal{I}_2)$	0	$(ET\ \mathcal{I}_1) = (ST\ \mathcal{I}_2)$
$(\textit{met-by}\ \mathcal{I}_1\ \mathcal{I}_2)$	0	$(ST\ \mathcal{I}_1) = (ET\ \mathcal{I}_2)$
$(\textit{starts}\ \mathcal{I}_1\ \mathcal{I}_2\ B_1)$	1	$(ST\ \mathcal{I}_1) = (ST\ \mathcal{I}_2)$ $\wedge (ET\ \mathcal{I}_2) - (ET\ \mathcal{I}_1) \in B_1 \wedge B_1^l > 0$
$(\textit{started-by}\ \mathcal{I}_1\ \mathcal{I}_2\ B_1)$	1	$(ST\ \mathcal{I}_2) = (ST\ \mathcal{I}_1)$ $\wedge (ET\ \mathcal{I}_1) - (ET\ \mathcal{I}_2) \in B_1 \wedge B_1^l > 0$
$(\textit{during}\ \mathcal{I}_1\ \mathcal{I}_2\ B_1\ B_2)$	2	$(ST\ \mathcal{I}_1) - (ST\ \mathcal{I}_2) \in B_1$ $\wedge (ET\ \mathcal{I}_2) - (ET\ \mathcal{I}_1) \in B_2$ $\wedge\ B_1^l > 0 \wedge B_2^l > 0$
$(\textit{contains}\ \mathcal{I}_1\ \mathcal{I}_2\ B_1\ B_2)$	2	$(ST\ \mathcal{I}_2) - (ST\ \mathcal{I}_1) \in B_1$ $\wedge (ET\ \mathcal{I}_1) - (ET\ \mathcal{I}_2) \in B_2$ $\wedge\ B_1^l > 0 \wedge B_2^l > 0$
$(\textit{finishes}\ \mathcal{I}_1\ \mathcal{I}_2\ B_1)$	1	$(ET\ \mathcal{I}_1) = (ET\ \mathcal{I}_2)$ $\wedge (ST\ \mathcal{I}_1) - (ST\ \mathcal{I}_2) \in B_1 \wedge B_1^l > 0$
$(\textit{finished-by}\ \mathcal{I}_1\ \mathcal{I}_2\ B_1)$	1	$(ET\ \mathcal{I}_2) = (ET\ \mathcal{I}_1)$ $\wedge (ST\ \mathcal{I}_2) - (ST\ \mathcal{I}_1) \in B_1 \wedge B_1^l > 0$
$(\textit{overlaps}\ \mathcal{I}_1\ \mathcal{I}_2\ B_1)$	1	$(ST\ \mathcal{I}_2) - (ST\ \mathcal{I}_1) > 0$ $\wedge (ET\ \mathcal{I}_2) - (ET\ \mathcal{I}_1) > 0$ $\wedge\ (ET\ \mathcal{I}_1) - (ST\ \mathcal{I}_2) \in B_1 \wedge B_1^l > 0$
$(\textit{overlapped-by}\ \mathcal{I}_1\ \mathcal{I}_2\ B_1)$	1	$(ST\ \mathcal{I}_1) - (ST\ \mathcal{I}_2) > 0$ $\wedge (ET\ \mathcal{I}_1) - (ET\ \mathcal{I}_2) > 0$ $\wedge\ (ET\ \mathcal{I}_2) - (ST\ \mathcal{I}_1) \in B_1 \wedge B_1^l > 0$
$(\textit{met-by-or-after}\ \mathcal{I}_1\ \mathcal{I}_2\ B_1)$	1	$((ST\ \mathcal{I}_1) - (ET\ \mathcal{I}_2)) \in B_1 \wedge B_1^l \geqslant 0$
$(\textit{before-or-meets}\ \mathcal{I}_1\ \mathcal{I}_2\ B_1)$	1	$((ST\ \mathcal{I}_2) - (ET\ \mathcal{I}_1)) \in B_1 \wedge B_1^l \geqslant 0$
$(\textit{during-or-equals}\ \mathcal{I}_1\ \mathcal{I}_2\ B_1\ B_2)$	2	$(ST\ \mathcal{I}_1) - (ST\ \mathcal{I}_2) \in B_1$ $\wedge (ET\ \mathcal{I}_2) - (ET\ \mathcal{I}_1) \in B_2$ $\wedge\ B_1^l \geqslant 0 \wedge B_2^l \geqslant 0$

Table 4.2: Semantics of all temporal constraints. $\mathcal{I}$, $\mathcal{I}_1$ and $\mathcal{I}_2$ refer to the intervals. B_i to the i'th bound and B_i^u/B_i^l to the upper/lower bound of B_i. Bounds and relations are constraints on start time (ST) and end time (ET) of the constrained intervals.

Relation	Description
(*duration-greater-than* $\mathcal{I}$ v)	$(\mathsf{EET}\ \mathcal{I}) - (\mathsf{EST}\ \mathcal{I}) > v$
(*duration-less-than* $\mathcal{I}$ v)	$(\mathsf{EET}\ \mathcal{I}) - (\mathsf{EST}\ \mathcal{I}) < v$
(*start-time-greater-than* $\mathcal{I}$ v)	$(\mathsf{EST}\ \mathcal{I}) > v$
(*start-time-less-than* $\mathcal{I}$ v)	$(\mathsf{EST}\ \mathcal{I}) < v$
(*end-time-greater-than* $\mathcal{I}$ v)	$(\mathsf{EET}\ \mathcal{I}) > v$
(*end-time-less-than* $\mathcal{I}$ v)	$(\mathsf{EET}\ \mathcal{I}) < v$

Table 4.3: Six temporal queries that are used to test if temporal intervals fulfill specific requirements on their start times, end times and durations.

If $\mathcal{Q}$ contains variable terms it may lead to a flaw since there may be multiple consistent ground instances (i.e., $|\textsc{PrologQuery}(\mathcal{Q}, \Phi)| > 1$). From this we can immediately define

$$\mathit{Flaws}(\mathit{prolog}, \Phi|\mathcal{B}) = \{\textsc{PrologQuery}(\Phi[\mathit{prolog}], \mathcal{B})\} \tag{4.6}$$

always creating exactly one flaw which can be resolved by any of the substitutions that were returned by PrologQuery:

$$\mathit{Resolvers}(\mathit{prolog}, \kappa, \Phi) = \{\{\sigma\}|\sigma \in \kappa\} \tag{4.7}$$

If PrologQuery fails it will return the empty set and consequently

$$\mathit{Resolvers}(\mathit{prolog}, \emptyset, \Phi) = \emptyset$$

leads to an unresolvable flaw. If there is only one possible substitution or no substitution is necessary, the resolver will contain exactly one element.

If background knowledge for a domain is static we often have the possibility to evaluate it only once as a preprocessing step rather than for every suggested plan as we will show later in this Section. This is similar to what many planners when detecting and compiling away static predicates. If background knowledge can change (e.g., by adding preferences of a new person in the environment), we either have to repeat the preprocessing or resolve flaws during solution search.

Example 4.11. *Below we see an example of a Prolog knowledge base describing the adjacency of locations that can be visited by agents. We load a background knowledge file into our domain definition (under the name "kb") and use it to put constraints on the adjacency between two locations wrt. knowledge base "kb":*

```
(:include (kb "kb.prolog") )
(:prolog kb (adjacent ?L1 ?L2))
```

where (*adjacent* ?*L1* ?*L2*) *is a query that must be satisfied by the Prolog knowledge base kb that is shown below. The code for kb is written in Prolog. Prolog variables are upper case,* :- *reads "if" and the comma reads "and".*

The full stop marks the end of a rule. For more details on Prolog see, e.g., Bratko [27]. According to the code below (*adjacent* ?*L1* ?*L2*) *is true if either* (*adjacencyTable* ?*L1* ?*L2*) *or* (*adjacencyTable* ?*L2* ?*L1*) *is true.*

```
adjacencyTable(location1,location2).
adjacencyTable(location1,location3).
adjacencyTable(location2,location4).
adjacent(L1,L2) :- adjacencyTable(L1,L2).
adjacent(L1,L2) :- adjacencyTable(L2,L1).
```

The procedure PROLOGQUERY *will return the following set of substitutions which will be interpreted as a single flaw by Equation 4.6.*

$$\{\ \{\mathtt{L1/location1}, \mathtt{L2/location2}\}, \mathtt{L1/location2}, \mathtt{L2/location1}\}$$
$$,\ \{\mathtt{L1/location1}, \mathtt{L2/location3}\}, \mathtt{L1/location3}, \mathtt{L2/location1}\}$$
$$,\ \{\mathtt{L1/location2}, \mathtt{L2/location4}\}, \mathtt{L1/location4}, \mathtt{L2/location2}\}\}$$

Each of these substitutions will then be proposed as a single resolver according to Equation 4.7.

For decidability and complexity it will become important that we stick to a decidable subset of Prolog. This is possible under Assumption 4.1 which essentially enforces a recursion free Prolog program. For more complex cases decidability has to be shown on a case by case basis. However, Assumption 4.1 should suffice in many practical cases.

Assumption 4.1. *Consider the graph* $\mathsf{G} = (\mathsf{V}, \mathsf{E})$ *where* V *are the predicates in a Prolog knowledge base and* E *are directed edges from predicates in the head of every clauses to every predicate in the body of that same clause. We assume* G *is acyclic.*

In practice Assumption 4.1 can be violated to permit more complex background knowledge (e.g., with recursion). In this case the time complexity behavior may change and Assumptions 4.4, 4.5 and 4.7 (see Section 4.5.1) for Prolog constraints have to be considered on a case-by-case basis.

To solve Prolog constraints we create a prolog query from the conjunction of all Prolog constraints. Each Prolog constraint has an associated Prolog program that is used to evaluate it. (This distinction is the purpose of the identifier *kb* in Example 4.11.) We considered two approaches.

The first approach implements TESTANDRESOLVE-*prolog*. For each Prolog program it takes all corresponding Prolog constraints in a CDB, creates the query and runs Prolog to get all answers.

The second approach implements PREPROCESS-*prolog* and is shown in Algorithm 1. Algorithm 2 replaces all operators and *ICs* by ones that are consistent with their Prolog constraints. The impact of using this is to remove the entire set of flaws originating from Prolog by changing the set of available operators.

This can be done if Prolog background knowledge is static. This frequently allows to move all the work to PREPROCESS-*prolog* and reduce the number of flaws that have to be resolved by Algorithm 1. This can have a significant impact on the size of the search space (see Section 4.5.2).

The complexity of answering a Prolog query depends on the background knowledge that is used to evaluate it. Under Assumption 4.1, let n_P be the maximum number of literals in the body of any clause. Let m_P be the maximum count of rules that use the same predicate in their head literal. Finally, let l be the longest possible path in the graph created in Assumption 4.1. Then the time complexity of answering a Prolog query can be bound by $O(\max(m_P, n_P)^{2l})$, which is the exploration of the corresponding and-or tree.

Algorithm 2 Preprocessing Prolog constraints

Require: Φ - a CDB, $\mathcal{O}$ - a set of operators, $\mathcal{B}$ - User provided knowledge (e.g., Prolog knowledge base)

```
 1: global B, O                                   ▷ Available to all functions
 2: function PREPROCESS-prolog(Φ)
 3:     for o ∈ O do        ▷ Replace operators with substitutions that satisfy Prolog constraints.
 4:         for κ ∈ FLAWS(prolog, C_o|B) do
 5:             for {σ} ∈ Resolvers(prolog, κ, Φ|B) do
 6:                 O ← (O \ {o}) ∪ σ(o)
 7:     for c ∈ Φ[ic] do        ▷ Replace ICs with substitutions that satisfy Prolog constraints.
 8:         for κ ∈ FLAWS(prolog, Condition(c)|B) do
 9:             for {σ} ∈ Resolvers(prolog, κ, Φ|B) do
10:                 C ← (C \ {c}) ∪ σ(c)
```

Example 4.12. *The Prolog query*

```
query(R,S,L1,L2,D,V,T) :-
    size(R,S), distance(L1,L2,D), speed(R,V), divide(D,V,T).
```

is created from the Prolog constraints of the Move operator (Example 4.17) that uses the distance between locations and the speed of the robot to determine the time T *it will take to move. It yields all substitutions of the variables* R ,S, L1, L2, D, V *and* T *that are consistent with the background knowledge. In case there are alternatives (e.g., multiple speed settings for the robot), a flaw needs to be resolved.*

4.4.5 Reusable Resources

A state-variable x, with an integer domain, becomes a reusable resource if there exists a *capacity constraint*

$$(\textit{reusable}\ x\ c_x)$$

where c_x is an integer representing the capacity. A reusable resource x is such that its capacity is consumed by a certain amount $u \leqslant c_x$ by a statement $(\mathcal{I}\ x\ u)$. The usage ceases after the interval $\mathcal{I}$ of the statement, and the resource does

not require replenishing. A resource flaw in a CDB Φ occurs when a resource is potentially used above its capacity during any period of time.

Example 4.13. *The resource constraint below allows a maximum of* 4 *units for the resource* (*space loc1*). *Both statements use* 3 *units of the resource. If they can overlap (which they can without any temporal constraints), there is a resource flaw. This flaw can be resolved by adding a temporal constraint* (*before* R1 R2 [1 inf]) *or* (*before* R2 R1 [1 inf]).

```
(:statement
    (R1 (space loc1) 3)
    (R2 (space loc1) 3) )
(:resource (reusable (space loc1) 4) )
```

The set of all usages of a specific resource χ in a CDB Φ is the set

$$Usages(\chi, \Phi) = \{s \in \Phi | SV(s) = \chi \wedge (reusable\ \chi\ c) \in \Phi\},$$

where $SV(s)$ is the state-variable of statement s. This set is empty for non-resource state-variables. All resource flaws in a CDB Φ are in the set

$$\begin{aligned} Flaws(resource, \Phi) = \bigcup_{\chi \in SV(\Phi)} \{\, S \in 2^{Usages(\chi,\Phi)} \quad | \quad & (intersect\ S) \\ \wedge \quad & (reusable\ \chi\ c) \in \Phi \\ \wedge \quad & \sum_{s \in S} Val(s) > c\} \end{aligned}$$

assuming integer values for all resource variables χ (see Equation 4.5 for the definition of (*intersect* S)).

Resolvers for these flaws add temporal constraints (*before* and *after* from Table 4.2) that remove intersections of intervals to assure resource usage stays within capacity.

$$\begin{aligned} Resolvers(resource, \kappa, \Phi) \quad = \quad & \{(before\ \mathcal{I}_1\ \mathcal{I}_2\ [1\ inf]), (before\ \mathcal{I}_2\ \mathcal{I}_1\ [1\ inf]) | \\ & \mathcal{I}_1 \neq \mathcal{I}_2 \wedge (\mathcal{I}_1\ \chi_1\ v_1) \in \kappa \wedge (\mathcal{I}_2\ \chi_2\ v_2) \in \kappa\} \end{aligned}$$

State variables can have only one value assigned at a time, which creates a similar (implicit) resource problem.

To solve the reusable resource problem we directly adopt the precedence constraint posting method proposed by Cesta, Oddi and Smith [36]. Their notion of flaws is called minimal conflict sets. A minimal conflict set (put in terms of this thesis) is a set of statements that together overuse a resource (creating a flaw) and temporally separating any two statements in this minimal conflict set resolves the flaw. Consider, for instance, a resource with capacity 1 and three resource usage statements whose intervals intersect. In this case there would be three minimal conflict sets containing two statements each.

They propose two sampling-based strategies to find minimal conflict sets that take $O(|\Phi[\textit{statement}]|)$ and $O(|\Phi[\textit{statement}]|^2)$ time respectively. The number of flaws $|K|$ can also be bound by $O(|\Phi[\textit{statement}]|)$ or $O(|\Phi[\textit{statement}]|^2)$ (depending on the strategy). Flaws are ordered based on temporal flexibility, so once the set of flaws K is created it takes $O(|\Phi[\textit{statement}]|)$ or $O(|\Phi[\textit{statement}]|^2)$ to select the least flexible flaw. Since flaws can be sorted while being discovered the effort for finding and selecting flaw are added.

Resolvers are precedence constraints between pairs of statements in a flaw. In the worst case we have $|\Phi[\textit{statement}]|$ flaws and it takes $O(|\Phi[\textit{statement}]|^2)$ time to create all resolvers. Resolvers are also selected based on the temporal flexibility after their application, choosing the most flexible one. The strategy of selecting the least flexible flaw and the most flexible resolver is a common strategy for variable- and value- selection in constraint processing [65]. Overall the time complexity of TestAndResolve-*resource* is bound by

$$O(|\Phi[\textit{statement}]|^2)$$

4.4.6 Costs

Costs can be used to evaluate the quality of solutions and to enforce a maximum on certain criteria. Common uses of costs involve required money or energy consumption. In this thesis we mainly use costs to account for negative impacts on user experience (i.e., to put a value on the "human-awareness" of a plan). For some constraints violations may be acceptable but lead to a decreased user experience modeled by a cost. Costs are assigned in the form of cost constraints and can be constrained by cost inequalities. They have no temporal interpretation and we only allow increasing them, which implies that one can safely prune any plan that violates a cost inequality. Cost constraints as introduced in the following do not create flaws (and thus need to resolvers). A cost is increased by a constraint

$$(\textit{add}\ x\ c)$$

where x is a term and c is an integer[3]. The meaning of this constraint is that value c is added to the cost represented by term x. Costs can be used to evaluate the quality of a plan or create inconsistencies when reaching a maximum value by providing a cost inequality constraint

$$(\textit{less-than-or-equal}\ x\ c_{max}).$$

[3]Note that while CDBs are sets we may need to add the same amount twice to the same cost. Here we assume that each $(\textit{add}\ x\ c)$ also has a unique key term, but we will not include it in our representation.

Constraint	Meaning
(add S e)	add element e to set S
(in S e)	true iff $e \in S$
(not-in S e)	true iff $e \notin S$
(subset S1 S2)	true iff $S1 \subseteq S2$
(proper-subset S1 S2)	true iff $S1 \subset S2$

Table 4.4: List of supported set constraints

We then can define

$$Satisfiable(cost, \Phi) \Leftrightarrow ((less\text{-}than\text{-}or\text{-}equal\ x\ c_{max}) \in \Phi \Rightarrow \Big(\sum_{(add\ x\ c_i\ o) \in \Phi} c_i \Big) < c_{max}).$$

Example 4.14. *Example of a set of cost constraints. In this thesis we use costs to measure the negative impact of planning decisions on the social acceptability.*

```
(:cost
    (add social 10)
    (add social 5)
    (less-than social 20) )
```

To evaluate costs we simply need to add up all costs assigned to cost terms. An inconsistency is found if a cost inequality is violated for any cost term. The time complexity of cost evaluation is bound by $O(|\Phi[cost]|)$, since we simply have to calculate all costs and check inequalities.

4.4.7 Sets

We use a simplified notion of set constraints that allows to construct sets by providing the elements they contain. The resulting fixed sets are tested for set membership and subset relations. Thus these set constraints do not require flaw-resolution and can be verified by creating all sets and testing all membership constraints. Table 4.4 summarizes the supported constraints. A more complex notion of set constraints could be substituted but this would also increase the complexity of solving them [2].

Example 4.15. *Example of set constraints we will use later to manage participants of a meeting.*

```
(:set
  (set meeting)
  (add meeting human1)
  (add meeting human2)
  (in meeting human1)
  (not-in meeting human3) )
```

4.4.8 Goals

One of the main requirements (FR_{Gl}) stated in Chapter 1 was the *HaPlEx*'s ability to reach goals by using available operators. To achieve this we use goal constraints. A goal constraint

$$(goal\ \mathfrak{g})$$

sets statement $\mathfrak{g} = (\mathfrak{a}\ \mathfrak{x}\ \mathfrak{v})$ as a goal. This in turn creates an open goal flaw. This flaw can be resolved by establishing a causal link from an existing statement $\mathfrak{s} = (\mathfrak{b}\ \mathfrak{x}\ \mathfrak{v}) \in \Phi[statement]$ to the goal statement. If no such statement exists actions may be applied in some combination to produce an effect that achieves this goal.

Example 4.16. *The constraints below define a simple planning problem in which we have a robot r at location a. The goal is to move it to location b. This could be achieved by applying a move action (see Example 4.17) that changes the robot's position from a to b.*

```
(:statement (s (at r) a) )
(:goal (g (at r) b) )
```

Goals are resolved by linking them to existing statements. Our notion of causal links is slightly different from the one we saw in Section 3.1.4. Here we rely on temporal constraints to establish causal links.

Definition 4.8 (Causal Link)**.** *A causal link between two statements* $(\mathfrak{a}\ \mathfrak{x}\ \mathfrak{v})$ *and* $(\mathfrak{b}\ \mathfrak{x}\ \mathfrak{v})$ *is established by a temporal constraint* (*equals* $\mathfrak{a}\ \mathfrak{b}$)*. In other words, the goal is reached by setting its interval to be equal to the interval of a statement with the same variable* $\mathfrak{x}$ *and value* $\mathfrak{v}$*. This is equivalent to replacing the interval* $\mathfrak{a}$ *of the goal with the interval* $\mathfrak{b}$*.*

As a consequence of Definition 4.8 both statements will be subject to the same set of temporal constraints. Whatever statement is chosen to reach a goal is constrained by all temporal constraints on that goal.

Operators (or actions) are provided with the domain definition. They describe ways of transforming one CDB into another by adding effect statements and constraints. Operators are used to reach open goals via their effect statements. In order to apply an operator we must add and satisfy all its constraints. The set of constraints usually contains temporal constraints to describe how long it will take to execute the operator. Temporal constraints also relate precondition and effect statements of the operator to the operator itself. Other

common choices of constraints are costs and resource usages. A concrete instantiation of such an operator is called an action. An operator

$$o = (\mathtt{name}_o\ \mathcal{P}_o\ \mathcal{E}_o\ \mathcal{C}_o)$$

consists of a $\mathtt{name}_o$ statement, two sets of statements $\mathcal{P}_o$ and $\mathcal{E}_o$ for preconditions and effects and a set of constraints $\mathcal{C}_o$. If we compare this definition of an operator to the one provided in Section 3.1.1, we use statements in preconditions and effects instead of state-variable assignments. This allows to directly reference the intervals of these statements in temporal constraints. Another key difference is the set of constraints $\mathcal{C}$ that can be of any type.

Example 4.17. *The move operator below has one precondition and two effects. The interval* ?*THIS refers to the interval associated to the operator itself. Precondition* ?*P ends right before* ?*THIS starts due to the temporal constraint* (*meets* ?P ?THIS). *Effect* ?*E1 starts when* ?*THIS ends. The duration of* ?*THIS is in the interval* ?*T and inf and models the time it takes to change location from* ?*P1 to* ?*E1. This allows for temporal flexibility in case the movement takes longer than expected which is often the case when robots are involved. To trigger an inconsistency if movement takes too long (which would lead to re-planning), it is possible to set a finite maximum.* ?*T is determined by Prolog constraints and depends on the distance between* ?*L1 and* ?*L2 and the speed* ?*S of the robot. Statement* ?*E2 uses a resource* (*location-usage* ?L2) *that restricts the space of location* ?*L2. (The fact that* (*location-usage* ?L2) *is a resource is not apparent from the operator alone. This is established by a resource constraint (Section 4.4.5), such as* (*reusable* location − usage 2), *which sets a resource capacity of two for this variable.) The amount of required units of the resource depends on the size of the robot* (*size* ?*R* ?*Size*). *The temporal constraint* (*equals* ?E1 ?E2) *ensures that the resource is used as long as the robot is at location* ?*L2. Note that in this formulation we do not need to explicitly delete the old value of the state-variable since we assign a new one. Along the same line we do not need to explicitly free any resource usage for the previous location since the temporal interval associated to such a usage ends as soon as the robot leaves the location.*

```
(:operator (move ?R - robot ?L1 - location ?L2 - location)
  (:preconditions (?P (at ?R) ?L1) )
  (:effects
    (?E1 (at ?R) ?L2)
    (?E2 (location-usage ?L2) ?Size) )
  (:constraints
    (:temporal
      (duration ?THIS [?T inf])
      (meets ?P ?THIS)
      (meets ?THIS ?E1)
      (equals ?E1 ?E2) )
    (:prolog kb
      (size ?R ?Size)
      (distance ?L1 ?L2 ?D)
      (speed ?R ?V)
      (divide ?D ?V ?T) ) ) )
```

To apply an operator o to a CDB Φ we simply convert o to a CDB

$$\Phi_o = \mathcal{E}_o \cup \mathcal{C}_o \cup \{\sigma_o\} \cup \mathcal{C}_{Goals} \text{ where } \mathcal{C}_{Goals} = \{(goal\ p) | p \in \mathcal{P}_o\} \tag{4.8}$$

where σ_o is a substitution that instantiates the variable terms in o and substitutes all non-instantiated variables with new unique variables. (This is necessary to make sure that adding the same operator twice does not add the same variable.) $\mathcal{C}_{Goals}$ contains a goal constraint for every precondition. This means that all preconditions will have to be satisfied in order to apply an action. Note that if we ignore the fact that $\mathcal{C}_o$ can contain any type of constraint this is very close to plan-space planning (see Section 3.1.4). We can now apply the operator to a CDB by using the transition:

$$\Phi' = \gamma(\Phi_o, \Phi) = \Phi \cup \Phi_o$$

The set of open goal flaws is simply the set of goals that have not been achieved by any statement in a CDB:

$$\begin{aligned} Flaws(goal, \Phi) = \{(a\ x\ v) \quad | \quad & (a\ x\ v) \in \Phi[goal] \\ \wedge \quad & \nexists (b\ x\ v) \in \Phi[statement] \\ & : (equals\ a\ b) \in \Phi[temporal]\} \end{aligned}$$

A resolver establishes a link $l = (equals\ g\ s)$ either to an existing statement or by adding operators in a way that resolves the flaw:

$$\begin{aligned} Resolvers(goal, (g\ x\ v), \Phi|\mathcal{O}) \quad = \quad & \{\Phi_r | \Phi_r = \{(equals\ g\ s)\} \\ & \wedge (s\ x\ v) \in \Phi[statement]\} \\ \cup \quad & \{\Phi_r | \Phi_r = \{(equals\ g\ s)\} \cup \Phi_o \\ & \wedge o \in \mathcal{O} \wedge (s\ x\ v) \in \mathcal{E}_o \end{aligned}$$

Obviously, $Resolvers(goal, (g \times v), \Phi|\mathcal{O})$ is finite as long as $\mathcal{O}$ models a finite set of operators. So it satisfies Assumption 4.5 (see Section 4.5.1) as long as all variables in $\mathcal{O}$ have a finite domain.

Example 4.18. *We can resolve the open goal in Example 4.16 by using the move operator from Example 4.17: First we apply the substitution*

$$\{?R/rob1, ?L2/loc2, ?E1/e1\_1, ?E2/e2\_1, ?P/p\_1\}.$$

Then we add a temporal constraint (*equals e1_1* g) *and* Φ_{move} *(as defined in Equation 4.8). The resulting CDB contains* (*goal* (*p_1* (*at rob1*) *?L1*)), *which can be resolved by linking to the statement in Example 4.16 by a resolver that contains the substitution* {?L1/loc1}, *and the link* (*equals* s *p_1*). *The substitutions of ?E1, ?E2 and ?P assure that the intervals used by the action are unique (i.e., when using the same operator twice different intervals will be used).*

Open goals are resolved by planning approaches. We discussed different ways to do this in the related work. Using plan-space planning in our representation is straight forward, since we defined satisfiability of goals in the same way (compare Sections 3.1.4 and 4.4.8). Causal links can be established by setting the intervals of statements that are preconditions temporally equal to statements of effects that achieve those preconditions (see Section 4.4.3 on temporal constraints). In addition, it is possible to leave threats on causal links to a resource scheduler. With this in mind we can apply flaw and resolver selection strategies directly from the literature on plan-space planning (e.g., Younes and Simmons [238]). All open goals can be found in $O(|\Phi[goal]|)$ (TestAndFindFlaws-*goal*) and it takes $O(|\mathcal{O}| + |\Phi[statement]|)$ to create all resolvers (InitResolvers-*goal*) from operators and existing statements (see Equation 4.9). The time it takes to select among flaws and resolvers is usually polynomial, but the exact term depends on the employed heuristic. A range of possible choices for heuristics was summarized, e.g., by Younes and Simmons [238].

In the following sections, we present an approach based on forward planning. The basic idea of this approach it to extract a state-space planning problem from the operators and statements in the constraint-based planning problem. We then solve this problem with forward planning heuristics and convert the actions in the solution plan back to constraint-based operators. The motivation behind this approach is that we can use existing forward planning heuristics to guide the search. As pointed out in Section 3.1.1 (State-space Planning), these heuristic have shown very good performance, e.g., in the international planning competition [1].

To preserve some of the temporal expressiveness, we allow to assign sequences of values to state-variables. This allows to use any value of assigned sequences as preconditions. In addition, these sequences contain precedence constraints between goals. This allows to capture temporal constraints on these

goals to some degree. All constraints other than open goals and temporal constraints are ignored when creating a plan. This approach resolves all open goals at once (i.e., each resolver represents a plan reaching all open goals). In the following few subsections we will detail the approach. First, we introduce the extended state-variable planning problem. Second, we discuss how it is extracted from a CDB. Third, we show how it is solved and finally we show how its solution is used to create a resolver to achieve open goals in a CDB.

Extended State-variable Planning Problem

To resolve open goals with a heuristic forward planning approach, we propose an extension of the state-variable based planning problem (Section 3.1.1). This will allow us to consider temporal information to some degree by assigning sequences of values to a state-variable rather than single values. In addition, goals are separated based on the earliest start time of their intervals. This problem is extracted for the set of all open goals in a CDB and enables the use of heuristic forward planning.

In the extended state-variable planning problem the state of the world is described as a set of state-variable assignments of the form

$$s = variable \leftarrow \langle value_1, \ldots, value_n \rangle \tag{4.9}$$

with a finite set of possible variables and values. We use a sequence of values to be able express an operator causing multiple changes of the same state-variable. If an operator is applied, the sequences will be assigned to the corresponding state-variables. This allows to use intermediate values as preconditions for other operators.

We use $Var(s)$ and $Var(S)$ to access the set of state-variables that are assigned by the state-variable assignment s or all assignments in set S of state-variable assignments. We use $S[x]$ to access the value sequence that S assigns to variable x (providing an empty sequence in case $x \notin Var(S)$). A set S of state-variable assignments where each variable is assigned at most once is called a state. In the same way we use $Values(s)$ to get the sequence of values assigned by s and $LastValue(s)$ to get the last value in the sequence. An operator

$$o = (name, \mathcal{P}, \mathcal{E})$$

consists of a *name*, a set of precondition assignments $\mathcal{P}$ and effect assignments $\mathcal{E}$. We require $\forall p \in \mathcal{P} : |Values(p)| = 1$ so that sequences of values are only permitted for effects. An operator describes how states can be changed given that they fulfill the requirements of the precondition. A concrete instantiation

of an operator is called an action. An action $a = (name, \mathcal{P}, \mathcal{E})$ is applicable to state S iff

$$\forall p \in \mathcal{P} : \begin{cases} \exists s \in S : Var(s) = Var(p) \wedge LastValue(p) \in Values(s) \\ \quad \text{if } Var(p) \notin Var(\mathcal{E}) \\ \exists s \in S : Var(s) = Var(p) \wedge LastValue(p) = LastValue(s) \\ \quad \text{otherwise} \end{cases}$$

The first case covers situations in which no effect of the action changes the precondition. In that case the precondition may be fulfilled by any value in the current sequence $Values(s)$. The second case considers that an effect overwrites the value of the precondition p. In this case that precondition must be satisfied by the last value of $Values(s)$ in s (i.e., $LastValue(s)$). The reason for this is that otherwise effects might interfere with already established sequences of values and interrupt them.

Applying action $a = (name, \mathcal{P}, \mathcal{E})$ with effects $\mathcal{E}$ to a state S yields a new state S′

$$S' = \gamma(a, S) = \{s \mid s \in S \wedge Var(s) \notin Var(\mathcal{E})\} \cup \mathcal{E} \tag{4.10}$$

such that any state-variable that appears in an effect takes the value sequence of that effect. We illustrate all these concepts in the following example.

Example 4.19. *The following simple move action models a robot moving from one location to another through an intermediate location* corridor.

```
(:operator (move r l1 l2)
  (:preconditions
    (?P (at r) l1) )
  (:effects
    (?E1 (at r) corridor)
    (?E2 (at r) l2) )
  (:constraints
    (:temporal
      (meets ?P ?THIS) (?THIS ?E1)
      (equals ?THIS ?E2) )
    (:cost (add cost 10)) ) )
```

This operator would be converted to to an action $a_1 = (name_1, \mathcal{P}_1, \mathcal{E}_1)$ *with*

$$\begin{aligned} name_1 &= (move\ r\ l1\ l2) \\ \mathcal{P}_1 &= \{(at\ r) \leftarrow l1\} \\ \mathcal{E}_1 &= \{(at\ r) \leftarrow \langle corridor, l2\rangle\}. \end{aligned}$$

Note that that non-temporal constriant types such as the cost in the original operator are ignored. The order of values in the sequence of $(at\ r)$ *is derived*

from the temporal constraints of the operator. Now consider the following two states

$$\begin{aligned} S_1 &= \{(at\ r) \leftarrow \langle a, b, c, d, l1\rangle\} \\ S_2 &= \{(at\ r) \leftarrow \langle a, b, l1, c, d\rangle\} \end{aligned}$$

The move action above can be applied to S_1 *and doing so would lead to the following state*

$$S_3 = \{(at\ r) \leftarrow \langle corridor, l2\rangle\}$$

However, a_1 *cannot be applied to* S_2 *since it overwrites the state-variable* $(at\ r)$. *The action* $a_2 = (name_2, \mathcal{P}_2, \mathcal{E}_2)$ *with*

$$\begin{aligned} name_2 &= (switch - light\ l1) \\ \mathcal{P}_2 &= \{(at\ r) \leftarrow l1\} \\ \mathcal{E}_2 &= \{(light\ l1) \leftarrow \langle on\rangle\} \end{aligned}$$

can be applied to both S_1 *and* S_2 *since it does not overwrite* $(at\ r)$.

A planning problem

$$\Pi = (I, \mathcal{O}, G = \langle G_1, \ldots, G_n\rangle)$$

consists of an initial state I a sequence of goal states G, a set of operators $\mathcal{O}$. A plan

$$\pi = \langle a_1, \ldots, a_n\rangle$$

is a sequence of actions. A plan π is applicable to a state s if all its actions are applicable in the given sequence. Its application yields a sequence of states

$$\langle S_0, \ldots, S_m\rangle$$

where $S_0 = I$ by applying all actions in π in turn (see Equation 4.10). A plan is a solution to a planning problem Π if it reaches all goals in the order of the sequence. More formally, this can be expressed with the following two conditions.

$$\forall i \in \{1 \ldots n\}, g = (x \leftarrow v) \in G_i : \exists j \in \{0, \ldots, n\} \text{ s.t. } v \in S_j[x]$$

This requires that each goal assigment is reached in some state in the sequence established by applying π.

$$\begin{aligned} &\forall i, j \in \{1 \ldots m\}, g_1 = (x_1 \leftarrow v_1) \in G_i, g_2 = (x_2 \leftarrow v_2) \in G_j : \\ &i < j \Rightarrow \exists k, l \in \{0, \ldots, n\} \text{ s.t. } k < l \wedge v_1 \in S_i[x] \wedge v_2 \in S_j[x] \end{aligned}$$

This condition enforces the sequence in which goals are reached to respect the sequence of goals $\langle G_1, \ldots, G_n \rangle$. Unlike classical planning, we do not require that all goals are achieved in the final state. The sequence $\langle G_1, \ldots, G_n \rangle$ allows to search for a plan that reaches G_1 first, then continue the search to reach G_2 and so on.

Heuristics

In this section we discuss how forward planning heuristics can be applied to the problem introduced in the previous section. Even though the approach assigns sequences of values to state-variables forward planning heuristics can still be used. This is possible by combining heuristic values for all combinations of single-valued states that can be created from the sequences in the multi-valued states (see Example 4.20). Since forward planning heuristics usually assume actions with a single effect per variable, we create combinations of single-valued effect actions from the multi-valued effect actions. The move action from Example 4.19 would lead to two actions (one reaching the corridor and one reaching the destination) as far as the heuristic is concerned.

Example 4.20. *Given a state*

$$S = \{x \leftarrow \langle a, b \rangle, y \leftarrow \langle c, d \rangle\}$$

we use the minimum of the four heuristic values of non-sequential states

$$\begin{aligned} S_1 &= \{x \leftarrow a, y \leftarrow c\} \\ S_2 &= \{x \leftarrow b, y \leftarrow c\} \\ S_3 &= \{x \leftarrow a, y \leftarrow d\} \\ S_4 &= \{x \leftarrow b, y \leftarrow d\}. \end{aligned}$$

This indicates that we want to keep the number of assignments of the same variable in an operator low for the approach to remain feasible. On the other side it provides a good way to keep some temporal information. In other approaches to temporal planning this problem is approached by compiling actions into start- and end- actions [54]. We refrained from selecting this solution, since we may have longer sequences of effect values in a single action.

We divide all open goals into the sequence $\langle G_1, \ldots, G_n \rangle$ according to their earlierst start-times. We then plan for these individual sets of goals and attempt to achieve the earliest ones first. The resulting plans are concatenated. This has several benefits. It might divide the planning problem into multiple smaller ones. It is more likely to satisfy temporal constraints since goals are achieved in the order of earlierst start-time. Finally, it allows to have multiple goals on the same state-variable. This allows to formulate sub-goals based on the results of other reasoners. As an example of this consider a transportation domain in which we have to decide which vehicles will load which objects. This problem

could be solved as a constraint satisfaction problem whose results are substituted into open-goals that have to be achieved before the final location of the cargo.

An interesting consideration for future work is to incorporate metrics from other reasoners as heuristic values. Costs, for instance, as the one that was discarded in Example 4.19 could be used by the plan search. A general solution to this would contribute to the transfer of knowledge from other constraint types to influence plan search. This in turn could lead to faster search and/or better plans due to more informed heuristics.

Extracting the Problem

The extended state-variable problem is created from a CDB Φ and a set of operators in the following way.

First, we extract the initial state I from the CDB Φ. To do this we propagate the temporal constraints of Φ and use the state-variables and values of the statements with the earliest start-times to form the initial state I.

Second, we need to create the set of operators (recall Example 4.19). A state-variable operator $o = (name, \mathcal{P}, \mathcal{E})$ is created from a constraint-based planning operator $o' = (name_o, \mathcal{P}_o, \mathcal{E}_o, \mathcal{C}_o)$ by removing the constraints $\mathcal{C}_o$. Multiple assignments of the same state-variables in effects are sequenced in order of their earliest start time (*EST*) and added as a single effect. Multiple preconditions on the same state-variable are not supported by the extended state-variable planning approach (leading to an error).

Finally, we create the set of open goals. To do this, all open goals in Φ are sorted by their earliest start-time. Then, the goal sequence $\langle G_1, \ldots, G_n \rangle$ is extracted, where all G_i are sets of goals. Goals with the same start-time are put into the same G_i. The goal sequences is sorted by the earliest start-time of the goals each set contains. To convert an individual open goal $g = (\mathcal{I}\ x\ v)$ from the CDB, we simply remove the interval to get the goal $g' = x \leftarrow v$.

Extracting the Resolver

Assuming that we found a solution to the extended state-variable problem we need to convert it back into a CDB that can be added as a resolver. To do this we take each action from the plan that solves the state-variable problem, take the corresponding constraint-based planning operators and add all effects and constraints to the resolver. We add a temporal constraint (*equals* I_p I_e) between a precondition and an effect if the latter achieves the former in the sequential plan. In addition, we create new temporal intervals for each effect statement. Preconditions that cannot be linked in this way must be linked to the initial CDB. TestAndResolve-*goal* performs the plan search and provides the next solution (if it exists) converted to a resolver and applied to the input CDB as described in this section.

For the computational complexity we need to look at the search space involved in the extended state-variable planning problem. Using systematic search we have to explore the entire state space in the worst-case. Observe that the sequences of states that can appear in any state are fixed by the operators. Thus, for the purpose of counting the number of possible states we can replace all sequences by unique values representing the sequence and map each possible state-variable assignment to one which assigns only a single value. Given domains D_i for n state-variables x_i the number of possible states is

$$\prod_{1 \leqslant i \leqslant n} |D_i| \in O(d^n) \text{ with } d = \max_i(|D_i|)$$

which is also the time complexity of TestAndResolve-*goal*.

In practice, this approach for TestAndResolve-*goal* provides an iterator over possible solutions rather than returning all solutions at once. This means that the set R of resolved CDBs in line 8 of Algorithm 1 is an iterator that provides members of the resolver set one at a time in line 12. Each iteration of the for-loop in line 12 continues the heuristic forward planning search from where it stopped previously. The search then treats the last solution as a failure and continues until it finds the next solution or until the search space has been exhausted. This avoids repeatedly exploring the same part of the search space. It also avoids finding all possible solutions before testing any of them (this would be the case if we were to expect all possible plans to be found in line 8 of Algorithm 1).

Limitations

This approach to resolving open goals is limited because it assumes a state-space: It only applies operators when their preconditions can be satisfied. This reduces the number of resolvers we can find and makes the approach incomplete. It enables, however, a much faster generation of resolvers due to the availability of forward planning heuristics. In principle it is always possible to fall back to a complete planning approach if no working resolver is found with the incomplete one. In practice, however, it may take a long time until the incomplete approach fails.

Another drawback of the presented approach is its limited consideration of future changes (e.g., scheduled changes or human activities). This could be addressed by adding transition operators that allow the planner to switch from the current value of a state-variable to a future value if that future value is known from statements in the CDB.

Example 4.21. *Consider the following statements using the same variable* door.

```
(:statement
  (i1 door closed)
  (i2 door open) )
```

```
(:temporal
  (at i1 [0 0] [50 50])
  (at i2 [50 50] [100 100]) )
```

The door is not controllable by the HaPlEx but it must be open to allow a robot to move through. In this case we can add the following transition operator[4] *that allows the HaPlEx to wait for the door to open.*

```
(:operator (transition-i1-i2)
  (:preconditions (i1 door closed))
  (:effects (i2 door open))
  (:constraints
    (:temporal
      (meets i1 i2) ) ) )
```

4.4.9 Interaction Constraints

Interaction Constraints (ICs) are our main tool to model human-aware planning knowledge. In Chapter 5 we will see how they can be used to provide social acceptability (FR_{SA}), proactivity (FR_{PA}) and context-awareness (FR_{CA}). In addition, they will prove very useful for planning for cooperation (FR_{PC}). *ICs* are used to enforce a set of constraints only under certain conditions. Whenever the condition of an *IC* applies to a context one of its resolvers must be applied. More formally, an *IC* c has the form

$$\begin{aligned} \mathrm{c} &= (\mathtt{name}, \mathit{Condition}(\mathrm{c}), \mathit{Resolvers}(\mathrm{c})) \\ \mathit{Resolvers}(\mathrm{c}) &= \langle \mathit{Resolver}_1(\mathrm{c}), \ldots, \mathit{Resolver}_n(\mathrm{c}) \rangle \end{aligned}$$

where $\mathtt{name}$ identifies the *IC* and $\mathit{Condition}(\mathrm{c})$ and all elements in the sequence $\mathit{Resolvers}(\mathrm{c})$ are CDBs. If there are no resolvers, the constraint-based search should fail as soon as the condition is satisfied.

Definition 4.9 (*IC* Flaw)**.** *An IC* c *leads to a flaw in a CDB* Φ *iff there exists a substitution* σ *s.t.*

$$\mathit{Condition}(\sigma(\mathrm{c}))[\mathit{statement}] \subseteq \Phi \ \wedge \ \mathit{Condition}(\sigma(\mathrm{c})) \cup (\Phi \setminus \Phi[\mathit{ic}])$$

has a solution when taking into consideration only the constraint types that appear in $\mathit{Condition}(\sigma(\mathrm{c}))$*. We say that* $\sigma(\mathrm{c})$ *is applicable to* Φ*.*

This means that the applicability of an *IC* to Φ does not depend on other *ICs* in Φ. Furthermore we ignore all constraint types that do not come up in *Condition* when testing its satisfiability. If there is a flaw for some *IC* c in a

[4]Note that we assume that the precondition and effect statements in this operator are already present in the CDB. For this reason, transition operators do not lead to the addition of new statements to the CDB.

CDB Φ we need to find a resolver $r \in \mathit{Resolvers}(c)$ such that $r \cup \Phi$ has a solution. Resolvers are prioritized in the order in which they are provided in the *IC* definition.

In human-aware planning the condition of an *IC* describes the properties of a context that needs to be further constrained (e.g., no vacuuming while child is sleeping) , could be acted upon (e.g., by posting a goal) or may indicate further information (e.g., by adding a statement that describes a group activity from several individual activities). Clearly, an *IC* of the form $(\mathit{Condition}, \emptyset)$ leads to an inconsistency as soon as it applies. This pattern can be used to forbid certain unwanted situations completely.

Example 4.22. *The constraint below applies whenever we have an activity of human ?H at location ?L. If human ?H dislikes robots and the activity intersects in time with a robot ?R also being at ?L. Whenever this condition holds in a CDB we have to apply one of the resolvers. The first two resolvers focus on making the temporal intersection impossible, while the third one offers to tolerate the situation by increasing the social cost of the plan.*

```
(:ic
  (hates-robots ?H ?L ?R ?A)
  (:condition
    (:statement
      (?I1 (human-activity ?H ?L ?A))
      (?I2 (at ?R) ?L) )
    (:temporal
      (intersection {?I1 ?I2}) )
    (:prolog kb
      (hasProperty ?H hatesRobots) ) )
  (:resolver (:temporal (before ?I1 ?I2 [1 inf]) ) )
  (:resolver (:temporal (after ?I1 ?I2 [1 inf]) ) )
  (:resolver (:cost (add social 5) ) ) )
```

As for operators, the *ICs* provided to the *HaPlEx* usually use variables and are used as templates for rules that apply to a variety of situations. To determine flaws we need to test all possible ways to match the statements in an *IC* to the statements in the CDB. To match the statements of two CDBs we define

$$\mathit{Match}(\Phi, \Psi) = \{\sigma_i | \sigma_i(\Phi[\mathit{statement}]) \subseteq \Psi[\mathit{statement}])\} \tag{4.11}$$

to be the set of all substitutions that match the set of statements in Φ (i.e., $\Phi[\mathit{statement}]$) to a subset of the set of statements in Ψ. If $\Phi[\mathit{statement}] \subseteq \Psi[\mathit{statement}]$ then $\mathit{Match}(\Phi, \Psi)$ will contain exactly one empty substitution.

If no match is possible we have $Match(\Phi, \Psi) = \emptyset$. Flaws are substitutions of an *IC* c whose condition is consistent (ignoring other *ICs* in Φ).

$$\begin{aligned} Flaws(ic, \Phi) \leftarrow \bigcup_{c \in \Phi[ic]} \{(c, \sigma) \quad | \quad & \sigma \in Match(Condition(c), \Phi) \\ \wedge \quad & Solvable((\Phi \setminus \Phi[ic]) \cup \sigma(Condition(c)), \\ & Types(Condition(c)) \\ \wedge \quad & Asserted(\sigma(c)) \notin \Phi\} \end{aligned} \tag{4.12}$$

where $Asserted(c)$ is added by any resolver of c to record that c has been resolved. This is needed since resolvers of *ICs* will not necessarily negate their associated conditions. As an example, consider the last resolver in Figure 4.22, where a cost is added and the situation is accepted.

Resolvers are part of the *IC* definition together with the special constraint $Asserted(c)$ to make sure we only resolve an *IC* once. For each statement that appears in a resolver we substitute its interval with one that has not been used yet (this is covered by the last two lines in the following equation) .

$$\begin{aligned} Resolvers(ic, (c, \sigma), \Phi) \quad \leftarrow \quad & \{\Phi_r \cup \{Asserted(\sigma(c)), \sigma_r\} \\ & \mid \Phi_r \in Resolvers(\sigma(c)) \\ & \wedge\, Ground(Int(\sigma_r(\Phi_r))) \\ & \wedge \nexists x \in Int(\sigma_r(\Phi_r)) \;:\; x \in Int(\Phi)\} \end{aligned} \tag{4.13}$$

The following assumption assures that the order in which *ICs* are evaluated does not influence the outcome.

Assumption 4.2. *If the condition of a IC* c *can be negated by applying the resolver* r *of any other flaw,* c *must have a resolver that achieves the same as* r.

Assumption 4.2 ensures that any way to make the condition of an *IC* inconsistent with a resolver of another flaw has to be available in the resolvers of the *IC* itself. This assumption is necessary to make sure that the order in which flaws are resolved does not matter. Otherwise it is possible to construct cases in which an *IC* only applies for a certain ordering of flaws which in turn means that we would be required to backtrack over the order in which flaws are resolved (rather than just backtracking over alternative resolvers). For constraint types we consider in the conditions of *ICs* in this thesis, Assumption 4.2 is easily satisfied. In other cases it might be harder. Consider, for instance, an *IC* that uses an open goal in its condition (i.e., the condition only applies if the goal can be reached). It is easy to see that resolvers of other constraints may make the goal unsatisfiable, but it seems difficult to specify an *IC* resolver that achieves the same.

Now we will discuss how *ICs* are resolved. Algorithm 3 goes over every *IC* (line 3). The procedure FindFlaw is used to find a substitution of an *IC* c

whose condition is satisfied. To test conditions all *ICs* are removed from Φ (line 2) according to Definition 4.9.

If no such substitution exists FindFlaw returns *Failure*. Otherwise a flaw was found (line 5). Algorithm 3 then tests if the substituted *IC* has already been asserted (line 6). If this is not the case it creates all modifications of Φ that resolve the flaw and returns them (lines 7 and 8). If no flaws are found it returns Φ (line 9).

We consider two alternative procedures for FindFlaw in this thesis. In earlier work [144] we used the procedure FindFlawBruteForce (Algorithm 4). This approach tries all ways to match the set of statements of the condition of *IC* c to a CDB Φ (line 2). It then tests if the condition is satisfied by using the procedure ResolveAll that was detailed in Algorithm 1. Before ResolveAll is called we modify Θ to only contain constraint types that occur in the condition.

There is a major drawback in FindFlawBruteForce. Consider an *IC* which has n statements in its precondition and that each of these statements can be matched to at most m existing statements in Φ. The loop in line 2 may have to consider m^n ways to match the statements of the condition to statements in Φ. This problem can be solved by casting the problem of finding a satisfied condition as a search problem. The recursive procedure FindFlawSearch (Algorithm 5) uses this idea. It takes all statements from the condition that are not yet matched to the CDB Φ (line 2). It then tests if the condition holds when all non-assigned statements are removed (lines 3 and 5). If this test fails it returns failure (line 6). Otherwise it will return substitution σ if there are no more unassigned statements (i.e., set S is empty, line 7). It will then pick one of the unmatched statements (line 8) and recursively try to find an assignment that matches all of them while satisfying the condition (lines 9 and 10). If no option works it fails (line 12). The advantage of Algorithm 5 over Algorithm 4 is in the fact that it can prune as soon as the condition fails for a partial assignment of matching substitutions.

Example 4.23. *Consider an IC with the following condition:*

```
(:statement
    (?I1 x1)
    (?I2 x2)
    (?I3 x3)
    (?I4 x4) )
(:temporal
    (intersect {?I1 ?I2 ?I3 ?I4}) )
```

Algorithm 4 will try all ways of matching the four intervals to intervals in a CDB. Algorithm 5 on the other hand will realize that when a substitution {?I1/0, ?I2/1} *fails because intervals* 0 *and* 1 *do not intersect there is not reason to even consider matches for* ?I3 *and* ?I4.

We will see in the benchmark part of the evaluation that this can lead to a significant improvement. If, on the other hand, every possible way to apply an *IC* is applicable, Algorithm 5 has a computational overhead compared to Algorithm 4. In practice this should not matter since it is hard to imagine meaningful *ICs* whose statements are not related to each other via temporal constraints. The statements in the condition of the *IC hates-robots*, for instance, are related by the intersection constraint. Without such a constraint the temporal context for these statements would be missing.

Both alternatives for FindFlaw will also match to partially specified statements. This is useful when only incomplete information is available about human-activities (TR_{Uncert}). Since the *HaPlEx* cannot choose, e.g., which human executes a certain activity it is forced to assure that the solution is consistent for every possibility. We provide some results for such partially specified human-activities in the evaluation.

Algorithm 3 Resolving a Single *IC* Flaw

Require: Φ - a CDB, Θ - a sequence of constraint types that determines the order of solvers

function TestAndResolve-*ic*(Φ)
  $\Phi' \leftarrow \Phi \setminus \Phi[ic]$
  for $c \in \Phi[ic]$ **do**
    $\sigma =$ FindFlaw($\emptyset, c, \Phi', \Theta$) ▷ Find substitution of c with satisfied condition.
    if $\sigma \neq$ *Failure* **then** ▷ Found a flaw
      if *Asserted*($\sigma(c)$) $\notin \Phi$ **then**
        $R \leftarrow \{\Phi \cup r | r \in$ *Resolvers*$(ic, \sigma(c), \Phi)\}$
        return R ▷ Return all ways to resolve flaw
  return $\{\Phi\}$ ▷ No flaws

Algorithm 4 Brute-force *IC* flaw detection

Require: c- a *IC*, Φ - a CDB, Θ - a sequence of constraint types that determines the order of solvers

function FindFlawBruteForce(_, c, Φ, Θ)
  for $\sigma \in$ *Match*(*Condition*(c), Φ) **do** ▷ Match condition to CDB
    $\Phi' \leftarrow (\Phi \cup$ *Condition*$(\sigma(c))$ ▷ Apply condition
    $\Theta' \leftarrow \Theta \setminus \{t \in \Theta | Condition(c)[t] = \emptyset\}$ ▷ Only keep types of condition
    if Resolve-All(Φ', Θ') **then** ▷ See Algorithm 1
      return σ ▷ Return substitution
  return *Failure*

Interaction constraints have many potential applications. We used them to formulate constraints for social acceptability (FR_{SA}) between robot plans and human activities. This has been investigated by Köckemann, et al. [144]. They can be used in a similar way to formulated constraint-based proactivity and context-awareness (FR_{PA}, FR_{CA}) which has been investigated in Köckemann, et al. [145]. A similar approach to proactivity was used, e.g., by Pecora et al. [185].

Algorithm 5 Search for *IC* flaw detection

Require: Φ - a CDB, Θ - a sequence of constraint types that determines the order of solvers
1: **function** FINDFLAWSEARCH(σ, c, Φ, Θ)
2: $S \leftarrow \{s | s \in \sigma(Condition(c)[statement]) \wedge s \notin \Phi\}$ ▷ All statements not yet matched.
3: $\Phi' \leftarrow \sigma(Condition(c) \setminus S) \cup \Phi$
4: $\Theta' \leftarrow \Theta \setminus \{t \in \Theta | Condition(c)[t] = \emptyset\}$ ▷ Remove unused types.
5: **if** RESOLVE-ALL(Φ', Θ) = *Failure* **then**
6: **return** *Failure*
7: **if** $S = \emptyset$ **then return** σ ▷ Found satisfied condition
8: Pick any $s \in S$
9: **for** $\sigma' \in Match(\{s\}, \Phi)$ **do** ▷ Recursively try substitutions
10: $\sigma'' \leftarrow$FINDFLAWSEARCH($\sigma \cup \sigma', Condition, \Phi, \Theta$)
11: **if** $\sigma'' \neq$ *Failure* **then return** σ''
12: **return** *Failure* ▷ No assignment works

ICs have similarities with approaches that utilize control knowledge such as TLPLAN or PTLPLAN [8, 131] or the way in which methods guide the search in HTN planning [179]. These approaches use control knowledge to guide the search and/or to reduce the search space of the planning problem. While *ICs* can be used in such a way (e.g., by forbidding certain combinations of actions), they can also have the effect to enlarge the search space. Statements in the resolver of an *IC*, for instance, could lead to a larger set of applicable actions. Goals in resolvers of *ICs* change the planning problem itself.

4.5 Formal Properties

In this section we will look into some formal properties of problem presented in Section 4.2.2 and the constraint-based planning algorithm we presented in Section 4.3.

4.5.1 Decidability

The question of decidability is of importance since it may limit the ability of any algorithm that attempts to solve problems in a given language. The constraint-based planning problem Π is decidable for a set of constraint types $\mathcal{T}$ if there exists an algorithm that can decide $Solvable(\Pi, \mathcal{T})$ for any constraint-based planning problem $\Pi = (\Phi, \mathcal{O}, \mathcal{B})$. This is the case if there exists a sequence of resolvers R that is a solution to Π (recall Definition 4.7) in the constraint-based search problem.

Definition 4.5. (Constraint-based Search) *The search space of constraint-based planning given a set of constraint types $\mathcal{T}$ is defined by its nodes, the solution test, and the successor function.*

- *Nodes: Constraint Databases Φ (Definition 4.1)*

- *Solution: Consistent*$(\Phi, \mathcal{T})$ *(Definition 4.2)*
- *Successor function: Successors*$(\Phi, \mathcal{T})$ *(Equation 4.4)*

The problem is semi-decidable if there exists an algorithm that will terminate if there exists a solution but may never halt if no solution exists. It is undecidable if there exists no algorithm that is guaranteed to terminate regardless of the problem having a solution or not.

In the following we consider two statements or constraints as distinct if they cannot be made equal by only renaming variables. Recall that CDBs are finite by Definition 4.1 and that resolvers can only add constraints but never delete them. Also recall that every constraint c regardless of its type has a relational form

$$c = (r\ t_1\ \ldots\ t_n),$$

where r is the relation and t_i are the arguments. For each interval

$$\mathcal{I} = ([EST, LST], [EET, LET])$$

we refer to $\mathcal{I}$ as the symbolic representation of the interval.

So far our discussion is completely independent of the properties of the types of constraints that we consider. We will make three assumptions on their properties that will help us to determine formal properties of the approach.

Assumption 4.3. (Finite Domains) *For every CDB* Φ *all variables* $x \in Var(\Phi)$ *have a finite domain* $D[x]$ *(with the exception variables representing temporal intervals).*

Assumption 4.3 allows infinite domains only for variables that represent temporal intervals. All other variables require finite domains. The following three assumptions restrict constraint types rather than CDBs.

Assumption 4.4. (Finite Flaws) *Given a set of types* $\mathcal{T}$*,* $Flaws(\Phi, \mathcal{T})$ *is finite for any CDB* Φ*.*

Assumption 4.5. (Finite Resolvers) *Given a set of types* $\mathcal{T}$*,* $Resolvers(\Phi, \mathcal{T})$ *is finite for any CDB* Φ*.*

Assumption 4.6. (Resolved Flaws) *Resolving a flaw of any type* t *removes it from the set of flaws for all succeeding CDBs:*

$$\begin{aligned} \forall \kappa \in Flaws(t, \Phi): \quad & \forall \quad r \in Resolvers(t, \kappa, \Phi) : \kappa \notin Flaws(t, \gamma(r, \Phi)) \\ & \wedge \quad \nexists \quad r' : \kappa \in Flaws(t, \gamma(r', \gamma(r, \Phi))). \end{aligned}$$

The second part assures that no resolver R *can cause the flaw* κ *to re-surface once* r *has been applied. Note that from Equation 4.1 it follows that a single resolver can lead to any successor and thus* r' *covers all possible successors of* Φ *in a single step.*

Assumption 4.5 assures that for each flaw there is a finite set of choices of how to resolve it (i.e.,the search space will have a finite breadth). Assumptions 4.4 and 4.6 control the depth of the search space.

These four assumptions and the following ones are restrictions on the types of constraints that can be used and on the language $\mathcal{L} = 2^{\Phi}$ which constitutes the set of all possible CDBs that we can express. The following assumption assures that deciding if a CDB is a solution ($Consistent(\Phi, \mathcal{T})$ from Definition 4.2) is decidable.

Assumption 4.7. *Given a set of constraint types $\mathcal{T}$, for each constraint type $t \in \mathcal{T}$ we assume that Consistent$(\Phi, \mathcal{T})$ is decidable.*

Given that *Consistent*$(\Phi, \mathcal{T})$ is decidable for the set of types $\mathcal{T}$ and that the problem at hand is a search problem. For any CDB Φ and set of constraint type $\mathcal{T}$, let the set S_Φ be defined as follows

$$\begin{aligned} \Phi &\in S_\Phi \\ \Phi'' &\in S_\Phi \Leftarrow (\Phi' \in S_\Phi \wedge \Phi'' \in \mathit{Successors}(\Phi', \mathcal{T})). \end{aligned}$$

In other words S_Φ is the set of all nodes that can be visited when starting the search from Φ. Using this we add the following assumption.

Assumption 4.8. *For each constraint type t given a CDB Φ the set of all constraints of this type that can appear is $\mathcal{C} = \bigcup_{s \in S_\Phi} s[t]$. Given the relational form of these constraints*

$$(\mathit{rel}\ t_1\ \ldots\ t_n),$$

let $Rel = \{rel | (rel\ t_1\ \ldots) \in \mathcal{C}\}$ be the set of all distinct relations that appear in all CDBs in the set S_Φ. We assume that Rel is finite.

This assumption restricts constraint types that might have an infinite amount of possible relations to a finite amount of actually used relations for any concrete CDB Φ. There exists, for instance, an infinite amount of possible Prolog constraints since r is an arbitrary string. However, given a concrete CDB the ones that can appear in the set S_Φ are finite and known and no new Prolog relations are generated during flaw resolution. If any term t_i in the relational form has an infinite domain, there may still be an infinite amount of constraints for any relation r.

If the set S_Φ is finite, the problem is obviously decidable because it can be explored systematically. If the search space can have an infinite depth but a finite branching factor, making the size of S_Φ infinite and the problem at least semi-decidable. This is the case because breadth-first search can explore it systematically and will find a solution if one exists. It may, however, never terminate if no solution exists. A constraint-based search tree with infinite depth means that there exists a path in the search tree along which there exists no

node that satisfies either $\mathit{Consistent}(\Phi, \mathcal{T})$ or $\mathit{DeadEnd}(\Phi, \mathcal{T})$ given the constraint types $\mathcal{T}$.

As a result of this discussion we would like to make sure that the depth of the search space is finite to ensure a finite search space. However, Assumptions 4.4, 4.5, 4.6, and 4.8 alone are not enough to guarantee this. Consider the following example.

Example 4.24. *The consistency of the problem below cannot be proven. The IC applies to the original statement and its resolver satisfies its own condition. If the number of distinct symbolic intervals is not limited this IC leads to an infinite chain for flaw-resolution. Hence, the second part of Definition 4.2 cannot be proven by systematic search. There might be methods to decide this type of inconsistency but we are not aware of any.*

```
(:statement (i1 x))
(:ic
  (infinite-flaws ?I1)
  (:condition
    (:statement (?I1 x)))
  (:resolver
    (:statement (?I2 x))))
```

The problem we observe in Example 4.24 originates from the fact that we do not include temporal intervals in Assumption 4.3. This means that the set of all possible statements becomes infinite which can lead to an infinite chain of new flaws. The problem considered so far resembles deciding PLAN-EXISTENCE [105, Ch. 3]. According to Assumption 4.3 (finite domains) the only terms that have an infinite domain are temporal intervals. Consider the following adaption of the search problem that resembles deciding PLAN-LENGTH [105, Ch. 3] with the difference that it limits the number of intervals used by a plan rather than the number of actions.

Definition 4.10. (Limited Interval Constraint-based Search) *The following defines the search for a solution with at most* k *statements using the set of constraint types* $\mathcal{T}$.

- *Nodes: Constraint Databases* Φ *(Definition 4.1)*
- *Solution:*

  $\mathit{Consistent}(\Phi, \mathcal{T}) \wedge |\mathit{Int}(\Phi)| \leqslant \mathrm{k}$ *(limiting max. number of statements)*
- *Successor function:*

  $\mathit{Successors}\text{-}\mathrm{k}(\Phi, \mathcal{T}) = \{\Phi' \in \mathit{Successors}(\Phi, \mathcal{T}) \mid |\mathit{Int}(\Phi')| \leqslant \mathrm{k}\}$

Theorem 4.1. *Under Assumptions 4.3, 4.4, 4.5, 4.6, and 4.8 for any CDB* Φ *the set* S_Φ *defined by*

$$\begin{aligned} \Phi &\in S_\Phi \\ \Phi'' &\in S_\Phi \Leftarrow (\Phi' \in S_\Phi \wedge \Phi'' \in \textit{Successors}\text{-k}(\Phi', \mathcal{T})) \end{aligned}$$

has a finite number of elements.

Proof. Given a set of constraint types $\mathcal{T}$ and a CDB Φ with finite domains for each variable (Assumption 4.3) and that we need to consider at most k temporal intervals, recall that each constraint c has a relational form

$$c = (r\ t_1 \ldots t_n)$$

with a finite scope. According to Assumption 4.8 we know there is a finite number of distinct relations r for each type in $\mathcal{T}$ given Φ. Each argument t_i of the relational form now has a finite domain $D[t_i]$. This means that the overall number of distinct constraints that can be expressed given the set of all domain D is finite. Since there exists only a finite number of distinct constraints that can be expressed for each type in $\mathcal{T}$ given Φ the number of possible successors must be finite as well. □

According to Theorem 4.1 we are left with a finite set of successors which can be explored with systematic search. As a result, solution existence for the constraint-based search problem with a limited number of intervals is decidable.

Assumption 4.4 (finite flaws) together with Assumption 4.5 restricts the branching factor of the search space to a finite number of possible successors for each CDB. As long as the maximum number of intervals is not limited, however, it may be the case that the depth of the search space becomes infinite (as shown in Example 4.24). Even under Assumptions 4.3 and using at most k intervals it can still be the case that the exact same flaws may re-appear, which is why in general, Assumption 4.6 (flaws do not re-appear) is needed. In our present framework resolvers only add constraints, so Assumption 4.6 is satisfied naturally in most cases. It would become hard to guarantee Assumption 4.6 if we were to allow removal of constraints as part of resolvers. Assumption 4.5 (finite resolvers) holds naturally for all constraint types and approaches discussed in this thesis, as long as we consider only distinct minimal resolvers, where we consider a resolver of a flaw minimal if the removal of any of its elements would disqualify it from being a resolver of that flaw. Assumption 4.8 is not required by any constraint type considered in this thesis but necessary to avoid cases in which new relations r in the relational form can be generated infinitely. We cannot think of any practical constraint type that would require relaxing this assumption.

This discussion is independent of the concrete types of constraints that we use, which makes it possible to extend the introduced model easily with new types of constraints as long as we make sure to satisfy the necessary assumptions.

4.5.2 Computational Complexity

The complexity analysis of Algorithm 1 can be divided into two quantities. The first one is the number of nodes visited in the corresponding search space (i.e., how often is line 13 reached). The second one is the effort required for each node.

The way in which Algorithm 1 is stated enables us to analyze the time complexity of constraint-based planning independently of the types of constraints that we actually use. We will do this by creating a template that can be used on a concrete instance of Algorithm 1 (i.e., given required procedures for all used constraint types). The question to be answered is: What is the time complexity of Algorithm 1, given a CDB Φ, a set of operators $\mathcal{O}$ an ordering of constraint types? In the following we use T as the set of all constraint types. As before, let $\mathcal{L}$ be the set of all possible CDBs. The function describing the overall time complexity is

$$f : \mathcal{L} \to \mathbb{N}.$$

Algorithm 1 is a depth-first search. We use $d(\Phi)$ to denote the maximum depth of the search space (i.e., the maximum number of flaws that are resolved before backtracking or finding a solution) and $b(\Phi)$ to denote the maximum branching factor (i.e., the maximum number of resolvers over all flaws), where CDB Φ is the root node of the search, the number of nodes can be bound by

$$\sum_{0 \leqslant i < d(\Phi)} b(\Phi)^i \in O(b(\Phi)^{d(\Phi)}). \tag{4.14}$$

Recall, that according to Assumption 4.5 (finite resolvers) the branching factor is finite. The maximum depth might be infinite even under Assumptions 4.3, 4.4, and 4.6 if we do not restrict the maximum number of available intervals (recall Definition 4.10). Starting from Equation 4.14 we will now analyze Algorithm 1 as a search that is composed of multiple searches. More concretely, each constraint type t spans a search space of its own. Let $d(\Phi, t)$ and $b(\Phi, t)$ be the maximum number of flaws and resolvers required by constraint type t. For any constraint type t that does not require flaws to be resolved we use

$$\begin{aligned} d(\Phi, t) &= 0 \\ b(\Phi, t) &= 1 \end{aligned}$$

Letting n be the total number of constraint types we will use an integer representation for each constraint type. We assume, for now, that all flaws of constraint type $t \in \{1, \ldots, n\}$ are resolved before moving on to the next one $t + 1$.

We can compute the size of the search space for a type t by counting the number of ways to reach type t and then for each such way, computing the search space for t. Combining the leaves of all previous search trees for $1 \leqslant$

$k < t$, there are $\prod_{1 \leqslant k < t} b(\Phi, k)^{d(\Phi,k)-1}$ ways to reach type t. For each way to reach the search tree for type t there are $\sum_{0 \leqslant i < d(\Phi,k)} b(\Phi, t)^i$ nodes in this tree. Using this we can sum over the number of nodes in each tree leading us to Equation 4.15. Applying Equation 4.14 leads us to Equation 4.16. By changing indices and exponents we can simplify to Equation 4.19, which is bound by Equation 4.20 (recall that n is a constant):

$$\sum_{1 \leqslant t \leqslant n} \Big(\Big(\prod_{1 \leqslant k < t} b(\Phi, k)^{d(\Phi,k)-1} \Big) \sum_{0 \leqslant i < d(\Phi,t)} b(\Phi, t)^i \Big) \tag{4.15}$$

$$\in O\Big(\sum_{1 \leqslant t \leqslant n} \Big(\Big(\prod_{1 \leqslant k < t} b(\Phi, k)^{d(\Phi,k)-1} \Big) b(\Phi, t)^{d(\Phi,t)} \Big) \Big) \tag{4.16}$$

$$\subseteq O\Big(\sum_{1 \leqslant t \leqslant n} \Big(\prod_{1 \leqslant k \leqslant t} b(\Phi, k)^{d(\Phi,k)} \Big) \Big) \tag{4.17}$$

$$\subseteq O\Big(\sum_{1 \leqslant t \leqslant n} \Big(\prod_{1 \leqslant k \leqslant n} b(\Phi, k)^{d(\Phi,k)} \Big) \Big) \tag{4.18}$$

$$\subseteq O\Big(n \Big(\prod_{1 \leqslant k \leqslant n} b(\Phi, k)^{d(\Phi,k)} \Big) \Big) \tag{4.19}$$

$$\subseteq O\Big(\prod_{0 \leqslant t \leqslant n} b(\Phi, t)^{d(\Phi,t)} \Big). \tag{4.20}$$

Note that in Algorithm 1 we cannot assume that we can resolve all flaws of one type before moving on to the next. Resolving an open goal, for instance, may create another resource flaw. However, Equation 4.20 is valid for arbitrary re-orderings of flaws since multiplication is commutative. As a result, the worst-case number of nodes considered by Algorithm 1 is bound by Equation 4.20.

The second quantity, the effort per node, is the effort of ResolveAll for some Φ without recursion (since recursion uses the successor nodes). Thus it can be estimated by summing the effort

$$f(\Phi, t) \tag{4.21}$$

required for each TestAndResolve-t. Combining the two quantities we can bound the effort required to solve the constraint-based planning problem for a CDB Φ by

$$f(\Phi) = \sum_{1 \leqslant t \leqslant n} f^{\text{PRE}}(\Phi, t) + \prod_{1 \leqslant t \leqslant n} b(\Phi, t)^{d(\Phi,t)} \sum_{1 \leqslant k \leqslant t} f(\Phi, k). \tag{4.22}$$

Equation 4.22 models the effort of exploring a search space composed of all constraints. The worst-case depth of this search space is $\sum_t d(\Phi, t)$. The maximum branching factor of this search space is $\max_t b(\Phi, t)$. With this we

can "plug-in" concrete instances of Equation 4.21 for every type t to analyze a specific approach. This, in turn, allows us to analyze concrete instances of Algorithm 1.

4.5.3 Soundness & Completeness

In this section we address formal properties of Algorithm 1. An algorithm is sound if any solution it returns is a solution to the problem it is supposed to solve. An algorithm is complete if it will eventually return a solution in case one exists [105].

Theorem 4.2. *If all* TestAndResolve-t *procedures are sound (i.e., they do not miss any flaw or inconsistency) Algorithm 1 is sound.*

Proof. Algorithm 1 only returns Φ as a solution if no flaws exist and no constraint type causes TestAndResolve-t to fail (i.e., to return $\emptyset$). This means all TestAndResolve-t return Φ unchanged and the only way in which such a solution can violate our definition of consistency (Definition 4.2), is a non-sound TestAndResolve-t procedure. □

Algorithm 1 is not complete since it implements a depth-first search that may never backtrack if there is no limit to search-depth (see Example 4.24). This could be remedied by changing the search strategy of the algorithm to use iterative deepening search.

As soon as we can provide a complete approach for every type of constraint, we can create an overall complete solver in the sense that it will eventually detect all flaws and try all available resolvers for each detected flaw and thus find any solution that is reachable in that way. This works under Assumptions 4.3, 4.5, 4.4, and 4.6 and using any search technique that avoids exploring an infinite path as shown in Example 4.24. Two examples of such search techniques are iterative deepening and breadth-first search.

4.6 Pruning

As was mentioned before some constraint types (especially goals) require a complex search of their own in order to resolve their flaws. Our solver for the extended state-variable planning problem described in Section 4.4.8 ignores all constraints when trying to find a plan. The advantage of this is that we may find a resolver quickly. If this resolver does not work, however, there might be a significant overhead in backtracking. To alleviate this we consider three different methods to discover dead-ends earlier and/or to prune the search space in case dead-ends are found. The first case simply considers partial plans considered by the open goal resolver. The two more complex ones use ideas from the constraint processing literature [65].

4.6.1 Testing Partial Solutions

The most basic way to prune is to test partial solutions. This is useful if generating a set of resolvers involves a complex search space of its own. This is the case for the extended state-variable planner. Instead of creating a solution that reaches all goals and only then test if there is a problem with another constraint type (e.g., a resource inconsistency), we extract a resolver r_π for a sequence of actions π that is the current node in the forward planning search. We then test use CB-PLAN$(\Phi \cup r, \Theta)$, where Θ is usually a subset of constraint types such as $\langle$*cost*, *temporal*, *resource*$\rangle$. If the partial resolver leads to an unsatisfiable problem we can safely prune all successors of the corresponding node π in the forward planning search.

The following two methods can be seen as generalizations of this idea: Once an inconsistency is detected how much can we generalize from it?

4.6.2 Minimal Conflicting CDB

In case a dead-end is discovered when attempting to solve a set of open goals with the method described in Section 4.4.8, we can attempt to prune by finding the shortest plan whose application is inconsistent. This can lead to significant number of pruned nodes in the search of TESTANDRESOLVE-*goal*. This pruning approach can be summarized as follows. Given a CDB Φ and a sequential plan $\pi = \langle a_1, \ldots, a_n \rangle$ and that $\Phi \cup \Phi_\pi$ is inconsistent (recall Section 4.4.8 on extracting a resolver $\Phi_p i$ from a sequential plan π). We find the shortest inconsistent plan via binary search [59, Ch. 27]. Let $l = 1$ and $r = n - 1$ we test the plan $\pi' = \langle a_1, \ldots, a_{(r-l)/2} \rangle$. If $\Phi \cup \Phi_{\pi'}$ is consistent we update $l := a_{(r-l)/2+1}$. In case of inconsistency we update $r := a_{(r-l)/2}$. If $l = r$ we return l and $\langle a_1, \ldots, a_l \rangle$ it the shortest conflicting plan that can be removed, together with all its successors, from the search space of the forward planner (TESTANDRESOLVE-*goal*).

This approach tests consistency of sub-sequences of actions by using binary-search. The time complexity is $f(\Phi) \log_2 |\pi|$, where $|\pi|$ is the length of the plan considered by FINDRESOLVERS-*goal* and $f(\Phi)$ is the time complexity of testing consistency (as defined in Equation 4.22) which may vary depending on the cause of the inconsistency and which constraint types we need to test. Obviously, this method does not add much to testing partial solutions (at least when we test partial solutions with the same or more constraint types). Which of the two is better depends on the domain. If we expect few inconsistencies, finding minimal conflicting CDBs may be superior to testing partial solutions. If inconsistencies are common (e.g., due to frequent resource violations from action choices), testing partial solutions will have less overhead.

Example 4.25. *Consider the following plan that loads a series of object on a truck, then moves the truck, and finally unloads all objects at the new location:*

```
 1) (load truck a)
 2) (load truck b)
 3) (load truck c)
 4) (load truck d)
 5) (load truck e)
 6) (move truck l1 l2)
 7) (unload truck a)
 8) (unload truck b)
 9) (unload truck c)
10) (unload truck d)
11) (unload truck e)
```

This plan solves a simple transportation problem. We assume that a failure occurs because the truck can only load object a *but no further object due to a resource constraint. Using Algorithm 1 together with the approach described in Section 4.4.8 and without testing partial solutions (previous section), this resource conflict will only surface after the whole plan was created. With the above method we can backtrack to the last feasible decision (1) and avoid a significant amount of backtracking, for instance, over the order in which objects are unloaded.*

4.6.3 Lifted Pruning

Lifted pruning attempts to replace constants by variables (also called lifting) and tests if the resulting CDB is still inconsistent. If this is the case all matching CDBs can be pruned. This method helps to avoid cases in which many alternative resolvers lead to similar inconsistencies. If the resolver still fails we can safely prune all matching resolvers. We refer to this method as LiftedPruning.

We have only employed this method on resolvers of the causal reasoner. There are some choices to consider in this approach. Deciding which constant terms to lift, which constraint types to test for consistency or how many resolvers could potentially be pruned by using it. Algorithm 6 is a greedy approach that starts by lifting a single term and, if still inconsistent, greedily lifts more. If lifting any single term leads to a consistent CDB, we can ignore it from then on. As before, the benefit of applying this method (and which strategy to choose for lifting) is domain dependent.

Example 4.26. *Consider the following plan (only showing action names for simplicity):*

```
1) (load ship cargo1)
2) (load ship cargo2)
```

Let us assume that the ship's capacity for loading cargo is reached by loading cargo1. *This creates a reusable resource inconsistency when attempting to load* cargo2. *Now, it may be the case that there is not just one but a large number of cargo objects that need transportation. Without any further information the*

Algorithm 6 Greedy Lifted Pruning

Require: Φ - a CDB, Φ_r - resolver
1: **function** LIFTEDPRUNING(Φ, Φ_r)
2: $\sigma = \{\}$ ▷ Empty substitution
3: *ignore* $\leftarrow \emptyset$
4: **while** *Ground*$(\sigma(\Phi_r)) \setminus$ *ignored* $\neq \emptyset$ **do**
5: $c \leftarrow$ SELECT(*Ground*$(\sigma(\Phi_r)),$ *ignored*)
6: $\sigma' \leftarrow \sigma \cup \{c/x\}$ where x is a variable and $x \notin Var(\Phi \cup \sigma(\Phi_r))$
7: **if** RESOLVEALL$(\Phi \cup \sigma_r)$ = *Failure* **then**
8: $\sigma \leftarrow \sigma'$
9: **else**
10: *ignore* $\leftarrow$ *ignore* $\cup$ c
11: **return** $\sigma(\Phi_r)$

causal reasoner may try each one in turn. The approach above will test the consistency of the same plan where cargo2 *is lifted to be a variable* C*:*

```
1) (load ship cargo1)
2) (load ship ?C)
```

If this it still fails, we can safely prune all matching plans from the search space of the causal reasoner.

The benefits of this method rely very much on the domain of the variables that can be lifted and the heuristic "attractiveness" of the other objects in their domain. If ?C in Example 4.26 has a domain with a large number of objects, we might be able to prune a large number of nodes from the plan search. If, however, none of these nodes would ever have been considered because the heuristic does not deem them interesting, it would be an overhead to remove them. The method could be extended to take both factors into account by checking domain sizes and heuristic values of potentially pruned nodes. The time complexity of this approach is determined by the number of generalizations that are tested and the constraint types included in this test. It may be interesting to use lifted pruning for other types of constraints as well but to justify the overhead of testing consistency we require a significant benefit.

4.6.4 Discussion

For the methods presented here the limiting factor is the complexity $f(\Phi)$ (Equation 4.22) and their use is determined by the amount of nodes that can be pruned compared to the overhead of finding prunable nodes. For this reason it would be interesting to investigate methods that can reach the same conclusions without actually testing consistency. In Example 4.26 knowing that it was the last decision that caused an inconsistency and knowing that it was a resource problem, we may be able to prune every lifted version of the last decision that uses at least the same amount of that resource. Such methods would

be less general since they have to be designed for (combinations of) specific constraint types, but could be much faster in reaching conclusions by exploiting constraint-specific features.

4.7 Online Planning & Execution

Solutions to a planning problem need to be maintained online and may need to be adapted when new human activities are observed (TR_{Online}), reflecting the fact that we may be uncertain about several aspects of human activities at planning time (TR_{Uncert}). In this section we describe how our approach integrates off-line and on-line reasoning about plans in a uniform way. As we explain in Section 4.7.2, we exploit the temporal aspect of statements (flexible temporal intervals) to realize a timeline-based online planning capability. To maintain a connection with hard- and software components (Figure 1.1) with use the *Robot Operating System* ROS[5]. ROS maintains a network of nodes that can publish data to topics and/or subscribe to topics to receive data. ROS nodes are programs that, e.g., publish a robot's sensor data, read sensor data from another node and apply a filter, or calculate a robot's path given a navigation goal together with mapping and localization information. ROS can also be used to send goals (such as target poses) to robotic systems.

To manage the advancement of time in a CDB during execution it is common practice [93, 184, 190] to maintain two temporal intervals representing the past and future, e.g., with the following set of temporal constraints.

```
(release past [t0 t0])
(deadline past [t t])
(meets past future)
(deadline future [inf inf])
```

Here, t_0 is the earliest time of relevance to the system, t is the current time and *inf* is the temporal horizon. Statements that are executed will be constrained wrt. *past* and *future* according to the current state of execution (see Section 4.7.2). When an execution update is performed the deadline constraint is replaced with the current time.

Algorithm 7 shows how a CDB is updated during execution. It first updates the current time by replacing the deadline constraint as mentioned above (lines 2 and 3). It then handles communication with the robot middleware, which here is assumed to be the Robot Operating System (ROS) [188] (line 4). It reads and writes data from and to ROS topics according to ROS constraints (see Section 4.7.1 for details). This information may come from sensor data processing algorithms (e.g., observed human activities). The CDB is then tested (line 5) for new flaws without regarding goals. If new open goals are added or some flaw cannot be resolved (line 6) replanning is triggered. In this case a CDB Φ_{start} is created (line 7) that only contains actions and resolvers whose

[5] www.ros.org

execution has already been started (i.e., the current state of execution). A new plan is created from this current state of execution (line 8). If replanning is not successful, the algorithm fails (line 9). Otherwise, reactors are updated (line 13) (see Section 4.7.2). Finally, in order to maintain scalability when planning online over long horizons, we use a simple "forgetting" mechanism (line 14) for statements and temporal constraints which is detailed in Section 4.7.3.

Online flaw-resolution, as illustrated by Algorithm 7 makes proactivity possible (FR_{PA}): Opportunities to act are flaws which are derived online when the CDB is updated. We discuss *ICs* for proactivity in Chapter 5. The drawback of re-planning in case of new open goals is that it requires re-solving parts of the problem that have been solved before in order to accommodate new goals. A more sophisticated approach would attempt plan-repair rather than re-planning.

Algorithm 7 Execution update

Require: $\mathtt{t}$ - the current time, $\mathtt{t}_{last}$ - time of last execution update, Φ - the execution CDB, Θ - a sequence of constraint types

1: **function** ExecutionUpdate($\mathtt{t}, \Phi, \Theta$)
2: $\quad\Phi \leftarrow \Phi \setminus \{(\mathit{deadline\ past}\ [\mathtt{t}_{last}\ \mathtt{t}_{last}])\}$
3: $\quad\Phi \leftarrow \Phi \cup \{(\mathit{deadline\ past}\ [\mathtt{t}\ \mathtt{t}])\}$
4: $\quad\Phi \leftarrow$ UpdateROS(Φ)
5: $\quad\Phi' \leftarrow$ ResolveAll($\Phi, \Theta - \mathit{goal}$)
6: $\quad$**if** $\Phi' = \mathit{Failure} \vee \mathit{Flaws}(\mathit{goal}, \Phi') \neq \emptyset$ **then**
7: $\quad\quad\Phi_{start} \leftarrow \Phi$ without future resolvers
8: $\quad\quad\Phi \leftarrow$ ResolveAll(Φ_{start}, Θ) ▷ Replanning
9: $\quad\quad$**if** $\Phi = \mathit{Failure}$ **then**
10: $\quad\quad\quad$**return** *Failure* ▷ No recovery possible.
11: $\quad$**else**
12: $\quad\quad\Phi \leftarrow \Phi'$
13: $\quad\Phi \leftarrow$ UpdateReactors(Φ)
14: $\quad\Phi \leftarrow$ Forget(Φ)
15: $\quad$**return** Φ

4.7.1 Interfacing with the environment: ROS Constraints

Various choices are available when it comes to interfacing the *HaPlEx* with the real world (i.e., the hard- and software components in Figure 1.1). The current implementation of the *HaPlEx* supports ROS. Many existing robotic platforms are compatible with ROS, which also provides convenient mechanisms to access sensor data and integrate sensor processing modules.

To interface with ROS we use a set of constraints that establish a connection between the *HaPlEx* and the ROS middleware. As a result the *HaPlEx* will have its own ROS node. Publisher constraints will make sure that the values of certain state-variables are published on a specified ROS topic. Subscriber constraints will make sure that values published on specific topics will be added as statements to the CDB. Usually ROS constraints are specified as part of the

initial CDB, but there are scenarios in which they could be added by operators (if, e.g., an operator requires to monitor a state-variable after it was applied).

Nodes in ROS are programs that are components of a distributed system. Nodes can represent a mobile robot or an array of sensors or software components, such as a filter, or the *HaPlEx*. Nodes exchange information by publishing messages to topics that other nodes subscribe to. A single node could, e.g., read data from a set of sensors and publish it to different topics. A node representing the *HaPlEx* could read the published data, publish inferred context, and send commands to nodes representing robots.

ROS messages range from simple data types such as integers, floats or strings, to structured types such as robot poses in specific reference frames, maps of the environment or data streams from a camera. Supporting ROS is convenient because of the wide support for robotic platforms and existing modules for localization, mapping, sensor data interpretation, motion planning, etc. To allow the *HaPlEx* to take part in the exchange of messages we provide two constraints. The subscriber constraint

```
(:ros (subscribe-to
          x ; Variable
          v ; Value
          topic ; ROS topic name
          m ) ) ; ROS message
```

assures that a statement with state-variable x and value v is added (or maintained) in the execution CDB based on messages published on a ROS topic by another node. The value of v is based on the ROS message msg that was last published to ROS topic *topic*.

Example 4.27. *As a concrete example consider the following constraint in which the value* ?L *of* (*at human*) *is taken from* ROS *topic* human_location *with message format* (String l ?L) *(where* l *is the variable name used by* ROS*).*

```
(:ros
    (subscribe-to
        (at human) ; Variable
        ?L ; Value
        human_location ; ROS topic name
        (String l ?L) ) ) ; ROS message
```

The publisher constraint

```
(:ros (publish-to
          x ; Variable
          v ; Value
          topic ; ROS topic name
          m ) ) ; ROS message
```

publishes the current value of v of a state-variable x as a message to a ROS topic. These constraints provide a convenient way to integrate constraint-based planning in a robotic system and with systems for activity recognition. This allows to update the *HaPlEx* online with observed human activities (TR_{Online}).

For more advanced cases ROS offers the *actionlib*[6], which can be used to send goals to ROS nodes. We therefore also provide support for actionlib goals.

```
(:ros (is-goal
    state-variable
    topic
    action
    goal-message ) )
```

Example 4.28. *As a concrete example of this consider the following goal to send a turtlebot to the coordinates* (X, Y) *in the reference frame* map *with a* MoveBaseAction *(see* `http://wiki.ros.org/move_base` *for details about* ROS *move_base and the message formats used here.).*

```
(:ros (is-goal
    (move-to ?X ?Y)
    /turtlebot_1/move_base
    MoveBaseAction
    (MoveBaseGoal g
      (PoseStamped target_pose
        (Header header (String frame_id /map))
        (Pose pose
          (Point position (float x ?X) (float y ?Y))
          (Quaternion orientation (float w 1.0)) ) ) ) ) )
```

4.7.2 Reactors

Reactors are processes that manage the execution of statements. These processes are responsible for changes in the real world, such as moving a robot. For statements $(\mathcal{I}\ x\ v)$ intended to trigger actuation, the *HaPlEx* will create a reactor online at the earliest start time $(EST\ \mathcal{I})$ and wait for a confirmation that execution has been started. After execution has been started the *HaPlEx* will wait for the reactor to signal that execution has finished. During this a reactor undergoes a sequence of states: *NotStarted*, *WaitingForStart*, *Started*, *WaitingForFinished*, and *Finished*. The transition between states leads to the addition and the removal of constraints that link the executed statement to *past* and *future* (see Table 4.5 for details). Our approach to reactors is similar to the one used by TREX [190, 191]. There are two transitions for a reactor that depend on feedback from the executed process:

- *WaitingForStart* $\rightarrow$ *Started*
- *WaitingForFinished* $\rightarrow$ *Finished*

A statement is assigned to a reactor in three cases.

1. If it is associated with a ROS goal;

[6] http://wiki.ros.org/actionlib

Transition	Change in the CDB
NotStarted → *WaitingForStart*	(A) (*met-by* $\mathcal{I}$ *past*)
WaitingForStart → *Started*	(R) (*met-by* $\mathcal{I}$ *past*) (A) (*release* $\mathcal{I}$ [t t]) (A) (*met-by-or-overlapped-by* future $\mathcal{I}$ [0 *inf*])
WaitingForFinished → *Finished*	(R) (*met-by-or-overlapped-by* future $\mathcal{I}$ [0 *inf*]) (A) (*deadline* $\mathcal{I}$ [t t])

Table 4.5: Changes of temporal constraints during reactor state transitions for some statement ($\mathcal{I}$ x v). (A) = Adding (R) = Removing. When reaching *Done* the execution CDB contains a fixed release and deadline for interval $\mathcal{I}$.

2. If it is an effect whose state-variable has a ROS subscription (i.e., the operator depends on an observation);

3. If it is associated with a custom reactor.

In the first case a ROS goal is sent to a ROS node and the reactor waits for the start confirmation. Once execution has started, the reactor waits for the finished confirmation. The ROS goal in Example 4.28 would be sent to a node. The executing process will confirm that that the movement of the robot has started and when it is finished.

In the second case the *HaPlEx* must wait until the effect has been observed on a topic subscribed to via ROS. Consider an operator that "moves the human". Since there is no control over human agents, all the system can do is ask and wait for the effect to be observed. Usually, plans are delayed via temporal constraints until the desired effect is observed (or a contingency condition is triggered). We will discuss a detailed example of this in our solution to the planning for cooperation use cases (Section 5.6).

The third case is used to allow easy addition of reactors without requiring ROS. In this thesis we use one custom reactor for the robotic test presented in Chapter 6: The special state-variable (*say phrase*) is executed by a reactor that uses text-to-speech to communicate a text associated to the term *phrase* to the user.

4.7.3 Forgetting

The amount of observations of human activities and other types of events (external or planned by the *HaPlEx*)) increases with the temporal horizon over which the *HaPlEx* is kept in operation. All these observations are represented by statements that contribute variables to the underlying Simple Temporal Problem (STP). The STP is solved via temporal constraint propagation

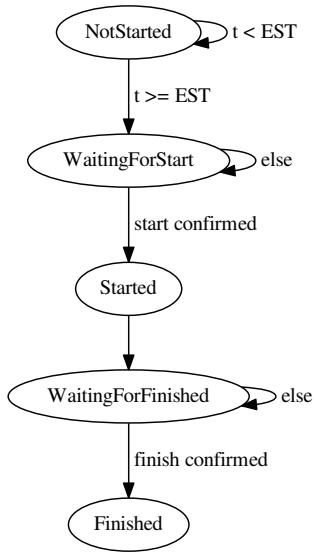

Figure 4.2: The state transitions that each reactor goes trough. Two of which depend on feedback from the software responsible for execution (e.g., feedback from a ROS node that its goal was reached).

at each update of the CDB, the computational overhead of which depends on the number of statements. The number of statements in a CDB may also have a large impact on the performance of resolving *ICs* (we will see this in detail in Section 6). In order to keep the performance of temporal constraint propagation and IC resolution feasible over time, we require a mechanism to "forget" past statements.

For this we use the following strategy. The set of statements whose intervals are fixed and lie in the past is

$$S = \{(\mathcal{I} \times v) | (EST\ \mathcal{I}) = (LST\ \mathcal{I}) \wedge (EET\ \mathcal{I}) = (LET\ \mathcal{I}) \wedge (LET\ \mathcal{I}) < T_{exec}\}$$

where T_{exec} is the current execution time. Any satisfied temporal constraint between two intervals whose statements are in S can safely be removed as it can never be violated and will have no impact on the solution of the problem. In a similar way we can remove any statement completely if its interval is not connected (via temporal constraints) to other intervals whose statements are in not S. The set of statements that we allow to forget is

$$\mathsf{F} = \{(\mathcal{I}\ \mathrm{x}\ \nu) \in \Phi \mid \nexists\, \mathcal{I}' : (\mathcal{I}'\ \mathrm{x}'\ \nu') \notin \mathsf{S} \wedge (\mathrm{r}\ \mathcal{I}\ \mathcal{I}') \in \Phi[T] \vee (\mathrm{r}\ \mathcal{I}'\ \mathcal{I}) \in \Phi[T]\}$$

Here we use $(\mathrm{r}\ \mathcal{I}\ \mathcal{I}')$ as an arbitrary temporal constraint between $\mathcal{I}$ and $\mathcal{I}'$. This is relatively easy to do since all required data is available as soon as the underlying STP has been solved. The drawback of this approach is in the assumption that the past will not be of interest as new statements are added based on observations. There are many practical cases where this is not true. Consider, for instance, planning to do the weekly laundry. Here it is important to know when the previous laundry was completed. Cases like this would require to develop more selective approaches to forgetting. In the spirit of the approach presented in this section a possible extension could be *execution constraints* that specify the rules of forgetting. This type of solution could be useful when the choice of which information to forget and when to forget is domain dependent. This is left for future work.

4.8 Discussion

In this chapter we presented an approach to constraint-based planning that fulfills the technical requirements that were posed in Chapter 1. The presented approach uses flaw-resolution to provide a modular and extendable (TR_{Modul}, TR_{Extend}) way to integrate many types of constraints without changing the theory. We also showed how to integrate it with execution (TR_{Online}) in order to deal with dynamically occurring events via replanning. In the next chapter we will see that this approach is indeed capable of addressing the requirements of human-aware planning and provides a convenient way to model solutions to the use cases for social acceptability, proactivity, and context-awareness (FR_{KR}).

An open-source implementation of this approach can be found at `http://spiderplan.org`. The set of available constraint types in this implementation contains more than was discussed in this thesis. Graph constraints, for instance, were added to describe graphs and query certain properties of these graphs (e.g., shortest path, maximum flow). We also provided a set of constraints that allow to provide input to the constraint processing language and solver MiniZinc [181, 166]. MiniZinc can be used in a similar way to Prolog (given a MiniZinc program as background knowledge), but it can also be used to specify optimization problems which can be useful to determine landmark goals (e.g., to decide who carries which cargo in a transportation domain).

In practice, the freedom of choice between different constraint types requires some care when writing domain and problem formulations. If a problem for some constraint type becomes very difficult there will be a huge impact on efficiency. In Chapter 6 we will also see that the order of solvers (Θ) can have a big impact on the time it takes to solve a problem. A good choice of constraint types, however, can be advantageous as it may detect inconsistencies

early and has the potential to divide an otherwise difficult problem into a set of manageable smaller problems.

Starting from the requirements analysis we proceeded in a bottom-up fashion. In this chapter we focused on the technical requirements of human-aware planning. In the next chapter we will build on this to provide solutions for the functional requirements by providing patterns of *ICs* for the social acceptability (FR_{SA}), proactivity (FR_{PA}), conext-awareness (FR_{CA}), and planning for cooperation (FR_{PC}). Together these patterns will provide a way for domain engineers to model human-aware planning knowledge for these requirements (FR_{KR}).

Chapter 5
Human-aware Reasoning & Planning

We will now define the human-aware planning problem and introduce patterns of *ICs* that provide solutions for social acceptability (FR_{SA}), proactivity (FR_{PA}) and context-awareness (FR_{CA}). These patterns also provide a guideline to modeling human-awareness (FR_{KR}). The usage of *ICs* contributes to most requirements since they allow to model complex conditions and to freely define resolvers. We will use the series of use cases introduced earlier to explain the use of *ICs* for human awareness. As we will see, small additions to human preferences lead to small increments in modeling, in partial achievement of functional requirement FR_{KR}[1].

The major advantages of the constraint-based planning approach presented in the previous chapter are its support for many different types of knowledge (TR_{Know}), its modularity, and its extendability (TR_{Modul}, TR_{Extend}). We use these properties to define the human-aware planning problem as follows.

Definition 5.1 (Human-aware Planning Problem). *The human-aware planning problem is a tuple*

$$\Pi = (\Phi \cup \Phi_{\mathcal{H}}, \mathcal{O} \cup \mathcal{O}_{\mathcal{H}}, \mathcal{B} \cup \mathcal{B}_{\mathcal{H}}).$$

It extends the Constraint-Based Planning Problem (see Definition 4.6) by adding the CDB $\Phi_{\mathcal{H}}$ that represents human activities (via statements) and interaction constraints for social acceptability, proactivity and context awareness (FR_{SA}, FR_{PA}, FR_{CA}). The set of operators is extended with $\mathcal{O}_{\mathcal{H}}$ to include operators that require human participation for planning for cooperation (FR_{PC}). The Prolog program $\mathcal{B}$ is extended with a description of human preferences, typically used to determine whether an IC applies or not.

[1] Minimal examples of all use cases and instructions to run them are available online at `spiderplan.org`.

In the following subsections we discuss commonly used constraint types and ways to model humans in the environment. Since we will compare our solutions to the use cases to possible PDDL solutions we will look into different features of the most recent PDDL version in some detail. Then we will discuss the challenges and provide an *IC* based solution for each of them, discuss complexity of the solution patterns and provide example applications.

5.1 Constraints for Human-Awareness

All of the solutions we provide make use of Interaction Constraints (*ICs*). We will start by providing a general pattern for *ICs* that tackle the functional requirements presented in Chapter 2. *ICs* are very flexible since they allow any type of constraint to be used in their condition and resolvers. In most cases temporal constraints are required to relate several statements in time. Human activities are modeled via statements. In most examples we use statements of the form

```
(:statement (?I (activity ?H ?L ?A)))
```

where ?*I* is the temporal interval of the activity, ?*H* represents a human, ?*L* represents a location and ?*A* represents the activity. The types of activities we consider depend on the application domain.

Social costs indicate that a resolver lowers the solution quality from a human-aware perspective. Increasing the social cost with a constraint such as

```
(:cost (add social 10))
```

is a useful way to resolve *ICs* that cannot be satisfied in any other way but should not lead to a failure of Algorithm 1.

Domain constraints can be used to apply an *IC* to a subset of possible values for a variable. For example,

```
(:domain (in ?A {reading working}))
```

Prolog constraints can be used in a similar way but are more powerful since they allow to express relations between objects. This in turn makes them useful when modeling preferences and properties of humans in the environment. The following example could be used to restrict access of someone's room to their work times in case they dislike robots:

```
(:prolog kb (roomOf ?L ?H) (hatesRobots ?H) (workingTime ?H ?T1 ?T2))
```

Statements in resolvers of *ICs* are used to infer context. If a statement in a resolver leads to a change in the environment, the corresponding *IC* has a proactive component. This provides simple proactive support in cases where no chain of actions is required. Open goals are used to achieve proactive support in more complex situations.

Planning for cooperation (FR_{PC}) requires a more sophisticated model of humans and their activities. If we need to create plans that include both humans and robots we need operator descriptions of possible human actions so the

HaPlEx can plan with them. We consider such operator models in Section 5.6 to create plans that involve both humans and robots. Operator descriptions of human activities also support plan recognition (which is outside the scope of this thesis).

Human goals and plans, given that they can and have been inferred, can be exploited by the *HaPlEx* to provide advanced support. Take, for instance, a human goal to be at work. The *HaPlEx* could deduce that this involves driving; knowing that the driving requires car keys would allow the *HaPlEx* to plan for fetching and delivering them to the user.

In the following sections we will show how information about humans can be modeled and then detail the patterns for *ICs* for social acceptability (FR_{SA}), proactivity (FR_{PA}) and context-awareness (FR_{CA}), as well as an approach to planning for cooperation (FR_{PC}). For each of these cases we discuss possible solutions for the corresponding set of use cases that our approach is capable of modeling (FR_{KR}).

5.2 Modeling Human-awareness with PDDL

Since the Planning Domain Definition Language (PDDL) is the standard when it comes to modeling planning problems, it is interesting to discuss the possibilities and limitations of its different versions viz-a-viz the requirements of human-aware planning, and to make a comparison to the possibilities and limitations of the approach presented in this thesis. We provide first a general discussion, then discuss possible PDDL solutions to the presented use cases in the respective subsections.

PDDL [103] provides a standard to express classical planning domains and is used by the International Planning Competition (IPC). PDDL was extended to support durative actions in PDDL2.1 [89], and *domain axioms* and *Timed-Initial Literals (TILs)* were subsequently added in PDDL2.2 [79]. The most recent version, PDDL3 [100], supports a notion of *constraints and preferences* on the states that are considered during plan search. PDDL+ [90] is a branch of PDDL2.2 that supports continuous time, events and processes.

Domain axioms and TILs, which were introduced in PDDL2.2, seem useful for human-aware planning. TILs are used to add information about future events. These could be used to allow a planner to consider future human activities which could be used in the preconditions of durative actions (PDDL2.1) to specify when specific actions are not allowed.

Example 5.1. *The following two TILs set an activity that starts at 20 and ends at 40.*

```
(at 20 (activity))
(at 40 (not (activity)))
```

This action take 100 timesteps to execute and its condition makes sure it is not executed during the activity.

```
(:durative-action do-something
  :parameters ()
  :duration (= ?duration 100)
  :condition
    (and (over all (not (activity))))
  :effect
    (and (at end task-complete) ) )
```

This equates to building social acceptability (FR_{SA}) into action applicability, which is inconvenient since it lacks modularity. Domain axioms (PDDL2.2) are realized via *derived predicates*, which are rules that determine the truth value of a literal in the state-space depending on a condition. It was shown [215] that domain axioms can lead to more compact domain formulations and plans, compared to compiling the same information into operators. Domain axioms can be used for modeling context-awareness (FR_{CA}) as they allow to derive predicates in the scope of a single state. They cannot, however, derive a future event from the current state as they have no notion of time.

Example 5.2. *The following example could be used to infer a simple cooking activity when a human is in the kitchen while the stove is on.*

```
(:derived
  (activity human-kitchen-cooking)
  (and
    (at human kitchen)
    (stove on) ) )
```

Constraints and preferences in PDDL3 can be used to limit the set of states that are considered when planning. This makes it possible, for example, to model rules for social acceptability (FR_{SA}): TILs can be used to describe a known schedule of human activities, and constraints to restrict the co-occurrence of these activities with planned actions. Combining TILs with constraints seems more convenient than using TILs in preconditions of durative actions as it allows to separate social acceptability from action applicability.

Example 5.3. *The following minimal constraint forbids an activity and robot action to occur at the same time.*

```
(:constraints
  (always (not (and
    (activity)
    (robot-action) ) ) ) )
```

The following example shows the above constraint as a preference that is stated together with a metric:

```
(:constraints
  (preference p1
```

```
    (always (not (and
      (activity)
      (robot-action) ) ) ) ) )
...
(:metric minimize (is-violated p1))
```

The drawback of this representation is that it lacks explicit and flexible temporal intervals, which makes it hard to express constraints that depend on complex temporal relations. It is also worth to note that we could not find a PDDL3 compatible planner (e.g., Optic [18], SGPlan5 [122]) that correctly handles PDDL constraints in conjunction with TILs. This makes it difficult to compare our approach to human-aware planning to a PDDL3 model using constraints and/or preferences.

The comparisons to PDDL in this chapter are bound to be inaccurate to some degree since the two approaches have different semantics. PDDL with durative actions and TILs establishes a series of "happenings" for each proposition. Our approach uses flexible intervals with earliest and latest start and end time, allowing different temporal interpretations of a solution. Note that in some cases we omit the PDDL solution as there is no direct way to model the corresponding use case. This clearly does not mean that the use case is impossible to model using PDDL. Since CDBs are finite sets (Definition 4.1) and, under the presented assumptions (see Section 4.5.1), have a finite expansion for each problem, it is in principle possible to map CDBs into a propositional state space (much like a planning problem with limited number of actions can be mapped to a SAT problem as discussed in Section 3.1.2). Such solutions, however, are of dubious practical value, and lie outside the scope of this thesis.

Context-awareness could benefit from the notion of processes and events presented in PDDL+ [90]. PDDL+ is based on PDDL2.2 and it is not clear how constraints and preferences introduced in PDDL3 would interact with PDDL+ processes and events. For this reason we do not consider PDDL+ based solutions in this thesis.

5.3 Social Acceptability

A human-aware planner needs to make plans socially acceptable for humans in the environment (FR_{SA}). Plans may require further constraints or can be rejected altogether depending on how they interact with human activities. In this section we deal only with acceptability and do not consider proactive aspects such as supporting observed human activities or cooperative plans that involve human activities. In the following we refer to *avoidable situations* as situations that can be avoided by adding further constraints to a CDB. *Acceptable* situations are not desirable but can be tolerated if they cannot be planned around.

Table 5.1 lists four different classes of constraints for social acceptability. Social acceptability *ICs* may lead to rejection of the plan if no resolver works

Avoidable?	Acceptable? No	Acceptable? Yes
No	Class 1 (*:ic* (*:condition* <constraints>))	Class 2 (*:ic* (*:condition* <constraints>) (*:resolver* <cost>))
Yes	Class 3 (*:ic* (*:condition* <constraints>) (*:resolver* <constraints>)$^+$)	Class 4 (*:ic* (*:condition* <constraints>) (*:resolver* <constraints>)$^+$ (*:resolver* <cost>))

Table 5.1: Different classes of constraints for social acceptability and *IC* pattern for different those classes (+ stands for 1 or more instances). Avoidable cases have resolvers that assure human-awareness. Acceptable cases offer at least one resolver that increases social cost.

(classes 1 and 3) or allow a resolver that increases the social cost (classes 2 and 4)[2]. They can also describe situations that can be resolved (class 3 and 4) or that have no possible resolver (class 1 and 2) apart from increasing the social cost.

Certain situations that can arise when applying actions or from observations of human activities must be prohibited in any human-aware plan. Once they materialize there is no way to fix them (class 1) and the planner needs to backtrack. A situation may not be avoidable but it can be accepted at a social cost (class 2). Often a set of constraints can be added whose fulfillment avoids an unacceptable situation (class 3). A situation might be avoidable but also acceptable if none of the possible fixes works (class 4). Classes 2 and 4 constitute soft constraints. Note that Algorithm 1 will not backtrack if a social cost allows it to accept a situation. Resolvers for *ICs* are simply tried in order and if one of them allows to accept the situation it will move on to the next flaw. Note that this can still cause backtracking if there is an upper limit on the social cost that is reached by accepting too many violations. All of these four cases can make use of Prolog constraints to assure that they only hold for a subset of humans depending on their individual features and preferences.

Example 5.4. *A robot should never enter the office (class 1). A robot should avoid entering the kitchen (class 2). A robot should never clean the bathroom while it is occupied by a human (class 3). A robot should avoid vacuuming in a room where a human is watching TV (class 4).*

[2]Note that increasing the social cost may also lead to a failure if there is a maximum allowed social cost.

To ensure that a solution to a planning problem is socially acceptable we use *ICs* in the following way. The condition of an *IC* describes a problematic situation that requires to be fixed or must not arise. The pattern for all four classes of social acceptability constraints are given in Table 5.1.

The computational effort required for resolving *ICs* using these patterns depends on the number of resolvers. If the situation cannot be fixed (no resolver), it leads to an immediate dead-end and therefore backtracking. Testing the condition is the only effort required by such constraints. In case of a single resolver, search continues after applying it but it has no influence on backtracking since there is no alternative path in the search. The worst case complexity grows rapidly, however, if we use more than one resolver in class 3. The type of *ICs* considered here is convenient to separate robot and human actions in time, which is essentially a scheduling problem forcing us to try all alternatives in the worst case. Recall that Algorithm 1 does not consider optimality. For *ICs* this means it will not try to change previous resolvers (e.g., chosen operators or resolvers of other *ICs*) to achieve a lower social cost unless there is an upper bound on said social cost.

Use Cases for Social Acceptability

Use Case SA-1: "No robots are allowed in the office."

Interaction Constraint:

```
(:ic (no-robots-in-office ?R ?I)
  (:condition
    (:statement
      (?I (at ?R) office) ) ) )
```

PDDL3:

```
(forall ?R - robot
  (always (not
    (at ?R office)) ) )
```

The first use case for social acceptability is an example of a situation that is not allowed and cannot be planned around once it becomes part of the context. This is easy to address in our approach with a simple *IC* that identifies the forbidden situation (i.e., the robot being in the office) and has no resolver. This use case can be easily expressed with a PDDL3 constraint as shown above.

Use Case SA-2: "Don't vacuum while the child is resting."

Interaction Constraint:

```
(:ic (resting ?H ?R ?L ?I1 ?I2)
  (:condition
    (:statement
      (?I1 (activity ?H ?L resting))
      (?I2 (action ?R ?L vacuum)) )
    (:temporal
      (intersection {?I1 ?I2}) )
    (:prolog kb
      (property ?H child) ) )
  (:resolver
    (:temporal
      (after ?I1 ?I2 [1 inf]) ) )
  (:resolver
    (:temporal
      (before ?I1 ?I2 [1 inf]) ) ) )
```

PDDL3:

```
(forall ?H - child
  (forall ?R - robot
    (forall ?L - location
      (always (not (and
        (activity ?H ?L
            resting)
        (action ?R ?L vacuum)
      ) ) ) ) ) )
```

Both formulations use a condition that depends on a human activity "resting" and a robot action "vacuum". The condition is satisfied if the human is a child and the human activity and robot action co-occur. In the *IC* formulation the condition is expressed with a temporal constraint (*intersection*) on two statements. In the PDDL3 formulation the condition is modeled with an *always* constraint. This constraint assures that the human activity *resting* and the robot action *vacuum* never occur in the same state. This condition would have to be satisfied wrt. a set of timed initial literals that state when a child is resting. As noted earlier, we could not solve this problem with any available and functional PDDL3 planner. The *IC* also does something that the PDDL constraint cannot do: It suggests resolvers. Here we use two resolvers that model two alternative ways of separating the resting activity from the vacuuming action in time. These are specified with temporal constraints *after* and *before*. PDDL3 constraints/preferences offer a way to limit sequences of states during search but they do not provide a means to specify how plans should be modified under certain conditions. Note that we can also easily adapt the amount of time by which resting and vacuuming should be separated by modifying the interval [1 inf] to, e.g., [10 inf].

The following solution is an alternative using PDDL2.2:

```
(:durative–action vacuum
  :parameters (?R - robot) (?L - location)
  :duration (= ?duration 10)
  :condition (and
               (at start (at ?R ?L))
               (at start (not (clean ?L)))
         (forall (?H - child)
           (over all (not (activity ?H ?L resting)))
         )
       )
  :effect (at end (clean ?L))
)
(:init
  (at 20 (activity child1 bedroom resting))
  (at 40 (not (activity child1 bedroom resting))) )
```

In this solution we have compiled the human-awareness into the action definition. However this obfuscates the distinction between action theory and user oriented requirements. With *ICs* we can clearly separate the action theory from user requirements. For example adding or modifying existing user preferences can be done without modifying the action definitions, thus allowing to use the same action definitions for different environments, humans and human preferences.

> **Use Case SA-3:** "Don't vacuum while the child is resting or pay social cost of 10."

Interaction Constraint:

```
(:ic (resting ?H ?R ?L ?I1 ?I2)
  (:condition
    (:statement
      (?I1 (activity ?H ?L resting))
      (?I2 (action ?R ?L vacuum)) )
    (:temporal
      (intersection {?I1 ?I2}) )
    (:prolog kb
      (property ?H child) ) )
  (:resolver
    (:temporal
      (after ?I1 ?I2 [1 inf]) ) )
  (:resolver
    (:temporal
      (before ?I1 ?I2 [1 inf]) ) )
  (:resolver
    (:cost (add social 10) ) ) )
```

PDDL3:

```
(forall ?H - child
  (forall ?R - robot
    (forall ?L - location
      (preference Sleeping
        (always (not (and
          (activity ?H ?L
              resting)
          (action ?R ?L
              vacuuming)
) ) ) ) ) )

(:metric
  minimize (* 10
    (is-violated Sleeping)))
```

Above we have two possible solutions that extend the previous use case. The *IC* version is a simple extension of the solution to use case SA-2. Here the

preference is provided by a third resolver which increases the social cost by 10 in case the first two resolvers lead to inconsistencies. Note that by using Prolog constraints we could easily adapt the social cost depending on the person that is disturbed. The PDDL solution is identical to the solution to use case SA-2 apart from the fact that it is a preference. Violations can be added up and weighted via metrics [100].

> **Use Case SA-4:** "Don't vacuum while the child is resting or pay a social cost of 5 or 10 for a small or large overlap respectively."

```
(:ic (resting ?H ?R ?L ?I1 ?I2)
  (:condition
    (:statement
      (?I1 (activity ?H ?L resting))
      (?I2 (action ?R ?L vacuum)) )
    (:temporal (intersection {?I1 ?I2}) ) )
  (:resolver
    (:temporal (before ?I1 ?I2 [1 inf]) ) )
  (:resolver
    (:temporal (after ?I1 ?I2 [1 inf]) ) )
  (:resolver
    (:temporal (overlaps ?I1 ?I2 [1 10]))
    (:cost (add social 5)) )
  (:resolver
    (:temporal (overlapped-by ?I1 ?I2 [1 10]))
    (:cost (add social 5)) )
  (:resolver
    (:cost (add social 10)) ) )
```

In case of *ICs* we simply extend use case SA-3 by adding two resolvers that allow the overlap at a social cost of 5. If both of these resolvers fail the last resolver adds a social cost of 10. While we limit the overlap to a maximum of 10 time units there is no optimization (i.e., minimization of the overlap) in the simple temporal problem. This means that once the overlap is allowed its size may be any value in the allowed range that does not create a temporal conflict. Increasing the number of resolvers in this way leads to a large search space for *ICs*. Backtracking can be avoided at this point if the last resolver simply accepts the solution at a cost. Here we run into a major problem for PDDL3: There is no obvious way in which the quantified overlaps can be expressed via durative actions or PDDL3 constraints.

5.4 Proactivity

Proactivity (FR_{PA}) goes beyond making plans socially acceptable and requires the system to actively support humans in the environment when opportunities arise. These opportunities usually arise when specific (combinations of) human

activities are observed during execution (e.g., activities recognized from sensors or activities inferred via context-awareness). There can also be opportunities when specific combinations of robot actions and predicted human activities arise during the planning process. The latter case can be seen as a form of conditional goals that are only relevant when the planner goes through a specific state. Opportunities are modeled as conditions of *ICs*. The proactive reaction of the system is modeled via resolvers. Proactive *ICs* interact with the environment or user in some way to provide active support when their condition is satisfied. We distinguish three cases for proactive *ICs*: The first case does not directly act in the environment, but makes decisions for variables that will directly influence human activities. This could mean that locations for meetings are decided, or devices in the environment are configured to support human activities.

Example 5.5. *The HaPlEx recognizes that a meeting is scheduled but no room has been assigned (i.e., the room is a variable). It proactively chooses a room for the meeting to take place in.*

In the second case proactivity is achieved by statements that can be directly executed (recall Section 4.7.2 about reactors) without causal dependencies. Consider, e.g., the statement $(\mathcal{I}\ (\mathit{light\ livingroom})\ \mathit{on})$ to turn on a light. The third case allows more complex proactive behavior by adding open goals as part of a resolver. In scenarios with robotic assistance the robot may be required to be in a specific location with the right equipment for a task. In cases like this open goals provide a convenient solution.

Example 5.6. *A robotic assistant is scheduled to serve coffee during the meeting that was scheduled in the above examples.*

For all three cases we may have to handle situations in which the proactive resolver leads to a flaw. Table 5.2 lists six classes of proactivity, covering the three cases with and without acceptable failures. Failure can be handled as for social acceptability by adding a social costs. This is useful when reaching the additional goal would be good, but ultimately does not threaten the success of the plan.

Example 5.7. *The HaPlEx should not fail if no robotic assistant is available to serve coffee during a meeting.*

The computational effort required by classes 1A and 1B depends on the constraints that need to be satisfied by the imposed choices. For classes 2A and 2B we trigger simple actions during execution without a need for replanning. This is realized by adding statements that have an effect on the environment as part of the *IC* resolver. We consider the action as simple since it does not require an operator definition and thus has no preconditions beyond what was already established by the condition of the *IC*. Simple actions could fail if they overlap

Resolver	Failure acceptable? No	Failure acceptable? Yes
Selection	Class 1A (*:ic* (*:condition* <opportunity>) (*:resolver* <select-val>))	Class 1B (*:ic* (*:condition* <opportunity>) (*:resolver* <select-val>) (*:resolver* <cost>))
Action	Class 2A (*:ic* (*:condition* <opportunity>) (*:resolver* <action>))	Class 2B (*:ic* (*:condition* <opportunity>) (*:resolver* <action>) (*:resolver* <cost>))
Open goal	Class 3A (*:ic* (*:condition* <opportunity>) (*:resolver* <open-goal>))	Class 3B (*:ic* (*:condition* <opportunity>) (*:resolver* <open-goal>) (*:resolver* <cost>))

Table 5.2: *IC* pattern for different classes of proactivity.

with a second simple action that uses a different value (e.g., a light switch is required to be in two states at the same time). If such situations are impossible the effort required by simple actions is very low since they only use one resolver and do not lead to additional flaws. Classes 3A and 3B pose open goals. Reaching these goals potentially requires a sequence of actions to assure that all causal dependencies are met. As pointed out before our current approach will re-plan from the current state of execution if new goals are added to the problem. As long as only non-goal flaws appear they will simply be resolved by Algorithm 1 and execution continues. Plan repair for open goals is difficult as it may require to remove constraints. This may invalidate the assumptions under which other flaws were resolved. A major advantage of the presented approach is that it can freely choose between different planning methods: Even if we create the original plan with a heuristic forward search we can attempt to satisfy new goals with a plan-space planner.

Use Cases for Proactivity

Use Case PA-1: "If the human gets out of bed at night, turn on the light."

Interaction Constraint:

```
(:ic (getting-up-light ?H ?I1 ?I2)
  (:condition
    (:statement
      (?I1 (activity
        ?H ?L getting-up))
      (?I2 phase night-time) )
    (:temporal
      (during ?I1 ?I2
        [0 inf] [0 inf])))
  (:resolver
    (:statement
      (?I3 (light) on) )
    (:temporal
      (equals ?I1 ?I3) ) ) )
```

PDDL2.2:

```
(:derived (light on)
  (and
    (activity ?H ?L
        getting-up)
    (phase night-time) ) )
```

For the *IC* solution we use two statements and require the activity to be during the night. For real world execution we could simply use a ROS publisher constraint (recall Section 4.7.1) to publish the value of the state-variable *light* to a node that controls the light.

Since this case requires a simple reaction with no need for actual planning, we could solve it with an axiom in PDDL2.2 assuming that the derived predicate will be executed. Note that this assumes also that the PDDL planner is used online to continuously solve an updated planning problem. The advantage of the *IC* solution is that it can be used offline to plan for turning on the light in case it is predicted that the human will get up. Note that the time for turning on the light is temporally constrained to be equal to the getting-up activity. Therefore if this activity is shifted in time during execution, so will the planned reaction of turning on the light.

Use Case PA-2: "If the human gets out of bed at night, turn on the light. Turn the light off 2 minutes after the human is back in bed."

```
(:ic (getting-up-light ?H ?I1 ?I2)
  (:condition
    (:statement
      (?I1 (activity ?H ?L getting-up))
      (?I2 phase night-time) )
    (:temporal
      (during ?I1 ?I2 [0 inf] [0 inf]) ) )
  (:resolver
    (:statement
      (?I3 (light) on)
      (?I4 (light) off) )
    (:temporal
      (equals ?I1 ?I3)
      (before ?I3 ?I4 [120 120]) ) ) )
```

The *IC* solution is the same as for the previous use case with the addition of one statement and one temporal constraint in the resolver. When exactly the light is turned off is conditioned on the end of the *getting up* activity (due to the temporal constraints). The temporal constraint *before* in the resolver assures that the light is turned off two minutes (120 seconds) after the getting up activity has ended.

Regarding PDDL, we can no longer use derived predicates because they would require knowledge of the previous state (that the light was on because a person got up during the night) to come to the correct conclusion. The delay between the end of the activity and turning off the light cannot be directly expressed either. We are also not aware of a direct way to express this fact in PDDL3 constraints.

Use Case PA-3: "If the human gets out of bed at night, turn on the light. Turn the light off 2 minutes after the human is back in bed. If the person is the grandfather, also provide a robotic assistant."

```
(:ic (getting-up-robot-assist ?L ?I1 ?I2)
  (:condition
    (:statement
      (?I1 (activity grandfather ?L getting-up))
      (?I2 phase night-time) )
    (:temporal
      (during ?I1 ?I2 [0 inf] [0 inf]) ) )
  (:resolver
    (:goal
      (?I3 (assist-to-bathroom-and-back grandfather ?L)) )
    (:temporal
      (finished-by ?I1 ?I3 [1 5]) ) ) )
```

In this case we need to add a goal that has to be achieved if the condition is fulfilled. As before the condition depends on the *getting-up* activity at night time. For the first part of the use case we use the solution to use case PA-2 (see above) since it applies to this one as well. We then add the above *IC*. In this case the resolver does not add a simple statement, but a goal the achievement of which brings about the robotic assistance. Note that this may require non-trivial planning depending on the operators specified in the action theory (which is independent of *ICs*).

To make this proactive behavior optional (i.e., only assist when possible) we can simply add a second resolver with a social cost. If social costs are irrelevant we can also add an empty resolver. Depending on the temporal constraints on the newly added goals we may not always be able to find a plan. In the above example we would require a robot to be close by to react in time. The goal cannot be reached if all robots are in another part of the building and moving them would take too long. This fact is modeled by the temporal constraint *finished-by* which gives the robot up to five minutes to start assisting. In addition, in an online setting, context awareness may further alleviate the problem of short notice since it could be used to predict *getting-up* before it happens (e.g., by measuring movement in the bed). This would more easily allow the fulfillment of the *finished-by* constraint.

We are not aware of any obvious way to achieve the above with PDDL derived predicates, TILs and/or constraints. As pointed out before, derived predicates cannot be goals, TILs cannot be modified, and PDDL constraints cannot be used to add resolvers.

5.5 Context Awareness

The *HaPlEx* can only react to situations it is aware of to begin with (FR_{CA}). By combining human activities (and created plans) the system can benefit from inferring context that is implied. If several humans are carrying out the same activity in the same location, it may be a group activity. If one of them is cooking we can expect all of them to be eating in the near future (see use case CA-4). Inferring context can help the system to find opportunities or simply to decrease the likelihood of violating constraints for social acceptability. Here it is important to point out the difference with activity recognition, where sensor signals are processed to derive human activities such as “sitting” or “walking”. In Figure 1.1 (the stage) we would position sensors and activity recognition based on sensor data as part of “Hard- & Software Components”, while context-awareness can use the information provided by activity recognition to infer abstract activities (context) based on observed activities.

Example 5.8. *If the whole family is watching a movie that lasts 90 minutes, the bedrooms can be cleaned without disturbing anyone.*

```
(:ic (:condition <context>)
     (:resolver <context-extension>))
```

Table 5.3: *IC* pattern for context-awareness.

Context awareness can be applied during execution when new activities are observed or during planning when a solution is applied to the initial context. The inferred context can be used by other *ICs* to identify situations that need attention.

ICs for context inference are often characterized by using statements as part of their resolver, where these statements model the inferred context. Temporal constraints are used to relate the inferred context with existing statements in the CDB. If there is uncertainty about the inferred context we can use constraint types that support uncertainty in the resolver. We will see an example of this in use case CA-2 below, where we use an uncontrollable variable to express uncertainty about who will execute an inferred activity. The *IC* pattern for context-awareness is shown in Table 5.3.

Every constraint for this purpose has exactly one resolver and (unlike in the previous section) it is not the purpose of the resolver to avoid the situation described in the condition but to extend it by providing additional information. The computational effort required for this type of context inference is low since it only uses one resolver that is supposed to provide information and should not cause inconsistencies. Obviously, this is only true for the effort imposed by resolvers and testing if an *ICs* applies may be very difficult depending on the constraint types used in the condition. Note that while this method could be used to perform activity recognition to some degree, it is not well suited to deal with noisy sensor data.

While we consider only *ICs* that use statements to describe derived context other applications are possible. It would be possible, for instance, to extend a Prolog background knowledge base as a result of observed context. This would allow *ICs* to deal with new humans in the environment by adding knowledge about their preferences.

Use Cases for Context Awareness

> **Use Case CA-1:** "If the stove is on while a person is in the kitchen, that person is cooking."

Interaction Constraint:

```
(:ic (is-cooking ?H ?I1 ?I2)
  (:condition
    (:statement
      (?I1 stove on)
      (?I2 (at ?H) kitchen) )
    (:temporal
      (intersection {?I1 ?I2}) ) )
  (:resolver
    (:statement
      (?I3 (activity ?H kitchen cooking))
    )
    (:temporal
      (during ?I3 ?I1 [1 inf] [1 inf])
      (during ?I3 ?I2 [1 inf] [1 inf]))))
```

PDDL2.2:

```
(:derived
  (activity ?H kitchen cooking)
  (and
    (at ?H kitchen)
    (stove on) ) )
```

The *IC* solution uses two statements and their temporal intersection to capture the fact that the inference of cooking requires a human to be in the kitchen while the stove is on.

The PDDL solution uses a derived predicate to add the activity *cooking* to the state whenever a human is in the kitchen while the stove is on. This solution lacks the temporal dimension which makes it impossible to plan based on the actual duration of cooking.

> **Use Case CA-2:** "If the stove is on while a sensor indicates human presence in the kitchen, an unknown family member is cooking."

```
(:ic (is-cooking ?I1 ?I2)
  (:condition
    (:statement
      (?I1 stove on)
      (?I2 sensor-state-kitchen-pir on) )
    (:temporal
      (intersection {?I1 ?I2}) ) )
  (:resolver
    (:statement (?I3 (activity ?H kitchen cooking)) )
    (:domain (uncontrollable {?H}))
    (:temporal
      (during ?I3 ?I1 [1 inf] [1 inf])
      (during ?I3 ?I2 [1 inf] [1 inf]) ) ) )
```

Compared to the previous *IC* there are only minor differences. Instead of requiring knowledge about who is in the kitchen, this *IC* simply uses a statement

that relies on an infrared sensor, which only indicates movement and cannot distinguish people. Instead of grounding the human ?*H* it is set as uncontrollable. This will force Algorithm 3 to enforce any *IC* whose condition includes

```
(:statement (?I (activity ?H kitchen cooking)) )
```

for any possible value of ?*H*. For example if there is an *IC* stating that the robot should not be in the kitchen while the grandfather is cooking, the robot will be kept out of the kitchen by the planner on the grounds that it is not known who is cooking (and it could be the grandfather). Considering the uncertainty aspect, we are not aware of a direct PDDL solution for this use case.

> **Use Case CA-3:** "If someone puts on his/her shoes and then puts on his/her jacket he/she will leave the house next."

```
(:ic (leaving-the-house ?H ?I1 ?I2)
  (:condition
    (:statement
      (?I1 (activity ?H entrance putting-on-shoes))
      (?I2 (activity ?H entrance putting-on-jacket)) )
    (:temporal (after ?I2 ?I1 [1 5]) ) )
  (:resolver
    (:statement (?I3 (activity ?H entrance leaving)) )
    (:temporal (after ?I2 ?I3 [1 5]) ) ) )
```

Here we use a sequence of two statements in the condition and use a temporal constraint to assure that they follow each other. In the resolver we add a third activity: the conclusion that the human will be leaving the house. Other *ICs* may trigger on this derived activity to, for instance, issue a warning if the stove was left on. Note that we would have to add a second *IC* to extend the use case with the possibility that the human puts on the jacket first. To capture this in a single *IC* would require disjunctions over temporal constraints which we do not consider in this thesis. To model this use case in PDDL we would be required to use information about past and current states to derive a predicate in a future state. This is, to the best of our knowledge, not possible.

Use Case CA-4: "If there is someone cooking in the evening there will be a 40 minute dinner activity 10 minutes after cooking finishes."

```
(:ic (dinner-after-cooking ?H ?I1 ?I2)
  (:condition
    (:statement
      (?I1 (activity ?H kitchen cooking))
      (?I2 phase evening) )
    (:temporal (during ?I1 ?I2 [1 inf] [1 inf]) ) )
  (:resolver
    (:statement (?I3 (activity family kitchen eating)) )
    (:temporal
      (before ?I1 ?I3 [10 10])
      (duration ?I3 [40 40]) ) ) )
```

A similar structure as in the previous use cases was used to model a similarly defined context: Again we use two statements and a temporal constraint to define the context we are interested in. The resolver contains the same elements as we have seen for other solutions and only differs in the choice of temporal constraints. This shows how use cases with a similar structure are captured by similar models. Again, a PDDL based solution becomes difficult due to the quantified temporal constraint *before*.

Note that the family activity in this use case does not necessarily exclude that activities of individual family members occur. As a consequence, *ICs* involving individual family members still apply, e.g., no vacuuming will occur during dinner if one of them is in bed with a fever.

5.6 Planning for Cooperation

Finally, we propose a solution to plan for cooperation with humans (FR_{PC}). If the goals of the *HaPlEx* and the human fully or partially coincide, or if one depends on the capabilities of the other (e.g., a robot not capable of opening a door), the *HaPlEx* and its users may have to cooperate in order to reach their goals. This in turn means that plans created by the *HaPlEx* have to include cooperation between both parties. In case of common goals a good solution would divide the work between human and robot in a way that accounts for their individual skills.

We divide all operators of the *HaPlEx* into three classes. The first one is the class of fully automated operators. These are executed by the *HaPlEx* without directly depending on human participation. The second one is the class of actions that involve both humans and the *HaPlEx*. A typical example would be handing over an object from a robot to a human or vice versa. The third one is the class of actions in which only humans are involved. For the latter two classes we assume that the humans will cooperate.

There are some general aspects that become interesting when considering operators that involve humans: The *HaPlEx* needs to communicate requests for actions to the user. Any plan that depends on actions involving humans should be robust wrt. their response time. It is not possible to impose a start time on intervals corresponding to statements that represent effects of human actions. After the action has been requested the *HaPlEx* must wait and see if the desired effect occurs. This is realized with reactors that observe when the action is started and completed and possibly report failure. We will see examples of this in the discussion of the use cases below. If it takes too long it may issue a reminder or simply replan. All of these aspects can be addressed directly with our approach with the online planning and execution techniques presented in Section 4.7.

The *HaPlEx* should be flexible when confronted with delays or human activities that do not follow the plan (TR_{Uncert}). Temporal uncertainty about human activities is often addressed by allowing uncontrollable intervals in temporal reasoning [228]. Here we model this by linking effects that depend on humans to ROS subscriptions, which means that the *HaPlEx* will wait until they can be observed (see Section 4.7.2 about reactors). Consequently, when such an effect is added as part of a cooperative action, the *HaPlEx* will wait for it (unless a contingency triggers). In this way delays can be handled with temporal flexibility that is offered by the *HaPlEx*'s use of flexible temporal intervals. Contingencies can be handled with *ICs*. In the real-world domain presented in Chapter 6, *ICs* are used to issue reminders when the waiting time exceeds a certain limit and when unexpected activities are executed. The following use cases were used in our robotics test presented in Chapter 6.

Use Cases for Planning for Cooperation

Use Case PC-1: "Ask a human to go somewhere."

```
(:operator
  (move-human ?H - human ?L1 - location ?L2 - location)
  (:preconditions (?P (at ?H) ?L1))
  (:effects
    (?E (at ?H) ?L2)
    (?S1 (say (human-moving ?H ?L2)))
    (?S2 (say thanks))
    (?RI (busy ?H) 1)
    (?HM (human-is-moving ?H ?L2)) )
  (:constraints
    (:prolog kb
      (location ?L1)
      (location ?L2)
      (notEqual ?L1 ?L2) )
    (:temporal
      (overlaps ?P ?THIS [1 inf])
      (overlaps ?THIS ?E [1 inf])
      (during ?S1 ?THIS [1 inf] [1 inf])
      (during ?S2 ?E [1 inf] [1 inf])
      (equals ?RI ?THIS)
      (starts ?HM ?THIS [1 inf])
      (meets ?HM ?E) ) ) )
```

This use case requires an action that is executed by a human. During execution the *HaPlEx* cannot expect this request to be fulfilled immediately. Any plan involving such an action must be flexible wrt. its execution time. We model this use case with the *move-human* operator above. The effect of this operator

```
(?E (at ?H) ?L2)
```

is linked to a ROS subscription

```
(:ros (subscribe-to (at ?H) ?L human_location (String 1 ?L) ) )
```

since they use the same state-variable (*at* human). The ROS topic *human_location* provides observed human locations. This indicates to the *HaPlEx* that it has to wait for this effect to manifest itself, and an indication of this manifestation is given by the content of the ROS topic *human_location*. Note that the ROS topic in the above example is hard coded. A better solution would use a Prolog constraint to look up the correct ROS topic for each human. When the desired location *?L2* is published, the corresponding statement is matched to the effect

```
(?E (at ?H) ?L2)
```

and the latter inherits the temporal bounds of this statement. Since the end time of the operator is constrained by the start time of the effect with the temporal constraint

```
(overlaps ?THIS ?E [1 inf])
```

the duration of the operator is dynamic wrt. the time it takes for the human to fulfill the request of the *HaPlEx*. The statement

```
(?HM (human-is-moving ?H ?L2))
```

represents the time from the start of the operator until the human arrives at the target location. This is used as input to the contingency *IC* that we use for use case PC-3.

Notice that several aspects are linked to execution: The statements

```
(?S1 (say (human-moving ?H ?L2)))
(?S2 (say thanks))
```

are associated with a text-to-speech reactor that executes them at the appropriate times. Phrases are included from a text file in the same way as the Prolog knowledge base in Example 4.11. The variables in the first statement are substituted into the phrase: "?*H* can you come to the ?*L2* please.". The effect statement

```
(?E (at ?H) ?L2)
```

of the *move-human* operator is associated to a ROS subscription and consequently has to be observed. This captures the fact that the *HaPlEx* can ask the human to go somewhere but not control the human directly. A PDDL solution of this use case could go as far as assuming that some operators are executed by humans, but lacks temporal flexibility wrt. human response times. A similar argument holds for the next two use cases.

If we consider operators as the one above the plan has to be communicated to everyone involved and the *HaPlEx* has to be able to deal with unpredictability in human behavior. One solution towards this end is to use *ICs* to monitor the execution of cooperative actions or action sequences that trigger an appropriate response in case of contingencies. One example of such an *IC* is provided by the solution to use case PC-3.

Use Case PC-2: "Ask a human to give an object to a robot."

```
(:operator
  (human-put ?H - human ?O - object ?R - robot ?L - location)
  (:preconditions
    (?P1 (object-location ?O) ?L)
    (?P2 (at ?R) ?L)
    (?P3 (at ?H) ?L) )
  (:effects
    (?E1 (object-location ?O) ?R)
    (?E2 (sensor-pressure-switch ?R) true)
    (?S1 (say (ask-for-object ?H ?O)))
    (?S2 (say thanks))
    (?RI (busy ?R) 1)
    (?RH (busy ?H) 1) )
  (:constraints
    (:temporal
      (overlaps ?P1 ?THIS [1 inf])
      (during ?S1 ?THIS [1 inf] [1 inf])
      (during ?S2 ?E2 [1 inf] [1 inf])
      (during ?THIS ?P2 [1 inf] [1 inf])
      (during ?THIS ?P3 [1 inf] [1 inf])
      (overlaps ?THIS ?E2 [1 inf])
      (overlaps ?E2 ?E1 [1 inf])
      (equals ?RI ?THIS)
      (equals ?RH ?THIS) ) ) )
```

This constitutes a cooperative action that involves both robot and human. Similarly to the previous use case, successful execution of the operator depends on a ROS topic, specifically a pressure sensor mounted on top of the robot, the activation of which indicates that an object has been placed on top of it. Differently from the previous use case, a controlled component (namely the robot) also partakes in the success of execution. As before the solution is flexible wrt. the time it takes for a human to fulfill the request.

Use Case PC-3: "If an expected human action takes too long issue a reminder."

```
(:ic
  (reminder-move ?THIS ?H ?L1 ?L2)
  (:condition
    (:statement (?I (human-is-moving ?H ?L2)))
    (:temporal (greater-than (duration ?I) 10) )
  )
  (:resolver
    (:statement (?S (say (human-moving-reminder ?H ?L2))) )
    (:temporal (during ?S ?I [1 inf] [1 inf]) ) ) )
```

This use case covers situations in which human actions take longer than expected. The *IC* above models a simple contingency for cases in which human movement takes longer than 10 time units. It uses the statement

```
(?I (human-is-moving ?H ?L2))
```

from the operator *move-human* and uses a temporal query

```
(greater-than (duration ?I) 10)
```

that is true if interval ?I lasts at least 10 time units. In this case the single resolver issues a proactive reminder that will use text-to-speech as explained for use case PC-1. To trigger replanning in case the human-action takes too long an empty resolver can be used: As soon as the time limit is reached the condition is satisfied and the *IC* cannot be resolved leading the *HaPlEx* search for another solution.

5.7 Discussion

We have provided *IC* solution patterns for social acceptability (FR_{SA}), proactivity (FR_{PA}) and context-awareness (FR_{CA}) and discussed possible solutions for planning for cooperation (FR_{PC}). We demonstrated the capability of our approach to solve these four functional requirements by discussing and solving a series of use cases. The use cases represent, in our opinion, a basis for addressing the functional requirements of human aware planning identified in Chapter 2. They could be extended in the future to include, for instance, a wider coverage of uncertainty. Where possible, we compared our solutions to the use cases to candidate solutions in PDDL.

All *IC*-based solutions are intuitive and allow to directly model the desired use case. These solutions benefit from the fact that temporal intervals have an explicit representation and from the variety of qualitative and quantified constraints that our approach supports. Whenever an *IC* only applies to a specific person/case we use prolog constraints to express this. As pointed out earlier, this allows us to preprocess constraints in a similar way to compiling away preconditions on static propositions in classical planning. In one case we used uncontrollable variables to express uncertainty in the derived context. In some cases we could see that adding complexity to an existing use case can be done by simply adding further resolvers or modifying the existing ones.

The approach presented in this thesis comes with its own domain definition language that allows to easily express conditions, axioms, preferences and constraints (FR_{KR}) that are hard or impossible to express in any of the above PDDL versions/variants as we saw when discussing the use cases. Domain axioms are similar to *ICs*. However the former do not support time, which is instrumental in many of our use cases. Moreover, *ICs* provide a lot of modeling freedom since they support every constraint type in their conditions and resolvers. As shown, we can use *ICs* to easily add new goals to the problem under certain conditions. Axioms cannot be used to add goals since actions are not allowed

to affect derived predicates [79]. In addition, domain axioms cannot directly be used to "forbid" situations as *ICs* do with resolvers that make the condition inconsistent. Domain axioms are rules that state facts that can be derived from the current state. *ICs* can be used in the same way but their resolvers can also make their own conditions inconsistent. This is a basic requirement for social acceptability since we need to model unwanted situations and provide ways to resolve them.

Another advantage of our approach is that it supports quantified temporal constraints, which are hard to express in PDDL. Also we support explicit intervals that allow operators and *ICs* to define more detailed temporal relations compared to PDDL2.1 durative actions, which only allow to relate preconditions and effects to the operator itself.

PDDL3 can express only a limited number of cases directly and would require overly verbose models to use quantified constraints. Specific features of other PDDL variants may mitigate the difficulty of modeling our use cases. However, no single feature of PDDL (or combination thereof) covers all the use cases of human-aware planning. In the following section we perform an evaluation of the algorithmic aspects of our solution. We employ problems that require the expressiveness of our modeling language to capture human-aware features of the domain.

The use cases we presented are conceived in the context of an online setting where plans are created and newly appearing flaws are resolved during execution. As shown in Section 4.7 and Section 5.6, our approach allows to include execution aspects, such as connections to ROS topics, in the domain model.

Chapter 6
Evaluation

In this chapter we describe a series of results obtained using the approach to human-aware planning outlined in the previous two chapters. We present four domains, each serving a different purpose:

The first one (*household*, Section 6.2) investigates how efficiently our approach handles social acceptability (FR_{SA}) in a household in which robots have to perform a set of tasks, such as cleaning or vacuuming. Valid solutions have to respect a set of *ICs* for social acceptability that will enforce scheduling decisions if necessary. Here, we expect some prior knowledge about human activities and allow partially specified human activities (i.e., activity statements with uncontrollable variables). Execution was not considered for these tests.

The second domain (*research-facility*, Section 6.3) investigates proactivity and context-awareness. It models a research facility environment in which the *HaPlEx* manages calendar entries of humans. Context awareness (FR_{CA}) is used to add meeting activities and proactive support (FR_{PA}) is employed to select rooms for these meetings and have a robot serve coffee during the meeting. The domain also manages lab tasks of different degrees of risk. For a medium risk task scheduled by a human, a robot will provide assistance if the task is scheduled alone. For high-risk tasks, a 30 minute break is scheduled proactively just before the task. We performed both online and offline evaluation for this domain. Results have appeared in previously published papers [144, 145].

The third domain (*assist-living*, Section 6.4) is an assisted living domain that was implemented and executed on a real robotic system in a smart home environment. It demonstrates how the *HaPlEx* operates the closed loop that includes human-aware planning, execution and simulated activities.

In the fourth domain (*ic-benchmark*, Section 6.5) we created a set of benchmarks using artificial *ICs* based on the social acceptability pattern presented in Section 3.2 to test their performance and find performance limiting factors.

The aim of all these experiments is to demonstrate the feasibility of the approach presented in this thesis in different settings. First we provide a short discussion on the implementation. Then, for each experiment, we introduce the

domain in more detail, describe the experimental setup and discuss the results. While we could express a subset of the *ICs* we use in this section in PDDL3 (as shown in Section 3.2) we did not manage to produce correct solutions for a minimal testing domain with any PDDL3 planner we are aware of[1]. We tried the planners Optic, Mips-XXL, Mips-BDD, and SGPlan. Problems that came up include bugs, and only partial PDDL3 support (e.g., Optic supports preferences but not constraints and does not support negative preconditions). We were able to produce two working examples for social acceptability with Optic. Both of these require to negate human-activities and would thus require to state the negation of all human activities at all times when they are not true. In one case we compile social acceptability into preconditions (as shown in use case SA-1). In the second case we use preferences to achieve a similar result. The likely explanation for the need to negate human activities in this case is that preferences are converted to operators and negated preconditions are not supported. The solution with preferences fails to work when changing timed initial literals. (I.e., it either fails to produce correct solutions or the search failed to find any solution and does not terminate even in trivial cases). While it might be possible to produce a working solution at least for a subset of the domains we consider in this section there is too much guesswork involved about how specific PDDL features are supposed to interact and which combinations of thereof are supported by a planner to make a reasonable comparison.

6.1 Implementation

All tests were run on a Java implementation of the *HaPlEx* which is publicly available at `http://spiderplan.org`. All domain definitions are available in the aforementioned repository. Prolog reasoning is carried out by the interpreter/compiler YAP 5.1.3 [60]. The platform used was an Intel® Core™ i7-2620M CPU with 4x2.70GHz and 4GB RAM with a 3GB memory limit.

Our implementation provides (a choice of) solvers for each type of constraint and allows to decide in which order flaws are detected and resolved. For each constraint type there exists a wide range of approaches and we prefer using available implementations or tools rather than re-implementing algorithms from scratch.

For forward planning, we apply some common heuristics, since using available tools as a black box does not offer the degree of control we require to manipulate the search and the problem. We implemented a couple of forward planning heuristics that have been shown to perform well: the *Fast Forward* [120], *Causal Graph* [113] and H^{add} [230] heuristics. They can be used individually or combined. Combinations either use alternating search queues [113] (one for each heuristic) or a lexicographical ordering. We also added a lookahead as described in [230] as an option. We use all of these approaches in

[1] Minimal PDDL3 and PDDL2.2 models are available at `spiderplan.org`

a solver that provides resolvers for all open goals at once. In this solver we use the idea of multiple alternating search queues [113] to combine different heuristics.

Solvers for each constraint types are exchangeable, which has several advantages. It provides an easy way to to switch between several alternative approaches for the same constraint type. This in turn makes it easy to investigate the impact of exchanging solvers. The resulting solvers can be evaluated not only in their run-times, but also in the number of flaws created and resolved and in the amount of backtracking and the gain of pruning.

The handling of *Uncontrollable*(x) is limited to *ICs*. There are approaches for temporal reasoning [229, 45] which can handle uncontrollable interval bounds, which would add another dimension of uncertainty. This could be used to lift the assumption of fixed intervals for human activities that we used in the running example.

All tests were run on a Java implementation of the *HaPlEx* which is publicly available at `spiderplan.org`. Prolog reasoning is carried out by the interpreter/compiler YAP 5.1.3 [60]. The platform used was an Intel® Core™ i7-2620M CPU @ 2.70GHz x 4, 4GB RAM platform with a 3GB memory limit.

6.2 Household Domain

This domain serves to evaluate our approach for social acceptability (FR_{SA}). It models a home with a set of human inhabitants that includes two parents, four children, two infants and four grandparents. The home consists of 20 rooms with specific purposes (such as bedrooms, bathrooms, laundry room, etc.). A set of robots have to solve daily goals without interfering with human activities. We leverage the capabilities of *ICs* to separate user preferences from action theory. Due to this operators are not required to encode a potentially large number of combinations of human preferences in their preconditions (as pointed out in Section 5.3). Even when all human activities are known in advance the resulting problem of reaching goals without interfering with full-day human schedules is challenging.

The temporal horizon of the problems in this domain is 1440 minutes (24 hours). This means that a plan has to take place between 0 and 1440 minutes (where 0 represents 6:00am). Human activities are pre-scheduled and linked to locations in the house, and there may be cases in which the human activity is unknown at planning time. The full set of activities we used here is *idle, out, eating, sleeping, reading, working, cooking, playing* and *usingBathroom*.

There are four robots that have to perform a set of tasks. The list of tasks we use is *vacuum, sortStuff, assist, collectTrash, takeOutTrash, collectLaundry, doLaundry, cleanRoom, entertain*. Robots can either move from one location to another adjacent location or perform a task at their current location. We use temporal constraints on goals for a given day to divide them into three batches, where every goal in each batch has to be achieved before every goal in

the next batch. One could also use fixed release times for each goal, to specify the latest time point when it has to be achieved. Arguably these goal batches make reaching goals somewhat easier but it shows how temporal constraints on goals can be used. The difficultly of resolving open goals (ignoring all other constraints, that is, the difficulty of the propositional planning sub-problem) is not our main concern in this benchmark.

Robot plans have to satisfy the following *ICs* (the * indicates class 4 social acceptability, all others are class 2 as found in Table 5.1): (1) No working where someone is sleeping*; (2) No working where a "light sleeper" is sleeping; (3) Don't interrupt easily distracted people*; (4) Robots not allowed when bathroom is occupied; (5) Robots avoid locations with people that dislike them* (see Example 4.22); (6) Collecting trash not allowed while people are eating*; (7) Robots not allowed in bedroom with two married people*; (8) No working in a child's rooms while the child is playing*; (9) Robots not allowed to vacuum while someone is reading*.

The purpose of the experiments is to show that the proposed planner is capable of creating plans that satisfy a set of *ICs* for problem instances of a reasonable size. As explained in the following section, we vary the number of goals of the *HaPlEx*, the number of humans whose schedules have to be considered, and the number of *ICs* that have to be satisfied. For all of these parameters we created one set of problems without uncertainty about human activities and one set of problems in which human activities may be partially specified (e.g., the location or exact type of activity may not be known during planning time). This allows us to show the impact of adding uncontrollable variables to human activities. In addition we will show the potential benefits of problem decomposition and constraint-based planning by comparing to a slight modification of Algorithm 1. This domain uses two operators and up to nine *ICs*. Appendix A contains the full domain, its Prolog knowledge base, and one of the problems used in the evaluation.

6.2.1 Setup

We randomly generated complete human activity schedules for three different sets of humans h1, h2 and h3 containing 5, 7 and 12 inhabitants, respectively. The number of goals varies between 10 and 30 (g10 – g30) and goals are randomly distributed into the three goal batches (keeping the number of goals in each batch approximately even). We used two configurations of uncertainty u0 (no uncertainty) and u1 (with uncertainty). For each activity created with u1 there is a 10% chance that this activity will be only partially specified (e.g., we may know the type of activity but not which human executes it) and we randomly pick between two and four random possible values. We created two domains d1 and d2 containing the first 6 and 9 *ICs* from the list presented above to evaluate how the number of *ICs* changes the problem solving time. For each configuration we created 20 random problems. We set a time limit

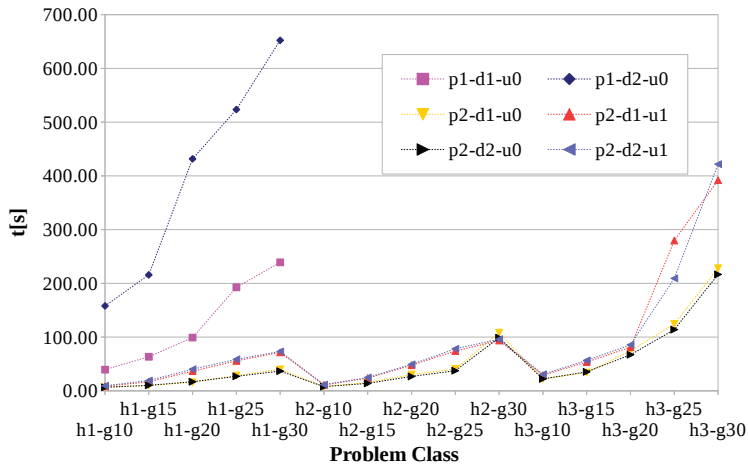

Figure 6.1: Average times on 20 random problems for 15 different configurations.

Problem	p1-d1	p1-d2	p2-d1	p2-d2
h1-g10	4.6 / 460.2	4.9 / 1118.3	4.6 / 140.7	4.9 / 155.4
h1-g15	6.6 / 731.8	6.9 / 1520.5	6.6 / 236.4	6.9 / 254.5
h1-g20	7.5 / 1008.2	8.2 / 2544.3	7.8 / 325.0	8.5 / 378.0
h1-g25	9.3 / 1677.3	9.8 / 3285.6	9.4 / 541.5	9.8 / 589.0
h1-g30	9.8 / 1861.4	10.6 / 3747.6	9.9 / 625.7	10.7 / 705.2

Table 6.1: Average number of applicable/tested *ICs* conditions for u0 with planner p1 and p2 for domains d1 and d2.

of 30 minutes to solve each problem. On this set of problems we used two different instances of Algorithm 1. Planner p1 solves Prolog constraints during preprocessing while planner p2 relies on TestAndResolve-*prolog*. Planner p1 shows that we can also solve Prolog constraints in the conditions of *ICs* during preprocessing. We do this for operators as well, which resembles the way in which most planners compile away static predicates. We argue that Prolog has some advantages since it allows for more compact representations due to the use of resolution (see Example 4.11).

6.2.2 Results

Figure 6.1 shows the average problem solving times on 20 random problems for each class. Planner p1 shows poor performance, but using planner p2 we were able to solve problems in a reasonable time, even for the most difficult set

that we used (g30, h3, u1). Comparing u0 to u1 we can see that the cost of uncertainty increases for the more difficult problems, which is not surprising, since more activities and goals are likely to increase the number of applicable *ICs*. An exception is h2 – g25 which takes less time with u1 than with u0. u1 has more potentially applicable *ICs* but slightly less actually applicable ones. In a similar way h3 – g25 has lower times for d2 than for d1. Our logs show that temporal reasoning and resource scheduling account for the difference which may be explained by *ICs* in the full set leading to fewer resource conflicts, which raises questions regarding the interplay between different constraint types.

Problem decomposition is enabled by the fact that we model each aspect of a domain with the appropriate type of constraint. In doing so we can separate reasoning about time, resources, logical constraints, causal dependencies among actions, *ICs* and costs from one another. This allows us to achieve planner p2, which significantly reduces the number of *IC* conditions that need to be tested. This effect is shown in Table 6.1, where we compare the number of applicable and tested *IC* conditions for a subset of problems.

Temporal expressiveness also contributes to the presented results. Removing the *intersection* constraint from the condition of *ICs* (see use cases SA-2, SA-3, and SA-4) adds a large amount of combinations to the set of applicable *ICs* that need to be resolved even if they are implicitly resolved already. For p2 in Table 6.1 this would force the planner to resolve all potentially applicable *ICs*, since they become applicable.

Except for one instance, all problems in our experiments were solvable within the allowed time of 30 minutes. In this case, however, the limit was reached when backtracking over planning decisions after an inconsistency was found when resolving *ICs*. Note that, unlike in the next domain, the order of solvers Θ for Algorithm 1 is irrelevant since all *ICs* relate actions of the *HaPlEx* to human activities (i.e., no *IC* will apply before actions are added to a CDB).

6.3 Research Facility Domain

In this section we test *ICs* for proactivity with a research facility domain. The purpose of these tests is to investigate how our approach scales when *ICs* are used for context-awareness and proactivity (FR_{CA}, FR_{PA}) when more goals and opportunities for proactivity are added. We perform two experiments. The first one is an offline experiment in which all information is known to the *HaPlEx* prior to planning. This is equivalent to assuming that meeting and experiment schedules of all workers at the research facility are known (e.g., the organization has a shared calendar). This includes sets with an increasing number of predefined goals and opportunities for proactivity. We evaluate the time to solve the resulting flaws. The second experiment is an online test in which opportunities for proactivity and goals appear online during simulated execution, over a period of three days, modeling the fact that researchers schedule meetings and experiments on the fly. The first experiment investigates the effect of two

alternative orderings of solvers in the human-aware planner. Other constraint types are sorted by increasing difficultly:

$$\begin{aligned}\Theta_a &= \langle \textsc{Domain}, \textsc{Costs}, \textsc{Sets}, \textsc{Prolog}, \textsc{Time}, \\ &\quad \textsc{Resources}, \text{IC}, \textsc{Goals}\rangle \\ \Theta_b &= \langle \textsc{Domain}, \textsc{Costs}, \textsc{Sets}, \textsc{Prolog}, \textsc{Time}, \\ &\quad \textsc{Resources}, \textsc{Goals}, \text{IC}\rangle\end{aligned}$$

The online experiment tests the effectiveness of the forgetting mechanism that was described in Section 4.7. As explained previously the aim of this mechanism is to keep the *HaPlEx* performant even if more and more statements are added (e.g., from observations) during long term execution. Forgetting statements reduces the number of temporal constraints, which in turn reduces the computational effort of temporal propagation of execution time. More importantly, forgetting statements also reduces the number of combinations of statements that have to be considered for *ICs*.

This domain consists of a set of robots in an environment with 20 offices, two meeting rooms and two laboratories. There are four basic types of goals: Cleaning rooms, object delivery, serving coffee during meetings, and assistance during dangerous work in the laboratories. Out of these four only the first two are added directly. The latter two are proactive goals: They are inferred as consequences of context derived from exogenous events. These exogenous events include new object delivery requests, human requests for meetings, and time scheduled in one of the laboratories. Meetings are handled through a chain of *ICs* for creating meetings (context-awareness), adding people to meetings (context-awareness), scheduling rooms (class 1A proactivity) and serving coffee during meetings (class 3A proactivity). When time in the laboratory is requested, *ICs* are used to create a goal for robot assistance (class 3A proactivity) in case the task is performed only by a single human. If the lab work is classified as a high risk task, a break of 30 minutes will be scheduled before it (class 2A proactivity). In the online case, calendar entries and goals are added during execution, and replanning is triggered to accommodate these new events and their consequences. The offline and online case only differ by the fact that we assume to know every goal and exogenous event for the offline version, while they appear at an unknown time during execution simulation in the online case. In addition, actions executed by the *HaPlEx* end at random times which become known online in the latter case. This domain uses six operators and seven *ICs*.

6.3.1 Offline Evaluation

In this set of problems we assume that every request and goal is known in advance. While this is not a realistic assumption it will allow us to evaluate

how *ICs* for proactivity affect scalability. We use combinations of four different settings: the number of initial delivery goals (5,10,15 and 20); the number meeting requests (0,2,6 and 10); the availability of meeting rooms (all available, one available, two partially available, none available); and the number of laboratory requests (0,4,8,12). Combining these settings we created a set of 256 problems. When there is at least one meeting request and no meeting room is available, the problem becomes unsolvable.

Figure 6.2 shows a comparison of run-times for all 256 problems, sorted by increasing number of goals. It is clearly visible that testing *ICs* first (Θ_a) outperforms solving goals first (Θ_b). This difference is most pronounced for the unsolvable problems that exclusively lead to a timeout when we solve goals first. Conversely, these problems are among the easiest when we solve *ICs* first. This can be observed in Figure 6.2 in the sudden peaks and drops of Θ_b and Θ_a respectively. In practice, ordering Θ_a creates and (attempts to) schedule meetings before it tries to reach open goals. It then detects (through repeated resource scheduling) that no room selection is feasible. Consequently, Algorithm 1 fails without ever trying to satisfy open goals. Ordering Θ_b, on the other hand, will lead to backtracking over all plans. Each plan will be rejected since no room can be scheduled. It seems that the intuition of CSP heuristics to choose the variable that is most likely to fail first [65] also applies to the order of solvers of Algorithm 1. In addition it seems a good approach to sort Θ in Algorithm 1 by increasing complexity of TestAndResolve-t procedures. Of course, in general, it is easy to construct examples where both these approaches fail. This leads to the interesting open question of how to determine the constraint type that is most likely to cause failure. This information could be used to provide Algorithm 1 with dynamic orderings of constraint types. Among all tested features, the number of initial open goals has the biggest impact on performance.

6.3.2 Online Evaluation

For online planning and execution we started each action as soon as possible and randomized their end time between earliest and latest possible times. Each time step has the duration of one minute and we simulate 4260 time steps (3 days). We simulate execution by making sets of exogenous events (i.e., new goals, activities, etc.) available to the *HaPlEx* at randomized times.

We use two simulation cases. In case #1 we triggered 28 exogenous events at 8 different time points distributed over three days. In case #2 we dispatched 21 exogenous events at randomized times (7 per day), to see how more frequent replanning affects performance. For case #1 we provide results both with and without the forgetting mechanism explained in Section 4.7.

Figure 6.3 shows the time used for each execution step. Peaks occur when replanning is triggered due to exogenous events. With the non-forgetting approach update and replanning times grow quickly without dropping down again. This is because replanning increases the size of the CDB and conse-

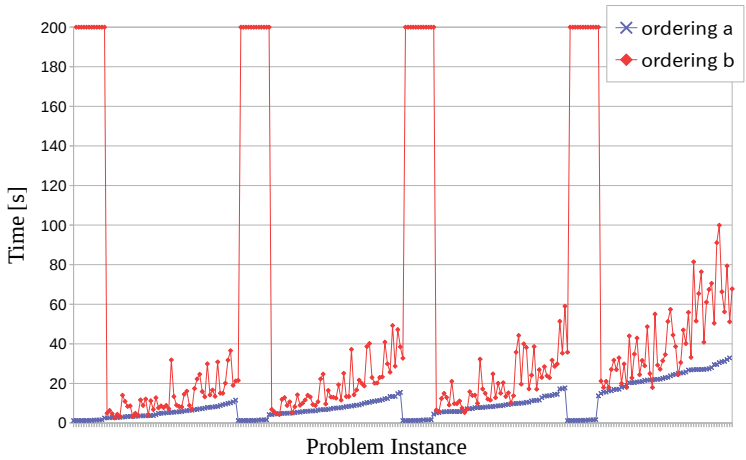

Figure 6.2: Comparing Θ_a to Θ_b. Sorted by increasing number of goals and by the time to solve the problem with order Θ_a. The 64 problems for each setting are sorted by the time it took to solve them with Θ_a.

quently leads to increasing computational overhead for both future replanning, as well as for the simple upkeep of the temporal network required by execution. If we employ the forgetting mechanism described in Section 4.7 to case #1 we obtain significantly better performance: The time required by temporal propagation still jumps after replanning, but it drops down as soon as actions are executed and statements and constraints become obsolete.

The results indicate that our approach is capable of providing human-aware plans including context-awareness and proactivity online (TR_{Online}). The forgetting mechanism suggested in Section 4.7.3 helps to reduce the impact of growing CDBs during execution. In the next section we go one step further and integrate the *HaPlEx* with a real robotic system.

6.4 Human-aware Planning for Robots in a Smart Home

In this test we demonstrate that our approach can be used for human-aware planning and execution with a real network robot system. This test also provides a practical implementation of planning for cooperation (FR_{PC}) that uses the solutions to the corresponding use cases (recall Section 5.6). Further, it demonstrates that cooperative plans can be executed with some robustness

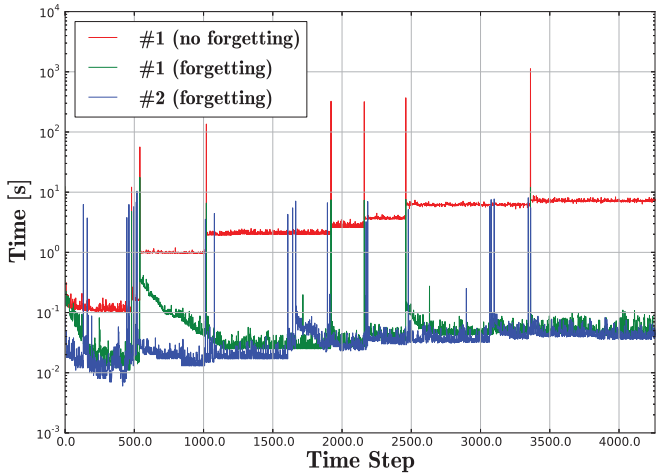

Figure 6.3: Update times for three runs over three days of simulation.

wrt. delays in the execution of human activities. We used the Turtlebot 2[2] as a robot platform and a sensor-equipped home at Örebro University for an assisted living domain.

The *HaPlEx* creates a plan to clean the apartment in collaboration with the human (FR_{PC}). It then executes the plan with the human while using context-awareness (FR_{CA}) and a set of sensors installed in the home to detect contingencies and will react to them in a proactive way (FR_{PA}). Here, our contingencies are reminders in case the human partner loses track of the plan.

We use several sensors (see Figure 6.4). A pressure sensor on top of the robot detects if the robot is carrying an object. A luminosity sensor on the TV detects if the TV is turned on. The human is located with a slipper equipped with an RFID reader that detects RFID tags embedded in the floor [140]. All sensors are connected to wireless XBee transmitter nodes and communicate with the *HaPlEx* via ROS.

Human movement is modeled with the cooperative *move-human* operator that we used as a solution to use case PC-1. We use two operators for the human to give or take objects and one cooperative operator to clean a room. We discussed two of these operators as use cases for FR_{PC} (Section 5.6). Every operator that includes the human contains an effect that depends on the human doing something. In case of *move-human*, we have the effect

```
(?E (at ?H) ?L2)
```

[2] `http://www.turtlebot.com/`

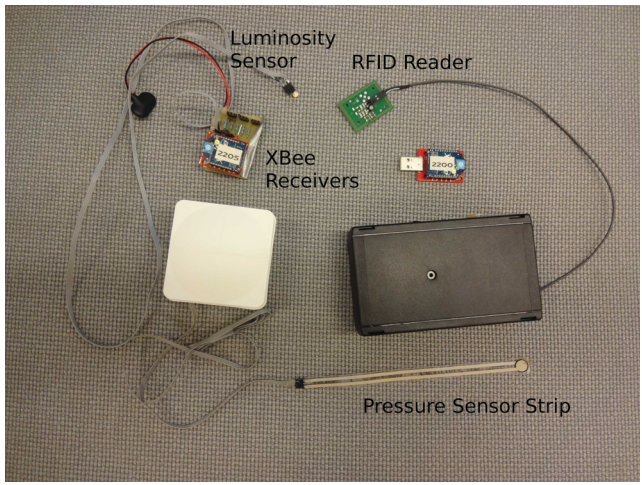

Figure 6.4: Sensors used in the test.

which is linked to a ROS subscriber (as described in Section 4.7). Whenever there is an effect that is linked to an observation the executive waits for the effect to be observed. In a similar way the *give* and *take* operators depend on the robot's pressure sensor, and the *cleaning* operator depends on a *done* signal. In our realization this signal is generated by pressing a button on a wireless joystick. In addition to moving capabilities we also provided the *HaPlEx* with a set of phrases it can use to let the human know about the next steps of the plan. The corresponding *say* statements in the operators are linked to a reactor that sends requests to a soundplay ROS node[3] that performs text-to-speech synthesis. The included files contain phrases that can be substituted if needed (e.g., with the object that the *HaPlEx* asks to put on the robot, or the name of the person the *HaPlEx* is interacting with). Note that the phrases are grounded via domain constraints, and their timing is the result of temporal constraint propagation. This is similar to previous work on speech synthesis via constraint reasoning [35].

The goal in the test is to change the state of all rooms from not tidy to tidy (using the clean operator). To execute the cooperative clean operator we require that the robot carries the trashcan. There is also a final goal (after all others) to return the trashcan to the kitchen. If it takes too long for the human to arrive at a target location the system will send a proactive reminder using the *IC* shown in the solution to use case PC-3 (Section 5.6). The same is true if the human starts to watch TV before the clean action is finished. This domain uses seven operators and up to four *ICs*.

[3] `http://wiki.ros.org/sound_play`

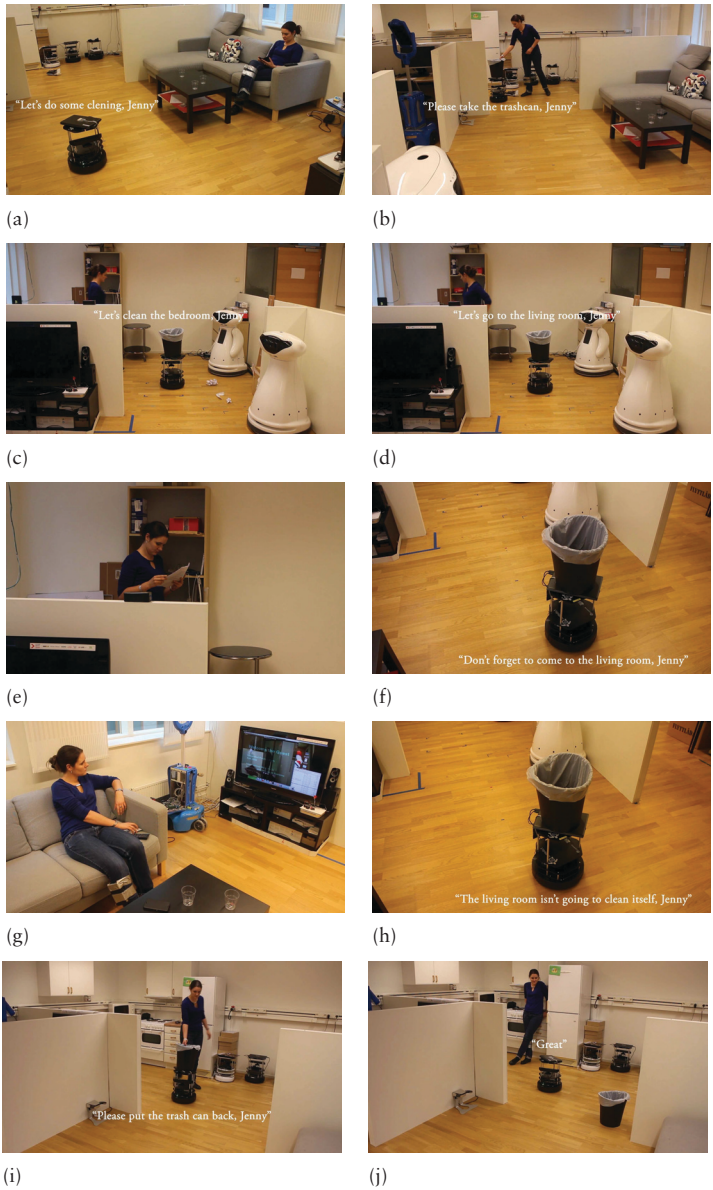

Figure 6.5: Representative moments of the robotics test.

Figure 6.6 shows a selection of timelines for a simulated execution in which the human and the robot clean three rooms. We see that on some occasions the movement reminder (use case PC-3, Section 5.6) has been triggered (a), the robot reminds the human to provide the trashcan (b) and on one occasion the television was turned on during a cleaning action which also led to a proactive reminder about the cooperative plan (c). A video of a real run of this test is available at `http://aass.oru.se/~fpa/HAP/CBHAP-robot-experiment.mp4`. Figure 6.5 shows the salient moments of plan execution. In the real run the human provided the trashcan before requiring the reminder.

Overall this test shows the capabilities of the *HaPlEx* to control a mobile robot by imposing human-awareness during planning and execution. This was achieved by using interaction constraints as introduced in Chapter 5 for context-awareness and proactivity (FR_{CA}, FR_{PA}) and by planning for cooperation (FR_{PC}). In the future we would like to extend this test featuring more than one human and a wider variety of sensors, robot capabilities, goals and human activities.

6.5 Benchmarking Interaction Constraints

We performed a series of tests to evaluate how well reasoning with *ICs* scales as more data is added to a CDB. The aim of these tests is to evaluate the performance of Algorithm 3 (recall Section 4.4.9 on resolving *ICs*) for specific cases that only use temporal constraints and statements. Obviously the performance of Algorithm 3 depends heavily on the constraint types involved in conditions and resolvers. Here we stick to tractable constraints (resulting in simple temporal problems) to test Algorithm 3 for simple conditions and resolvers. This allows us to evaluate the best case performance we can expect. Each test uses a simple domain with a single *IC* based on the *IC* pattern for social acceptability. We then test this *IC* by using an increasing number of statements that matches the context/condition part of the IC. Tests are generated in such a way that we know beforehand how many flaws have to be resolved and which resolvers work. In all of these tests we only use temporal constraints and statements to assure that testing conditions and resolvers can be done in polynomial time. In these tests we consider only social acceptability since in context-awareness we assume a single working resolver, and for proactivity we either need planning or also assume a single resolver that adds a statement that is executed.

As a basis we used the *IC* shown in Figure 6.7. It uses two boolean statements in its condition: if they intersect in their earliest time interpretation, one of two resolvers has to be applied. In a variation of this domain we used an *IC* with three/four statements (adding statements with state-variable x3/x4) in the condition and two/three temporal constraints in each resolver.

Starting out with the simplest possible case, *No Conflicts (NC)*, we test how a simple *IC* scales with an increasing number of applications that never cause any conflicts. We used pairs (or triples) of statements to which the *IC*

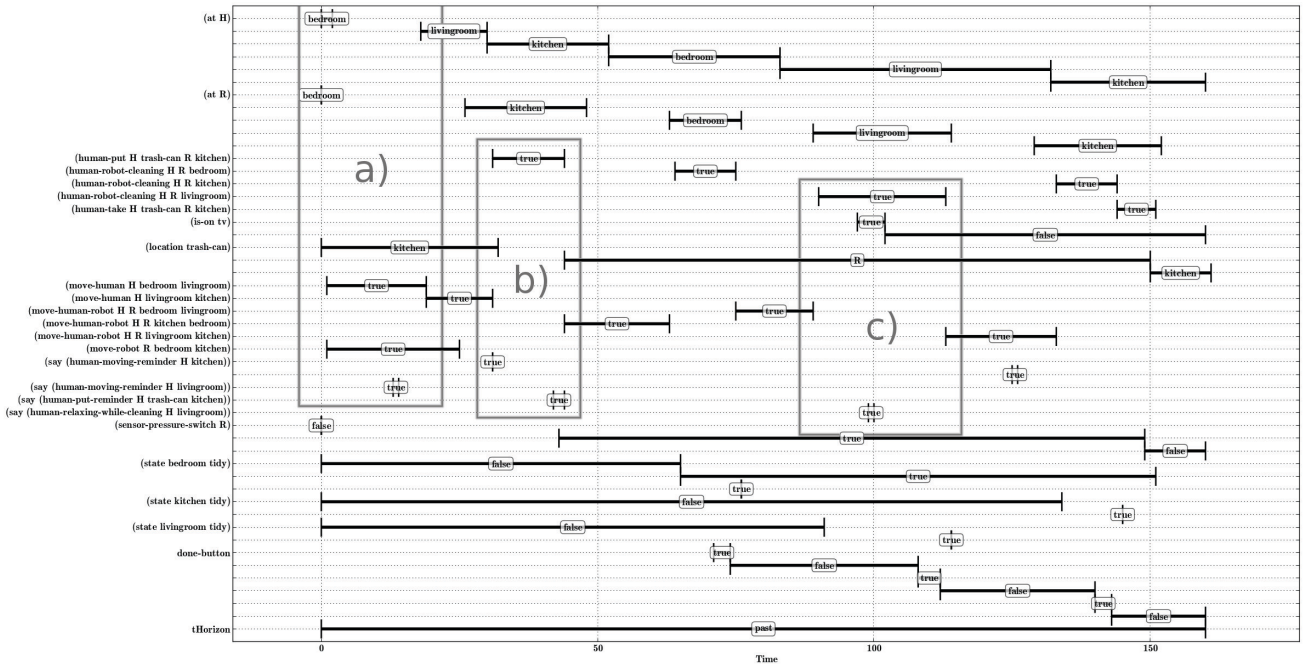

Figure 6.6: Timelines for execution of a cooperative plan highlighting three resolved *ICs*: A movement reminder (a), a reminder to provide the trashcan (b) and a cleaning reminder when the human turns on the TV during the cleaning action (c).

```
(:ic
  (social-acceptability-class-3 ?I1 ?I2)
  (:condition
    (:statement (?I1 (x1)) (?I2 (x2)) )
    (:temporal (intersection ?I1 ?I2) ) )
  (:resolver
    (:temporal (before ?I1 ?I2 [1 inf]) ) )
  (:resolver
    (:temporal (after ?I1 ?I2 [1 inf]) ) ) )

(:initial–context
  (:statement
    (1 (x1)) (2 (x2)) ; First combination
    (3 (x1)) (4 (x2)) ; Second combination
  )
  (:temporal
    (at 1 [0 3] [0 3]) (duration 1 [1 1])
    (at 2 [0 3] [0 3]) (duration 2 [1 1])
    (at 3 [4 7] [4 7]) (duration 3 [1 1])
    (at 4 [4 7] [4 7]) (duration 4 [1 1])))
```

Figure 6.7: Top: *IC* using the social acceptability pattern. It applies when intervals ?*I1* and ?*I2* intersect and is resolved by a before or after constraint between the two. Bottom: Setting up pairs of statements to which the *IC* on the top applies. This is the second problem of the series and each subsequent problem constraints one more pair.

applies and for which the first resolver always works. We used *ICs* with two and three statements (2INT, 3INT) in their condition and created 50 problems using from 1 to 50 combinations of statements that need to be resolved. Each combination is a set of statements whose intervals are constrained such that it is guaranteed that the *IC* will apply (Figure 6.7 is an example showing two combinations). On these two sets of 50 problems we also compare the performance between the two proposed approaches to finding flaws for *ICs* (recall Section 4.4.9): the brute-force approach (BF, Algorithm 4) and the one employing search (Algorithm 5). These tests should show how TestAndResolve-*IC* scales with an increasing number of potentially applicable *ICs*. Figure 6.8 (a) shows the results. While for two statements in the condition the brute-force version is slightly faster we can see a significant improvement in case of three statements.

In a second series, *First Fails (FF)*, we used an *IC* with n resolvers for which the first $n-1$ resolvers always fail. This series tests how simple *ICs* scale when every resolver except the last one fails. This is the worst-case that can be achieved without backtracking in Algorithm 1 (i.e., at least one resolver in line 12 of the algorithm works). We ran 50 tests each for $n \in \{2,\ldots,10\}$ to measure the impact of an increasing number failing resolvers. We manipulated the temporal constraints in Figure 6.7 (bottom) to make sure the first resolver in Figure 6.7 (top) fails for each flaw (i.e., we modified them so that the *before* constraint in the first resolver leads to an inconsistency). We then created *ICs* with n resolvers from Figure 6.7 (bottom) by repeating the first resolver $n-1$ times. Here the aim was to measure the impact on performance when *ICs* are difficult individually but do not require backtracking. Figure 6.8 (b) shows the results for 2 to 10 resolvers on 50 problems each (up to 50 applicable combinations). All *ICs* in these tests use 2 statements in their condition. We can see that the impact of going over all resolvers in order to find the working one is relatively small.

The third test uses a difficult test case for which we start with a series of satisfiable combinations of applicable statements but always end with one that cannot be satisfied. As a result, our current approach is forced to try all combinations of previous resolvers in an attempt to resolve the last *IC*. This test shows the worst-case in terms of the search space of *IC* resolvers. Although the outcome is predictable, this type of tests may yield insights into possible modifications to overcome an unfavorable choice of flaws (choosing the unresolvable flaw first would lead Algorithm 1 to terminate without search for each of these problems). In Figure 6.7 (bottom), for instance, we would change the *at* constraint of intervals 3 and 4 to

```
(at 3 [4 4] [5 5])
(at 4 [4 4] [5 5])
```

to have them intersect (satisfying the condition in Figure 6.7) and assure no resolver works (since the intervals are equal they cannot be before or after one another). Since the solver will resolve the unsatisfiable pair last it will test all

combinations of resolvers of satisfiable *ICs* before it concludes that the problem cannot be solved. Figure 6.8 (c) shows that the solving times for these problems grows exponentially as we increase the number if *ICs* that need to be resolved. This might be the worst-case for resolving *ICs* with our current approach, since every combination has to be tested and discarded before Algorithm 1 can decide that the problem has no solution. This issue could be overcome by modifying the flaw selection strategy for *ICs*. Testing, for instance, for any *IC* that cannot be satisfied instead of simply resolving the first applicable one that was found would avoid the issue at least for the above case. Of course this would create a computational overhead for easier problems. Other ways to overcome this issue include modifications of the backtracking strategy to employ ideas such as backjumping [65, Ch. 6] to find the earliest point in the search space at which an unresolvable flaw exists (methods as the ones described in Section 4.6 could be considered). However, it must also be said that even if TestAndResolve-*IC* is modified to solve this set of problems efficiently, the problems could always be modified in such a way that the unsatisfiability is only detectable after all other *ICs* have been resolved (i.e., by assuring that the condition of the unsatisfiable *IC* will only be satisfied after all other *ICs* have been resolved).

In the fourth set of problems we investigate how solving time scales when we increase the number of statements in the condition of an *IC*. For this test we use a fixed number of combinations of statements for which the *IC* applies and increase the number of statements in the condition. Again, each combination of statements resembles one applicable *IC* condition. Figure 6.8 (d) compares our two approaches to finding flaws for *ICs* on these problems (again we use "BF" for Algorithm 4 and "Search" for Algorithm 5). This is probably the most favorable possible comparison between the two approaches. In Figure 6.8 (a) we show how they compare on the first series of problems (No Conflicts) for two and three statements in the condition. Algorithm 4 is slightly faster for two intervals. In this case the search in Algorithm 5 provides a slight computational overhead since it will always start by testing a match for a single statement which will, in these problems, never be inconsistent. On problems with three statements it can be seen that Algorithm 5 clearly outperforms Algorithm 4 even if the difference is less pronounced than in Figure 6.8 (d).

Overall we conclude that *ICs* scale reasonably well as we increase the number of statements in their conditions. The number of resolvers we have to test before finding a working one (without backtracking) does not have a major effect on performance. In general, this obviously depends on the difficulty of the constraints involved in the resolvers. However, in cases where search has to backtrack over combinations of resolvers of *ICs* the problem becomes very difficult. In such cases it seems that approaches such as backjumping or forward checking used in constraint processing [65, Ch. 6] may provide help but are unlikely to remove the issue.

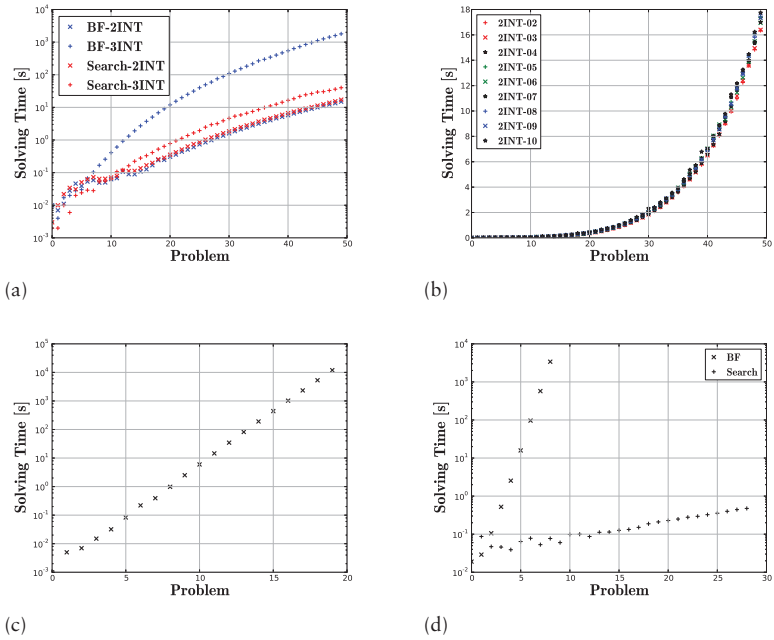

Figure 6.8: Benchmarking results: (a) Comparing solving times when using Algorithms *5* and 4 on the No Conflicts test set with 2 and 3 variables in the condition. (b) Benchmark results for *ICs* with n resolvers and problems that are created in such a way that the first $n - 1$ resolvers always fail. (c) Benchmark results for difficult unsatisfiable problems. As expected the time it takes to prove unsatisfiability is exponential with the current approach. (d) Solving times when scaling the number of statements in the condition of the *IC* from 2 to 30. For each problem 5 flaws have to be resolved. In case of Algorithm 4 we only went up to 10 variables in the *IC*'s condition.

6.6 Discussion

We investigated our approach on a series of domains to test how our algorithms scale for human-aware planning problems with social acceptability, proactivity and context-awareness. We also tested the integration with a real robotic system for planning for cooperation with humans.

We found that our approach is capable of satisfying our functional requirements. Our experiments in the *household* and *research-facility* domains show the feasibility of social acceptability (FR_{SA}), proactivity (FR_{PA}) and context-awareness (FR_{CA}). In case of proactivity we also integrated planning with execution to handle contingencies caused by exogenous events over multiple days of simulation. We also show the impact of the ordering of solvers in Algorithm

1. The integration with ROS was used in a test with a real robot and sensors that takes into account sensor data during execution of a cooperative plan to handle contingencies (FR_{PC}). The *IC* benchmarks investigate some of the limits of *ICs* and compare the two approaches presented in Section 4.4.9. Overall, we found that *ICs* provide a very useful tool to approach the functional requirements of human-aware planning.

Chapter 7
Conclusions

First, we will summarize the contributions of this thesis in the next section. Then we will provide overview of limitations and an outlook on possible further research possibilities in three different directions.

7.1 Summary of Contributions

We started this thesis with a top-down analysis of functional and technical requirements of human-aware planning and execution. We identified functional requirements for social acceptability (FR_{SA}), proactivity (FR_{PA}), context awareness (FR_{CA}), and planning for cooperation (FR_{PC}). For each of these requirements we provided a series of use cases that allow to verify the fulfillment of the requirements. Furthermore, we have provided the tools to adequately model human-aware planning domains and problems (FR_{KR}). Based on these functional requirements we derived a series of technical requirements for implementing a Human-aware Planner and Executive (*HaPlEx*). We found that reasoning about human-awareness of plans requires a variety of types of knowledge (TR_{Know}), should be extendable with new types of knowledge (TR_{Extend}), and modular (TR_{Modul}). We further found that temporal reasoning should support flexibility (TR_{Flex}), human-awareness has to be maintained online during execution (TR_{Online}), and that the *HaPlEx* has to deal with uncertainty of various types (TR_{Uncert}). The framework of flaw resolution seemed most promising to integrate reasoning about human-awareness with all required knowledge types in an extendable and modular way.

Based on this analysis, we then proceeded bottom up, starting from the technical requirements. This led us to develop a constraint-based planning and execution approach. This approach combines many different types of constraints in a modular and extendable way by using flaw resolution. In this way we integrated planning with flexible temporal reasoning, resource scheduling, costs, sets, Prolog constraints, and Interaction Constraints (*ICs*) (TR_{Know}). We use a representation based on flexible temporal intervals (TR_{Flex}) and reactor-based

execution (TR_{Online}) that is integrated with the Robot Operating System (ROS) that allows communication, for instance, with sensors or robots in the environment. In order to facilitate execution over longer temporal horizons we included an approach to "forgetting" sets of constraints that are only concerned with the past. Since the search space of flaw-resolution can be very complex, we consider a series of pruning methods that can be used to find problems early on in case creating a single resolver involves a search space of its own (as is the case for the open goal resolver described in Section 4.4.8). These methods include testing partial solutions, finding the shortest conflicting plan, and lifting constants in a plan to variables and test if the conflict remains in this more general plan. The supported set of constraints can be extended by any type of constraint that fulfills a series of assumptions about the flaws it can create and their resolvers. We provided a formal analysis of the constraint-based planning problem and our approach that is based only on these assumptions. In this way our approach facilitates modularity and extendability (TR_{Modul}, TR_{Extend}) in theory, as well as in the algorithm that solves the problem.

Building upon the proposed constraint-based planning approach, we then turned our attention to addressing the functional requirements. Towards this aim, we introduced patterns of *ICs* and discussed how they can be used to enable social acceptability (FR_{SA}), proactivity (FR_{PA}) and context awareness (FR_{PC}) during planning and execution. We gave a basic analysis of the effort required by those patterns. *ICs* are expressive enough to provide domain engineers with the means to model a variety of cases of the above functional requirements (FR_{KR}). We demonstrated the modeling advantages of our approach on a series of use cases. These use cases also serve to show that our approach is capable of addressing the above functional requirements and also the requirement for planning for cooperation (FR_{PC}). We also demonstrated the modeling advantages of *ICs* over potential solutions in PDDL on our use cases. While basic cases can often be handled in PDDL, minor modifications would make PDDL solutions either cumbersome or impossible.

We evaluated the approach on a series of domains. We tested a set of *ICs* for social acceptability in a household domain. We used a research facility domain for proactivity and context-awareness. In both cases we scaled problems by different factors and reached overall satisfactory results. We also presented a real-world assisted living test in which we control a mobile robot and use the *HaPlEx* to plan for cooperation between human and robot while using *ICs* to detect contingencies derived online from real sensors. While smaller in scale than the other tests, this demonstrates how the *HaPlEx* can be used for human-aware planning and execution in a real world domain. Finally, we evaluated the performance of *ICs* in a benchmark domain in which we tested how our approach to resolving *ICs* scales when increasing the number of cases in which their condition applies. Overall, the results of these evaluations point to the practical applicability of the approach and to interesting possibilities for future work.

7.2 Extending Requirements, Use Cases & Evaluation Methodology

As pointed out before, this thesis only provides a first step fulfilling the formulated requirements. In the future, further requirements may arise from practical experience. More complex use cases could be created to provide a more in depth analysis of how different approaches can be used to fulfill the requirements. Other questions that remain open are: how far can use cases go when it comes to testing the fulfillment of requirements? What other methods could be used to determine that a given approach fulfills the requirements?

Recently, interaction in environments with humans and robots was formalized by Chakraborti et al. [38]. Their work also introduces several metrics that could provide useful insights when evaluating human-aware planning and execution.

7.3 Increasing Efficiency

In general, *ICs* are very powerful, our evaluation showed some limitations in cases where all combinations of resolvers have to be explored. If we use many *ICs* with more than one resolver and are provided with an unsolvable or difficult problem, the number of combinations that have to be considered before a solution is found (or inconsistency is decided) can grow exponentially. This could be countered to some degree by using a flaw selection strategy to pick *ICs* that are most likely to fail first. A very simple form of this strategy will try *ICs* with fewer resolvers first. This is a common strategy in the constraint processing literature, where we also find other possibilities, such as forward checking and backjumping, that are worth investigating for *ICs* in the future.

More advanced strategies for improving performance could rely on information about the constraint types that are involved. In resource scheduling, for instance, flaws and resolvers are sorted based on their impact on temporal flexibility [36]. Our approach could accommodate more general ways for reasoners to exchange information. For example we could exploit the modularity of our approach to include specialized reasoners for dynamic flaw selection.

Possible directions for future work regarding performance include better heuristics that could account for information such as costs. A deeper understanding and an evaluation of pruning methods as the ones outlined in Section 4.6 would be very interesting as well.

7.4 Extending the Constraint-Based Planner

One of the main advantages of the presented planning approach is the fact that it can use any type of constraint that satisfies the established assumptions. In the

future we would like to extend the set of constraint types that are considered beyond the ones that appear in this thesis.

The implementation already contains support for the constraint processing language and solver MiniZinc [181, 166]. We also added a way of creating and reasoning about properties of graphs and evaluation of simple equations.

We would like to extend our representation in different ways. While our approach presents a basis towards fulfilling the requirements of human-aware planning and execution, it is so far limited in its handling of uncertainty. It is capable of reacting to human activities online, and our approach to solving *ICs* handles uncontrollable variables in a conformant way. Conditional planning [208, 131, 234] might be preferable since a conformant approach may fail to find a solution or may produce overly restricted solutions. We would also like to investigate advanced constraint types for uncertainty in our framework (e.g., constraints describing Bayesian or Markov networks).

Another direction for future work is optimization of social costs. In the approach presented we can put an upper limit on the social cost but there is no attempt to optimize it. A standard way to achieve this would be to keep searching for better solutions once the first solution has been found. Any (partial) solutions that increases the cost to be higher than the current best solution can be discarded. This becomes more challenging if we consider optimization of multiple parameters.

If a plan fails during execution or new goals are added to the problem our approach creates a new problem from the current state of execution. If this happens for a plan that was hard to find we would prefer to use a mechanism for plan repair instead of creating a new plan. Goals for proactivity may often require only minor adaptations to an existing plan. A possible approach here would be to remove all causal links and add an open goal for each causal link that was removed along with the newly added goals. To resolve the open goal flaws in the resulting problem the approach of plan-space planning seems more appropriate.

We considered three preliminary methods that can be used to prune the search space of the open goal resolver and to generalize inconsistent constraint databases. We believe that there is potential for interesting advances by generalizing approaches from constraint processing to constraint-based planning via flaw-resolution.

7.5 Extending the Formal Analysis

We provided a formal analysis of the constraint-based planning approach that was aimed at restricting allowed constraint types only by means of assumptions. This contributes to making the approach extendable to all constraint types that satisfy the given assumptions. In the future it would be interesting to advance this analysis by considering different sets of assumptions.

Introducing assumptions to make solution existence decidable without limiting the maximum number of statements would be an interesting area to look into. If we could, for instance, devise an algorithm that decides if a CDB could lead to an infinite number of flaws, then we could create an assumption based on this algorithm. This type of algorithm would need to analyze the structure of resolvers and the types of constraints that these resolvers may include.

Appendix A
Household Domain

A.1 Domain

```
(:initial-context
  ;; Loading Prolog background knowledge (see next section)
  (:include ( kb "./kb.prolog" ) )
  (:domain
    ;; Type definitions
    ( enum robot ( list r1 r2 r3 r4 ) )
    ( enum location ( list robotRoom floor1 floor2 basement livingRoom
        kitchen diningRoom bedroomParents bedroomGrandparents1
        bedroomGrandparents2 kidsRoom1 kidsRoom2 bathroom1 bathroom2
        bathroom3 bathroom4 laundryRoom storage out study ) )
    ( enum locationType ( list bedroom bathroom kidsRoom ) )
    ( enum robotTask )
    ( enum robotTaskClass ( list vaccuum sortStuff assist collectTrash
        takeOutTrash collectLaundry laundry cleanRoom entertain
        recharge_r1 recharge_r2 recharge_r3 recharge_r4 ) )
    ( enum robotTaskState ( list finished beingAttended waiting ) )
    ( enum human ( list mother father kid1 kid2 kid3 kid4 infant1
        infant2 grandmother1 grandmother2 grandfather1 grandfather2 ) )
    ( enum property ( list lightSleeper normalSleeper heavySleeper
        easilyDistracted ) )
    ( enum role ( list parent grandparent kid infant ) )
    ( enum activity ( list idle out eating sleeping reading working
        cooking playing usingBathroom ) )
    ;; Object is composed of all elements from types human and robot
    ( enum object ( list human robot ) )
    ;; Integer types are defined by range
    ( int intType ( interval 0 1000 ) )
    ( int costRange ( interval 0 200 ) )
    ;; Define state-variable signatures
    ( sig ( at object ) location )
    ( sig ( state robotTask robotTaskClass location ) robotTaskState )
```

```
  ( sig ( busy robot ) intType )
  ( sig ( goalBatchDivider intType ) boolean )
  ( sig ( hasProperty human property ) boolean )
  ( sig ( married human human ) boolean )
  ( sig ( locationType location locationType ) boolean )
  ( sig ( adjacent location location ) boolean )
  ( sig ( speed robot intType ) boolean )
  ( sig ( capability robot robotTaskClass ) boolean )
  ( sig ( hasTaskClass robotTask robotTaskClass ) boolean )
  ( sig ( robotTaskLocation robotTask location ) boolean )
  ( sig ( executionTime robot robotTaskClass intType ) boolean )
  ( sig ( movingTime robot location location intType ) boolean )
  ( sig ( humanActivity human location activity ) boolean )
  ( sig ( hasRole human role ) boolean )
  ( sig socialCost costRange ) )
(:resource
  ;; A robot can only do one thing at a time
  (reusable ( busy robot ) 1 ) )

(:ic
  ( LightSleeper ?H ?L ?T ?C ?I1 ?I2 )
  (:condition
    (:statement
      ( ?I1 ( humanActivity ?H ?L sleeping ) true )
      ( ?I2 ( state ?T ?C ?L ) beingAttended ) )
    (:domain
      ( in ?C ( list vaccuum cleanRoom sortStuff ) ) )
    (:temporal
      ( intersection {?I1 ?I2} ) )
    (:prolog kb
      ( hasProperty ?H lightSleeper ) ) )
  (:resolver
    (:temporal
      ( before ?I1 ?I2 [1 1440] ) ) )
  (:resolver
    (:temporal
      ( after ?I1 ?I2 [1 1440] ) ) ) )
(:ic
  ( NormalSleeper ?H ?L ?T ?I1 ?I2 )
  (:condition
    (:domain
      ( in ?C ( list vaccuum cleanRoom sortStuff ) ) )
    (:statement
      ( ?I1 ( humanActivity ?H ?L sleeping ) true )
      ( ?I2 ( state ?T ?C ?L ) beingAttended ) )
    (:temporal
      ( intersection {?I1 ?I2} ) )
```

```
    (:prolog kb
      ( hasProperty ?H normalSleeper ) ) )
  (:resolver
    (:temporal
      ( before ?I1 ?I2 [1 1440] ) ) )
  (:resolver
    (:temporal
      ( after ?I1 ?I2 [1 1440] ) ) )
  (:resolver
    (:cost
      ( add socialCost 10 ) ) ) )
(:ic
  ( NoDistractions ?H ?L ?T ?C ?I1 ?I2 )
  (:condition
    (:statement
      ( ?I1 ( humanActivity ?H ?L working ) true )
      ( ?I2 ( state ?T ?C ?L ) beingAttended ) )
    (:temporal
      ( intersection {?I1 ?I2} ) )
    (:prolog kb
      ( hasProperty ?H easilyDistracted ) ) )
  (:resolver
    (:temporal
      ( before ?I1 ?I2 [1 1440] ) ) )
  (:resolver
    (:temporal
      ( after ?I1 ?I2 [1 1440] ) ) )
  (:resolver
    (:cost
      ( add socialCost 20 ) ) ) )
(:ic
  ( Occupied ?H ?L ?R ?I1 ?I2 )
  (:condition
    (:statement
      ( ?I1 ( humanActivity ?H ?L usingBathroom ) true )
      ( ?I2 ( at ?R ) ?L ) )
    (:temporal
      ( intersection {?I1 ?I2} ) )
    (:prolog kb
      ( locationType ?L bathroom ) ) )
  (:resolver
    (:temporal
      ( before ?I1 ?I2 [1 1440] ) ) )
  (:resolver
    (:temporal
      ( after ?I1 ?I2 [1 1440] ) ) ) )
(:ic
```

```
  ( HatesRobots ?H ?L ?R ?A ?I1 ?I2 )
  (:condition
    (:statement
      ( ?I1 ( humanActivity ?H ?L ?A ) true )
      ( ?I2 ( at ?R ) ?L ) )
    (:temporal
      ( intersection {?I1 ?I2} ) )
    (:prolog kb
      ( hasProperty ?H hatesRobots ) ) )
  (:resolver
    (:temporal
      ( before ?I1 ?I2 [1 1440] ) ) )
  (:resolver
    (:temporal
      ( after ?I1 ?I2 [1 1440] ) ) )
  (:resolver
    (:cost
      ( add socialCost 5 ) ) ) )
(:ic
  ( LetMeEat ?H ?L ?T ?I1 ?I2 )
  (:condition
    (:statement
      ( ?I1 ( humanActivity ?H ?L eating ) true )
      ( ?I2 ( state ?T collectTrash ?L ) beingAttended ) ) )
  (:resolver
    (:temporal
      ( after ?I2 ?I1 [1 1440] ) ) )
  (:resolver
    (:temporal
      ( before ?I2 ?I1 [1 1440] ) ) )
  (:resolver
    (:cost
      ( add socialCost 10 ) ) ) )
(:ic
  ( AloneTime ?H1 ?H2 ?L ?R ?A ?I1 ?I2 )
  (:condition
    (:statement
      ( ?I1 ( humanActivity ?H1 ?L ?A ) true )
      ( ?C2 ( humanActivity ?H2 ?L ?A ) true )
      ( ?I2 ( at ?R ) ?L ) )
    (:temporal
      ( intersection {?I1 ?C2 ?I2} ) )
    (:prolog kb
      ( married ?H1 ?H2 )
      ( locationType ?L bedroom ) ) )
  (:resolver
    (:temporal
```

```
      ( before ?I1 ?I2 [1 1440] )
      ( before ?C2 ?I2 [1 1440] ) ) )
  (:resolver
    (:temporal
      ( after ?I1 ?I2 [1 1440] )
      ( after ?C2 ?I2 [1 1440] ) ) )
  (:resolver
    (:cost
      ( add socialCost 50 ) ) ) )
(:ic
  ( LetMePlay ?H ?L ?T ?I1 ?I2 )
  (:condition
    (:domain
      ( in ?c ( list vaccuum sortStuff collectTrash takeOutTrash
          collectLaundry laundry cleanRoom ) ) )
    (:statement
      ( ?I1 ( humanActivity ?H ?L playing ) )
      ( ?I2 ( state ?T ?C ?L ) beingAttended ) )
    (:temporal
      ( intersection {?I1 ?I2} ) )
    (:prolog kb
      ( hasRole ?H kid )
      ( locationType ?L kidsRoom ) ) )
  (:resolver
    (:temporal
      ( after ?I2 ?I1 [1 1440] ) ) )
  (:resolver
    (:temporal
      ( before ?I2 ?I1 [1 1440] ) ) )
  (:resolver
    (:cost
      ( add socialCost 5 ) ) ) )
(:ic
  ( LetMeRead ?H ?L ?T ?I1 ?I2 )
  (:condition
    (:statement
      ( ?I1 ( humanActivity ?H ?L reading ) )
      ( ?I2 ( state ?T vaccuum ?L ) beingAttended ) )
    (:temporal
      ( intersection {?I1 ?I2} ) ) )
  (:resolver
    (:temporal
      ( after ?I1 ?I2 [1 1440] ) ) )
  (:resolver
    (:temporal
      ( before ?I1 ?I2 [1 1440] ) ) )
  (:resolver
```

```
      (:cost
        ( add socialCost 10 ) ) ) )

;; Move robot from one location to another
(:operator
  ( Move ?R - robot ?L1 - location ?L2 - location )
  (:preconditions
    ( ?P1 ( at ?R ) ?L1 ) )
  (:effects
    ( ?E1 ( at ?R ) ?L2 ) )
  (:constraints
    (:prolog kb
      ( adjacent ?L1 ?L2 )
      ( movingTime ?R ?L1 ?L2 ?T ) )
    (:temporal
      ( duration ?THIS [?T 1440] )
      ( meets ?P1 ?THIS )
      ( meets ?THIS ?E1 )
      ( duration ?E1 [1 1440] ) ) ) )

;; Use robot to solve a task
(:operator
  ( SolveTask ?R - robot ?Task - robotTask ?TaskClass - robotTaskClass
      ?L -location )
  (:preconditions
    ( ?P1 ( at ?R ) ?L )
    ( ?P2 ( state ?Task ?TaskClass ?L ) waiting ) )
  (:effects
    ( ?E1 ( state ?Task ?TaskClass ?L ) beingAttended )
    ( ?E2 ( state ?Task ?TaskClass ?L ) finished )
    ( ?E3 ( busy ?R ) 1 ) )
  (:constraints
    (:prolog kb
      ( capability ?R ?TaskClass )
      ( executionTime ?R ?TaskClass ?T ) )
    (:temporal
      ( duration ?THIS [?T 1440] )
      ( during ?THIS ?P1 [1 1440] [1 1440] )
      ( equals ?THIS ?E1 )
      ( equals ?E1 ?E3 )
      ( meets ?P2 ?E1 )
      ( meets ?E1 ?E2 ) ) ) )
```

A.2 Prolog Knowledge Base

```
adjacent(L1,L2) :- adjacencyTable(L1,L2).
adjacent(L1,L2) :- adjacencyTable(L2,L1).

adjacencyTable(out,floor1).
adjacencyTable(floor1,floor2).
adjacencyTable(basement,floor1).
adjacencyTable(basement,floor2).

adjacencyTable(floor1,bathroom1).
adjacencyTable(floor2,bathroom2).
adjacencyTable(floor1,livingRoom).
adjacencyTable(floor1,diningRoom).
adjacencyTable(diningRoom,kitchen).

adjacencyTable(floor1,kidsRoom1).
adjacencyTable(floor1,kidsRoom2).
adjacencyTable(floor2,bedroomParents).
adjacencyTable(floor2,study).

adjacencyTable(floor2,bedroomGrandparents1).
adjacencyTable(floor2,bedroomGrandparents2).
adjacencyTable(bedroomGrandparents1,bathroom3).
adjacencyTable(bedroomGrandparents2,bathroom4).

adjacencyTable(basement,robotRoom).
adjacencyTable(basement,laundryRoom).
adjacencyTable(basement,storage).

movingTimeTable(r1,floor1,floor2,5).
movingTimeTable(r1,floor1,basement,5).
movingTimeTable(r1,basement,floor2,5).
movingTimeTable(r2,floor1,floor2,5).
movingTimeTable(r2,floor1,basement,5).
movingTimeTable(r2,basement,floor2,5).
movingTimeTable(r3,floor1,floor2,5).
movingTimeTable(r3,floor1,basement,5).
movingTimeTable(r3,basement,floor2,5).
movingTimeTable(r4,floor1,floor2,5).
movingTimeTable(r4,floor1,basement,5).
movingTimeTable(r4,basement,floor2,5).

defaultMovingtime(r1,1).
defaultMovingtime(r2,1).
defaultMovingtime(r3,1).
defaultMovingtime(r4,1).
```

```
movingTime(R,L1,L2,T) :- movingTimeTable(R,L1,L2,T).
movingTime(R,L1,L2,T) :- movingTimeTable(R,L2,L1,T).
movingTime(R,L1,L2,T) :- defaultMovingtime(R,T).

capability(r1,vaccuum).
capability(r1,recharge_r1).
capability(r2,sortStuff).
capability(r2,laundry).
capability(r2,collectLaundry).
capability(r2,recharge_r2).
capability(r3,collectTrash).
capability(r3,takeOutTrash).
capability(r3,cleanRoom).
capability(r3,recharge_r3).
capability(r4,entertain).
capability(r4,assist).
capability(r4,recharge_r4).

executionTime(r1,vaccuum,30).
executionTime(r1,recharge_r1,120).
executionTime(r2,sortStuff,60).
executionTime(r2,laundry,60).
executionTime(r2,collectLaundry,10).
executionTime(r2,recharge_r2,240).
executionTime(r3,collectTrash,60).
executionTime(r3,takeOutTrash,5).
executionTime(r3,cleanRoom,40).
executionTime(r3,recharge_r3,60).
executionTime(r4,entertain,60).
executionTime(r4,assist,20).
executionTime(r4,recharge_r4,120).

married(mother,father).
married(grandmother1,grandfather1).
married(grandmother2,grandfather2).

locationType(bedroomParents,bedroom).
locationType(bedroomGrandparents1,bedroom).
locationType(bedroomGrandparents2,bedroom).

locationType(bathroom1,bathroom).
locationType(bathroom2,bathroom).
locationType(bathroom3,bathroom).
locationType(bathroom4,bathroom).

locationType(kidsRoom1,kidsRoom).
```

```
locationType(kidsRoom2,kidsRoom).

hasRole(mother,parent).
hasRole(father,parent).
hasRole(kid1,kid).
hasRole(kid2,kid).
hasRole(kid3,kid).
hasRole(kid4,kid).
hasRole(infant1,infant).
hasRole(infant2,infant).
hasRole(grandmother1,grandparent).
hasRole(grandmother2,grandparent).
hasRole(grandfather1,grandparent).
hasRole(grandfather2,grandparent).

hasProperty(mother,lightSleeper).
hasProperty(father,heavySleeper).
hasProperty(grandfather1,heavySleeper).
hasProperty(grandmother1,lightSleeper).

hasProperty(grandfather1,hatesRobots).

hasProperty(mother,lightSleeper).
hasProperty(father,lightSleeper).
hasProperty(kid1,normalSleeper).
hasProperty(kid2,normalSleeper).
hasProperty(kid3,normalSleeper).
hasProperty(kid4,normalSleeper).
hasProperty(infant1,lightSleeper).
hasProperty(infant2,lightSleeper).
hasProperty(grandmother1,normalSleeper).
hasProperty(grandmother2,heavySleeper).
hasProperty(grandfather1,normalSleeper).
hasProperty(grandfather2,heavySleeper).

hasProperty(kid1,easilyDistracted).
hasProperty(kid2,easilyDistracted).
hasProperty(kid3,easilyDistracted).
hasProperty(kid4,easilyDistracted).
```

A.3 Example Problem

```
(:initial-context
  (:domain
    ;; IDs for robot tasks
    ( enum robotTask ( list tCollectlaundry1 tSortstuff2 tAssist3
        tCollecttrash4 tTakeouttrash5 tEntertain6 tVaccuum7
        tCollectlaundry8 tEntertain9 tTakeouttrash10 ) ) )
  (:statement
    ;; Statements represting human activities
    ( a1 ( humanActivity mother kitchen cooking ) true )
    ( a2 ( humanActivity mother bathroom2 usingBathroom ) true )
    ( a3 ( humanActivity mother kitchen cooking ) true )
    ( a4 ( humanActivity mother bedroomParents sleeping ) true )
    ( a5 ( humanActivity mother kitchen cooking ) true )
    ( a6 ( humanActivity mother diningRoom eating ) true )
    ( a7 ( humanActivity mother out out ) true )
    ( a8 ( humanActivity mother basement idle ) true )
    ( a9 ( humanActivity mother livingRoom sleeping ) true )
    ( a10 ( humanActivity mother kitchen working ) true )
    ( a11 ( humanActivity mother out out ) true )
    ( a12 ( humanActivity mother livingRoom sleeping ) true )
    ( a13 ( humanActivity mother diningRoom eating ) true )
    ( a14 ( humanActivity mother study idle ) true )
    ( a15 ( humanActivity mother diningRoom reading ) true )
    ( a16 ( humanActivity mother kitchen cooking ) true )
    ( a17 ( humanActivity mother basement working ) true )
    ( a18 ( humanActivity mother diningRoom reading ) true )
    ( a19 ( humanActivity mother kitchen idle ) true )
    ( a20 ( humanActivity mother diningRoom eating ) true )
    ( a21 ( humanActivity mother kitchen sleeping ) true )
    ( a22 ( humanActivity father livingRoom idle ) true )
    ( a23 ( humanActivity father livingRoom sleeping ) true )
    ( a24 ( humanActivity father out playing ) true )
    ( a25 ( humanActivity father out out ) true )
    ( a26 ( humanActivity father storage working ) true )
    ( a27 ( humanActivity father kitchen reading ) true )
    ( a28 ( humanActivity father bathroom3 usingBathroom ) true )
    ( a29 ( humanActivity father livingRoom idle ) true )
    ( a30 ( humanActivity father diningRoom eating ) true )
    ( a31 ( humanActivity father bedroomParents sleeping ) true )
    ( a32 ( humanActivity father out out ) true )
    ( a33 ( humanActivity father kitchen cooking ) true )
    ( a34 ( humanActivity father bedroomParents sleeping ) true )
    ( a35 ( humanActivity father out working ) true )
    ( a36 ( humanActivity father bathroom4 usingBathroom ) true )
    ( a37 ( humanActivity father kitchen eating ) true )
```

```
( a38 ( humanActivity father out out ) true )
( a39 ( humanActivity father kitchen cooking ) true )
( a40 ( humanActivity father floor2 idle ) true )
( a41 ( humanActivity father bedroomParents reading ) true )
( a42 ( humanActivity father bedroomParents sleeping ) true )
( a43 ( humanActivity father bedroomParents sleeping ) true )
( a44 ( humanActivity kid1 kitchen eating ) true )
( a45 ( humanActivity kid1 kidsRoom2 idle ) true )
( a46 ( humanActivity kid1 diningRoom playing ) true )
( a47 ( humanActivity kid1 diningRoom idle ) true )
( a48 ( humanActivity kid1 livingRoom playing ) true )
( a49 ( humanActivity kid1 diningRoom playing ) true )
( a50 ( humanActivity kid1 out out ) true )
( a51 ( humanActivity kid1 kidsRoom2 playing ) true )
( a52 ( humanActivity kid1 kitchen working ) true )
( a53 ( humanActivity kid1 study reading ) true )
( a54 ( humanActivity kid1 out working ) true )
( a55 ( humanActivity kid1 livingRoom eating ) true )
( a56 ( humanActivity kid1 livingRoom reading ) true )
( a57 ( humanActivity kid1 out out ) true )
( a58 ( humanActivity kid1 kidsRoom1 playing ) true )
( a59 ( humanActivity kid1 livingRoom reading ) true )
( a60 ( humanActivity kid1 kitchen eating ) true )
( a61 ( humanActivity kid1 diningRoom eating ) true )
( a62 ( humanActivity kid1 diningRoom eating ) true )
( a63 ( humanActivity kid1 out out ) true )
( a64 ( humanActivity kid1 kidsRoom2 sleeping ) true )
( a65 ( humanActivity kid2 kidsRoom1 idle ) true )
( a66 ( humanActivity kid2 bathroom4 usingBathroom ) true )
( a67 ( humanActivity kid2 out out ) true )
( a68 ( humanActivity kid2 study reading ) true )
( a69 ( humanActivity kid2 kidsRoom2 sleeping ) true )
( a70 ( humanActivity kid2 out out ) true )
( a71 ( humanActivity kid2 kidsRoom1 reading ) true )
( a72 ( humanActivity kid2 out out ) true )
( a73 ( humanActivity kid2 diningRoom reading ) true )
( a74 ( humanActivity kid2 livingRoom playing ) true )
( a75 ( humanActivity kid2 diningRoom eating ) true )
( a76 ( humanActivity kid2 livingRoom eating ) true )
( a77 ( humanActivity kid2 diningRoom eating ) true )
( a78 ( humanActivity kid2 livingRoom reading ) true )
( a79 ( humanActivity kid2 livingRoom reading ) true )
( a80 ( humanActivity kid2 out out ) true )
( a81 ( humanActivity kid2 out out ) true )
( a82 ( humanActivity kid2 out working ) true )
( a83 ( humanActivity kid2 storage working ) true )
( a84 ( humanActivity kid2 out out ) true )
```

```
( a85 ( humanActivity kid2 bathroom3 usingBathroom ) true )
( a86 ( humanActivity kid2 diningRoom playing ) true )
( a87 ( humanActivity kid2 livingRoom sleeping ) true )
( a88 ( humanActivity grandfather1 out out ) true )
( a89 ( humanActivity grandfather1 kitchen cooking ) true )
( a90 ( humanActivity grandfather1 bathroom1 usingBathroom ) true )
( a91 ( humanActivity grandfather1 livingRoom eating ) true )
( a92 ( humanActivity grandfather1 out playing ) true )
( a93 ( humanActivity grandfather1 kidsRoom2 playing ) true )
( a94 ( humanActivity grandfather1 bedroomGrandparents2 reading )
    true )
( a95 ( humanActivity grandfather1 kitchen cooking ) true )
( a96 ( humanActivity grandfather1 bedroomGrandparents2 sleeping )
    true )
( a97 ( humanActivity grandfather1 laundryRoom working ) true )
( a98 ( humanActivity grandfather1 bathroom3 usingBathroom ) true )
( a99 ( humanActivity grandfather1 kidsRoom1 idle ) true )
( a100 ( humanActivity grandfather1 floor2 idle ) true )
( a101 ( humanActivity grandfather1 kitchen reading ) true )
( a102 ( humanActivity grandfather1 storage working ) true )
( a103 ( humanActivity grandfather1 kitchen cooking ) true )
( a104 ( humanActivity grandfather1 out out ) true )
( a105 ( humanActivity grandfather1 out out ) true )
( a106 ( humanActivity grandfather1 bedroomGrandparents1 sleeping )
    true )
( a107 ( humanActivity grandfather1 out playing ) true )
( a108 ( humanActivity grandfather1 storage working ) true )
( a109 ( humanActivity grandfather1 kitchen cooking ) true )
( a110 ( humanActivity grandfather1 study reading ) true )
( a111 ( humanActivity grandfather1 bedroomGrandparents1 sleeping )
    true )
;; Initial locations of robots
( key1 ( at r1 ) robotRoom )
( key2 ( at r2 ) robotRoom )
( key3 ( at r3 ) robotRoom )
( key4 ( at r4 ) robotRoom )
;; Initial states for goals
( key5 ( state tCollectlaundry1 collectLaundry bathroom3 ) waiting )
( key6 ( state tSortstuff2 sortStuff bedroomParents ) waiting )
( key7 ( state tAssist3 assist storage ) waiting )
( key8 ( state tCollecttrash4 collectTrash kitchen ) waiting )
( key9 ( state tTakeouttrash5 takeOutTrash out ) waiting )
( key10 ( state tEntertain6 entertain livingRoom ) waiting )
( key11 ( state tVaccuum7 vaccuum kitchen ) waiting )
( key12 ( state tCollectlaundry8 collectLaundry bathroom3 )
    waiting )
( key13 ( state tEntertain9 entertain kidsRoom1 ) waiting )
```

```
  ( key14 ( state tTakeouttrash10 takeOutTrash out ) waiting )
  ;; Statements representing goal batches
  ( batchDiv1 ( goalBatchDivider 1 ) true )
  ( batchDiv2 ( goalBatchDivider 2 ) true ) )
(:goal
  ( ?G1 ( state tCollectlaundry1 collectLaundry bathroom3 ) finished )
  ( ?G2 ( state tSortstuff2 sortStuff bedroomParents ) finished )
  ( ?G3 ( state tAssist3 assist storage ) finished )
  ( ?G4 ( state tCollecttrash4 collectTrash kitchen ) finished )
  ( ?G5 ( state tTakeouttrash5 takeOutTrash out ) finished )
  ( ?G6 ( state tEntertain6 entertain livingRoom ) finished )
  ( ?G7 ( state tVaccuum7 vaccuum kitchen ) finished )
  ( ?G8 ( state tCollectlaundry8 collectLaundry bathroom3 ) finished )
  ( ?G9 ( state tEntertain9 entertain kidsRoom1 ) finished )
  ( ?G10 ( state tTakeouttrash10 takeOutTrash out ) finished ) )
(:temporal
  ( planning-interval [0 1440] )
  ;; Timing of activities
  ( at a1 [0 0] [59 59] ) ( at a2 [60 60] [106 106] )
  ( at a3 [107 107] [162 162] ) ( at a4 [163 163] [217 217] )
  ( at a5 [218 218] [249 249] ) ( at a6 [250 250] [294 294] )
  ( at a7 [295 295] [327 327] ) ( at a8 [328 328] [381 381] )
  ( at a9 [382 382] [432 432] ) ( at a10 [433 433] [468 468] )
  ( at a11 [469 469] [513 513] ) ( at a12 [514 514] [567 567] )
  ( at a13 [568 568] [623 623] ) ( at a14 [624 624] [674 674] )
  ( at a15 [675 675] [734 734] ) ( at a16 [735 735] [786 786] )
  ( at a17 [787 787] [838 838] ) ( at a18 [839 839] [872 872] )
  ( at a19 [873 873] [930 930] ) ( at a20 [931 931] [968 968] )
  ( at a21 [969 969] [1440 1440] )
  ( at a22 [0 0] [52 52] )
  ( at a23 [53 53] [98 98] ) ( at a24 [99 99] [159 159] )
  ( at a25 [160 160] [200 200] ) ( at a26 [201 201] [257 257] )
  ( at a27 [258 258] [306 306] ) ( at a28 [307 307] [367 367] )
  ( at a29 [368 368] [415 415] ) ( at a30 [416 416] [447 447] )
  ( at a31 [448 448] [479 479] ) ( at a32 [480 480] [538 538] )
  ( at a33 [539 539] [590 590] ) ( at a34 [591 591] [621 621] )
  ( at a35 [622 622] [661 661] ) ( at a36 [662 662] [704 704] )
  ( at a37 [705 705] [753 753] ) ( at a38 [754 754] [804 804] )
  ( at a39 [805 805] [856 856] ) ( at a40 [857 857] [917 917] )
  ( at a41 [918 918] [950 950] ) ( at a42 [951 951] [991 991] )
  ( at a43 [992 992] [1440 1440] )
  ( at a44 [0 0] [47 47] ) ( at a45 [48 48] [100 100] )
  ( at a46 [101 101] [136 136] ) ( at a47 [137 137] [193 193] )
  ( at a48 [194 194] [247 247] ) ( at a49 [248 248] [282 282] )
  ( at a50 [283 283] [332 332] ) ( at a51 [333 333] [371 371] )
  ( at a52 [372 372] [408 408] ) ( at a53 [409 409] [451 451] )
  ( at a54 [452 452] [509 509] ) ( at a55 [510 510] [559 559] )
```

```
( at a56 [560 560] [618 618] ) ( at a57 [619 619] [675 675] )
( at a58 [676 676] [717 717] ) ( at a59 [718 718] [775 775] )
( at a60 [776 776] [832 832] ) ( at a61 [833 833] [875 875] )
( at a62 [876 876] [925 925] ) ( at a63 [926 926] [978 978] )
( at a64 [979 979] [1440 1440] )
( at a65 [0 0] [48 48] ) ( at a66 [49 49] [84 84] )
( at a67 [85 85] [118 118] ) ( at a68 [119 119] [175 175] )
( at a69 [176 176] [218 218] ) ( at a70 [219 219] [263 263] )
( at a71 [264 264] [299 299] ) ( at a72 [300 300] [345 345] )
( at a73 [346 346] [406 406] ) ( at a74 [407 407] [452 452] )
( at a75 [453 453] [499 499] ) ( at a76 [500 500] [552 552] )
( at a77 [553 553] [584 584] ) ( at a78 [585 585] [627 627] )
( at a79 [628 628] [669 669] ) ( at a80 [670 670] [714 714] )
( at a81 [715 715] [775 775] ) ( at a82 [776 776] [810 810] )
( at a83 [811 811] [841 841] ) ( at a84 [842 842] [899 899] )
( at a85 [900 900] [935 935] ) ( at a86 [936 936] [984 984] )
( at a87 [985 985] [1440 1440] )
( at a88 [0 0] [32 32] ) ( at a89 [33 33] [66 66] )
( at a90 [67 67] [105 105] ) ( at a91 [106 106] [141 141] )
( at a92 [142 142] [179 179] ) ( at a93 [180 180] [211 211] )
( at a94 [212 212] [252 252] ) ( at a95 [253 253] [290 290] )
( at a96 [291 291] [334 334] ) ( at a97 [335 335] [375 375] )
( at a98 [376 376] [430 430] ) ( at a99 [431 431] [464 464] )
( at a100 [465 465] [521 521] ) ( at a101 [522 522] [558 558] )
( at a102 [559 559] [617 617] ) ( at a103 [618 618] [663 663] )
( at a104 [664 664] [699 699] ) ( at a105 [700 700] [739 739] )
( at a106 [740 740] [779 779] ) ( at a107 [780 780] [827 827] )
( at a108 [828 828] [871 871] ) ( at a109 [872 872] [912 912] )
( at a110 [913 913] [959 959] ) ( at a111 [960 960] [1440 1440] )
;; Initial state
( release key1 [0 0] ) ( release key2 [0 0] )
( release key3 [0 0] ) ( release key4 [0 0] )
( release key5 [0 0] ) ( release key6 [0 0] )
( release key7 [0 0] ) ( release key8 [0 0] )
( release key9 [0 0] ) ( release key10 [0 0] )
( release key11 [0 0] ) ( release key12 [0 0] )
( release key13 [0 0] ) ( release key14 [0 0] )
;; Dividing goal batches
( before ?G1 batchDiv1 [1 1440] )
( before ?G3 batchDiv1 [1 1440] )
( before ?G6 batchDiv1 [1 1440] )
( before ?G10 batchDiv1 [1 1440] )
( after ?G2 batchDiv1 [1 1440] )
( after ?G5 batchDiv1 [1 1440] )
( after ?G9 batchDiv1 [1 1440] )
( before ?G2 batchDiv2 [1 1440] )
( before ?G5 batchDiv2 [1 1440] )
```

```
    ( before ?G9 batchDiv2 [1 1440] )
    ( after ?G4 batchDiv2 [1 1440] )
    ( after ?G7 batchDiv2 [1 1440] )
    ( after ?G8 batchDiv2 [1 1440] ) )
  (:cost
    ;; Maximum allowed social cost
    ( less-than-or-equals socialCost 100 ) ) )
```

References

[1] *The 2014 International Planning Competition*, 2014. (Cited on pages 19 and 72.)

[2] A. Aiken, D. Kozen, M. Y. Vardi, and E. L. Wimmers. The complexity of set constraints. In *Proceedings of the 7th Workshop on Computer Science Logic (CSL)*, 1993. (Cited on page 68.)

[3] R. Alami, R. Chatila, A. Clodic, S. Fleury, M. Herrb, V. Montreuil, and E. A. Sisbot. Towards Human-Aware cognitive robots. In *Proceedings of the 5th International Cognitive Robotics Workshop (AAAI Workshop, CogRob)*, Boston, MA, USA, 2006. (Cited on pages 24 and 35.)

[4] R. Alami, A. Clodic, V. Montreuil, Sisbot, and R. Chatila. Toward Human-Aware robot task planning. In *AAAI Spring Symposium: To Boldly Go Where No Human-Robot Team Has Gone Before*, California, USA, 2006. (Cited on page 35.)

[5] J. Allen. Towards a general theory of action and time. *Artificial Intelligence*, 23:123–154, 1984. (Cited on page 60.)

[6] J. Allen and G. Ferguson. Human-machine collaborative planning. In *Proceedings of the 3rd International NASA Workshop on Planning and Scheduling for Space*, pages 27–29, 2002. (Cited on pages 14 and 39.)

[7] F. Bacchus and M. Ady. Planning with resources and concurrency a forward chaining approach. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI)*, 2001. (Cited on page 25.)

[8] F. Bacchus and F. Kabanza. Using temporal logic to control search in a forward chaining planner. In *Proceedings of the 3rd European Workshop on Planning*, pages 141–153. Press, 1995. (Cited on pages 19, 25, 37, and 84.)

[9] C. Bäckström. Computational aspects of reordering plans. *Journal of Artificial Intelligence Research*, 9:99–137, 1998. (Cited on page 19.)

[10] C. Bäckström and B. Nebel. Complexity results for SAS+ planning. *Computational Intelligence*, 11:625–655, 1995. (Cited on page 19.)

[11] J. A. Baier, F. Bacchus, and S. A. McIlraith. A heuristic search approach to planning with temporally extended preferences. *Artificial Intelligence*, 173:593 – 618, 2009. Advances in Automated Plan Generation. (Cited on page 41.)

[12] J. A. Baier and S. A. McIlraith. Planning with preferences. *AI Magazine*, 29(4):25–36, 2008. (Cited on page 41.)

[13] C. Barrett, R. Sebastiani, S. Seshia, and C. Tinelli. *Satisfiability modulo theories*, volume 185, chapter 26, pages 825–885. Frontiers in Artificial Intelligence and Applications, 2009. (Cited on pages 30 and 34.)

[14] R. Barták. On constraint models for parallel planning: The novel transition scheme. In *Proceedings of the 11th Scandinavian Conference on Artificial Intelligence (SCAI)*, pages 50–59, 2011. (Cited on page 20.)

[15] R. Barták and D. Toropila. Reformulating constraint models for classical planning. In *Proceedings of the 21st International Florida Artificial Intelligence Research Society (FLAIRS)*, pages 525–530, 2008. (Cited on page 20.)

[16] R. Barták and D. Toropila. Revisiting constraint models for planning problems. In *Foundations of Intelligent Systems*. Springer Berlin / Heidelberg, 2009. (Cited on page 20.)

[17] R. Barták and N.-F. Zhou. Using tabled logic programming to solve the petrobras planning problem. In *Proceedings of the 30th International Conference on Logic Programming (ICLP)*, 2014. (Cited on page 32.)

[18] J. Benton, A. J. Coles, and A. Coles. Temporal planning with preferences and time-dependent continuous costs. In *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS)*, 2012. (Cited on pages 26 and 107.)

[19] J. Bidot, L. Karlsson, F. Lagriffoul, and A. Saffiotti. Geometric backtracking for combined task and motion planning in robotic systems. *Artificial Intelligence*, 2015. (Online). (Cited on page 32.)

[20] M. Bienvenu, C. Fritz, and S. A. McIlraith. Planning with qualitative temporal preferences. In *Proceedings of the 10th International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 134–144, Lake District, UK, June 2–5 2006. (Cited on page 41.)

[21] M. Bienvenu, C. Fritz, and S. A. McIlraith. Specifying and computing preferred plans. *Artif. Intell.*, 175(7-8):1308–1345, 2011. (Cited on page 41.)

[22] A. Blum and J. C. Langford. Probabilistic planning in the graphplan framework. In *Proceedings of the 5th European Conference on Planning (ECP)*, pages 8–12, 1999. (Cited on page 28.)

[23] A. L. Blum and M. L. Furst. Fast Planning Through Planning Graph Analysis. *Artificial Intelligence*, 90(1):1636–1642, 1995. (Cited on page 21.)

[24] J. Blythe. Planning with external events. In *Proceedings of the 10th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 94–101. Morgan Kaufmann Publishers, 1994. (Cited on page 29.)

[25] C. Boutilier, T. Dean, and S. Hanks. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 11:1–94, 1999. (Cited on page 28.)

[26] J. M. Bradshaw, P. Beautement, M. R. Breedy, L. Bunch, S. V. Drakunov, P. Feltovich, R. R. Hoffman, R. Jeffers, M. Johnson, S. Kulkarni, J. Lott, A. K. Raj, N. Suri, and A. Uszok. *Making Agents Acceptable to People*, volume 2691 of *Lecture Notes in Computer Science*, chapter 12, pages 1068–1068. IOS Press, 2003. (Cited on page 38.)

[27] I. Bratko. *Prolog Programming for Artificial Intelligence*. Addison Wesley, September 2000. (Cited on pages 20, 61, and 64.)

[28] J. Bresina, A. Jónsson, P. Morris, and K. Rajan. Activity planning for the mars exploration rovers. In *Proceedings of the 15th International Conference on Automated Planning and Scheduling (ICAPS)*, 2005. (Cited on pages 27 and 31.)

[29] F. Broz, I. R. Nourbakhsh, and R. G. Simmons. Planning for human-robot interaction using time-state aggregated POMDPs. In *Proceedings of the 23rd AAAI Conference*, pages 1339–1344, 2008. (Cited on page 40.)

[30] C. Burghardt and T. Kirste. Inferring intentions in generic context-aware systems. In *Proceedings of the 6th International Conference on Mobile and Ubiquitous Multimedia (MUM)*, 2007. (Cited on page 39.)

[31] S. Cambon, R. Alami, and F. Gravot. A hybrid approach to intricate motion, manipulation and task planning. *International Journal of Robotics Research*, 28(1):104–126, Jan. 2009. (Cited on page 32.)

[32] F. M. Carlucci, L. Nardi, L. Iocchi, and D. Nardi. Explicit representation of social norms for social robots. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4191–4196. IEEE, 2015. (Cited on page 39.)

[33] L. A. Castillo, J. Fernández-Olivares, Ó. García-Pérez, and F. Palao. Temporal enhancements of an HTN planner. In *Proceedings of the 11th Conference of the Spanish Association for Artificial Intelligence (CAIPA)*, pages 429–438, 2005. (Cited on page 23.)

[34] L. A. Castillo, J. Fernández-Olivares, Ó. García-Pérez, and F. Palao. Efficiently handling temporal knowledge in an htn planner. In *In Proceedings of the 16th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 63–72. AAAI, 2006. (Cited on page 23.)

[35] A. Cesta, G. Cortellessa, R. Rasconi, F. Pecora, M. Scopelliti, and L. Tiberio. Monitoring elderly people with the robocare domestic environment: Interaction synthesis and user evaluation. *Computational Intelligence*, 27:60–82, 2011. (Cited on page 139.)

[36] A. Cesta, A. Oddi, and S. Smith. A constraint-based method for project scheduling with time windows. *Journal of Heuristics*, 8:109–136, 2002. (Cited on pages 27, 34, 48, 55, 60, 66, and 151.)

[37] T. Chakraborti, G. Briggs, K. Talamadupula, Y. Zhang, M. Scheutz, D. E. Smith, and S. Kambhampati. Planning for serendipity. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5300–5306, 2015. (Cited on page 36.)

[38] T. Chakraborti, K. Talamadupula, Y. Zhang, and S. Kambhampati. A formal framework for studying interaction in human-robot societies. In *Proceedings of the AAAI 2016 Workshop on Symbiotic Cognitive Systems*, 2016. (Cited on page 151.)

[39] T. Chakraborti, Y. Zhang, D. Smith, and S. Kambhampati. Planning with stochastic resource profiles: An application to human-robot cohabitation. In *Proceedings of the 3rd ICAPS Workshop on Planning and Robotics (PlanRob-15)*, 2015. (Cited on page 35.)

[40] T. Chakraborti, Y. Zhang, D. E. Smith, and S. Kambhampati. Planning with resource conflicts in human-robot cohabitation. In *Proceedings of the 15th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 1069–1077, 2016. (Cited on page 35.)

[41] Y. Chen, B. W. Wah, and C. wei Hsu. Temporal planning using subgoal partitioning and resolution in SGPlan. *Journal of Artificial Intelligence Research*, 26:323–369, 2006. (Cited on page 26.)

[42] S. Chien, R. Knight, A. Stechert, R. Sherwood, and G. Rabideau. Using iterative repair to improve responsiveness of planning and scheduling. In *Proceedings of the 5th International Conference on Artificial Intelligence Planning and Scheduling (AIPS)*, pages 300–307, 2000. (Cited on page 27.)

[43] S. A. Chien, D. Tran, G. Rabideau, S. R. Schaffer, D. Mandl, and S. Frye. Timeline-based space operations scheduling with external constraints. In *Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 34–41, 2010. (Cited on page 31.)

[44] K. Christoffersen and D. D. Woods. How to make automated systems team players. *Advances in Human Performance and Cognitive Engineering Research*, 2:1–12, 2002. (Cited on page 36.)

[45] A. Cimatti, A. Micheli, and M. Roveri. Solving temporal problems using smt: Strong controllability. In *Principles and Practice of Constraint Programming*. Springer Berlin Heidelberg, 2012. (Cited on page 131.)

[46] M. Cirillo. *Planning in Inhabited Environments : Human-Aware Task Planning and Activity Recognition*. PhD thesis, Örebro Universitet, 2010. (Cited on pages 13, 14, 25, and 37.)

[47] M. Cirillo, L. Karlsson, and A. Saffiotti. A framework for human-aware robot planning. In *Proceedings of the 10th Scandinavian Conference on Artificial Intelligence (SCAI)*, pages 52–59, Amsterdam, The Netherlands, The Netherlands, 2008. IOS Press. (Cited on page 37.)

[48] M. Cirillo, L. Karlsson, and A. Saffiotti. A human-aware robot task planner. In *Proceedings of the 11th International Conference on Automated Planning and Scheduling (ICAPS)*, 2009. (Cited on page 37.)

[49] M. Cirillo, L. Karlsson, and A. Saffiotti. Human-aware task planning for mobile robots. In *Proceedings of the International Conference on Advanced Robotics (ICAR)*, 2009. (Cited on page 37.)

[50] M. Cirillo, F. Pecora, H. Andreasson, T. Uras, and S. Koenig. Integrated motion planning and coordination for industrial vehicles. In *Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS)*, Portsmouth, USA, 2014. (Cited on page 34.)

[51] A. Clodic, V. Montreuil, R. Alami, and R. Chatila. A decisional framework for autonomous robots interacting with humans. In *IEEE International Workshop on Robot and Human Interactive Communication (ROMAN)*, pages 543 – 548, aug. 2005. (Cited on page 40.)

[52] A. Coles, A. J. Coles, A. Clark, and S. Gilmore. Cost-sensitive concurrent planning under duration uncertainty for service-level agreements. In F. Bacchus, C. Domshlak, S. Edelkamp, and M. Helmert, editors, *Proceedings of the 21st International Conference on Automated Planning and Scheduling (ICAPS)*. AAAI, 2011. (Cited on page 32.)

[53] A. Coles, M. Fox, K. Halsey, D. Long, and A. Smith. Managing concurrency in temporal planning using planner-scheduler interaction. *Artificial Intelligence*, 173(1):1–44, Jan. 2009. (Cited on page 26.)

[54] A. Coles, M. Fox, D. Long, and A. Smith. Planning with problems requiring temporal coordination. In *Proceedings of the 23rd AAAI Conference*, 2008. (Cited on pages 26 and 76.)

[55] A. J. Coles, A. Coles, M. Fox, and D. Long. Incremental constraint-posting algorithms in interleaved planning and scheduling. In *In Proceedings of COPLAS'2009: ICAPS Workshop on Constraint Satisfaction Techniques for Planning and Scheduling Problems*, 2009. (Cited on page 26.)

[56] A. J. Coles, A. Coles, M. Fox, and D. Long. Temporal planning in domains with linear processes. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1671–1676, 2009. (Cited on page 26.)

[57] A. J. Coles, A. Coles, M. Fox, and D. Long. Forward-chaining partial-order planning. In *Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 42–49, 2010. (Cited on page 26.)

[58] S. Coradeschi and A. Saffiotti. Symbiotic robotic systems: Humans, robots, and smart environments. *IEEE Intelligent Systems*, 21:82 –84, 2006. (Cited on pages 38 and 40.)

[59] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009. (Cited on page 92.)

[60] V. S. Costa, R. Rocha, and L. Damas. The YAP Prolog System. *Theory and Practice of Logic Programming*, 12(1-2):5–34, 2012. (Cited on pages 130 and 131.)

[61] K. Currie and A. Tate. O-Plan: The open planning architecture. *Artificial Intelligence*, 52(1):49–86, Nov. 1991. (Cited on page 25.)

[62] W. Cushing, S. Kambhampati, M. Mausam, and D. Weld. When is temporal planning really temporal? In *Proceedings of the 20th International*

Joint Conference on Artificial Intelligence (IJCAI), pages 1852–1859, 2007. (Cited on pages 25 and 26.)

[63] M. de la Asunción, L. Castillo, J. Fdez-Olivares, O. Garc'ıa-Pérez, A. González, and F. Palao. SIADEX: An interactive artificial intelligence planner for decision support and training in forest fire fighting. *Artificial Intelligence Communications*, 18(4):257–268, 2005. (Cited on page 23.)

[64] M. de Weerdt and B. Clement. Introduction to planning in multiagent systems. *Multiagent and Grid Systems*, 5(4):345–355, Dec. 2009. (Cited on page 40.)

[65] R. Dechter. *Constraint Processing*. Elsevier Morgan Kaufmann, 2003. (Cited on pages 53, 67, 91, 136, and 145.)

[66] R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *Artificial Intelligence*, 49(1-3):61–95, 1991. (Cited on pages 20, 48, and 61.)

[67] M. Di Rocco, F. Pecora, and A. Saffiotti. When robots are late: Configuration planning for multiple robots with dynamic goals. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 9515–5922. IEEE, 2013. (Cited on pages 33 and 34.)

[68] M. B. Do and S. Kambhampati. SAPA: A multi-objective metric temporal planner. *Journal of Artificial Intelligence Research*, 20:155–194, 2003. (Cited on pages 25 and 26.)

[69] P. Doherty and J. Kvarnström. TALplanner: An empirical investigation of a temporal logic-based forward chaining planner. In *Proceedings of the 6th International Workshop on the Temporal Representation and Reasoning (TIME'99)*., pages 47–54, Orlando, FL, 1999. (Cited on page 25.)

[70] P. Doherty, J. Kvarnström, and F. Heintz. A Temporal Logic-Based Planning and Execution Monitoring Framework for Unmanned Aircraft Systems. *Automated Agents and Multi-Agent Systems*, 2(2):332–377, 2010. (Cited on page 31.)

[71] C. Domshlak and Y. Dinitz. Multi-agent off-line coordination: Structure and complexity. In *Proceedings of the 6th European Conference on Planning (ECP)*, 2001. (Cited on page 40.)

[72] C. Dondrup, N. Bellotto, M. Hanheide, K. Eder, and U. Leonards. A computational model of human-robot spatial interactions based on a qualitative trajectory calculus. *Robotics*, 4(1):63, 2015. (Cited on pages 15 and 40.)

[73] C. Dondrup, N. Bellotto, F. Jovan, and M. Hanheide. Real-time multisensor people tracking for human-robot spatial interaction. In *Proceedings of the Workshop on Machine Learning for Social Robotics at the IEEE International Conference on Robotics and Automation (ICRA)*, 2015. (accepted). (Cited on page 40.)

[74] C. Dornhege, P. Eyerich, T. Keller, S. Trüg, M. Brenner, and B. Nebel. Semantic attachments for domain-independent planning systems. In *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 114–121, 2009. (Cited on page 30.)

[75] C. Dornhege, M. Gissler, M. Teschner, and B. Nebel. Integrating symbolic and geometric planning for mobile manipulation. In *Proceedings of the IEEE International Workshop on Safety, Security and Rescue Robotics (SSRR)*, 2009. (Cited on page 30.)

[76] B. Drabble and A. Tate. The use of optimistic and pessimistic resource profiles to inform search in an activity based planner. In K. J. Hammond, editor, *Proceedings of the 2nd Artificial Intelligence Planning Systems Conference (AIPS)*, pages 243–248. AAAI, 1994. (Cited on page 25.)

[77] F. Dvořák and R. Barták. Integrating time and resources into planning. In *Proceedings of the IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, ICTAI '10, pages 71–78, Washington, DC, USA, 2010. IEEE Computer Society. (Cited on page 27.)

[78] F. Dvořák, R. Barták, A. Bit-Monnot, F. Ingrand, and M. Ghallab. Planning and acting with temporal and hierarchical decomposition models. In *Proceedings of the IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 115–121, Nov 2014. (Cited on pages 24, 27, and 40.)

[79] S. Edelkamp and J. Hoffmann. PDDL2.2: The language for the Classical Part of the 4th International Planning Competition. Technical Report 195, Albert-Ludwigs-Universität Freiburg, Institut für Informatik, Jan. 2004. (Cited on pages 105 and 127.)

[80] E. A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science (Vol. B)*, chapter Temporal and Modal Logic, pages 995–1072. MIT Press, Cambridge, MA, USA, 1990. (Cited on page 25.)

[81] E. Erdem, E. Aker, and V. Patoglu. Answer set programming for collaborative housekeeping robotics: representation, reasoning, and execution. *Intelligent Service Robotics*, 5(4):275–291, Nov. 2012. (Cited on pages 21 and 32.)

[82] K. Erol, J. A. Hendler, and D. S. Nau. UMCP: A sound and complete procedure for hierarchical task-network planning. In *Proceedings of the 2nd Artificial Intelligence Planning Systems Conference (AIPS)*, pages 249–254, 1994. (Cited on page 23.)

[83] G. Ferguson, J. F. Allen, and B. W. Miller. TRAINS-95: Towards a mixed-initiative planning assistant. In B. Drabble, editor, *Proceedings of the 3rd Artificial Intelligence Planning Systems Conference (AIPS)*, pages 70–77. AAAI, 1996. (Cited on page 14.)

[84] R. E. Fikes and N. J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3-4):189–208, 1971. (Cited on page 18.)

[85] A. Finzi and A. Orlandini. Human-robot interaction through mixed-initiative planning for rescue and search rovers. In *Proceedings of the 9th conference on Advances in Artificial Intelligence (AI*IA)*, pages 483–494, 2005. (Cited on page 39.)

[86] R. W. Floyd. Algorithm 97: Shortest Path. *Communications of the ACM*, 5:345–345, June 1962. (Cited on page 61.)

[87] T. Fong, I. Nourbakhsh, and K. Dautenhahn. A survey of socially interactive robots. *Robotics and Autonomous Systems*, 42(3):143–166, 2003. (Cited on page 40.)

[88] M. Fox, A. Gerevini, D. Long, and I. Serina. Plan stability: Replanning versus plan repair. In *Proceedings of the 16th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 212–221, 2006. (Cited on pages 14 and 40.)

[89] M. Fox and D. Long. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *Journal of Artificial Intelligence Research*, 20:61–124, 2003. (Cited on pages 24 and 105.)

[90] M. Fox and D. Long. Modelling mixed discrete-continuous domains for planning. *Journal of Artificial Intelligence Research*, 27:235–297, 2006. (Cited on pages 105 and 107.)

[91] G. Francès and H. Geffner. Modeling and computation in planning: Better heuristics from more expressive languages. In *Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 70–78, 2015. (Cited on page 30.)

[92] J. Frank and A. Jónsson. Constraint-based attribute and interval planning. *Constraints*, 8:339–364, 2003. (Cited on pages 27 and 32.)

[93] S. Fratini, F. Pecora, and A. Cesta. Unifying planning and scheduling as timelines in a component-based perspective. *Archives of Control Sciences*, 18(2):231–271, 2008. (Cited on pages 27 and 95.)

[94] C. Freksa. Temporal Reasoning Based on Semi-Intervals. *Artificial Intelligence*, 54:199–227, 1992. (Cited on page 60.)

[95] A. Gabaldon. Activity recognition with intended actions. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI)*, 2009. (Cited on pages 14 and 39.)

[96] C. Galindo, J.-A. Fernandez-Madrigal, and J. Gonzalez. Multihierarchical interactive task planning: Application to mobile robotics. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 38:785 –798, 2008. (Cited on pages 14 and 39.)

[97] A. Garrido, E. Onaindia, and F. Barber. Time-optimal planning in temporal problems. In *Proceedings of the 6th European Conference on Planning (ECP)*, 2001. (Cited on page 26.)

[98] A. Gaschler, R. P. A. Petrick, M. Giuliani, M. Rickert, and A. Knoll. KVP: A knowledge of volumes approach to robot task planning. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 202–208. IEEE, 2013. (Cited on page 32.)

[99] H. Geffner. Functional STRIPS: a more flexible language for planning and problem solving. In J. Minker, editor, *Logic-based Artificial Intelligence*, chapter Functional strips: a more flexible language for planning and problem solving, pages 187–209. Kluwer Academic Publishers, Norwell, MA, USA, 2000. (Cited on page 30.)

[100] A. Gerevini and D. Long. Plan constraints and preferences in PDDL3. Technical report, Department of Electronics for Automation, University of Brescia, Ital, 2005. (Cited on pages 41, 105, and 112.)

[101] A. Gerevini, A. Saetti, and I. Serina. Integrating planning and temporal reasoning for domains with durations and time windows. In *Proceedings of the 19th international joint conference on Artificial intelligence (IJCAI)*, IJCAI'05, pages 1226–1231, San Francisco, CA, USA, 2005. Morgan Kaufmann Publishers Inc. (Cited on page 26.)

[102] A. Gerevini, A. Saetti, and I. Serina. An approach to temporal planning and scheduling in domains with predictable exogenous events. *Journal of Artificial Intelligence Research*, 25:187–231, 2006. (Cited on pages 26 and 29.)

[103] M. Ghallab, C. Isi, S. Penberthy, D. Smith, Y. Sun, and D. Weld. PDDL - The Planning Domain Definition Language. Technical report, CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, 1998. (Cited on pages 3 and 105.)

[104] M. Ghallab and H. Laruelle. Representation and control in IxTeT, a temporal planner. In *Proceedings of the 2nd Artificial Intelligence Planning Systems Conference (AIPS)*, pages 61–67, 1994. (Cited on page 25.)

[105] M. Ghallab, D. Nau, and P. Traverso. *Automated Planning: Theory and Practice*. Morgan Kaufmann, 2004. (Cited on pages 18, 19, 20, 22, 23, 24, 48, 87, and 91.)

[106] F. Graf, M. Hans, and R. Schraft. Mobile robot assistants: issues for dependable operation. *IEEE Robotics and Automation*, 11 (2):67–77, 2004. (Cited on page 40.)

[107] F. Gravot, S. Cambon, and R. Alami. aSyMov: A planner that deals with intricate symbolic and geometric problems. In P. Dario and R. Chatila, editors, *Proceedings of the 11th International Symposium on Robotics Research (ISRR)*, volume 15 of *Springer Tracts in Advanced Robotics*, pages 100–110. Springer, 2003. (Cited on page 32.)

[108] P. Gregory, D. Long, M. Fox, and J. Beck. Planning modulo theories: Extending the planning paradigm. In *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS)*, 2012. (Cited on pages 30 and 34.)

[109] K. Halsey, D. Long, and M. Fox. CRIKEY - A temporal planner looking at the integration of scheduling and planning. In *Proceedings of the ICAPS Workshop on Integrating Planning Into Scheduling (WIPIS)*, 2004. (Cited on page 26.)

[110] S. Hanks and D. S. Weld. A domain-independent algorithm for plan adaptation. *Journal of Artificial Intelligence Research*, 2:319–360, March 1995. (Cited on pages 14 and 40.)

[111] P. Haslum. Improving heuristics through relaxed search: an analysis of TP4 and HSPa* in the 2004 planning competition. *Journal of Artificial Intelligence Research*, 25:233–267, 2006. (Cited on page 26.)

[112] P. Haslum and H. Geffner. Heuristic planning with time and resources. In *Proceedings of the 6th European Conference on Planning (ECP)*, 2001. (Cited on pages 25 and 26.)

[113] M. Helmert. The fast downward planning system. *Journal of Artificial Intelligence Research*, 26(1):191–246, 2006. (Cited on pages 19, 36, 130, and 131.)

[114] L. M. Hiatt, A. M. Harrison, and J. G. Trafton. Accommodating human variability in Human-Robot teams through theory of mind. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI)*, 2011. (Cited on page 36.)

[115] J. Hoey and M. Grzés. Distributed control of situated assistance in large domains with many tasks. In *Proceedings of the 21st International Conference on Automated Planning and Scheduling (ICAPS)*, 2011. (Cited on page 36.)

[116] G. Hoffman and C. Breazeal. Cost-based anticipatory action selection for human-robot fluency. *IEEE Transactions on Robotics*, 23(5):952 –961, oct. 2007. (Cited on page 36.)

[117] G. Hoffman and C. Breazeal. Effects of anticipatory action on human-robot teamwork efficiency, fluency, and perception of team. In *Proceedings of the ACM/IEEE international Conference on Human-Robot Interaction*, HRI '07, pages 1–8, New York, NY, USA, 2007. ACM. (Cited on page 36.)

[118] J. Hoffmann. FF: The Fast-Forward Planning System. *AI Magazine*, 22:57–62, 2001. (Cited on page 19.)

[119] J. Hoffmann. Where ignoring delete lists works: Local search topology in planning benchmarks. *Journal of Artificial Intelligence Research*, 24(1):685–758, 2005. (Cited on page 19.)

[120] J. Hoffmann and B. Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14, 2001. (Cited on page 130.)

[121] J. Hoffmann, J. Porteous, and L. Sebastia. Ordered landmarks in planning. *Journal of Artificial Intelligence Research*, 22(1):215–278, Nov. 2004. (Cited on page 19.)

[122] C. W. Hsu, B. W. Wah, R. Huang, and Y. X. Chen. Handling soft constraints and preferences in SGPlan. In *ICAPS Workshop on Preferences and Soft Constraints in Planning*, 2006. (Cited on page 107.)

[123] H. Igreja, J. R. Silva, and F. Tonidandel. ICKEPS 2012 Challenge Domain: Planning Ship Operations on Petroleum Platforms and Ports, 2012. (Cited on page 32.)

[124] A. Jaimes and N. Sebe. Multimodal human-computer interaction: A survey. *Computer Vision and Image Understanding*, 108:116 – 134, 2007. (Cited on page 14.)

[125] O. C. Jenkins, G. González, and M. M. Loper. Tracking human motion and actions for interactive robots. In *Proceedings of the 2nd ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 365–372, 2007. (Cited on pages 14 and 39.)

[126] A. K. Jónsson, P. H. Morris, N. Muscettola, K. Rajan, and B. D. Smith. Planning in interplanetary space: Theory and practice. In *Proceedings of the 5th International Conference on Artificial Intelligence Planning Systems (AIPS)*, pages 177–186, 2000. (Cited on page 32.)

[127] E. Kamar and E. Horvitz. Collaboration and shared plans in the open world: studies of ridesharing. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI)*, 2009. (Cited on page 37.)

[128] S. Kambhampati. Planning graph as a (dynamic) CSP: exploiting EBL, DDB and other CSP search techniques in Graphplan. *Journal of Artificial Intelligence Research*, 12:1–34, February 2000. (Cited on page 20.)

[129] S. Kambhampati and B. Srivastava. *Universal classical planner: an algorithm for unifying state-space and plan-space planning*, pages 61–75. IOS Press, Amsterdam, The Netherlands, The Netherlands, 1996. (Cited on page 22.)

[130] S. Kambhampati and X. Yang. On the role of disjunctive representations and constraint propagation in refinement planning. In *Proceedings of the 5th International Conference on the Principles of Knowledge Representation and Reasoning (KR)*, pages 135–146, 1996. (Cited on page 20.)

[131] L. Karlsson. Conditional progressive planning under uncertainty. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 431–438, 2001. (Cited on pages 25, 28, 37, 84, and 152.)

[132] H. Kautz. *A formal theory of plan recognition*. PhD thesis, Bell Laboratories, 1987. (Cited on page 39.)

[133] H. Kautz and J. Allen. Generalized plan recognition. In *Proceedings of the 5th National Conference on Artificial intelligence (AAAI)*, 1986. (Cited on page 39.)

[134] H. A. Kautz, D. McAllester, and B. Selman. Encoding plans in propositional logic. In *Proceedings of the 5th International Conference on the Principles of Knowledge Representation and Reasoning (KR)*, 1996. (Cited on page 20.)

[135] H. A. Kautz and B. Selman. Planning as satisfiability. In *Proceedings of the 10th European Conference on Artificial Intelligence (ECAI)*, 1992. (Cited on page 20.)

[136] H. A. Kautz and B. Selman. Pushing the envelope: Planning, propositional logic and stochastic search. In *Proceedings of the 13th National Conference on Artificial intelligence (AAAI)*, 1996. (Cited on page 20.)

[137] H. A. Kautz and B. Selman. Unifying SAT-based and graph-based planning. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI)*, 1999. (Cited on page 20.)

[138] H. A. Kautz, B. Selman, and J. Hoffmann. SatPlan: Planning as satisfiability. In *In Proceedings of the 5th International Planning Competition (IPC)*, 2006. (Cited on page 20.)

[139] B. R. Kavuluri, N. B. Saladi, and D. Khemani. Planning for PDDL3 - an OCSP based approach. In *Proceedings of the 16th International Conference on Automated Planning and Scheduling (ICAPS)*, 2006. (Cited on page 21.)

[140] A. A. Khaliq, F. Pecora, and A. Saffiotti. Inexpensive, reliable and localization-free navigation using an rfid floor. In *Proceedings of the European Conference on Mobile Robots (ECMR)*, 2015. (Cited on page 138.)

[141] R. Kirby, R. Simmons , and J. Forlizzi. COMPANION: A constraint optimizing method for person-acceptable navigation. In *IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, pages 607–612, September 2009. (Cited on pages 15 and 40.)

[142] A. Kirsch, T. Kruse, E. A. Sisbot, R. Alami, M. Lawitzky, D. Brščić, S. Hirche, P. Basili, and S. Glasauer. Plan-based control of joint human-robot activities. *Künstliche Intelligenz*, 24(3):223–231, 2010. (Cited on page 37.)

[143] R. Kirsch and T. Kruse. An integrated planning and learning framework for human-robot interaction. In *In Proc. of the 4th Workshop on Planning and Plan Execution for Real-World Systems (at ICAPS)*, 2009. (Cited on page 14.)

[144] U. Köckemann, F. Pecora, and L. Karlsson. Grandpa Hates Robots — Interaction Constraints for Planning in Inhabited Environments. In *Proceedings of the 28th AAAI Conference*, 2014. (Cited on pages 82, 83, and 129.)

[145] U. Köckemann, F. Pecora, and L. Karlsson. Inferring context and goals for online human-aware planning. In *Proceedings of the IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, 2015. (Cited on pages 83 and 129.)

[146] J. Koehler. Planning under resource constraints. In *Proceedings of the 13th European Conference on Artificial Intelligence (ECAI)*, pages 489–493, 1998. (Cited on page 26.)

[147] T. Kruse and A. Kirsch. Towards opportunistic action selection in human-robot cooperation. In *Proceedings of the 33rd Annual German Conference on AI (KI)*, pages 374–381, 2010. (Cited on page 38.)

[148] T. Kruse, A. Kirsch, E. A. Sisbot, and R. Alami. Dynamic generation and execution of human aware navigation plans. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2010. (Cited on page 40.)

[149] T. Kruse, A. Kirsch, E. A. Sisbot, and R. Alami. Exploiting human cooperation in human-centered robot navigation. In *Proceedings of the IEEE International Symposium in Robot and Human Interactive Communication (Ro-Man)*, 2010. (Cited on page 40.)

[150] J. Kvarnström, P. Doherty, and P. Haslum. Extending TALplanner with concurrency and resources. In *Proceedings of the 14th European Conference on Artificial Intelligence (ECAI)*, pages 501–505, 2000. (Cited on page 25.)

[151] P. Laborie and M. Ghallab. IxTeT: an integrated approach for plan generation and scheduling. In *Proceedings of the Symposium on Emerging Technologies and Factory Automation*, pages 485 – 495, 1995. (Cited on pages 25 and 46.)

[152] P. Laborie and M. Ghallab. Planning with sharable resource constraints. In *Proceedings of the 14th International Joint Conference on Artificial intelligence (IJCAI)*, pages 1643–1649, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc. (Cited on page 25.)

[153] F. Lagriffoul, D. Dimitrov, J. Bidot, A. Saffiotti, and L. Karlsson. Efficiently combining task and motion planning using geometric constraints. *International Journal of Robotics Research*, 33(14):1726–1747, 2014. (Cited on page 32.)

[154] F. Lagriffoul, D. Dimitrov, A. Saffiotti, and L. Karlsson. Constraint propagation on interval bounds for dealing with geometric backtracking. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 957–964, Oct 2012. (Cited on page 32.)

[155] R. Lallement, L. de Silva, and R. Alami. HATP: An HTN Planner for Robotics. In *Proceedings of the 2nd ICAPS Workshop on Planning and Robotics (PlanRob)*, 2014. (Cited on pages 24 and 35.)

[156] S. Lemai and F. Ingrand. Interleaving temporal planning and execution in robotics domains. In *Proceedings of the 19th National Conference on Artificial intelligence (AAAI)*, AAAI'04, pages 617–622. AAAI Press, 2004. (Cited on page 31.)

[157] N. Lesh, C. Rich, and C. Sidner. Using plan recognition in human-computer collaboration. In *Proceedings of 7th International Conference on User Modeling*, 1999. (Cited on page 39.)

[158] S. J. Levine and B. C. Williams. Concurrent plan recognition and execution for human-robot teams. In *Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS)*, 2014. (Cited on page 38.)

[159] V. Lifschitz. Answer set programming and plan generation. *Artificial Intelligence*, 138(1-2):39–54, 2002. (Cited on page 20.)

[160] J. Löhr, M. Wehrle, M. Fox, and B. Nebel. Symbolic domain predictive control. In *Proceedings of the 28th AAAI Conference*, pages 2315–2321, 2014. (Cited on page 30.)

[161] D. Long and M. Fox. Exploiting a graphplan framework in temporal planning. In *Proceedings of the 13th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 52–61, 2003. (Cited on page 26.)

[162] A. Lopez and F. Bacchus. Generalizing GraphPlan by formulating planning as a CSP. In *In Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 954–960. Morgan Kaufmann Publishers, 2003. (Cited on page 20.)

[163] S. Magrelli and F. Pecora. A constraint based approach to integrated planning and scheduling. In *Proceedings of the 18th Workshop of the UK Planning and Scheduling SIG*, 2010. (Cited on page 27.)

[164] M. Mansouri and F. Pecora. More knowledge on the table: Planning with space, time and resources for robots. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2014. (Cited on pages 10, 27, 33, 34, and 55.)

[165] F. Maris and P. Regnier. TLP-GP: New results on temporally-expressive planning benchmarks. In *Proceedings of 20th IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 507–514, 2008. (Cited on page 26.)

[166] K. Marriott, N. Nethercote, R. Rafeh, P. J. Stuckey, M. G. de la Banda, and M. Wallace. The design of the zinc modelling language. *Constraints*, 13(3):229–267, 2008. (Cited on pages 101 and 152.)

[167] D. McAllester and D. Rosenblitt. Systematic nonlinear planning. In *Proceedings of the 9th National Conference on Artificial intelligence (AAAI)*, pages 634–639, 1991. (Cited on pages 22 and 55.)

[168] C. McGann, F. Py, K. Rajan, J. Ryan, and R. Henthorn. Adaptive control for autonomous underwater vehicles. In *Proceedings of the 23th AAAI Conference*, 2008. (Cited on pages 27 and 31.)

[169] C. McGann, F. Py, K. Rajan, H. Thomas, R. Henthorn, and R. S. McEwen. A deliberative architecture for AUV control. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 1049–1054. IEEE, 2008. (Cited on pages 27 and 31.)

[170] I. Meiri. Combining qualitative and quantitative constraints in temporal reasoning. *Artificial Intelligence*, 87:260–267, 1996. (Cited on page 60.)

[171] M. D. Moffitt and M. E. Pollack. Optimal rectangle packing: A meta-CSP approach. In *Proceedings of the 16th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 93–102, 2006. (Cited on page 34.)

[172] V. Montreuil, A. Clodic, M. Ransan, and R. Alami. Planning human centered robot activities. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, pages 2618–2623, Oct 2007. (Cited on pages 10 and 35.)

[173] P. Morris, N. Muscettola, and T. Vidal. Dynamic control of plans with temporal uncertainty. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 494–502, 2001. (Cited on page 29.)

[174] L. Mösenlechner and M. Beetz. Fast temporal projection using accurate physics-based geometric reasoning. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2013. (Cited on page 34.)

[175] A. Müller, A. Kirsch, and M. Beetz. Transformational planning for everyday activity. In *Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 248–255, Providence, USA, September 2007. (Cited on page 37.)

[176] V. Narayanan, Y. Zhang, N. Mendoza, and S. Kambhampati. Plan or not: Remote human-robot teaming with incomplete task information. *Computing Research Repository (CoRR)*, abs/1412.2824, 2014. (Cited on page 35.)

[177] V. Narayanan, Y. Zhang, N. Mendoza, and S. Kambhampati. Automated planning for peer-to-peer teaming and its evaluation in remote human-robot interaction. In *Proceedings of the 10th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 161–162, 2015. (Cited on page 35.)

[178] D. Nau, T.-C. Au, O. Ilghami, U. Kuter, D. Wu, F. Yaman, H. Munoz-Avila, and J. Murdock. Applications of SHOP and SHOP2. *IEEE Intelligent Systems*, 20(2):34 – 41, march-april 2005. (Cited on pages 31 and 42.)

[179] D. S. Nau, T.-C. Au, O. Ilghami, U. Kuter, J. W. Murdock, D. Wu, and F. Yaman. SHOP2: An HTN planning system. *Journal of Artificial Intelligence Research*, 20:379–404, 2003. (Cited on pages 23, 24, 35, and 84.)

[180] D. S. Nau, Y. Cao, A. Lotem, and H. Muñoz-avila. SHOP: Simple hierarchical ordered planner. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 968–975, 1999. (Cited on page 23.)

[181] N. Nethercote, P. J. Stuckey, R. Becket, S. Brand, G. J. Duck, and G. Tack. *Principles and Practice of Constraint Programming – CP 2007: 13th International Conference, CP 2007, Providence, RI, USA, September 23-27, 2007. Proceedings*, chapter MiniZinc: Towards a Standard CP Modelling Language, pages 529–543. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007. (Cited on pages 101 and 152.)

[182] X. Nguyen and S. Kambhampati. Reviving partial order planning. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI)*, 2001. (Cited on page 22.)

[183] A. Pandey, M. Ali, and R. Alami. Towards a task-aware proactive sociable robot based on multi-state perspective-taking. *International Journal of Social Robotics*, 5(2):215–236, 2013. (Cited on pages 36 and 40.)

[184] F. Pecora and M. Cirillo. A constraint-based approach for multiple non-holonomic vehicle coordination in industrial scenarios. In *Proceedings of TAMPRA 2012: ICAPS Workshop on Combining Task and Motion Planning for Real-World Applications (TAMPRA)*, pages 45–52, 2012. (Cited on page 95.)

[185] F. Pecora, M. Cirillo, F. Dell'Osa, J. Ullberg, and A. Saffiotti. A constraint-based approach for proactive, context-aware human support. *Journal of Ambient Intelligence and Smart Environments*, 4(4):347–367, 2012. (Cited on pages 13, 14, 34, 37, and 83.)

[186] S. J. Penberthy and D. S. Weld. UCPOP: A sound, complete, partial order planner for ADL. In *Proceedings of the 3rd International Conference on the Principles of Knowledge Representation and Reasoning (KR)*, 1992. (Cited on page 22.)

[187] L. R. Planken. *Algorithms for Simple Temporal Reasoning*. PhD thesis, Delft University of Technology, 2014. (Cited on pages 25 and 61.)

[188] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. B. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. ROS: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009. (Cited on page 95.)

[189] E. Rabenau, A. Donati, M. Denis, N. Policella, J. Schulster, A. Cesta, Cortellessa, S. G., Fratini, and A. Oddi. The RAXEM Tool on Mars Express - Uplink Planning Optimisation and Scheduling Using AI Constraint Resolution. In *Proceedings of the 10th International Conference on Space Operations (SpaceOps-08)*, 2008. (Cited on page 32.)

[190] K. Rajan and F. Py. T-REX: Partitioned Inference for AUV Mission Control. In G. N. Roberts and R. Sutton, editors, *Further Advances in Unmanned Marine Vehicles*. The Institution of Engineering and Technology (IET), 2012. (Cited on pages 95 and 98.)

[191] K. Rajan, F. Py, and J. Barreiro. Towards Deliberative Control in Marine Robotics. In M. Seto, editor, *Marine Robot Autonomy*. Springer Verlag, December 2012. (Cited on page 98.)

[192] M. Ramírez and H. Geffner. Plan recognition as planning. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI)*, IJCAI'09, pages 1778–1783, San Francisco, CA, USA, 2009. Morgan Kaufmann Publishers Inc. (Cited on pages 14 and 40.)

[193] M. Ramírez and H. Geffner. Probabilistic plan recognition using off-the-shelf classical planners. In *Proceedings of the 24th AAAI Conference*, 2010. (Cited on pages 35 and 40.)

[194] S. Reddy, M. Iatauro, E. Kürklü, M. Boyce, J. Frank, and A. Jónsson. Planning and monitoring solar array operations on the iss. In *Proceedings of the Scheduling and Planning Applications Workshop (SPARK)*, 2008. (Cited on page 32.)

[195] S. Richter and M. Westphal. The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research*, 39:127–177, 2010. (Cited on page 19.)

[196] D. Roggen, G. Troster, P. Lukowicz, A. Ferscha, J. del R.Millan, and R. Chavarriaga. Opportunistic human activity and context recognition. *IEEE Computer*, 46(2):36–45, Feb 2013. (Cited on page 39.)

[197] S. Rosenthal, J. Biswas, and M. M. Veloso. An effective personal mobile robot agent through symbiotic human-robot interaction. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 915–922, 2010. (Cited on pages 12 and 38.)

[198] S. Rosenthal and M. M. Veloso. Mobile robot planning to seek help with spatially-situated tasks. In *Proceedings of the 26th AAAI Conference*, 2012. (Cited on page 38.)

[199] S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach (International Edition)*. Pearson US Imports & PHIPEs, November 2002. (Cited on pages 28 and 30.)

[200] D. Sanchez, M. Tentori, and J. Favela. Hidden markov models for activity recognition in ambient intelligence environments. In *Proceedings of the 8th Mexican International Conference on Current Trends in Computer Science (ENC)*, pages 33 –40, sept. 2007. (Cited on pages 14 and 39.)

[201] B. Schattenberg. *Hybrid Planning And Scheduling*. PhD thesis, Ulm University, Institute of Artificial Intelligence, 2009. (Cited on page 27.)

[202] B. Schattenberg, J. Bidot, S. Geßler, and S. Biundo. A framework for interactive hybrid planning. In *Proceedings of the 32th Annual German Conference on AI (KI)*, 2009. (Cited on pages 14 and 39.)

[203] C. Schmidt, N. Sridharan, and J. Goodson. The plan recognition problem: an intersection of psychology and artificial intelligence. *Artificial Intelligence*, 11(1-2):45–83, 1978. (Cited on pages 14 and 39.)

[204] E. Sisbot, R. Alami, T. Simeon, K. Dautenhahn, M. Walters, and S. Woods. Navigation in the presence of humans. In *In Proceedings of the 5th IEEE-RAS International Conference on Humanoid Robots*, pages 181 –188, dec. 2005. (Cited on page 40.)

[205] E. Sisbot, L. Marin, and R. Alami. Spatial reasoning for human robot interaction. In *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*, pages 2281 –2287, 29 2007-nov. 2 2007. (Cited on pages 15 and 40.)

[206] E. A. Sisbot, A. Clodic, M. Marin, M. Fontmarty, L. Brethes, and R. Alami. Implementing a Human-Aware robot system. In *Proceedings of the 15th IEEE International Symposium on Robot and Human Interactive Communication (ROMAN)*, pages 727–732, Sept. 2006. (Cited on page 40.)

[207] D. E. Smith. Temporal planning with mutual exclusion reasoning. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 326–337, 1999. (Cited on page 26.)

[208] D. E. Smith and D. S. Weld. Conformant graphplan. In *Proceedings of the 16th National Conference on Artificial intelligence (AAAI)*, 1998. (Cited on pages 28 and 152.)

[209] B. Srivastava and S. Kambhampati. Efficient planning through separate resource scheduling. In *Proceedings of the AAAI Spring Symposium on Search Strategy Under Uncertainty and Incomplete Information*. AAAI Press, 1999. (Cited on page 26.)

[210] B. Srivastava and S. Kambhampati. Scaling up Planning by Teasing out Resource Scheduling. In *Proceedings of the 5th European Conference on Planning: Recent Advances in AI Planning*, pages 172–186, London, UK, 1999. Springer-Verlag. (Cited on page 26.)

[211] K. Stergiou and M. Koubarakis. Backtracking algorithms for disjunctions of temporal constraints. *Artificial Intelligence*, 120:81 – 117, 2000. (Cited on page 29.)

[212] S. Stock, M. Mansouri, F. Pecora, and J. Hertzberg. Hierarchical hybrid planning in a mobile service robot. In *Proceedings of the 38th Annual German Conference on AI*, pages 309–315. Springer International Publishing, 2015. (Cited on page 24.)

[213] K. Talamadupula, G. Briggs, T. Chakraborti, M. Scheutz, and S. Kambhampati. Coordination in human-robot teams using mental modeling and plan recognition. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2957–2962, Sept 2014. (Cited on page 35.)

[214] A. Tate, B. Drabble, and R. Kirby. *O-Plan2: An Open Architecture for Command, Planning and Control*, chapter 7, pages 213–239. Morgan Kaufmann, 1994. (Cited on page 25.)

[215] S. Thiébaux, J. Hoffmann, and B. Nebel. In defense of PDDL axioms. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 961–966. Morgan Kaufmann, 2003. (Cited on pages 30 and 106.)

[216] K. Tierney, A. J. Coles, A. Coles, C. Kroer, A. M. Britt, and R. M. Jensen. Automated planning for liner shipping fleet repositioning. In L. McCluskey, B. Williams, J. R. Silva, and B. Bonet, editors, *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS)*. AAAI, 2012. (Cited on page 32.)

[217] G. D. Tipaldi and K. O. Arras. I want my coffee hot! learning to find people under spatio-temporal constraints. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 1217–1222, 2011. (Cited on page 37.)

[218] G. D. Tipaldi and K. O. Arras. Planning problems for social robots. In *Proceedings of the 21st International Conference on Automated Planning and Scheduling (ICAPS)*, 2011. (Cited on page 37.)

[219] G. D. Tipaldi and K. O. Arras. Please do not disturb! Minimum interference coverage for social robots. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1968–1973, 2011. (Cited on page 37.)

[220] S. Tomic, F. Pecora, and A. Saffiotti. Too cool for school adding social constraints in human aware planning. In *Proceedings of the 9th International Cognitive Robotics Workshop (ECAI Workshop, CogRob)*, 2014. (Cited on page 37.)

[221] D. Toropila, F. Dvořak, O. Trunda, M. Hanes, and R. Barták. Three approaches to solve the petrobras challenge: Exploiting planning techniques for solving real-life logistics problems. In *Proceedings of the 24th IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, volume 1, pages 191–198, Nov 2012. (Cited on page 32.)

[222] I. Tsamardinos and M. E. Pollack. Efficient solution techniques for disjunctive temporal reasoning problems. *Artificial Intelligence*, 151:43 – 89, 2003. (Cited on page 29.)

[223] I. Tsamardinos, T. Vidal, and M. E. Pollack. CTP: A new constraint-based formalism for conditional, temporal planning. *Constraints*, 8(4):365–388, 2003. (Cited on pages 21 and 26.)

[224] J. Ullberg, A. Loutfi, and F. Pecora. Towards continuous activity monitoring with temporal constraints. In *Proceedings of the 4th Workshop on Planning and Plan Execution for Real-World Systems at ICAPS 2009*, 2009. (Cited on pages 14 and 39.)

[225] P. van Beek and X. Chen. CPlan: A constraint programming approach to planning. In *Proceedings of the 16th National Conference on Artificial intelligence (AAAI)*, AAAI '99/IAAI '99, pages 585–590, Menlo Park,

CA, USA, 1999. American Association for Artificial Intelligence. (Cited on page 20.)

[226] T. S. Vaquero, G. Costa, F. Tonidandel, H. Igreja, J. R. Silva, and C. Beck. Planning and scheduling ship operations on petroleum ports and platforms. In *Proceedings of SPARK 2012: ICAPS Workshop on Scheduling and Planning Applications*, pages 8–16, 2012. (Cited on page 32.)

[227] D. Vasquez, B. Okal, and K. O. Arras. Inverse reinforcement learning algorithms and features for robot navigation in crowds: An experimental comparison. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1341–1346, 2014. (Cited on page 37.)

[228] T. Vidal. Controllability characterization and checking in contingent temporal constraint networks. In A. G. Cohn, F. Giunchiglia, and B. Selman, editors, *Proceedings of the 7th International Conference on the Principles of Knowledge Representation and Reasoning (KR)*, pages 559–570. Morgan Kaufmann, 2000. (Cited on pages 29 and 122.)

[229] T. Vidal and H. Fargier. Handling contingency in temporal constraint networks: from consistency to controllabilities. *Journal of Experimental and Theoretical Artificial Intelligence*, 11:23–45, 1999. (Cited on pages 29 and 131.)

[230] V. Vidal. YAHSP2: Keep It Simple, Stupid. In *Booklet of the 7th International Planning Competition (IPC)*, 2011. (Cited on pages 19 and 130.)

[231] M. Vilain and H. Kautz. Constraint propagation algorithms for temporal reasoning. In *Proceedings of the 5th National Conference on Artificial intelligence (AAAI)*, 1986. (Cited on page 25.)

[232] D. H. Warren. WARPLAN: A system for generating plans. Technical Report 76, Dept. of Computational Logic, University of Edinburgh School of Artificial Intelligence, June 1974. (Cited on page 20.)

[233] D. S. Weld. An introduction to least commitment planning. *AI Magazine*, 15(4):27–61, 1994. (Cited on page 22.)

[234] D. S. Weld, C. R. Anderson, and D. E. Smith. Extending graphplan to handle uncertainty and sensing actions. In *Proceedings of the 15th National Conference on Artificial intelligence (AAAI)*, AAAI '98/IAAI '98, pages 897–904, Menlo Park, CA, USA, 1998. American Association for Artificial Intelligence. (Cited on pages 28 and 152.)

[235] S. A. Wolfman and D. S. Weld. The LPSAT engine & its application to resource planning. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI)*, 1999. (Cited on page 34.)

[236] F. Wu, S. Zilberstein, and X. Chen. Online planning for ad hoc autonomous agent teams. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI)*, pages 439–445, 2011. (Cited on page 38.)

[237] N. Yorke-Smith. Exploiting the structure of hierarchical plans in temporal constraint propagation. In *Proceedings of the 20th National Conference on Artificial intelligence (AAAI)*, AAAI'05, pages 1223–1228. AAAI Press, 2005. (Cited on page 24.)

[238] H. L. S. Younes and R. G. Simmons. VHPOP: Versatile heuristic partial order planner. *Journal of Artificial Intelligence Research*, 20:405–430, 2003. (Cited on pages 22 and 72.)

[239] Y. Zhang, S. Sreedharan, and S. Kambhampati. Capability models and their applications in planning. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 1151–1159, 2015. (Cited on page 36.)

Publications *in the series*
Örebro Studies in Technology

1. Bergsten, Pontus (2001) *Observers and Controllers for Takagi – Sugeno Fuzzy Systems*. Doctoral Dissertation.
2. Iliev, Boyko (2002) *Minimum-time Sliding Mode Control of Robot Manipulators*. Licentiate Thesis.
3. Spännar, Jan (2002) *Grey box modelling for temperature estimation*. Licentiate Thesis.
4. Persson, Martin (2002) *A simulation environment for visual servoing*. Licentiate Thesis.
5. Boustedt, Katarina (2002) *Flip Chip for High Volume and Low Cost – Materials and Production Technology*. Licentiate Thesis.
6. Biel, Lena (2002) *Modeling of Perceptual Systems – A Sensor Fusion Model with Active Perception*. Licentiate Thesis.
7. Otterskog, Magnus (2002) *Produktionstest av mobiltelefonantenner i mod-växlande kammare*. Licentiate Thesis.
8. Tolt, Gustav (2003) *Fuzzy-Similarity-Based Low-level Image Processing*. Licentiate Thesis.
9. Loutfi, Amy (2003) *Communicating Perceptions: Grounding Symbols to Artificial Olfactory Signals*. Licentiate Thesis.
10. Iliev, Boyko (2004) *Minimum-time Sliding Mode Control of Robot Manipulators*. Doctoral Dissertation.
11. Pettersson, Ola (2004) *Model-Free Execution Monitoring in Behavior-Based Mobile Robotics*. Doctoral Dissertation.
12. Överstam, Henrik (2004) *The Interdependence of Plastic Behaviour and Final Properties of Steel Wire, Analysed by the Finite Element Metod*. Doctoral Dissertation.
13. Jennergren, Lars (2004) *Flexible Assembly of Ready-to-eat Meals*. Licentiate Thesis.
14. Jun, Li (2004) *Towards Online Learning of Reactive Behaviors in Mobile Robotics*. Licentiate Thesis.
15. Lindquist, Malin (2004) *Electronic Tongue for Water Quality Assessment*. Licentiate Thesis.
16. Wasik, Zbigniew (2005) *A Behavior-Based Control System for Mobile Manipulation*. Doctoral Dissertation.

17. Berntsson, Tomas (2005) *Replacement of Lead Baths with Environment Friendly Alternative Heat Treatment Processes in Steel Wire Production*. Licentiate Thesis.

18. Tolt, Gustav (2005) *Fuzzy Similarity-based Image Processing*. Doctoral Dissertation.

19. Munkevik, Per (2005) *"Artificial sensory evaluation – appearance-based analysis of ready meals"*. Licentiate Thesis.

20. Buschka, Pär (2005) *An Investigation of Hybrid Maps for Mobile Robots*. Doctoral Dissertation.

21. Loutfi, Amy (2006) *Odour Recognition using Electronic Noses in Robotic and Intelligent Systems*. Doctoral Dissertation.

22. Gillström, Peter (2006) *Alternatives to Pickling; Preparation of Carbon and Low Alloyed Steel Wire Rod*. Doctoral Dissertation.

23. Li, Jun (2006) *Learning Reactive Behaviors with Constructive Neural Networks in Mobile Robotics*. Doctoral Dissertation.

24. Otterskog, Magnus (2006) *Propagation Environment Modeling Using Scattered Field Chamber.* Doctoral Dissertation.

25. Lindquist, Malin (2007) *Electronic Tongue for Water Quality Assessment*. Doctoral Dissertation.

26. Cielniak, Grzegorz (2007) *People Tracking by Mobile Robots using Thermal and Colour Vision*. Doctoral Dissertation.

27. Boustedt, Katarina (2007) *Flip Chip for High Frequency Applications – Materials Aspects*. Doctoral Dissertation.

28. Soron, Mikael (2007) *Robot System for Flexible 3D Friction Stir Welding*. Doctoral Dissertation.

29. Larsson, Sören (2008) *An industrial robot as carrier of a laser profile scanner. – Motion control, data capturing and path planning.* Doctoral Dissertation.

30. Persson, Martin (2008) *Semantic Mapping Using Virtual Sensors and Fusion of Aerial Images with Sensor Data from a Ground Vehicle*. Doctoral Dissertation.

31. Andreasson, Henrik (2008) *Local Visual Feature based Localisation and Mapping by Mobile Robots*. Doctoral Dissertation.

32. Bouguerra, Abdelbaki (2008) *Robust Execution of Robot Task-Plans: A Knowledge-based Approach*. Doctoral Dissertation.

33. Lundh, Robert (2009) *Robots that Help Each Other: Self-Configuration of Distributed Robot Systems*. Doctoral Dissertation.

34. Skoglund, Alexander (2009) *Programming by Demonstration of Robot Manipulators*. Doctoral Dissertation.

35. Ranjbar, Parivash (2009) *Sensing the Environment: Development of Monitoring Aids for Persons with Profound Deafness or Deafblindness*. Doctoral Dissertation.

36. Magnusson, Martin (2009) *The Three-Dimensional Normal-Distributions Transform – an Efficient Representation for Registration, Surface Analysis, and Loop Detection*. Doctoral Dissertation.

37. Rahayem, Mohamed (2010) *Segmentation and fitting for Geometric Reverse Engineering. Processing data captured by a laser profile scanner mounted on an industrial robot*. Doctoral Dissertation.

38. Karlsson, Alexander (2010) *Evaluating Credal Set Theory as a Belief Framework in High-Level Information Fusion for Automated Decision-Making*. Doctoral Dissertation.

39. LeBlanc, Kevin (2010) *Cooperative Anchoring – Sharing Information About Objects in Multi-Robot Systems*. Doctoral Dissertation.

40. Johansson, Fredrik (2010) *Evaluating the Performance of TEWA Systems*. Doctoral Dissertation.

41. Trincavelli, Marco (2010) *Gas Discrimination for Mobile Robots*. Doctoral Dissertation.

42. Cirillo, Marcello (2010) *Planning in Inhabited Environments: Human-Aware Task Planning and Activity Recognition*. Doctoral Dissertation.

43. Nilsson, Maria (2010) *Capturing Semi-Automated Decision Making: The Methodology of CASADEMA*. Doctoral Dissertation.

44. Dahlbom, Anders (2011) *Petri nets for Situation Recognition*. Doctoral Dissertation.

45. Ahmed, Muhammad Rehan (2011) *Compliance Control of Robot Manipulator for Safe Physical Human Robot Interaction*. Doctoral Dissertation.

46. Riveiro, Maria (2011) *Visual Analytics for Maritime Anomaly Detection*. Doctoral Dissertation.

47. Rashid, Md. Jayedur (2011) *Extending a Networked Robot System to Include Humans, Tiny Devices, and Everyday Objects*. Doctoral Dissertation.

48. Zain-ul-Abdin (2011) *Programming of Coarse-Grained Reconfigurable Architectures*. Doctoral Dissertation.

49. Wang, Yan (2011) *A Domain-Specific Language for Protocol Stack Implementation in Embedded Systems*. Doctoral Dissertation.

50. Brax, Christoffer (2011) *Anomaly Detection in the Surveillance Domain*. Doctoral Dissertation.

51. Larsson, Johan (2011) *Unmanned Operation of Load-Haul-Dump Vehicles in Mining Environments*. Doctoral Dissertation.

52. Lidström, Kristoffer (2012) *Situation-Aware Vehicles: Supporting the Next Generation of Cooperative Traffic Systems*. Doctoral Dissertation.

53. Johansson, Daniel (2012) *Convergence in Mixed Reality-Virtuality Environments. Facilitating Natural User Behavior.* Doctoral Dissertation.

54. Stoyanov, Todor Dimitrov (2012) *Reliable Autonomous Navigation in Semi-Structured Environments using the Three-Dimensional Normal Distributions Transform (3D-NDT).* Doctoral Dissertation.

55. Daoutis, Marios (2013) *Knowledge Based Perceptual Anchoring: Grounding percepts to concepts in cognitive robots*. Doctoral Dissertation.

56. Kristoffersson, Annica (2013) *Measuring the Quality of Interaction in Mobile Robotic Telepresence Systems using Presence, Spatial Formations and Sociometry.* Doctoral Dissertation.

57. Memedi, Mevludin (2014) *Mobile systems for monitoring Parkinson's disease*. Doctoral Dissertation.

58. König, Rikard (2014) *Enhancing Genetic Programming for Predictive Modeling*. Doctoral Dissertation.

59. Erlandsson, Tina (2014) *A Combat Survivability Model for Evaluating Air Mission Routes in Future Decision Support Systems*. Doctoral Dissertation.

60. Helldin, Tove (2014) *Transparency for Future Semi-Automated Systems. Effects of transparency on operator performance, workload and trust.* Doctoral Dissertation.

61. Krug, Robert (2014) *Optimization-based Robot Grasp Synthesis and Motion Control*. Doctoral Dissertation.

62. Reggente, Matteo (2014) *Statistical Gas Distribution Modelling for Mobile Robot Applications*. Doctoral Dissertation.

63. Längkvist, Martin (2014) *Modeling Time-Series with Deep Networks*. Doctoral Dissertation.

64. Hernández Bennetts, Víctor Manuel (2015) *Mobile Robots with In-Situ and Remote Sensors for Real World Gas Distribution Modelling*. Doctoral Dissertation.

65. Alirezaie, Marjan (2015) *Bridging the Semantic Gap between Sensor Data and Ontological Knowledge*. Doctoral Dissertation.

66. Pashami, Sepideh (2015) *Change Detection in Metal Oxide Gas Sensor Signals for Open Sampling Systems*. Doctoral Dissertation.

67. Lagriffoul, Fabien (2016) *Combining Task and Motion Planning*. Doctoral Dissertation.

68. Mosberger, Rafael (2016) *Vision-based Human Detection from Mobile Machinery in Industrial Environments*.

69. Mansouri, Masoumeh (2016) *A Constraint-Based Approach for Hybrid Reasoning in Robotics*.

70. Albitar, Houssam (2016) *Enabling a Robot for Underwater Surface Cleaning*.

71. Mojtahedzadeh, Rasoul (2016) *Safe Robotic Manipulation to Extract Objects from Piles: From 3D Perception to Object Selection*.

72. Köckemann, Uwe (2016) *Constraint-based Methods for Human-aware Planning*.