**Dissertation**

# Achieving Dependability of High-Level Robot Programs by using Belief Management

Clemens Mühlbacher

Graz, 2017

*Institute for Software Technology*
*Graz University of Technology*

Supervisor/First reviewer: Univ.-Prof. Dipl.-Ing. Dr. techn. Franz Wotawa
Second reviewer: Univ.-Prof. Gerhard Lakemeyer Ph.D.

# Abstract

Mobile robots are getting increasingly deployed in everyday environments. Applications range from simple vacuum cleaning robots to fleets of transport robots used in warehouse environments. One can expect that more robots which perform even more complex tasks will get deployed in everyday environments soon. Even though these robots are very different, each of these robots uses some kind of belief about the environment to take a decision which action should be performed next to achieve the task given to the robot. To allow the robot to make a reasonable decision the belief needs to be continuously updated if the robot changes the environment.

Unfortunately, as the robot is deployed in an everyday environment one cannot assume that every performed action is executed successfully. Due to this wrong execution, the robot needs to deal with faults during the execution. Thus, the robot needs to continuously use the observations taken from the environment to check if the execution yields the desired result. Furthermore, the robot needs to be able to reason which action was executed incorrectly, if a discrepancy between expected and real observation occurs.

To reason which action was executed incorrectly, the belief of the robot about the world needs to be managed. Such a belief management allows the robot to draw reasonable conclusions even in the presence of faulty action outcomes. Thus, a belief management should be part of a robotic system. However, to integrate it into a robotic system the method needs to be simple to use and needs to be incorporated into the high-level control of the robot.

In this thesis, we show how one can simplify the use of such a belief management approach. We extend an existing approach to reduce the necessarily detailed specification how an action has failed. Furthermore, we propose a method to reuse common sense from other knowledge bases to allow the robot to detect a fault faster.

Besides the simplification of the belief management, we also show in this thesis how one can use the belief of the robot about the environment to draw a safe conclusion which action should be performed next. This method does not only allow the robot to decide which action to perform next but also to determine that it lacks the knowledge to make a decision. If the robot is lacking knowledge about the environment, the robot can use a method presented in this thesis to gather enough knowledge about the environment to take a decision.

Finally, we show a method in this thesis a method which allows one to automatically derive a reactive program from a given sequential program, thus increasing the robot's capability to react to faults which have occurred. Therefore, leading to a more robust and autonomous robotic system.

# Kurzfassung

Mobile Roboter werden heutzutage immer mehr im Alltag verwendet. Dies reicht von einfachen Staubsaugrobotern bis hin zu Robotern, welche in Lagerhallen selbständig ihre Arbeit verrichten. Durch die technische Entwicklung auf diesem Gebiet und dem erfolgreichen Einsatz bisheriger Roboter ist davon auszugehen, dass zukünftig im Alltag noch komplexere Robotiksysteme eingesetzt und entwickelt werden. Obwohl diese Roboter auf den ersten Blick sehr unterschiedlich sind, verwenden sie doch alle eine Art von Wissen über ihre Umgebung. Dieses Wissen ist notwendig, um eine Entscheidung darüber treffen zu können, welche Aktion als nächstes durchgeführt werden soll, damit das Ziel, welches der Roboter verfolgt, erreicht wird. Da die Aktionen des Roboters die Welt beeinflussen, müssen diese Änderungen auch im Wissen des Roboters über die Welt abgebildet werden.
Leider kann nicht davon ausgegangen werden, dass jede Aktion, welche durch den Roboter ausgeführt wird, den gewünschten Effekt erzielt. Deswegen muss der Roboter durch entsprechende Sensormessungen das erfolgreiche Ausführen einer Aktion selbständig überprüfen. Weiters muss er auf Grund der Messungen schließen können, welche Aktion fehlerhaft war, um damit sein Wissen über die Welt im Einklang mit der Welt zu halten.
Um dies zu ermöglichen, sollte ein Wissensmanagementsystem Teil des Robotiksystems sein, welches es dem Roboter, auch im Fall einer fehlerhaften Ausführung, erlaubt, sinnvolle Schlüsse über die Welt zu ziehen. Für die Integration in das Robotiksystem ist es unerlässlich, dass die Verwendung solch eines Systems einfach ist, sowie dass es nahtlos in der höchsten Ebene der Entscheidungsfindung des Roboters integriert ist.
In dieser Dissertation wird erläutert, wie ein bestehendes Wissensmanagementsystem erweitert werden kann, damit es einfach zu verwenden ist. Zum einen wird die Notwendigkeit einer genaueren Spezifikation, welche Effekte eine fehlerhafte Ausführung einer Aktion auf die Welt hat, aufgehoben. Zum anderen wird alltägliches Wissen über die Umgebung von externen Wissensdatenbanken in das System integriert, um so das Wissen über die Welt, welches der Roboter besitzt, einfach zu erweitern. Weiters wird in der gegenständlichen Dissertation eine Methode präsentiert, welche ein Ableiten von sicherem Wissen aus dem Wissensmanagementsystem erlaubt. Diese Methode erlaubt es dem Roboter eine Entscheidung zu treffen, welche Aktion als nächstes ausgeführt werden soll. Zudem kann der Roboter mit Hilfe dieser Methode feststellen, dass es einen Mangel an Wissen über die Welt gibt. Diese Wissenslücken können anschließend vom Roboter durch geeignete Messungen geschlossen werden.
Zu guter Letzt wird im Rahmen dieser Dissertation eine Methode präsentiert, welche es erlaubt, aus einem sequenziellen Programm ein reaktives Programm zu erzeugen. Dadurch ist es für den Roboter möglich, auf Änderungen des Wissens über die Welt zu reagieren, ohne dabei seinen ursprünglichen Plan für die Erreichung des Zieles aufzugeben.

# Acknowledgement

First of all, I would like to thank Gerald Steinbauer for his support during this thesis. Through the discussions with him many ideas began to crystallize, and without these discussions, most of the resulting work would not have been possible. Additionally, his passion for providing a working environment which allowed to experiment with different methods in theory and in practice allowed me to evolve my knowledge about robustness in robotic systems.
I would also like to thank my supervisor Franz Wotawa for the discussion about science and his passion about model based methods. Due to this passion, he inspired me to foster my knowledge about model based methods to create more robust systems.
Furthermore, I would like to thank Gerhard Lakemeyer for being my external reviewer. His work was always inspiration how logic can be used to model behavior.
Special thanks go to Michael Reip, Gerald Steinbauer, Stephan Gspandl, Ingo Pill and Alexander Ferrein who created the basis for this thesis. Without their work, this thesis would not have been possible.
Moreover, I would like to thank my colleagues for the long discussions about the topic of this thesis, robotics and the off-topic discussions which allowed me to clear my head.
Additionally, I would like to thank the members of the TEDUSAR team; Konstantin Lassnig, Julia Nitsch, Johannes Maurer, Alexander Buchegger, Stefan Imlauer and Stefan Loigge, which made it possible to bring robotic systems to life.
Finally, I would like to thank my sister Claudia Mühlbacher and my colleague Alexander Buchegger for reviewing an earlier version of this thesis.
Last but not least I would like to thank my family which always supported me. Without them, I would have never been able to finish my studies or my thesis.

Clemens Mühlbacher in Graz, 2017

## Statutory Declaration

*I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.*

Graz, ______________________ ______________________

Place, Date Signature

## Eidesstattliche Erklärung

*Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommene Stellen als solche kenntlich gemacht habe.*

Graz, am ______________________ ______________________

Ort, Datum Unterschrift

# Contents

# List of Figures

# List of Tables

Chapter 1

# Introduction

Robotic systems nowadays are used more and more in daily life. As an example, one could consider a vacuum cleaning robot [60] which is widely used. Even though these systems comprise only a simple behavior, they show the first applications of robots which operate in an unknown environment and react according to sensing information.
Starting from these basic robotics settings more and more complex robotics systems are developed such as autonomous cars [55], [146] which aim to change how we move between locations. Whereas such systems are designed to transport people robotics system which transport goods within a warehouse [155], [53] are already widely used. Those systems allow more flexibility and are cheaper than traditional conveyor belts.
Even though systems perform complex transportation tasks robots can perform even more complex tasks in real world settings such as serving as a museum tour guide [144], navigating in a crowded city center [12], collecting garbage in a city [40] exploring Mars [145] or being used for rescue missions in earth quake scenarios [62].
All these systems ranging from simple vacuum cleaning to exploring Mars have in common that they should operate without human intervention. Thus following [73] the demand that a system be dependable. One meaning of dependability is that the system should be fault tolerant. Thus, one does not assume the system always works perfectly, instead the system should be able to deal with faults in such a way that their consequences are negligible. To provide a dependable autonomous system one can follow the definition of [72]. In [72] dependability is defined as a concept which incorporates two sub-concepts.
The first concept is robustness in order to deliver a service in contrarious situations. Thus, a system needs to deal with uncertainty in its environment. Such uncertainty often arises due to the openness of the environment, e.g. an obstacle blocking the route of the robot or due to the non-determinism of the environment such as changes in the light conditions during the day.
The second concept is fault tolerance which imposes that a robot performs the task regardless of faults affecting system resources, e.g. sensor faults. Thus, the second concept strongly relates to the capability of a robot to deal with its shortcomings during the execution. Using the ISO standard 26262 [135] one can define fault tolerance more accurately. In the standard one distinguishes between a fault, an error, and a failure. A fault is an abnormal condition that can cause an element or an item to fail. The error is the discrepancy between the computed, observed or measured value or condition, and the true, specified, or theoretically correct value or condition. A failure is the termination of the ability of an element, to perform a function as required. Thus, fault tolerance means that regardless of a fault

the robot encounters, the robot does not show a failure.
As robotics systems are complex often these systems are created with the help of different layers. One very broadly applied version is the three-tier architecture [46]. The architecture uses a high-level which is responsible for planning which actions the robot should perform to reach a certain goal. This layer uses a high degree of the actions e.g. “pick up the milk box”. Thus, these actions cannot be directly executed. To execute such actions properly the middle layer is used. The actions may be decomposed further into little skills which can be executed directly by the robot. For each skill, the middle layer ensures that it can be executed on the robot without the interference of another skill. The result of the middle layer is an action which can be executed by the robot e.g. “move the arm at position 21 24 1”. The last layer ensures that the robot can execute actions by providing a proper implementation of the actions e.g. a method to move the arm without collisions in the environment to a specified location.
Ensuring a high degree of fault tolerance imposes several aspects such as testing, run-time diagnosis, and adaption during run time. All these problems can be addressed through model based approaches; such a concept was published in [133]. As the robot consists of several layers, different methods for testing need to be applied. See the publications [88] and [96] for model-based testing approaches of robotics systems. Besides model-based testing methods also methods to adapt such models are important. See the publications [90] and [89] for model adaption methods. Although all these methods improve the overall fault tolerance of the robot, we will focus on the run-time diagnosis in this thesis.
To have a robust and fault-tolerant and thus dependable robotics system one needs to deal with faults in a robotics system on all layers of abstraction. Each layer has its demands and possibilities for robustness and fault-tolerance. As it would not be possible in one thesis to address all these different layers, we will focus on the high-level of the robot which is responsible for the decision making of the robot. We refer the interested reader to [71] for a discussion about fault handling in the lower layer, the basic principles where also published in [131]. In the next section, we will explain in more detail why the high-level of a robot is of particular interest to guarantee a dependable robotics system.
After we have explained why we address dependability in the high-level of the robot, we will detail the research objectives which guided the performed work. This section is followed by a section describing which contributions were made in the thesis to achieve the research objectives. Afterward in Section 1.4 we will discuss a running example which is used throughout the thesis. Finally, we will give an outline of the thesis.

## 1.1. Background and Motivation

As we have motivated above robotics systems are becoming a part of our society. Furthermore, these systems should take decisions in such a way that a task is achieved regardless of the occurrence of faults. Following the classical three-layer architecture [47] the decision which action should be taken to achieve a task is taken in the high-level control of the robot. Thus, to achieve a dependable decision making of the robot the high-level control of the robot, needs to consider dependability as a central part.
To make a decision, at the high level, the robot uses the knowledge it possesses about the world. This knowledge comprises which actions are possible and which conditions need to hold so that these actions can be performed. Additionally, it comprises the knowledge which objects are available in the environment, where these objects are, and additional information about an object or a property of the object. Furthermore, the robot needs some knowledge about itself, ranging from its location to which object it carries. Additionally, not only information about objects is part of the knowledge of

the robot but also how objects and their properties relate to other objects and their properties. This part of the knowledge is the common sense of the robot, e.g. specifying that milk is perishable and that perishable goods are usually in the fridge.
Using its knowledge the robot can answer questions needed to take a decision, e.g. where should I store the milk. Besides such questions, it is also of interest that the robot can ask a question about the future, e.g. where is the milk if I pick it up and drive into the kitchen. Such questions are those questions which are used to derive a plan to achieve a task. To answer the question how to store the milk the robot needs to respond to questions about general statements about the world, e.g. where the milk should be stored, and the future state of the world, e.g. will I be able to put the milk in the fridge if I drive into the kitchen. Especially the question what is possible after performing a certain action is essential to find actions which lead to the desired goal, e.g. moving into the kitchen alone does not allow the robot to put the milk into the fridge as the robot first needs to open the fridge.
To answer a question about the future, the robot needs to be able to reason how the world changes after an action is performed. However, this reasoning is not only of interest to answer a question about the future but also about the present. As a robot performs actions during the execution of a task, the knowledge about the world needs to be properly updated. Thus, the robot needs to change the knowledge it has about the world after each action execution. Furthermore, the robot does not only perform actions to modify the world but also often senses the world to gain knowledge how the world looks like, e.g. sense if the milk is in front of it.
As the robot should continuously answer questions with the help of the reasoning this reasoning needs to be kept manageable. To maintain the reasoning manageable not every possible outcome is considered an action may have. Instead, the robot considers only the most likely outcome. Using only the most likely outcome imposes that the robot only considers a non-faulty action outcome, e.g. after driving to the kitchen the robot thinks it is in the kitchen. Thus, if a fault in the action execution occurs the robot's knowledge about the world and the real world do no longer agree. This discrepancy can cause further faults during the execution of other actions or even lead to dangerous decisions, e.g. if the robot thinks it is in the kitchen, it can decide to drive forward but if the robot is in front of the stairs driving forward is not a preferable decision.
The discrepancy between the real world and the knowledge about the world the robot has can only be detected through the execution of sensing actions. As the sensing actions measure some property of the reality, the same properties need to hold in the knowledge the robot has about the world. Thus, if the sensing information contradicts the knowledge about the world, the real world and the knowledge about the real world are no longer consistent.
This inconsistency of the real world and the knowledge the robot has about the world lead to an inconsistency in the knowledge if the sensing information is just assumed to hold. Due to this inconsistency, the robot is no longer able to answer a question reasonably. Thus causing the robot to be no longer able to derive a meaningful plan to achieve a goal. Instead, a robot may take arbitrary actions or don't perform actions at all.
As ignoring the inconsistency between the measured real world and the knowledge the robot has is not an option one needs to consider a different approach. Instead of ignoring the inconsistency the robot can reason what went wrong to reason how the world may look like.
Let's consider a simple example outlining this idea. The robot needs to move the folder from room D to room C. The start state of the environment is depicted in Figure 1.1.
After picking up the folder and moving to room C the robot assumes the environment is as depicted in Figure 1.2. But instead the robot senses that the calculator is also in the same room as the robot. Thus, the world seems to be as depicted in Figure 1.3.
As an object does not move on its own the real world cannot have changed from the environment as

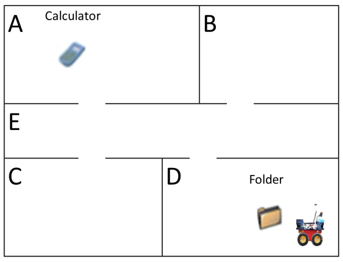

Figure 1.1.: Environment at the beginning of the deliver task.

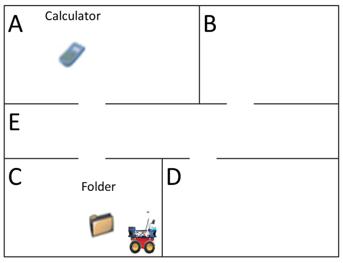

Figure 1.2.: Expected environment after the delivery task was performed.

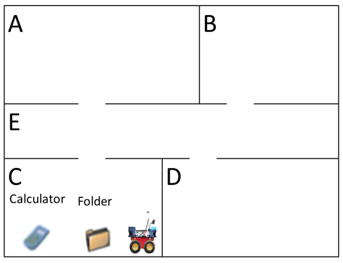

Figure 1.3.: Sensed environment after the delivery task was performed.

it is depicted in Figure 1.1 to the environment depicted in Figure 1.3. Thus, the robot can perform a reasoning and can conclude that either it drove to room A instead of C or the sensing was faulty and the calculator is not in the same room. Both cases are depicted in Figure 1.4. Using this information, the robot can derive a plan to safely navigate to its destination and finish the delivery.
As this kind of reasoning allows the robot to conclude how the world looks like, the robot can detect and handle faults. In [51] such a reasoning was formally defined. We will use the definition of this reasoning as a basis for the thesis. We extend the proposed method to derive the secured knowledge the robot has about the world. In the remainder of the thesis we will call the method to derive explanations for an inconsistency, together with the method to derive secure knowledge from this approach belief management. Furthermore, we will extend the approach presented in [51] such that the robot can react to changes in the belief management. This reaction to changes is necessary as the belief management may reveal that the secure knowledge invalidates a previously made plan.

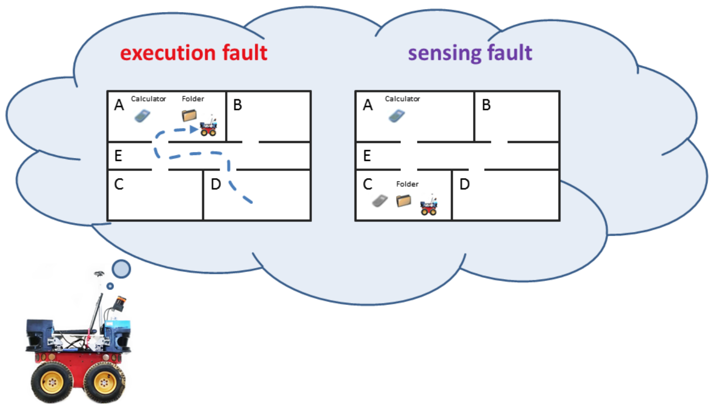

Figure 1.4.: Possible explanation how the environment looks like.

## 1.2. Research Objectives

In the previous section, we have motivated why we are interested in a belief management, which ensures that the robot can perform a reasoning even in cases of faulty action executions. Using the method described in [51] as basis several research objectives guided the work of this thesis, which is described in this section.

### 1.2.1. Reduce the burden of modeling of the belief management

To perform the belief management of the robot several aspects of the world need to be properly modeled to allow an automatic reasoning. To use the belief management, this modeling effort should be reduced.
The modeling effort consists of two main parts. Firstly, the robot needs to know which faults can occur. This knowledge allows the robot to draw a definite conclusion of which fault might have occurred. However, this has the drawback that such a model is often hard to derive and even harder to maintain. Secondly, to detect inconsistencies the robot uses commonsense knowledge about the environment, it is operating in. Through this knowledge, the robot can detect if an environment constellation is physically not plausible, e.g. an object can be only at one place at a time. If the robot has more knowledge about its environment, a fault can be detected easier. Thus, the broader the commonsense knowledge is, the better the belief management works. However, the specification of a big, sound and complete commonsense knowledge base can be complicated.
The research objective is to reduce both modeling efforts. Firstly, reduce the need to model faults in detail. Secondly, the effort to specify proper commonsense knowledge should be as minimal as possible.

### 1.2.2. Increase the efficiency of the belief management

As the robot uses the knowledge about the word to decide which action should be taken next, it is important that the reasoning used to answer certain questions be as fast as possible. Thus, an efficient method for the belief management is of interest. To improve the efficiency of the belief management, an efficient method to derive the possible faulty actions and a method to answer queries using the information of possible faulty actions, are necessary.

### 1.2.3. Integration of the belief management in the high-level control of the robot

To allow the robot to use the belief management, there is a need to integrate the reasoning method into the high-level control of the robot. This integration needs to consider whether it is safe to act with the current knowledge. Additionally, this integration needs to consider whether the robot is confident that the secure knowledge implies that the plan needs to be aborted. Furthermore, the robot needs to be able to distinguish cases where knowledge is lacking to make a conclusion about future actions with those cases where the robot can derive an answer. This distinction also includes how the robot handles a lack of knowledge.

### 1.2.4. A systematic evaluation of the belief management

After the integration of the belief management into the high-level control of the robot, an evaluation is of interest. The evaluation should answer the question if the belief management has the desired positive effect on the dependability. Furthermore, the evaluation should answer the question which kind of high-level program can benefit from the belief management most.

## 1.3. Contributions

In the previous section, we discussed the research objectives which guided the work presented in this thesis. The performed work resulted in the contributions outlined in this section.

### 1.3.1. Belief management without models of action faults

To address the problem of modeling every fault of an action a different approach to identify possibly faultily executed action needs to be used. Instead of modeling the faulty action one follows the idea of consistency-based diagnosis [113]. The idea is that a faulty action may influence the truth value of a subset of fluents. Thus, instead of defining all faults, an action has, one only defines which parts of the environment may be influenced if the action fails. This definition eases the modeling for the belief management as well as improves the efficiency as one does not need to reason how the action has failed. Instead, it is sufficient to reason if the action has failed at all. The concept of this method was published in [94].

### 1.3.2. Using common sense from external sources

Besides the modeling of faulty actions, the modeling of the commonsense knowledge of the robot is a cumbersome task. To address this problem, one can reuse the commonsense knowledge which is already available. Over the last decades big commonsense knowledge bases have been created. Thus, instead of modeling the same statements again one can reuse this knowledge by incorporating this knowledge into the knowledge base the robot has. The theoretical concept was published in [132]. In [93] the impact of the belief management was evaluated.

### 1.3.3. Integration of belief management into high-level program execution

After the robot has derived which actions are faulty, the robot can derive the knowledge it is certain about. This knowledge consists of those statements which are valid in all possible explanations. With the help of this knowledge, the robot can reason which actions are safe to perform. Furthermore, the robot can reason if it lacks information to proceed safely. This reasoning is achieved by defining the execution semantic which is used by the agent to choose its next action. The concept of this integration was published in [92].

### 1.3.4. Active gathering of knowledge

As the robot can decide if its knowledge is not enough to safely take a decision, the robot also needs the capability to react to such a situation. Thus, instead of aborting its mission, the robot can gather knowledge about the world through the execution of safe sensing actions. This method was published in [91].

### 1.3.5. Deriving programs that react

To achieve a certain goal, it is often sufficient to specify a simple imperative program. Such programs are simple recipes which actions should be performed after each other. Although such simple programs are often enough to achieve the goal they often fail in case of a fault. This failure is caused by a fault which may yield a different state of the world as expected by the robot after executing a certain action. Thus, the robot can no longer follow this simple program to achieve its goal. However, often it is sufficient to step back in the program and continue the execution from this point. To allow the robot to step back in its program one needs to change the imperative program into a reactive program. In this thesis, we will show how one can automatically derive a reactive program from an imperative program.

### 1.3.6. Evaluating the benefit of belief management

The belief management is designed to increase the dependability of the robot. As such one can not only state theoretical properties about the knowledge, the robot possesses during execution. Instead, an empirical evaluation is of interest, showing the benefit of the belief management in comparison to other approaches. Such a preliminary evaluation was published in [95].

## 1.4. Running example

To discuss the different methods in this thesis, we will use a running example. This example will be utilized as a simple version of a real-world robotics application. Furthermore, we will use this example during the evaluation for comparing the impact of different belief management methods.
The running example is a simple delivery robot. It can fetch objects and move those objects between different locations. As such it is an abstract version of any transport robot used already in the industry, see [155] or [53] for example. A similar example was already used to evaluate the performance of a robot using another belief management method [52].
The delivery robot can perform three actions. $goto(R_1)$ moves the robot from the current room to room $R_1$. To pick an object $O_1$ the robot can perform the action $pick(O_1)$. Additionally, the robot can perform the action $put(O_1)$ to put the object $O_1$ down.
Besides the action to change the world, the robot can perform two different sensing actions. To check if the robot is holding something in its gripper, it can perform the action *senseHolding*. Through the sensing action, $senseIsAt(O_1)$ the robot can check if the object $O_1$ is at the same place as the robot. Both sensing actions yield a simple true or false as a sensing result.
As we consider the robot to fail during the execution of some actions we consider the following faults the robot may encounter. The *goto* action can fail such that the robot moves to another place as

specified, e.g. the robot ends up in room $R_2$ instead of $R_1$. The action *pick* can fail such as that the robot is holding no object at all. Moreover, the action can fail, such it can pick up a wrong object which is in the same room. Thus, the robot possibly picks up object $O_2$ instead of $O_1$. If the robot performs the *put* action it may fail by it still holding the object. Finally, both sensing actions may produce a wrong result. Thus, instead of yielding true as a result the action yields false as result or vice versa.
To model further uncertainty in the environment, we consider exogenous events. These events occur without the knowledge of the robot and change the environment. We consider two different events. The first event *exogMove* moves an object from one location to another. The second event *snatch* simulates that somebody snatches the object from the robot.
The task of the robot is always a set of objects which need to be delivered. Each object has specified its target location. Furthermore, we assume the robot has initially complete knowledge about the object's location, as well as its location. Additionally, the robot does not hold an object in the initial situation.

## 1.5. Outline

The remainder of the thesis is structured as follows. The next chapter discusses some preliminaries necessary for the thesis. In Chapter 3 we discuss research which is related to the different parts proposed in this thesis. Additionally, we discuss related research which tackles belief management for robotics systems. Please note that a more detailed related research is discussed in every of the remaining chapters which discuss the different contributions. The follow-up chapter discusses how belief management could be performed without fault modes. Afterwards in Chapter 5 the method to use common sense from external sources is discussed. The proceeding chapter discusses the integration of the belief management into the high-level program execution of the robot. This chapter is followed by a chapter discussing how the agent can actively gather information about the world. In Chapter 8 we discuss how to derive a reactive program from a sequential one. This chapter is proceeded by the chapter discussing aspects of the practical implementation of the belief management. Afterwards, we discuss the evaluation of the belief management in comparison to other classical approaches for a robust high-level control in Chapter 10. Finally, we conclude the paper and point out some future work.

Chapter 2

# Preliminaries

In this chapter, we will discuss the theoretical foundations for the remainder of the thesis. Throughout the thesis, we are interested in modeling a robot which performs actions in an environment. To be more specific we are interested in the high-level control of such a robot. Thus, we represent the robotic system on an abstract level, allowing us to talk about actions and discrete states of the world.

The first problem which we need to address is to model the changes an action causes in the world. For each action one can specify the influence it has on the world, e.g. the *goto* action changes the location of the robot. But to ultimately define the impact an action has in the world one also needs to specify which parts of the world are not changed, e.g. the *goto* does not alter the color of the robot. As the number of things which are not changed is arguably infinite, the specification of non-effects cannot be performed quickly. The problem to address the specification of non-effects is known in artificial intelligence as the frame problem [82].

The frame problem was first identified by McCarthy and Hayes in [82]. It is one of the well-known problems which need to be addressed to model an intelligent agent. To address this issue, McCarthy proposed in [80] to use a form of non-monotonic reasoning called circumscription [79]. To avoid non-monotonic reasoning, Reiter suggested an alternative solution in [114] which uses the situation calculus [78] and successor state axioms to deal with the frame problem. We will discuss the situation calculus and the solution to the frame problem in Section 2.1 in more detail.

Besides the modeling of the effect, an action has one also needs to model the conditions which must hold to be able to perform the action. The problem of modeling this precondition of an action is called the qualification problem [81]. The problem arises from the observation that there is a (nearly) infinite set of conditions which need to hold to perform an action successfully. Let's consider the running example. To do the action $put(O_1)$ the agent needs to hold the object $O_1$. Furthermore, the object needs not to be glued to the robot or mounted in another way. Furthermore, the arm of the robot needs to work properly ....

To address the qualification problem, McCarthy proposed in [81] to use circumscription with the idea that the precondition of an action holds unless proven otherwise. We will use a similar approach to address the problem. We split the precondition of an action into two parts. The first part of the precondition can be easily modeled with the knowledge the agent has, e.g. for the *put* we only model that the agent is holding $O_1$. The second part of the precondition captures all the other conditions which are needed but cannot easily be modeled, e.g. that the object is glued to the robot's hand. We assume that the second part of the precondition holds if no observation is made that the action was not successfully executed. Thus, we use the idea of McCarthy in a restricted form, as we consider only

parts of the precondition to be modeled.
The first part of the precondition can be modeled by standard means of the situation calculus and will be presented in Section 2.1. Whereas the second part of the precondition is modeled using the history-based diagnosis approach and will be presented in Section 2.2.
After the actions, effects and the precondition of the actions are modeled, one needs to define how the robot should use the actions to achieve a certain task. We will use the high-level action language called IndiGolog [20] to specify the actions the robot should perform to achieve a task. In Section 2.3 we will discuss how the language is defined and how the robot decides which action to perform next.

## 2.1. Situation Calculus

To model the impact an action has the situation calculus [78] is an expressive method. The situation is a second order logic with three different sorts. Actions ($\mathcal{A}$), objects ($O$) and situations ($\mathcal{S}$). Situations are used to model a sequence of executed actions. This is done with the help of the constant $S_0$ representing the initial situation and the function $do : \mathcal{A} \times \mathcal{S} \rightarrow \mathcal{S}$. Thus, to represent that the robot has only executed action $goto(R_1)$ we have the following situation term $do(goto(R_1), S_0)$.
Besides modeling the action sequence with the situation calculus, the situation term is also used to represent the state of the world. This representation is done by so-called fluents $\mathcal{F}$. A fluent is either a situation dependent predicate, e.g. $holding(O_1, s)$. This type of fluent is called relational fluent. Or a fluent is a situation dependent function, e.g. $objectTemperature(O_1, s) = 20$. Such fluents are called functional fluent. As such the last argument of a fluent is a situation term. This situation term is used to specify if the fluent holds after the action sequence was performed. For the sake of simplicity, we will concentrate on relational fluents only in this thesis.
With the help of the situation term, one can model the sequence of actions performed so far. But to model, the current state of the world one needs to model the effect an action has. To do so, one can use so-called effect axioms. Each of these axioms defines a condition which needs to hold such that the action sets the fluent to true or false after its execution.
The specification of the effect axioms is not sufficient as we argued above. This modeling issue is called the frame problem and was addressed by Reiter in [114]. The proposed solution of Reiter is based on the work suggested in [54], [100] and [122]. To solve the frame problem, Reiter offered to rewrite the effect axioms to so-called successor state axioms by considering some mild assumptions.
First, let's consider the most general case of an effect axiom. The action $\alpha$ with the precondition $\Pi_\alpha$ changes the truth value of a situation dependent formula $\phi$ under the condition $\kappa$.

$$\forall s. \Pi_\alpha(s) \wedge \kappa(s) \rightarrow \phi(do(\alpha, s)) \tag{2.1}$$

The first assumption made to tackle the frame problem is to restrict the formula $\phi$ to $F(do(\alpha, s))$, where $F$ is a fluent. Thus, instead of specifying the effect an action has on a formula one specifies the effect the action has on one fluent $F$.

$$\forall s. \Pi_\alpha(s) \wedge \kappa(s) \rightarrow F(do(\alpha, s)) \tag{2.2}$$

The second assumption made is that every possibility an action can influence a fluent is specified. This assumption is refereed as completeness assumption. To show the impact of this assumption let's consider two actions $\alpha_1$ and $\alpha_2$ influencing $F$ specified as follows.

$$\forall s. \Pi_{\alpha_1}(s) \wedge \kappa_{\alpha_1}(s) \rightarrow F(do(\alpha_1, s)) \tag{2.3}$$

$$\forall s.\Pi_{\alpha_2}(s) \land \kappa_{\alpha_2}(s) \rightarrow F(do(\alpha_2, s)) \tag{2.4}$$

If we assume that each action has a unique name, we can now apply the completeness assumption to rewrite the above equations into.

$$\forall s, \alpha. (\alpha = \alpha_1 \land \Pi_{\alpha_1}(s) \land \kappa_{\alpha_1}(s)) \lor (\alpha = \alpha_2 \land \Pi_{\alpha_2}(s) \land \kappa_{\alpha_2}(s)) \rightarrow F(do(\alpha, s)) \tag{2.5}$$

We can simplify the notation by grouping the conditions of the effect of action $\alpha$ into one formula $\psi(\alpha, s)$. Thus, we have the formula.

$$\forall s, \alpha.\psi(\alpha_1, s) \lor \psi(\alpha_2, s) \rightarrow F(do(\alpha, s)) \tag{2.6}$$

If we now consider $N$ actions instead of two we can define the formula $\Psi_F^+(\alpha, s)$ which gathers the left-hand side of implications for all actions.

$$\bigvee_{i.1 \leq i \leq N} \psi(\alpha_i, s) \tag{2.7}$$

We can do the same transformations steps for those effects setting a fluent value to false instead of true. We denote the formula which defines all actions and their conditions to set fluent $F$ to false as $\Psi_F^-(\alpha, s)$. Using this formula, we derive the following two equations for each fluent.

$$\forall s, \alpha.\Psi_F^+(\alpha, s) \rightarrow F(do(\alpha, s)) \tag{2.8}$$

$$\forall s, \alpha.\Psi_F^-(\alpha, s) \rightarrow \neg F(do(\alpha, s)) \tag{2.9}$$

To avoid situations which cause a contradiction we assume that the action effects do not contradict each other, thus it needs to hold that:

$$\nexists s, \alpha.\Psi_F^+(\alpha, s) \land \Psi_F^-(\alpha, s) \tag{2.10}$$

Combining the completeness assumption and assuming no contradicting effects we can conclude from Equation 2.8 and Equation 2.9 the following.

$$\forall s, \alpha.\neg F(s) \land F(do(\alpha, s)) \rightarrow \Psi_F^+(\alpha, s) \tag{2.11}$$

$$\forall s, \alpha.F(s) \land \neg F(do(\alpha, s)) \rightarrow \Psi_F^-(\alpha, s) \tag{2.12}$$

Combining those two equations into one equation we get the so-called successor state axioms.

$$\forall s, \alpha.F(do(\alpha, s)) \equiv \Psi_F^+(\alpha, s) \lor F(s) \land \neg\Psi_F^-(\alpha, s) \tag{2.13}$$

Using the successor state axiom one needs only to define one axiom per fluent instead of $2 \times |\mathcal{A}|$ axioms per fluent. Furthermore, the successor state axiom only incorporates actions and their conditions which change the truth value of the fluent. Thus, the length of the axioms is kept to a minimum.

### 2.1.1. Basic action theory

With the help of the successor state axiom the frame problem can be solved. But to use the situation calculus for reasoning purposes one needs to gather several axioms including the successor state axioms defined above. The combination of these axioms is called basic action theory (BAT) [116]. The BAT defines a second-order logical language with the following alphabet:

- three distinct sorts: actions $\mathcal{A}$, situations $\mathcal{S}$ and objects $O$.
- a constant $S_O$ of type $\mathcal{S}$. The constant defines the initial situation.
- a function $do : \mathcal{A} \times \mathcal{S} \to \mathcal{S}$. The function defines a successor relation between two situations. Thus, situation $s'$, $s' = do(\alpha, s)$ is the successor situation of $s$ after performing $\alpha$. In the remainder of the thesis we use $do([\alpha_1, \dots \alpha_n], S_0)$ as an abbreviation for $do(\alpha_n, \dots do(\alpha_1, S_0))$.
- a binary predicate $\sqsubset$: $\mathcal{S} \times \mathcal{S}$, we write $s \sqsubseteq s'$ in the remainder of the thesis as an abbreviation of $s \sqsubset s' \vee s = s'$. The predicate defines the ordering of the situations.
- for each $n \geq 0$, countable infinite predicates with arity $n$ of the form $(\mathcal{A} \cup O)^n$, these predicates are used to state situation independent relations.
- for each $n \geq 0$, countable infinite functions with arity $n$ of the form $(\mathcal{A} \cup O)^n \to O$, these functions are used to state situation independent functions.
- for each $n \geq 0$, finite or countable infinite functions with arity $n$ of the form $(\mathcal{A} \cup O)^n \to \mathcal{A}$, these define performed actions. In the remainder of the thesis we will use $\alpha$ with superscripts to donate actions.
- a binary predicate $Poss : \mathcal{A} \times \mathcal{S}$, we write in the remainder of the thesis $\Pi_\alpha(s)$ as an abbreviation of $Poss(\alpha, s)$. The predicate defines the condition which needs to hold such that an action can be performed.
- for each $n \geq 0$, finite or countable infinite predicates with arity $n$ of the form $(\mathcal{A} \cup O)^n \times \mathcal{S}$, these are relational fluents.
- for each $n \geq 0$, finite or countable infinite functions with arity $n$ of the form $(\mathcal{A} \cup O)^n \times \mathcal{S} \to O$ , these are functional fluents.

In addition to the above alphabet the language contains the standard alphabet of second order logic including equality. With the help of the defined language, we can state the different axioms used. First, we state the axioms used to define a valid situation.

$$do(\alpha_1, s_1) = do(\alpha_2, s_2) \supset \alpha_1 = \alpha_2 \wedge s_1 = s_2 \tag{2.14}$$

$$\forall P.[P(S_0) \wedge \forall \alpha, s.P(s) \supset P(do(\alpha, s))] \supset \forall s.P(s) \tag{2.15}$$

$$\neg s \sqsubset S_0 \tag{2.16}$$

$$s \sqsubset do(s, s') \equiv s \sqsubseteq s' \tag{2.17}$$

To ensure proper use of situation terms a unique name assumption of situations is defined in Equation 2.14. Equation 2.15 uses an induction axiom to define the successor relation $do$ on the situation. Furthermore, each situation is rooted in $S_0$. Thus $S_0$ is used as the initial situation. To ensure that $S_0$ is the initial situation Equation 2.16 is used. Finally, Equation 2.17 defines the ordering relation of the situations. These axioms form the so-called foundation axioms $\Sigma$.
Besides the foundation axioms, we use additional axiom sets. $\mathcal{D}_{SS}$ is the set of all successor state axioms. Thus, for each fluent, we create a successor state axiom as discussed above. To do so, one needs a unique name assumption for actions which are gathered in the axiom set $\mathcal{D}_{una}$. In the axiom set $\mathcal{D}_{ap}$ all preconditions of the actions are defined. Thus, a complete definition of the predicate $Poss$

is contained in this set. Finally, we define the axiom set $\mathcal{D}_{S_0}$ which defines the values of the fluent in the initial situation. In this set, only the situation term $S_0$ is used to specify the fluents. Additionally, no other use of a situation term is allowed in this set.
We can now summarize the axioms of the BAT into the set $\mathcal{D}$ as follows:

$$\mathcal{D} = \Sigma \cup \mathcal{D}_{SS} \cup \mathcal{D}_{una} \cup \mathcal{D}_{ap} \cup \mathcal{D}_{S_0} \tag{2.18}$$

With the help of the BAT, the robot can answer queries about the state of the world after a sequence of actions is executed. Furthermore, the robot can respond to questions about future states of the world. The reasoning performed to answer these queries solves the so-called projection problem [116]. Following [116] the projection problem can be stated as: decide if a logical formula holds after a given sequence of actions is executed.
Although with the help of the BAT one could solve the projection problem through theorem proving, it is not feasible in practice to perform such a theorem proving. Instead one can perform two different methods to solve the projection problem.

### 2.1.2. Regression

The first method to solve the projection problem is through regression [152]. The idea of regression in the situation calculus [114] is to start from a given goal formula and move this formula backward in time. Thus, instead of answering if the formula holds in the current situation, one regresses back the formula to the initial situation and answers the formula using the initial situation only ($\mathcal{D}_{S_0} \models \mathcal{R}[\phi]$).
To use only the initial situation to answer a question one regresses the formula back with the help of $\mathcal{D}_{SS}$ and $\mathcal{D}_{una}$. Thus, the complex reasoning imposed by the second-order induction axiom of the situation is not necessary.
The idea behind regression in the situation calculus is the following: Let's assume a formula $\psi(s)$ which only mentions $s$ as situation term. We call such a formula uniform in $s$. Furthermore, let's assume $s$ is the result of $s = do(\alpha, s')$. Additionally, let's assume the successor state axiom of the fluents have the following form $F(\bar{x}, do(\alpha, s')) \equiv \Phi(\bar{x}, \alpha, s')$. We now replace all occurrences of the fluents with their equivalent right-hand side of the successor state axiom. The resulting formula is $\phi'$. As we have replaced every fluent with the right-hand side, the formulas are equivalent. Furthermore, as the right-hand side only mentions $s'$ instead of $s$ the formula $\phi'$ is uniform in $s'$. Thus, the formula is closer to $S_0$ as it holds that $s' \sqsubset s$. One can now perform this replacement until only the situation term $S_0$ is mentioned.
This simple rewriting method can be used for formulas fulfilling certain restrictions [105]. A formula is regressable if it is a first order formula, every situation term which is mentioned has the form $do(\alpha_n, \ldots do(\alpha_1, S_0))$ for some $n \geq 0$, the formula does not quantify over situations and for every mentioned $Poss(\alpha, s)$ in the formula it holds that $\alpha$ is part of the actions defined in $\mathcal{A}$.
To perform the rewriting method to rewrite a formula such that only the initial situation is mentioned a regression operator is used [105], [116]. The regression operator $\mathcal{R}$ is defined for a formula $\phi$ as follows:

- if $\phi$ is an atom
  - if $\phi$ has the form $s = s'$, where $s = [\alpha_1, \ldots \alpha_m]$ and $s' = [\alpha'_1, \ldots \alpha'_n]$ then the regression is defined as follows:
    * if $m = n$ and $m = 0$ thus $s = S_0$ and $s' = S_0$ then $\mathcal{R}[\phi] = \top$
    * if $m \neq n$ then $\mathcal{R}[\phi] = \bot$

    - if $m = n$ and $m > 0$ then $\mathcal{R}[\phi] = \mathcal{R}[\alpha_m = \alpha'_m \wedge \ldots \alpha_1 = \alpha'_1]$
  - if $\phi$ has the form $s \sqsubset s'$, where $s = [\alpha_1, \ldots \alpha_m]$ and $s' = [\alpha'_1, \ldots \alpha'_n]$ then the regression is defined as follows:
    - if $m = 0$ and $n > 0$ then $\mathcal{R}[\phi] = \top$
    - if $m \geq n$ then $\mathcal{R}[\phi] = \bot$
    - if $m < n$ and $m > 0$ then $\mathcal{R}[\phi] = \mathcal{R}[\alpha_m = \alpha'_m \wedge \ldots \alpha_1 = \alpha'_1]$
  - if $\phi$ has the from $Poss(\alpha, s)$ then $\mathcal{R}[\phi] = \mathcal{R}[\Pi_\alpha(s)]$
  - if $\phi$ is situation independent then $\mathcal{R}[\phi] = \phi$
  - if $\phi$ is a term of the form $F(\bar{t}, S_0)$, where $F$ is a relation fluent then $\mathcal{R}[\phi] = \phi$
  - if $\phi$ is a term of the form $f(\bar{t}, S_0)$, where $f$ is a functional fluent then $\mathcal{R}[\phi] = \phi$
  - if $\phi$ is a term of the form $F(\bar{t}, do(\alpha, s))$, where $F$ is a relation fluent and the fluent has the following successor state axiom $F(\bar{x}, do(\alpha, s)) \equiv \Phi_F(\bar{x}, \alpha, s)$ then $\mathcal{R}[\phi] = \mathcal{R}[\Phi_F(\bar{x}, \alpha, s)]$
  - if $\phi$ is a term of the form $f(\bar{t}, do(\alpha, s))$, where $f$ is a functional fluent and the fluent has the following successor state axiom $f(\bar{x}, do(\alpha, s)) = y \equiv \phi_F(\bar{x}, y, \alpha, s)$ then $\mathcal{R}[\phi] = \mathcal{R}[\exists y. \phi_F(\bar{x}, y, \alpha, s) \wedge \phi|_y^{f(\bar{t}, do(\alpha, s))}]$, where $\phi|_{t'}^{t}$ donates the replacement of all occurrences of the term $t$ in $\phi$ by the term $t'$.

- if $\phi$ is not an atom. Please note that the connectives $\neg$, $\wedge$ and $\exists$ are sufficient to describe a first order formula.
  - if $\phi$ has the form $\neg\phi'$ then $\mathcal{R}[\phi] = \neg\mathcal{R}[\phi']$.
  - if $\phi$ has the form $\phi_1 \wedge \phi_2$ then $\mathcal{R}[\phi] = \mathcal{R}[\phi_1] \wedge \mathcal{R}[\phi_2]$.
  - if $\phi$ has the form $\exists v. \phi'$ then $\mathcal{R}[\phi] = \exists v. \mathcal{R}[\phi']$.

We refer the interested reader to [105] for a proof of the correctness of the regression operator.

### 2.1.3. Progression

While regression is proven to be a sound and complete solution to the projection problem under mild assumptions it has the inherent problem that the efficiency depends on the length of the executed action sequence. As such regression gets slower the more actions, an agent performs. Thus, if one assumes a robot which performs several deliveries a day regression may not be suitable for this job.
Instead of rolling the query back to the initial situation one can use the opposite approach. One can progress [43] the initial situation to reflect the effect an action has. Thus, one updates the initial database after an action is performed and uses the updated database to answer the query. This method has two benefits. First, the efficiency of the reasoning does not depend on the length of the executed actions anymore. Second in most practical cases, one performs several queries before executing an action. Thus, the effect an action has needs only be considered once for several queries. Due to this reuse, the computational costs of the progression are amortized over several queries.
Thus, instead of defining a regression operator we use a progression operator $\mathcal{P}$ to progress $\mathcal{D}_{S_0}$ after executing $\alpha$ to $\mathcal{D}_\alpha$. After progressing the initial situation, we replace it in the BAT thus creating a new BAT $\mathcal{D}' = \mathcal{D} \setminus \mathcal{D}_{S_0} \cup \mathcal{D}_\alpha$.
Unfortunately, first results of progression for the situation calculus published in [68] indicated it is not possible in general that progression can be expressed with first-order logic. Later it was shown in [149] that it is not feasible to perform progression with first-order logic for arbitrary models.

As no general method for progression exists several fragments of the BAT were identified which allow a progression, see for example [68], [69] or [148] for such fragments. An overview of fragments of the BAT which allow progression can be found in [150]. These fragments all have in common that they restrict some expressions within the BAT. This restriction can either be the form of the successor state axioms, the form of the initial situation or a combination of both. We refer the interested reader to [147] for a detailed description of several progression methods.

### 2.1.4. Sensing

With the help of regression and progression, one can solve the projection problem efficiently. But until now we have only discussed how to model actions which change the world. We call this action, following [116], primitive actions. Whereas in contrast sensing actions are also very important for a robotic system.
To incorporate the information a sensing action provided in the situation calculus the predicate $SF(\alpha, s)$ was proposed in [119]. The predicate is used to capture the result of a sensing action if the product of the sensing action $\alpha$ which was performed in $s$ was $\top$ $SF(\alpha, s)$ is asserted. Otherwise $\neg SF(\alpha, s)$ is asserted to hold.
To link the result of a sensing action $SF(\alpha, s)$ to the property which was measured by the sensing action in the current situation one defines a formula $\psi_\alpha(s)$ $(SF(\alpha, s) \equiv \psi_\alpha(s))$ for each sensing action. Considering our running example, the action *senseHolding* has the sensing formula $\exists o.holding(o)$. Thus, if the action returns true, it is expected that the robot is holding an object. If the action returns false, one expects that the robot does not hold any object.
To capture the results of the sensing actions the set $\{sensed\}$ is added to the BAT [22]. Thus, one asserts the sensing result to hold. This assertion implies that the sensing formula holds for a particular situation. Due to the assertion, the theory becomes inconsistent if the sensing information contradicts the current situation. We will use this observation to detect if the robot's knowledge about the world does no longer reflect the real world.
To distinguish the expected sensing result and the measurement the robot has taken we will use the predicate $AO(\alpha, s)$ in the remainder of the thesis. Instead of asserting $SF$ directly we assert $AO$ and assert the following formula:

$$\forall \alpha, s. AO(\alpha, s) \leftrightarrow SF(\alpha, s) \tag{2.19}$$

Due to this separation into two different predicates, we can later separate expected with received sensing result. This separation allows us to perform a reasoning on this difference. Additionally, the predicates allow to remove the binding between expected and real sensing result if necessary. In [51] a very similar predicate was used to distinguish between expected and actual sensing value. The main difference is that we not only use the predicate to detect a fault but instead, we use the predicate $AO$ and a modified version of the formula 2.19 to define that a sensing result should be ignored.
Please note that regression for sensing actions is still possible, we refer to an example how to regress with sensing formulas [120]. In the case of progression, no general statement is possible as it depends on the progression method used and the sensing formulas. We will discuss this problem in more detail in Chapter 6.

### 2.1.5. Knowledge

The information provided by sensing is closely related to the knowledge the robot possesses. Instead of just modeling how the world looks the agent is now reasoning about what it knows about the

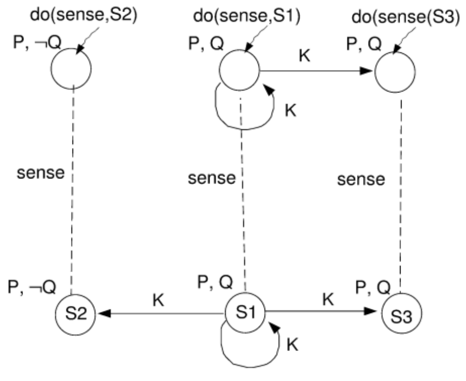

Figure 2.1.: Knowledge-Producing Actions and the K relation [120]

world. Furthermore, the robot can reason what it does not know. Thus, it can also detect the lack of knowledge and perform countermeasures, e.g. perform sensing action.
One method to define knowledge is with the possible world semantic [86]. The idea is that the robot could be in one of the several worlds. Each world differs in which properties hold. To denote that the robot does not know which world is the true one the predicate $K(s', s)$ is used. It states that the robot could be in situation $s'$ or $s$ as far as the robot knows. Thus, increasing the knowledge is caused by ruling out possible situations.
To define that a robot knows a formula $\phi$ in situation $s$ one uses the macro $\forall s'.K(s', s) \rightarrow \phi(s')$. Thus, the robot only knows a formula if it holds in all possible worlds. This reasoning causes that the robot only knows things if it has the same truth value in each possible world. We will use this reasoning to ensure that the robot only performs actions which it is certain it can conduct safely.
The semantic of possible worlds was introduced into the situation calculus in [120]. The initial situation now defines what the agent knows and as such which situations are accessible from the initial situation. After executing an action, all situations which are accessible are advanced. If the action is a sensing action, only those situations which are consistent with the sensing result are accessible for further use. Thus, the agent can gather knowledge about the world through the execution of sensing actions. The relation between accessible situations and performed sensing actions is depicted in Figure 2.1.
In situation $S_1$ the agent performs the action *sense* which asserts that $Q$ needs to hold. As $Q$ does not hold in situation $S_2$, the situation is no longer accessible.
Through the combination of knowledge and sensing action, one can now distinguish sensing and primitive action by the effect they have on the accessibility relation. Primitive actions do not change the accessibility relation. Instead, they advance all accessible situations. In contrast to primitive actions, sensing actions don't advance the situations. Instead, they change the accessibility relation. To allow an easy discussion on how the knowledge of the agent changes we assume that an action is either a primitive or a sensing action and not both.
With the definition of the knowledge of robots, the situation calculus offers all the theoretical foundations necessary to deal with actions and the changes they cause. We refer the interested reader to [116] for further information about the situation calculus.

## 2.2. History-Based Diagnosis

In the previous section, we have discussed how to tackle the frame problem. Furthermore, we have defined general conditions which need to hold to perform an action successfully. But as mentioned above the complete specification of the precondition is not possible due to the qualification problem. To address the qualification problem, we instead assume that as long as no indication says otherwise the action was executed successfully. If we make an observation which contradicts our knowledge about the world we try to detect which action was not successfully performed.
The reasoning about which action was not performed successfully is addressing the question "what happened" as it was defined in [83]. This method contrasts with other diagnosis methods which try to answer the question "what is wrong". Let's consider a simple example to point out the difference.
Let's consider our running example. The robot needs to deliver object $O_1$ from room $R_1$ to room $R_2$. After picking up the object and moving to $R_2$ the robot puts down the object. Afterward, the robot checks if the object is in the room. It discovers that the object $O_1$ is not in room $R_2$. Thus, the sensing contradicts the knowledge of the robot that the object $O_1$ is in room $R_2$. If we now ask the question "what is wrong" the answer is simply that object $O_1$ is not allowed to be in this room. But we cannot further infer where the object could be. If we ask the question "what happened" instead, we can draw a more detailed conclusion. We can conclude that either the robot was not able to pick up the object, it failed to put down the object, or somebody has snatched the object during the delivery. Depending on the case we can conclude that the object is in $R_1$, the robot still holds the object or that the object is somewhere else. Thus, through asking the question "what happened" we can draw a more detailed conclusion how the world looks.
To answer this question in the context of the situation calculus [56] proposed the concept of history-based diagnosis, which was later extended in [51]. As the name of the method implies, the history of performed actions is diagnosed.
In classical consistency-based diagnosis [113] one uses a predicate $AB(x)$ to indicate that component $x$ is faulty. If a component is faulty, it can behave arbitrarily. Thus it does not longer define the relationship which needs to hold between its inputs and outputs. As the output for a faulty component can be arbitrary, a contradicting observation of the output value can be resolved by searching for a minimal set of components (diagnosis) which need to be faulty such that no observations contradict the expected observations.
The classical consistency-based diagnosis approach contrasts with abductive diagnosis [17]. Abductive diagnosis replaces the behavior of a faulty component by a defined behavior specific for a fault mode. Thus, instead of an arbitrary behavior, a component can have a particular behavior. This specification allows a more detailed information about the behavior of the complete system after some particular fault has occurred. History-based diagnosis follows this approach but instead of replacing components by their faulty counterparts actions are replaced by their faulty counterparts. In the remainder of the thesis, we use the term history to specify the executed actions and there sensing outcomes. The term situation is utilized for the performed actions within the definitions.
To specify how the action sequence can be changed history-based diagnosis defines two predicates *Varia* and *Inser*. We slightly adapt the notations of the original paper [56] to have a uniform representation throughout the thesis. The first one is used to define when to replace an action by another action in the history. The second predicate specifies when an action can be inserted into a history.
The variation of an action is defined as.

$$Varia(\alpha', \alpha, s) \equiv \Theta_\alpha \tag{2.20}$$

The predicate defines that action $\alpha'$ is a variation of $\alpha$ in situation $s$ if condition $\Theta_\alpha$ holds. Let's consider our running example we could state

$$Varia(\alpha', put(o), s) \equiv \alpha' = putNothing \tag{2.21}$$

Thus, action *put* can be replaced by action *putNothing* in every situation to represent a possible fault. This specifies that the action has failed to actually put the object down.
The insertion of an action is defined as.

$$Inser(\alpha, s) \equiv \Theta \tag{2.22}$$

The predicate defines that action $\alpha$ can be inserted in situation $s$ if condition $\Theta$ holds. Let's consider our running example we could state

$$Inser(snatch(o), s) \equiv holding(o, s) \tag{2.23}$$

Thus, action *snatch* can be inserted into the history of performed actions if the robot is holding an object.
Using the above predicates one can define how a situation can be changed with the help of the predicate *ExtVariation*. Following the definition of [51] and [56] we define the predicate *ExtVariation* to hold as follows:

**Definition 1.**

$$\begin{aligned}
ExtVariation(S_0, S_0) &\doteq \top \\
ExtVariation(S_0, do(\alpha, s)) &\doteq \bot \\
ExtVariation(do(\alpha', s'), S_0) &\doteq Inser(\alpha', do(s', S_0)) \wedge ExtVariation(s', S_0) \\
ExtVariation(do(\alpha', s'), do(\alpha, s)) &\doteq [Varia(\alpha', \alpha, do(s', S_0)) \wedge ExtVariation(s', s)] \vee \\
&\quad [Inser(\alpha', do(s', S_0)) \wedge ExtVariation(s', do(\alpha, s))]
\end{aligned}$$

The first two equations specify the base cases with an empty history. The third equation specifies the case of an insertion into an empty history. The last equation specifies that a history can be changed by either inserting an action or by replacing the action by a variation.
Let's consider our running example. The robot moves from to room $R_1$, picks up object $O_1$, moves to room $R_2$ and puts down the object. This results in the following action sequence:

$$[goto(R_1), pick(O_1), goto(R_2), put(O_1)] \tag{2.24}$$

A possible variation using the definition from above would be:

$$[goto(R_1), pick(O_1), goto(R_2), putNothing] \tag{2.25}$$

This variation represents the case that a robot fails to put down the object after picking up the object.
As not every variation of history is possible to be realized in practice one needs to rule out those variations which can't be executed. To rule out invalid variations we first define a history of actions to be executable [116] defined through the predicate *executable* as follows:

$$executable(s) \equiv \forall a, s^*. do(a, s^*) \sqsubseteq s \rightarrow Poss(a, s^*) \tag{2.26}$$

Thus, the definition of the executable history ensures that every action within the history is possible to be performed. With the help of this definition we can rule out those variations which can't be executed. Additionally, we are only interested in variations of the history if the original history contradicts with a sensing result. We combine both criteria into the definition of an explanatory history-based diagnosis which is per [51] defined as follows:

**Definition 2.** *An explanatory history-based diagnosis for an observation $\phi$ and a history $s$ is an extended variation $s'$ of $s$ such that $s'$ is executable and $\phi$ holds in the situation after the action sequence $s'$, i.e., $\mathcal{D} \cup \{sensed\} \models \phi[do(s', S_0)]$.*

$$ExplDiagnosis(s', \phi, s) \doteq ExtVariation(s', s) \wedge executable(s') \wedge \phi[s']$$

The definition of the history-based diagnosis was extended in [51] to incorporate commonsense knowledge about the environment. The commonsense knowledge of the robot is collected a set $I$ of different invariant $i(s)$. Additionally, the predicate *Invaria* is used which is the collection of all the invariant.

$$Invaria(s) \equiv \bigwedge_{i \in I} i(s) \tag{2.27}$$

Due to the use of the common-sense knowledge the definition of a valid variation needs to be adapted. This is achieved by defining a situation to be consistent if the situation does not lead to a contradiction with the invariants imposed. Following [51] the consistency is defined by the predicate *Cons* as follows:

**Definition 3.** *A history $s$ is consistent if and only if $\mathcal{D} \models Cons(s)$ with $Cons(\cdot)$ inductively defined as:*

1. $Cons(S_0) \doteq Invaria(S_0)$
2. $Cons(do(\alpha, s)) \doteq Cons(s) \wedge Invaria(do(\alpha, s)) \wedge [AO(\bar{s}) \leftrightarrow SF(\alpha, \bar{s})]$

The definition of the consistency is recursive. The base case defines that the common-sense knowledge needs to hold for the initial situation. If the situation consists of a sequence of actions the sequence is recursively checked for consistency. During each check the invariants are checked to hold. Furthermore, with the help of the predicate *AO* and *SF* it is checked if the measurement is consistent with the expected values.
With the help of the definition of a consistent situation we can rule out those variations of a history which are not possible to be realized. But we do not only want a history which can be realized. Instead we want a variation of the history which is very likely. To define how likely a variation is, we use a function *val* which defines how likely a certain insertion or variation is. The function uses insertions or variations to define a real value greater zero ($\mathbb{R}^+$) for the likelihood of the change. The lower the value of an insertion or variation is the more likely it is that this insertion or variation has happened. With the help of this function we can define a function *cv* defining the likelihood of a complete situation per [51] as follows:

**Definition 4.** *Let $cv : \mathcal{S} \times \mathcal{S} \rightarrow \mathbb{R}^+$, $cv(s', s) = v$ with $v \geq 0$ evaluates the difference between a diagnosis $s'$ and a history $s$. cv is called the* change value *and is inductively defined as:*

1. $cv(s', s') = 0$
2. $cv(do(\alpha', s'), s) = cv(s', s) + val(Inser(\alpha', s'))$
3. $cv(do(\alpha', s'), do(\alpha, s)) = cv(s', s) + val(Varia(\alpha', \alpha, s'))$

The definition of the likelihood of a variate history is inductively defined. If the history is the same as the variation the likelihood is very high thus the value is 0. Please note that this case holds at least for the initial situation $S_0$. If the situations are different the value is increased if the difference is caused

due to an insertion or variation. The increase of the value is caused by the definition of the function *val*.
Combining the definition of an extended variation of a situation with the definition of a consistent situation with its ranking we can define a diagnosis of a situation with its associated costs. Following [51] we define a diagnosis for a history as follows:

**Definition 5.** *Let $s'$ and $s$ be histories. Let $Diag(s',s,v) \doteq ExtVariation(s',s) \wedge executable(s') \wedge Cons(s') \wedge v = cv(s',s)$, denoting that $s'$ is an extended variation of $s$ that is executable and consistent. $s'$ is a proper* history-based diagnosis *(or diagnosis for short) based on $s$ if and only if $\mathcal{D} \models Diag(s',s,v)$.*

The definition states that a diagnosis of a situation is a variation of this situation which is consistent. Furthermore, the value how likely this variation is, is used in the diagnosis. To capture all possible diagnoses one would need to treat in the worst case an exponential number of diagnoses [51]. To be more precise if we assume only finitely many objects there are exponentially many variations of an action. Furthermore, if we assume a fixed maximum length of the history, there are exponentially many variations of the history. Thus, in total one has a double exponential problem under the restricting assumptions.
As such it is not possible in practice to calculate all these diagnoses. Instead the method in [51] only calculates a pool of *n* diagnoses. Thus, regardless of the real number of diagnoses one has a fixed number of diagnoses to deal with. Each diagnosis is represented in the pool with the help of one tuple. The tuple is composed of the variate history and its change value.

**Definition 6.** *Let s be a history. Then Pool is defined as*

$$\begin{aligned} Pool(s) = \{\langle s'_1, v_1\rangle, \cdots, \langle s'_n, v_n\rangle\} \doteq & Cons(s) \wedge s'_1 = s \wedge v_1 = 0 \wedge \cdots \wedge s'_n = s \wedge v_n = 0 \vee \\ & \neg Cons(s) \wedge Diag(s'_1, s, v_1) \wedge \cdots \wedge Diag(s'_n, s, v_n) \end{aligned}$$

The definition of the pool states that if the situation is consistent the situation itself is repeated *n* times to fill the pool. Otherwise, the pool consists of *n* diagnoses. Due to this definition, the pool may include a diagnosis which is very unlikely. Thus, instead, we want to have a pool which has those diagnoses with the lowest costs. The lower the value, the higher the likelihood of the diagnosis. To prefer those diagnoses with a lower cost for chances is very common, see for example [25].
To incorporate only the most likely diagnosis into the pool we place the following condition on the pool:

$$\nexists \langle s', v'\rangle \wedge Diag(s', s, v') \wedge \langle s', v'\rangle \notin Pool(s). \exists \langle s_i, v_i\rangle \in Pool(s) v' < v_i \tag{2.28}$$

With the help of the pool, we have a fixed set of diagnoses which are most likely. One could use the pool to perform the necessary reasoning for the robot. But instead, the method described in [51] chooses one diagnosis as the preferred diagnosis and uses this diagnosis for further reasoning.

**Definition 7.** *Let s be a history. The preferred diagnosis prefDiag is defined as:*

$$\begin{aligned} prefDiag(s) = s' \doteq & \exists s'_1, v_1, \ldots, s'_n, v_n. Pool(s) = \{\langle s'_1, v_1\rangle, \cdots, \langle s'_n, v_n\rangle\} \wedge \\ & [s' = s'_1 \wedge v_1 \leq v_2 \wedge \cdots \wedge v_1 \leq v_n \vee \\ & s' = s'_2 \wedge v_2 \leq v_1 \wedge v_2 \leq v_3 \wedge \cdots \wedge v_2 \leq v_n \vee \\ & \vdots \\ & s' = s_n \wedge v_n \leq v_1 \wedge \cdots \wedge v_n \leq v_{n-1}] \end{aligned}$$

For further information, how the preferred diagnosis can be used in a robotic system we refer the interested reader to [109].

## 2.3. IndiGolog

With the help of the history-based diagnosis, one can address the problem of faulty action execution. Thus, the robot can recognize a fault during its execution and can conclude what might have happened. This reasoning allows the robot to answer queries about the current state of the world as well as future states of the world.
To do work towards a goal the robot needs not only to perform reasoning about how the world looks. Additionally, the robot needs a method to decide what would be the next action to choose to achieve the task. One method is the use of a high-level programming language.
Using the situation calculus as a basis, the high-level programming language Golog was proposed in [67]. The idea is to provide macros which can be extended into logical formulas to allow to write programs. The programs can have imperative parts such as conditions, loops or sequences. Additionally, one can use declarative parts in the language such as pick an argument or choose one program branch to follow further execution. Due to its expansion into a logical formula, one can search for a valid sequence of actions which need to be executed to achieve the task.
The expressiveness of Golog was furthermore extended in ConGolog [19]. ConGolog added language elements to deal with interrupts and concurrent executions. Additionally, the semantic was changed from a macro expansion into a transition semantic. This change has not affected the basic idea to use logic to derive a valid action sequence. Instead of the macro expansion one now performs the reasoning on the transition semantic which is defined with the help of logic.
Golog and ConGolog both plan first a whole sequence of actions which need to be executed before executing the actions. But after the planning step, no further reasoning is applied. Such an approach cannot just be used in a robotic system as the robot should react to changes in its environment. To tackle this problem IndiGolog [20] was created.
IndiGolog uses the same language constructs as ConGolog but adds the possibility to decide if only the next best action to execute should be selected or if the robot should perform a look ahead. This distinction is realized using the so-called search operator $\Sigma$. If the search operator is applied for a program, IndiGolog searches for an action sequence for the program before it executes an action. Otherwise, IndiGolog searches only for the next best action to be executed. This distinction allows switching smoothly between deliberation and reactive behavior [118].
To define a high-level program IndiGolog offers the following programming constructs [20].

- $\alpha$
  primitive action
- $\phi?$
  test/wait for condition
- $\delta_1;\delta_2$
  sequence
- $\delta_1 \mid \delta_2$
  non-deterministic branch
- $\pi x.\delta$
  non-deterministic choice of arguments
- $\delta^*$
  non-deterministic iteration

- **if** $\phi$ **then** $\delta_1$**else** $\delta_2$ **endIf**
  conditional branching
- **while** $\phi$ **do** $\delta$ **endWhile**
  conditional loop
- $\delta_1 \mid\mid \delta_2$
  concurrent execution with equal priority
- $\delta_1 \rangle\rangle \delta_2$
  concurrent execution where $\delta_1$ has a higher priority
- $\delta^{||}$
  concurrent iteration
- $\langle\phi \rightarrow \delta\rangle$
  interrupt
- **proc** $P(x)$ $\delta$ **endProc**
  procedure definition
- $P(\Theta)$
  procedure call
- $\Sigma(\delta)$
  search operator

To execute an IndiGolog program a transition semantic is used. This transition semantic is defined through the two predicates *Trans* and *Final*. $Trans(\delta, s, \delta', s')$ defines that the program $\delta$ in situation $s$ can be further executed to remaining program $\delta'$ and the resulting situation is $s'$. $Final(\delta, s)$ defines that program $\delta$ can legally terminate in situation $s$. To define these two predicates, we will use an additional program construct *nil* defining the empty program.
We follow the definition in [20] to define the predicate *Trans* as follows:

- $Trans(nil, s, \delta', s') \equiv \perp$. The empty program cannot lead to any further execution.
- $Trans(\alpha, s, \delta', s') \equiv Poss(\alpha[s], s) \wedge \delta' = nil \wedge s' = do(\alpha[s], s)$. If the action precondition holds the action can be executed and yield an empty remainder program. Furthermore, the resulting situation is created by extending the current situation by the executed action.
- $Trans(\phi?, s, \delta', s') \equiv \phi[s] \wedge \delta' = nil \wedge s = s'$. If the condition holds the program can proceed to the empty program.
- $Trans(\delta_1;\delta_2, s, \delta', s') \equiv \exists\delta''.\delta' = \delta'';\delta_2 \wedge Trans(\delta_1, s, \delta'', s') \vee Final(\delta_1, s) \wedge Trans(\delta_2, s, \delta', s')$. The program performs a transition in $\delta_1$ if this is possible. If $\delta_1$ can terminate the program performs a transition according to $\delta_2$.
- $Trans(\delta_1 \mid \delta_2, s, \delta', s') \equiv Trans(\delta_1, s, \delta', s') \vee Trans(\delta_2, s, \delta', s')$. The program takes either the transition of $\delta_1$ or the transition of $\delta_2$.
- $Trans(\pi x.\delta, s, \delta', s') \equiv \exists t.Trans(\delta|^x_t, s, \delta', s')$. The program performs a transition if a valid instance for the parameter $x$ can be found. Please note that $\delta|^x_t$ substitutes every occurrence of $x$ with $t$ in the program $\delta$.

- $Trans(\delta^*, s, \delta', s') \equiv \exists \delta''.\delta' = \delta''; \delta^* \wedge Trans(\delta, s, \delta'', s')$. The program performs a transition in $\delta$ to yield $\delta''$. This program is further executed till it is finished. Afterwards the loop can be started again. Thus, the loop is unrolled and executed.
- $Trans(\textbf{if } \phi \textbf{ then } \delta_1 \textbf{else } \delta_2 \textbf{ endIf}, s, \delta', s') \equiv \phi[s] \wedge Trans(\delta_1, s, \delta', s') \vee \neg\phi[s] \wedge Trans(\delta_2, s, \delta', s')$. If the condition holds $\delta_1$ is considered for the transition. If the negation of the condition holds $\delta_2$ is considered for the transition.
- $Trans(\textbf{while } \phi \textbf{ do } \delta \textbf{ endWhile}, s, \delta', s') \equiv \exists \delta''.(\delta' = \delta''; \textbf{while } \phi \textbf{ do } \delta \textbf{ endWhile}) \wedge \phi[s] \wedge Trans(\delta, s, \delta'', s')$. If the condition holds the transition is proceeds further with the inner program. This program is executed till it finishes. Afterwards the loop can be started again. Thus, like the non-deterministic iteration the loop is unrolled.
- $Trans(\delta_1 \ || \ \delta_2, s, \delta', s') \equiv \exists \delta''.\delta' = \delta'' \ || \ \delta_2 \wedge Trans(\delta_1, s, \delta'', s') \vee \exists \delta''.\delta' = \delta_1 \ || \ \delta'' \wedge Trans(\delta_2, s, \delta'', s')$. Thus, either a transition is taken for the program $\delta_1$ or a transition is taken for the program $\delta_2$. In either cases the remainder of the program is executed further in concurrently with another program.
- $Trans(\delta_1 \ \rangle\rangle \ \delta_2, s, \delta', s') \equiv \exists \delta''.\delta' = \delta'' \ || \ \delta_2 \wedge Trans(\delta_1, s, \delta'', s') \vee \exists \delta''.\delta' = \delta_1 \ || \ \delta'' \wedge Trans(\delta_2, s, \delta'', s') \wedge \nexists \bar{\delta}, \bar{s}.Trans(\delta_1, s, \bar{\delta}, \bar{s})$. Thus, program $\delta_1$ is further progressed if a transition is possible. Otherwise program $\delta_2$ is used for further transitions.
- $Trans(\delta^{||}, s, \delta', s') \equiv \exists \delta''.\delta' = \delta'' \ || \ \delta^{||} \wedge Trans(\delta, s, \delta'', s')$. Thus, the program performs a transition by executing $\delta$ and running the concurrent loop in further concurrent execution.

Due to the definition of *Trans* we can decide which transitions the program can take. Please note that not all program constructs need a separate transition semantic as some of these constructs can be rewritten. We refer the interested reader to [20] for further information. Beside the predicate *Trans* the predicate *Final* needs to be defined. Following the definition in [20] we define the predicate as follows:

- $Final(nil, s) \equiv \top$. The empty program is a legal termination.
- $Final(\alpha, s) \equiv \bot$. An action is never considered as a legal termination.
- $Final(\phi?, s) \equiv \bot$. A condition is never considered as a legal termination.
- $Final(\delta_1; \delta_2, s) \equiv Final(\delta_1, s) \wedge Final(\delta_2, s)$. The sequence is a legal termination if both parts of a sequence can terminate legally.
- $Final(\delta_1 \mid \delta_2, s) \equiv Final(\delta_1, s) \vee Final(\delta_2, s)$. If either of the program can terminate the non-determinism branch can terminate legally.
- $Final(\pi x.\delta, s) \equiv \exists t.Final(\delta|_t^x, s)$. The program can terminate if a valid instance for the parameter $x$ can be found. Please note that $\delta|_t^x$ substitutes every occurrence of $x$ with $t$ in the program $\delta$.
- $Final(\delta^*, s) \equiv \top$. A legal termination is always possible. Thus, the non-deterministic loop can terminate instead of iterate.
- $Final(\textbf{if } \phi \textbf{ then } \delta_1 \textbf{else } \delta_2 \textbf{ endIf}, s) \equiv \phi[s] \wedge Final(\delta_1, s) \vee \neg\phi[s] \wedge Final(\delta_2, s)$. If the condition holds the program can terminate if $\delta_1$ can terminate. If the negation of the condition holds $\delta_2$ is used to decide legal termination.

- $Final(\textbf{while } \phi \textbf{ do } \delta \textbf{ endWhile}, s) \equiv \neg\phi[s] \vee Final(\delta, s)$. The loop stops if either the condition does not hold any longer or the body of the loop can terminate.
- $Final(\delta_1 \mid\mid \delta_2, s) \equiv Final(\delta_1, s) \wedge Final(\delta_2, s)$. The program can only terminate if both concurrent branches can terminate.
- $Final(\delta_1 \rangle\rangle \delta_2, s) \equiv Final(\delta_1, s) \wedge Final(\delta_2, s)$. The program can only terminate if both concurrent branches can terminate.
- $Final(\delta^{||}, s) \equiv \top$. The program can always legally terminate thus no execution can be performed or further parallel executions can be performed.

We gather the the definitions of the predicate *Trans* and *Final* together with the program reified as terms (See [118] for further details) in the set $\mathcal{C}$. Using this set of definitions we can extend the basic action theory to decide through a reasoning process which action to perform next. We define the extended basic action theory $\mathcal{D}^*$ as follows:

$$\mathcal{D}^* = \mathcal{D} \cup \mathcal{C} \cup \{Sensed\} \tag{2.29}$$

Where $\{Sensed\}$ defines all the sensing results. Additionally the definition of *SF* is part of the set $\{Sensed\}$. Thus, the extended basic action theory contains the information the robot has about the world, how the world changes through actions, how to choose the next action and the high-level program which should be used to achieve the task.
With the help of the extended basic action theory the robot can decide which action to perform next by following the *online execution semantics*. Following [20] this semantics decides how to proceed from a given program $\delta$ in each situation $s$ as follows:

1. if $\mathcal{D}^* \models Final(\delta, s)$ terminate the program.
2. if $\mathcal{D}^* \models Trans(\delta, s, \delta', s)$ proceed with program $\delta'$.
3. if $\mathcal{D}^* \models Trans(\delta, s, \delta', do(\alpha, s))$, execute action $\alpha$ and proceed with program $\delta'$.

As we discussed above IndiGolog also provides the possibility to plan ahead. This is realized through the search operator $\Sigma(\delta)$. The search operator is defined using the *offline execution semantics*. This semantics mimics the behavior of the original Golog language. Following [20] the semantics is defined with the help of the predicate $Do(\delta, s, s')$ as follows:

$$Do(\delta, s, s') \doteq \exists \delta'. Trans^*(\delta, s, \delta', s') \wedge Final(\delta', s') \tag{2.30}$$

Where $Trans^*$ is the reflexive transitive closure of the transition predicate *Trans* and is defined as follows:

$$Trans^* \doteq \forall T.[\cdots \rightarrow T(\delta, s, \delta', s')] \tag{2.31}$$

Where ... is the conjunction of the universal closure of the inductive definition of a possible transition which is defined as follows:

$$\begin{aligned} \top &\rightarrow T(\delta, s, \delta, s) \\ Trans(\delta, s, \delta'', s'') \wedge T(\delta'', s'', \delta', s') &\rightarrow T(\delta, s, \delta', s') \end{aligned} \tag{2.32}$$

Thus, the definition states that a program transition is performed with the help of the predicate *Trans* till a program $\delta'$ is found with the corresponding situation $s'$ which can legally terminate.

Using this definition of the *offline execution semantics* one can define the transition semantic of the search operator as follows:

$$Trans(\Sigma(\delta), s, \Sigma(\delta'), s') \equiv Trans(\delta, s, \delta', s') \wedge \exists s^*.Do(\delta', s', s^*) \tag{2.33}$$

Thus, the transition of the search operator performs only a transition if it is possible and leads to a final state $s^*$ which allows a legal termination.

Chapter 3

# Related Research

In the introduction, we have motivated why a robot needs to consider that an action may fail in its reasoning process. During the execution of an action, the action may fail and produce a different result than expected. If the robot does not consider the different result in the execution, it may perform wrong or even dangerous actions. Thus, to properly incorporate this awareness into a robotic system is the main objective of the thesis. But this thesis is not the first attempt to make a robot aware of faults during its execution. In this chapter, we will discuss related research which addressed some problems of making a robot aware of its fault. We will focus our attention on related research in the area of diagnosis of action sequences and action languages. We mainly concentrate on the situation calculus and the usage of the action languages to serve as the high-level control. Please note that this chapter discusses the related research which is common to all chapters. Additional related research specific for each chapter can be found in the following chapters.
This chapter is structured as follows: the next section will discuss methods to detect faulty components of a system which can change through actions. As the working of components is not the most important aspect of a high-level fault detection mechanism we discuss how to detect if a goal can still be reached in Section 3.2. This section is followed by a section discussing how such approaches can be incorporated into the fluent calculus. Afterwards in Section 3.4 we will discuss methods which allow managing the belief of the robot. Following the idea of a managed belief, we show how faulty actions are handled in the situation calculus and how it leads to a belief management in Section 3.5. As the situation calculus uses second order axioms to define its semantic one needs to perform the reasoning accordingly. To avoid such higher order logic axioms particular logics can be utilized which allow reasoning about changes of actions and belief natively. We will discuss such approaches in Section 3.6. The approach discussed so far deals with possible alternative situations or world states. This is often very complex. To avoid such a complexity one may just define possible alternatives on the level of fluents. We will discuss the impact of such a fluent based reasoning in Section 3.7. Besides defining how the robot thinks the world looks like one also needs to address how faults are dealt with in the high-level control. In Section 3.8 we will discuss different methods to deal with faults in the high-level control. Finally, in Section 3.9 we will discuss how to monitor the execution to detect faults.

## 3.1. Detecting faulty components

We start our discussion with the work proposed in [85]. The method tackles the problem to detect a fault in a system which has static constraints as well as undergoes dynamic changes. One has (beside the description of the system) also a sequence of actions which were performed and which describe the changes of the system. Furthermore, observations are generated for the system. The observations are compared to the prediction the system makes according to its description. The system description, as well as the description how the system changes after an action is performed, is done with the help of the situation calculus. To describe a fault in the system special predicates are used which indicate if a part of the system has failed. Initially one assumes that all the system components function correctly. If an observation is now in contradiction with the expectation, one inserts actions which cause a component to be faulty. Thus, the changed history explains the observation. This contrasts with our approach as we diagnose action faults and not component faults. The method we use as a basis for the reasoning is an extension, as it allows to model action faults but also actions which set components faulty. Thus, the method proposed in [85] can be integrated into our system easily. To describe not only the system and how it changes but also the procedure controlling it the method was extended in [84]. To specify the control procedures, Golog was used. As a faulty component is specified through a predicate, this predicate can be utilized within the program to perform certain actions if a component is faulty. This contrasts with our approach as we use the knowledge about the faults in the background for the reasoning. Thus, the robot does not need to directly reason which action has failed but instead which conclusion can still be drawn from the current knowledge about the situation.
A similar approach to detect faulty components was proposed in [5]. The method uses a specific action language to describe system constraints as well as action changing the system. As in the previous case, one uses a description of a dynamic system which can fail. To describe a fault of a component specific predicates are used. Furthermore, specific actions are used to specify that a component became broken. Like [85] a history which explains the observations made is used in [5] as a diagnosis. Besides this diagnosis, a planning mechanism was described which allows creating a conditional plan to derive a more detailed diagnosis. The idea is to create a conditional plan which performs sensing actions and depending on the sensing outcome the plan proceeds with a certain branch. As sensing actions can rule out certain explanations based on their sensing result, one can use a sequence of sensing actions to derive a finer grained diagnosis. With the help of the conditional plan, one can derive a more detailed diagnosis regardless of future sensing results. We will show a similar mechanism which is performed online to reduce the uncertainty a robot has about its world. As in the previous case, the method proposed in this thesis is not restricted to diagnose a system component to be faulty. But instead it addresses the problem of faulty action executions.

## 3.2. Detect whether the goal can be reached

The above methods have tackled the problem if a component of a system is faulty. But for a robotic system, the major concern is not which components are faulty but if there is still a chance to reach the desired goal with a given plan. One way to address this problem was proposed in [34]. The method uses the current state of the world together with the plan to check if the goal can still be reached. The plan may only impose a partial order on the action execution, thus allowing the system to perform different trajectories to reach the goal. Within this context, a diagnosis is defined as a state which was reached but does not allow the robot to reach its goal. Thus, any further trajectory does not yield to a goal state. An alternative definition of the same method was proposed in [33]. One definition only considers the history how a state was reached whereas the other definition uses the state itself. Furthermore, it was shown in [33] that if all histories and states which don't reach its goal have the same initial state the two different definitions are equal. Thus, it is sufficient for most cases just to check if the current state cannot reach the goal any longer, but the previous state could do so. To perform this test, the method expects that the complete state can be observed. Furthermore, no dedicated sensing actions were assumed, thus not allowing to model faults in the perception. This contrasts with the approach presented in this thesis that does neither assume complete observation of the state nor faultless sensing. Both aspects are important for robotic systems. No one can assume that the full state can be observed all the time for non-trivial tasks. Furthermore, one needs to consider that a sensing action may fail, thus yielding a wrong observation.

## 3.3. Fault handling in the fluent calculus

With the above methods, we have addressed either the problem of a system component to be faulty or the problem of being in a state that does not allow further reaching the goal. In a robot system, the major task is to reach a given goal. To do so, actions are performed which may be influenced by the system components. Thus, if a system component is faulty, an action may not be possible to be performed. In [142] a method was proposed which tackles this problem. If no contradicting observation is made, one assumes that the system components are not faulty and thus allow to perform an action without a fault. Thus, per default, an action is executed successfully. But if a fault of an action has been observed or an observation indicates a contradiction between expectation and real world one can state that the system components which were needed for the action were faulty. This is done with the help of a predicate which specifies that the system components used in the precondition are not abnormal. In the case of a fault, one uses exogenous actions to set these predicates to false. As in the case of [85] or [5] the exogenous actions are used to model fault occurrences. Using these actions, one states that a component is broken till a repair of this component is performed. Additionally, the method proposed in [142] allows to model faults of one action execution only. Thus, an action can accidentally fail. This allows to model faults during the action execution which is not caused due to a permanent problem in the robotic system. To achieve this, a default rule is applied stating that the action execution is successful if not detected otherwise. Instead of exogenous actions default logic rules are used to reason about these faults. Summarizing the method presented in [142] allows to model permanent as well as intermittent faults during the action execution. Both faults are modeled with the help of default rules. As multiple explanations are possible, these default rules are prioritized, thus allowing that only a small set of explanations be used during the further reasoning. The proposed method uses the fluent calculus [138] to perform the reasoning. The difference between the fluent calculus and the situation calculus, presented above, is that the fluent calculus uses a description of a state whereas the situation calculus use a description of the history of performed actions to perform its reasoning. As it was shown in [140] these two representations can be translated into each other for many interesting cases. With the help of this formal representation the robot high-level control language FLUX [143] is defined. FLUX allows controlling a robot towards achieving a certain goal. Thus, FLUX is like Golog which can also be translated into each other as presented in [121]. To incorporate the reasoning of failing actions a combination of FLUX and the method presented in [142] was discussed in [76]. The main difference to the approach presented in this thesis is how the reasoning which action has failed is integrated into the high-level control of the robot. The method presented in this thesis allows to distinguish the case where the robot knows that no further execution is possible from the case where it only lacks the knowledge to make a decision. This furthermore allows

gathering knowledge if required. A method to perform this acquisition is also presented in this thesis. In [41] the fluent calculus [138] and the corresponding action language FLUX [143] was used together with several extensions to create an intelligent robotic system. The robotic system is comprised of several low-level components performing reactive behavior to control the hardware of the robot. Furthermore, a high-level control manages which actions should be performed by the robot. The high-level control is realized through FLUX with several extensions. The extensions comprise the method described in [142] to specify a fault of an action. Additionally, the knowledge of the robot can be represented with the method described in [141]. The method allows defining the knowledge of the robot with the help of a possible world semantics. Thus, the robot describes its knowledge by considering several different worlds which are possible. Through the execution of sensing actions, different worlds can be ruled out. Finally, the system presented in [41] is also uses fluents with a time stamp attached to them to remove knowledge which is too old, thus, allowing the robot to forget old knowledge. To allow the high-level control to react to changes in the environment the approach simply performs a planning step after receiving new information. Thus, the robot plans each time a new perception enters the knowledge base, allowing the robot to maximize its success. One difference to our approach is that the method does not define the knowledge of the agent together with the faulty action execution, thus distinguishing between the information of the knowledge of the robot and the effects of faulty action execution. This also leads to the problem that it is not simply possible to deal with several explanations of an action failure. In contrast, this is a key part of the method presented in this thesis. This also allows the method presented in this thesis to gather active knowledge if knowledge is lacking which is not simply possible in [41]. Finally, the method presented in this thesis uses a high-level program to steer the robot behavior. It is left to the programmer of this high-level control how much freedom the robot has, to choose an action, thus allowing a balance between planning and simple imperative procedures. To allow a reactive behavior to changes in the environment an automatic procedure to derive reactive programs from sequential ones is described in this thesis. This makes it possible that the programmer still has control over which action the robot takes and how much planning it performs to allow a fast and successful execution.
To incorporate the belief of a robot into the fluent calculus the method presented in [57] can be used. The method uses a ranking of different possible situations to describe which state of the world the robot believes to hold. The lower the ranking the more likely the world is to hold. The belief of the robot is formed by those worlds which have the lowest ranking. Furthermore, as sensing actions are executed the ranking of the worlds is changed. This is done by reducing the ranking of those worlds which predict the sensing result and increase the ranking of those worlds which do not predict the sensing result. Thus, the robot believes those worlds to hold which correspond best to the sensing results. The major contrast between this approach and the approach presented in this thesis is the ranking which is used. This thesis represents the likelihood of a fault instead of how likely a particular world is. This imposes that the method presented in this thesis changes the ranking differently as the last sensing action may also have failed.

## 3.4. Fault handling through belief management

The idea to form the belief of the agent according to the fault it experiences was already outlined in [134]. The idea was to derive a knowledge base per fault and only consider those facts to be true which hold in all these knowledge bases. Furthermore, the idea also discussed that if no further execution can be made one should perform active knowledge gathering to allow the robot to execute its program further. One can see this work as an important source of inspiration for which problem needs to

be tackled in this thesis. The ideas presented in [134] where formalized in many ways in this thesis. Furthermore, through this formalization, it was possible to derive theoretical results and methods from implementing the ideas outlined in [134].
As a first attempt to tackle the ideas presented in [134] the method shown in [51] can be mentioned. The method calculates a diagnosis according to the executed actions. This diagnosis considers that an action can fail in particular ways. Each of these diagnoses is ranked according to their likelihood. The more likely a diagnosis is the lower the ranking. The reasoning is performed, by choosing one diagnosis which has the lowest ranking. This contrasts with the method presented in this thesis which uses all the lowest ranked diagnoses to form the belief of the robot. Thus, also allowing to reason about which actions are safe to be performed and can be used to increase the knowledge of the robot. The method was later extended in [110] to allow more efficient diagnosis calculation. A summary of different improvements is described in more detail in [109].

## 3.5. Fault handling in the situation calculus

As the approach presented in [51] our method is also based on the situation calculus. The situation calculus offers an easy solution to the frame problem and is often studied to form the belief of robots. In an early work [119] the robot belief was modeled through the predicate $K(s', s)$ stating that the robot could be in situation $s'$ if it considers $s$ as the current situation. To reason about faulty actions, the first step was to change this static relation to a relation specifying a ranking among these situations. Such a method was proposed in [129]. Instead of just stating the relation between situations each situation has a plausibility value assigned. This value allows distinguishing between more and less plausible situations. The set of situations with the lowest ranking were used to form the belief of the robot. Sensing actions are used to change the belief by ruling out those situations which are inconsistent. Thus, neither action failures nor sensing failures were considered, which were tackled later by other methods. But it is important to mention that the ranking of a situation is one central part which was used later to incorporate failing actions into the belief of the robot. Although the approach is not able to deal with faults it has been applied to a robotic system [99] to deal with imprecise or contradicting information in the initial situation.
One attempt to represent failing actions using the situation calculus was presented in [3]. The approach extended the situation calculus such that action can be non-deterministic. Thus, one does not consider only one successor situation but instead a set of successor situations. This allows modeling faulty action execution simply. With the help of sensing actions, the situations which were not consistent were ruled out. The approach uses all situations which are not ruled out to form the robot's knowledge. Thus, the robot gets either overly cautious or too brave, depending if all faults are modeled as possible successor situations. To address this problem, one can assign a likelihood for each situation. Afterwards one can incorporate this likelihood into the reasoning process. Later, the work was extended in [8] to deal with probability distributions which are continuous or discrete, allowing a variety of probability distributions to be used to model action failures. Furthermore, the extensions allow using continuous fluents, thus making it possible that an action defines the result of a continuous function in a probabilistic manner. Although this approach allows to incorporate failing actions and their likelihood it loses the simplicity of a ranked history which forms the belief of the robot, as one needs to perform a probabilistic reasoning to answer a query. The approach was later extended in [9] and in [10] to specify a high-level program semantics. This semantics allows programming the robot using the reasoning with the probabilistic belief. The main difference to the semantics presented in this thesis is that the semantics of this thesis ensures proper safe transitions whereas the semantics

of [10] only allows querying the belief in the program but does not use this belief for the program transition.
To address wrong sensing information a simple method based on the ranking of histories was presented in [127]. As in [129] one assumes the initial situation specifies several situations which determine different alternatives. Furthermore, each of the situations has a ranking assigned. The belief is formed by those situations which have the lowest ranking. The ranking is given by the previous value and the difference between expected and real sensing. To define the difference between expected and actual sensing two separate predicates are used. To specify the expected sensing the predicate *SF* is used. This predicate is defined as discussed in [129]. The second predicate *SR* defines the sensing result of the robot. If both predicates have the same truth value, the sensing result is as expected. In such a case the ranking of the situation is reduced such that the minimum ranked situations are those who predicted the sensing correctly. Otherwise, the ranking is increased by one, thus indicating a wrong prediction. This allows that a sensing action can be incorrect as it only reduces the likelihood if the sensing action is not as expected. But it does not impose that a situation is ruled out completely. A similar approach was presented in [74] which only differs in the value which is used to increase wrong sensing results. In contrast to the approach which is used in this thesis the actions performed are not allowed to fail. Additionally, the ranking is different as the method proposed in this thesis focus on the failed actions instead of ranking the last sensing action to be predicted by the belief.
Another method to address wrong sensing was presented in [32]. Instead of ranking different alternatives the method uses the following observation: After performing a sensing action, the robot should believe the sensing action result. If the robot only senses a fluent value, the observations imply that the robot needs to believe a certain truth value for a fluent. Thus, instead of calculating a complex belief, one could simply set the value of the fluent to the sensing result. This allows overriding the expectation with the latest sensor measurement. Although this approach does not perform a belief management as it was done in the previous case one can argue that the robot uses the most recent results of the sensing actions. Thus, if the robot performs sensing actions, often enough the belief of the robot is consistent with the real world. This contrasts with our approach as we do not assume that the robot needs to perform frequently sensing actions to ensure consistency. Instead, we use a costlier reasoning with the help of the belief to ensure this consistency.
With the above presented methods, one could address the problem of failing actions. But often one also needs to consider that the environment is changed by another agent without notification. Thus, the environment changes and the belief of the robot needs to reflect these changes with the help of exogenous events properly. In [128] a method was presented which allows dealing with such exogenous events. The method allows a situation to be extended by exogenous events. These extensions allow that events which were not observed can be inferred if otherwise the situation would be inconsistent. To form the belief, one uses a ranking on the possible situations preferring the initial ranking of possible situations. If the initial ranking is the same, one prefers shorter situations to longer situations. Thus, only those exogenous actions are considered which are necessary to make the situation consistent. As in the previous cases, the lowest ranked situations are used to form the belief of the robot. The main difference to the approach employed in this thesis is that the approach is restricted to exogenous events only. Thus, the approach does not allow faulty actions, which is also addressed by the method used in this thesis.
To combine the ability to deal with failing actions together with a simple representation of the belief the method proposed in [29] can be used. It specifies how a situation can be altered in case the original situation is inconsistent. To create this alternative one considers fault models of different actions. Each fault model has associated how it impacts the ranking of the situation. As in the previous cases, those situations which have the lowest ranking are the most plausible ones. Using the most plausible

situations, the belief of the robot is formed. Through this specification, the belief of the robot can be kept consistent but still prefer simple situations to explain inconsistencies. Additionally, through the execution of sensing action, the belief is changed as those situations which are inconsistent with the sensing actions are removed. Thus, the robot belief, the sensing result or the sensing action is considered to be faulty. With the help of the ranking proposed in [29] it is ensured that the latest sensing result is always believed, thus allowing to show the strong relation of the method to other belief revision methods. The method was changed in [30] to impose no restrictions on the way the ranking is created, thus making it possible that the robot does not believe the sensing result after executing a sensing action. This might be preferable in some situations where the sensing action is not very reliable, or the other actions are at least as reliable as the sensing action. The main difference to the approach used in this thesis is that no exogenous events are allowed in [30], whereas the method applied in this thesis also deals with exogenous events. Additionally, the method used in this thesis uses invariants of the environment to detect inconsistencies.
In [36] a method was presented to reason about the belief within a multi-robot setting. The belief is formed with the help of a plausibility ranking of different situations. The plausibility is ranked according to the plausibility of the previous situation and the performed action. The performed action has a higher impact on the plausibility than the ranking of the previous situation. This allows to model faulty actions as usual; a faulty action is not as plausible as a normal action. Thus, one prefers situations with normal actions instead of faulty ones. If later a sensing action detects a fault the plausibility ranking is changed, and a faulty action can now become more plausible, than a correctly executed action. To perform an efficient reasoning, a method to progress the knowledge base was presented in [37] which builds on the definitions of [36] for an agent's belief. The method used forgetting to forget the fluents which are settled by the action. The main difference to the approach employed in this thesis is that the approach used in this thesis allows dealing with exogenous events. Thus, allowing other agents to change the environment without notification. Additionally, invariants which need to hold for the environment are used to detect a fault as early as possible in the method applied in this thesis.

## 3.6. Logics for actions and belief

The above methods rely on a representation of the belief of the robot in the situation calculus. Although this allows a sound definition it imposes to use the reasoning associated with the situation calculus. This implies that one needs to deal with either a second order definition or needs to use regression or progression to answer queries of the current state. Furthermore, regression and progression are often not applicable or complex to perform if the belief of the robot is considered. To address the problem of a complex reasoning, an alternative logic was proposed in [65]. The logic $\mathcal{ES}$ defined in [65] captures most of the expressions of the situation calculus as it was shown in [66]. This is of special interest as the logic $\mathcal{ES}$ defines the knowledge of the robot, thus allowing to perform reasoning about the knowledge natively in the logic. Later, the logic was extended in [124] to deal with a belief of the robot which can change over time. The belief of a robot is defined by the set of worlds which have the lowest ranking. The logic itself defines how to interpret a sentence using the robot beliefs. As in the case of [129] the ranking is changed in such a way that the robot believes the last sensing result. Additionally, the logic proposed in [124] also allows defining that a robot only knows a certain set of beliefs, thus making it possible to simplify the reasoning about what the robot knows and what it does not know. Finally, the logic allows specifying conditional belief such that one can state the robot believes $\psi$ if it believes $\phi$. This further simplifies the definition of the belief of the robot. As the logic

is designed to perform the reasoning one would need a proper implementation of the logic. To avoid a separate implementation of the logic and reuse existing automatic logical reasoning mechanisms a mapping was shown in [125] to first-order logic. First, the formula which is queried is regressed back to the initial situation. Afterwards, the initial knowledge base together with the query is translated to first order to answer the query. To do so, one exploits that a robot defines its belief through only believing a certain set of formulas. Together with the properties of conditional belief, the mapping can be performed allowing to reuse existing automatic logical reasoning mechanisms. As an important remark, one needs to consider that regression for the method presented in [129] would not be simply possible as the belief of the robot is defined by quantifying over situations, thus the standard regression operator cannot be used. As regression, can get inefficient for long histories of executed actions a method to perform progression was shown in [126], thus allowing to progress the belief of the robot and use the progressed belief to answer the query. Through the usage of progression, the efficiency of a robot using a belief is significantly increased. Thus, we will sketch a method how to perform progression for the belief of the robot as we define the belief in this thesis. We refer the interested reader to [123] for further details how the logic proposed in [124] relates to other logical frameworks and how to perform regression and progression. The major difference to the approach used in this thesis is how the belief is formed as the method proposed in [124] defines the belief of the robot in the initial situation only. Afterwards, the belief is revised according to the sensing actions. Thus, neither faulty actions nor failing sensing actions are addressed. Although the results of regression and progression are interesting to perform, an efficient reasoning for robots with belief, they cannot be simply carried over to a belief which is formed by considering failing actions.
One way to incorporate failing actions is through the use of stochastic actions. To define the effect a stochastic action has on the belief the method of [65] was extended with probabilistic extensions in [45]. This allows to perform a reasoning as it was presented in [3] with the efficiency of [65]. The idea is to assign each action a set of actions which can also be the result of the execution. If an action is executed, one of the associated actions is executed in reality. For each of these actions, one defines a probability. This allows specifying that it is more likely that the robot performs the correct action, or a slight mistake, than an entirely wrong action. Additionally to the stochastic action, one uses a probability distribution about the different worlds which define the knowledge of the robot. After performing an action, these probability distributions are combined to specify how likely a certain sentence is, thus allowing to reason how likely it is that a query is true after executing a sequence of actions. To use this information in the reasoning process of the robot one would need to use a probabilistic reasoning to answer queries which we want to avoid due to its complexity.
Another logic which can deal with beliefs was proposed in [70]. The idea is that one only performs reasoning about the belief of a robot with the focus to choose a simple reasoning and a reasoning yielding all implications which are made by the belief. This is done by defining a belief operator with a certain level. The reasoning of the belief operator is based on splitting the formulas if these formulas are not simple clauses. Each of these splitting operations reduces the remaining possible splittings. Thus, a higher level of the belief operator allows drawing a more implicit conclusion of the belief as further splittings are allowed. Due to this limitation, the reasoning can be performed efficiently even if one specifies the belief of a robot. This reasoning was used in a robotic system, by extending the logic as described in [14] to also incorporate action executions. The reasoning is performed with the help of a regression operator such that a query can always be answered in the initial situation. To respond to the query in the initial situation the semantics proposed in [70] is used. Besides the reasoning also a Golog variant is defined in [14]. The variant uses the belief of the robot to determine if a formula holds. This allows that the test conditions in the Golog variant are answered by the belief of the robot. Thus, the robot performs an execution which is belief-aware. As the Golog variant is

applied to a robot system one uses a dedicated search operator to plan ahead only for those parts of the program which are necessary. The search operator does not only allow to define which part of the program should be planned but also specifies how much effort is applied in the reasoning process. As the belief is defined by different levels, the definition of the reasoning effort in the search operator is done through the level of the belief used. Thus, one may find a plan with a low-level of the belief where less effort is spent in the reasoning process. Additionally, if one needs to find a plan one can define that the search operator uses the highest belief level possible, thus, using the complete inference of the belief to find the plan. Another variant to use the logic proposed in [70] for a robotic system was presented in [35]. The method uses progression to change the knowledge base after an action is executed. This allows answering the query always with the currently progressed database using the semantics defined in [70]. Although the method presented in [14] as well as in [35] proposes a Golog variant which uses the belief of the robot to determine the actions the robot should be performed. The limitation is that the method is based on [70] which specifies the belief through a fixed set of possible worlds at the initial situation.

## 3.7. Uncertainty about fluents instead of possible worlds

Besides the use of a specific logic to simplify the reasoning about the belief of a robot one can also address the problem by defining the belief differently. The above methods used a possible world semantics [86] to define the belief. This means that one uses different situations to represent different possible worlds the robot could be in. Each of these worlds is one possibility, and the robot believes only those things which hold in all of these worlds. Thus, the reasoning needs to consider all these worlds which cause a complex reasoning task. A different approach to model the belief of the robot was used in [31]. The idea is instead of defining different worlds for the belief one defines the belief only on the level of a fluent. Thus, the robot believes a certain truth value of a fluent or not. The belief of the robot is now formed by the belief for each fluent. To answer a query one replaces the fluents in the query with their belief counterpart. Thus, the answer is based on the belief of the robot. As the belief is defined for each fluent, the change of the belief is defined in the successor state axioms of an action, thus allowing simply to modify the belief through an action. But this imposes a constraint on the sensing actions as the successor state axiom can only change one fluent value. Thus, one cannot simply model a disjunctive sensing action. To analyze the restriction imposed by this form of belief "Cartesian situations"[101] were proposed in [101]. A Cartesian situation defines a possible world semantics such that the difference between two possible worlds is only defined by one fluent term. This poses a strong restriction as only one fluent term can be different between the worlds but allows to model the belief of the robot on the level of a fluent. To be more precise if one considers a Cartesian situation, the possible world semantic yields the same results as if one uses only the belief of a fluent as was also shown in [104]. To preserve this property in the presence of performed sensing action one needs to restrict sensing actions to conditional sensing of fluents. Additionally, each fluent used in the condition needs to be sensed by the same action. Thus, the sensing action can sense a set of fluents which depend on each other in a noncyclic way. Besides the sensing actions also the action which changes the world needs to be restricted to ensure that the Cartesian situation is preserved. In [101] different classes of actions where shown which preserve the Cartesian situation. The restrictions mainly impose that the used variables of a fluent are defined in the arguments of the action which change the fluent. Additionally, it poses constraints on the condition which allows the fluent to be changed. As an important effect of this restriction one preserves not only the Cartesian situation property but one could also perform progression on these theories, thus allowing an efficient implementation of the method. Such an effective implementation was shown in [102] and [103]. Due to the structure of the Cartesian situation and the progression property, one can use several databases to represent the belief of the robot. This efficient database implementation allows creating a planning system on top of it. The planning system can construct a plan which depends on the belief of the robot. Thus, one can also incorporate sensing actions into the plan and state conditions depending on the sensing result of the plan. Although the planning system is very powerful and the underlying belief is changed by the sensing action, the method does not consider failing actions. This imposes that one can use the planning system to plan ahead for the agent but can't use the method during the execution as the belief of the robot would not be adequately changed in case of a faulty action execution. Additionally, the restriction to Cartesian situations seems very limiting if one considers failing actions. It is left for future work to see if there exist fragments of failing actions which preserve Cartesian situations but are still expressive enough to be practically useful.

## 3.8. Handling faults in the high-level control

Above we have mainly discussed methods to specify the belief of the robot. To deal with faulty action execution not only the belief is of major concern but also how to deal with this faulty action in the high-level control of the robot. One method to deal with the possible faulty execution of action was presented in [49]. The focus of this approach is to allow the robot to create a plan such that the robot is certain that a specific query holds with a certain likelihood. This is achieved using a nondeterministic statement which triggers a branch with a certain probability. Thus, one can model a faulty action by using this non-deterministic statement and specifying that the normal action is an outcome of one branch and the faulty action is the outcome of another branch. By extending the transition semantic of Golog one can calculate the probability for a program that a certain query holds. This can be furthermore used to find a plan, such as that the query holds with a given probability. To model different initial worlds with different probabilities, a possible world semantics is used. Besides which situation is a reasonable alternative to the current situation a probability for this alternative is given. The method was extended in [50] to be used on a robotic system. The extensions define how the high-level and the low-level of the robot interact. This allows planning for the robotic system, also considering the reaction the low-level has on a specific high-level request. Additionally, time was added to the method presented in [49] to the formalism allowing to perform reasoning based on time, e.g. after 15 seconds the robot has started to paint an object. Due to the use of probabilities, one can create a more detailed description how the world may look like. But this imposes the cost of probabilistic reasoning and thus loses the simplicity of the method we use in this thesis. Additionally, the method proposed in [49] allows querying if a certain formula holds with a certain probability but does not incorporate the belief in the transition semantics as we do it in this thesis. Our incorporation in the transition semantics allows the programmer to ignore how the belief looks. The proper usage of the belief through the robot is ensured with the help of the transition semantics.
To not only plan for a program but to derive a policy to ensure that the program performs best the method proposed in [130] can be used. The method uses a transition semantics which extends Golog by incorporating a likelihood for a transition as well as a reward. This allows deriving a policy which maximizes the reward and the probability of a successful execution. To model the faulty action one uses a nondeterministic choice of actions. This allows the robot to perform an action and considers both action outcomes as possible until noticed otherwise later. To determine which action outcome has occurred one can perform sensing actions. But unlike, the sensing actions modeled like above which assert a formula, the sensing actions used in [130] use a successor state axiom to set the truth value

of a fluent. Thus, the latest sensing results override any previous knowledge. This is in contrast to the approach used in this thesis as we use an inconsistency between sensing result and the knowledge of the robot to check if an action was performed successfully. This allows the approach employed in this thesis to determine which action may have failed, including sensing actions, and to derive which indirect effects hold.
A combination of the above two approaches was described in [38]. The Golog derivative described in [38] allows combining a reasoning which incorporates probabilities for action together with a non-deterministic choice of actions. Additionally, the planning allows finding a policy which is optimal concerning a defined reward. Besides the possibilities to deal with faulty action, the method also incorporates constructs for concurrent execution interrupts and conditions which are checked during the execution. Although the language offers many features to specify the behavior of the robot special care was taken to ensure that the reasoning is performed in a fast manner. Thus, as in the previous approach, the latest sensing result was taken for the reasoning process. The assumption behind this method is that the sensing action is performed with a "high" frequency, thus ruling out wrong sensing results in a fast way. Due to its efficient reasoning, the proposed method was successfully applied to different robotics systems. The main difference to the approach used in this thesis is that no reasoning is performed, which fault has occurred which allows the method applied in this thesis to conclude indirect effects. We assume that such a reasoning is preferable for many robotic systems even though it imposes a more complex reasoning.

## 3.9. Monitoring the execution of actions

Besides the integration of the belief into the execution of the high-level control of a robotic system, one also needs to consider that during the execution the robot needs to consider to monitor the execution and react appropriately. The belief of the robot may be changed appropriately but only due to its use within the high-level execution the execution is monitored, and it is ensured that the robot behaves properly. Additionally, one needs to react to changes if the monitoring indicates that the current execution can be performed further. In [24] such a monitoring and reaction method was proposed. The method uses the transition semantics of Golog to incorporate the monitoring and the reaction. The method assumes that all external changes which are not caused by the robot can be observed. Furthermore, no faults are considered. If an external change happens the program is checked to see if it can still legally terminate with a predefined goal condition. If no such termination exists a program is searched which allows recovering from the external event. To recover from a fault, a linear sequence of actions with minimal length is searched, thus preferring simple recoveries over a complex ones. This simple method was used in [27] to monitor a process management system. The main difference to the approach presented in this thesis is that we don't assume to observe external influences, instead we use a belief which keeps track of such changes. Additionally, in this thesis, we extend the standard IndiGolog transition semantics to use the belief to check that the robot performs only safe transitions. Finally, we don't create a plan to recover from a fault. Instead, we only plan for sensing actions if the robot is lacking knowledge.
Another approach to monitor the execution of a generated plan was described in [44]. The method is based on regressing back the formulas used in the plan generation to check if the program can be further executed successfully. Thus, instead of searching in each iteration if the program can terminate successfully, one uses regression to pre-calculate which formula needs to hold to allow the program to terminate successfully. Additionally, the method can be extended to allow monitoring the execution of an optimal plan. This is done by regressing back the optimal value, allowing to decide at each point

during the execution if the robot still uses the best method to achieve its goal. The main difference to the approach presented in this thesis is that we only use a one-step look ahead to determine if the program can be performed further. Additionally, in the thesis, we presented a method which uses regression and the structure of the program to create a reactive program. Thus, we extend the idea presented in [44] for the high-level program instead of plans.

Chapter 4

# Consistency History Based Diagnosis

As we have discussed in Chapter 2, to model a robotic system one needs to address the qualification problem. The problem arises from the fact that it is not feasible to specify every condition which needs to hold to execute an action successfully. This issue is partly caused by to the complexity involved to capture all the necessary conditions, e.g. the humidity in the room could influence the grasp performance of the robot. Thus one would need to model the humidity and how it changes. However, the problem is also caused as not every condition is known to the program specifying the conditions for an action, e.g. which influence the color of the object has on the grasp performance.

To address the qualification problem, we have discussed the usage of history based diagnosis. The idea is to alter the history of performed actions in such a way that it reflects the real execution sequence. Thus, one replaces executed actions with a faulty counterparts. Additionally, it is checked if these variations of the original history are consistent with the observations which were made by the robot. If such a variation is consistent, it can be considered a possible explanation.

Although the approach addresses the qualification problem, it has several drawbacks. Two major drawbacks are the following: firstly, if the robot acts in an open environment one cannot assume a finite number of objects. This drawback implies that one cannot assume that a finite number of faulty versions of an action exist. Thus, the approach can become computationally infeasible if one considers the world to have an unknown size. The second drawback is that every possibility an action fails needs to be considered. Thus, one needs not only to model one fault an action can have but all of them. Additionally, the modeled faults need to ensure a complete description of the faulty behavior. This modeling effort can become an infeasible problem on its own. This issue can be even stated more drastically as it is the counterpart to the qualification problem. If one is not able to specify every condition necessary to perform an action, why should one be able to describe every possibility how an action may fail.

Let's consider a simple example to point out the first drawback in more detail. Let's consider our running example added with the additional fault that the robot can lose the object during its delivery. As the robot does not know the exact traversal route, the object could be lost in any room. Thus, if the robot drives from room $R_1$ to room $R_2$ and loses the object, the robot needs to consider the fault action variation $gotoLostObject(R_3)$, $gotoLostObject(R_4) \ldots gotoLostObject(R_n)$. Thus if $n$ gets big, the number of possible alternative actions also becomes large. Furthermore, if there is an unknown number of rooms which needs to be considered, e.g. the movement was not just within a building but between buildings or even cities, one cannot calculate all alternatives. Thus, in either case, the number of alternatives become infinite or at least too large to be handled properly.

To point out the second drawback in more detail, let's consider another example. Let's consider our running example, but instead of transporting just one object the robot can also transport a box with numerous objects. If the robot fails to deliver the box, during the delivery several different fault outcomes may occur. The robot has dropped the complete box, but the objects are still in the box. The robot has only "spilled" one object from the box but still holds the box with the other objects. .... One could imagine that the number of possible faults which may have occurred is enormous if one considers dropping the box or spilling objects from the box. The modeling of all these possibilities becomes even more complicated if one needs to consider that the box may have no fixed upper bound on the number of objects within the box. Thus, one would need to pay extra attention for the faulty actions as one needs to consider a different number of objects in the box, and thus different possibilities of faults.
To address both problems at the same time, we propose to use a description of an action influence instead of an alternative action to specify the impact a faulty action may have. The method, presented in this chapter to address the drawbacks mentioned above is based on the work presented in [94].
The remainder of the chapter is organized as follows: In the next section, we will briefly discuss some preliminaries for this chapter. In Section 4.2 we define the approach how to calculate a diagnosis as well as how the robot can draw a conclusion from this diagnosis. This section is followed by a section which discusses how to calculate the diagnosis using a conflict driven search. The proceeding section discusses how one can use default logic to perform the reasoning. Afterward, we discuss the relation of the proposed method with the classical history based diagnosis procedure in Section 4.5. This section is followed by a discussion of related research particular to this method. Finally, we conclude the chapter and point out some future work.

## 4.1. Preliminaries

History based diagnosis as it was presented in Chapter 2 follows the idea of assigning fault modes to components. This method is known as abductive diagnosis [17]. As we want to avoid to handle fault modes of an action we follow a different approach. In consistency based diagnosis [113] one assigns components to be either faulty or not, thus no fault mode of the components is necessary.
To declare a component $C_1$ to be faulty one uses the predicate $AB(C_1)$ in consistency based diagnosis. Additionally, one uses a formula $\phi_{C_1}$ to specify the behavior of the component $C_1$. Furthermore, one uses additional formulas to link the communication between the components and the observations of the components. To link the $AB(C_1)$ with the behavior of the component the following formula is used:

$$\forall c . \neg AB(c) \rightarrow \phi_c \tag{4.1}$$

If the component is assumed to function correctly one asserts that $\neg AB(C_1)$ holds. In combination with the definition of $\phi_{C_1}$ and the formulas linking observations and communication between components, a logical contradiction is caused if an observation differs from a predicted value from the system.
To resolve such a contradiction one can blame a component as faulty. Due to the formula if the component is considered to be faulty $AB(C_1)$ the implication is trivially true and the formula $\phi_{C_1}$ does not need to hold. Thus, one can use this method to retract constraints imposed by the components which can lead to a resolution of the contradiction.
As one could now blame all components within a system to be faulty to resolve an inconsistency one follows Occam's razor. Instead of blaming any set of components to be faulty one tries to find a minimum set of components to be faulty. We refer the interested reader to [113] and [25] for further information about consistency based diagnosis.

## 4.2. Consistency History Based Diagnosis

As consistency based diagnosis allows to resolve inconsistencies without the need for fault modes, it is desirable to resolve the drawbacks which are inherent to classical history based diagnosis.
In classical history based diagnosis one alters the history of performed actions through the usage of declared alternative actions. Thus, one replaces an inconsistent history with another history which is consistent. Those histories which have a small cost in their changes are preferred.
As we argued above the problem with this approach is that it is not always feasible to calculate all (optimal) alternatives. Additionally, one cannot assume to be able to model all fault modes of an action. Thus, instead of changing the history, we blame a set of actions to be faulty. A faulty action may have an arbitrary behavior, thus allowing to resolve the observed inconsistencies. This idea follows closely the idea presented in [117].
To blame an action to be faulty we use the predicate $fail(\alpha, s)$. With the help of the predicate, we state that action $\alpha$ which was performed in situation $s$ is considered to be faulty. Thus, the action does not settle its effect in the situation $s' = do(\alpha, s)$.
To model that an action does not settle its effect as expected we need to define which effect a faulty action has. We cannot assume in general that a faulty action does not have an effect at all. Additionally, we cannot assume that an action does change everything, e.g. if the robot fails to move from one room to another the color of the robot does not change. Thus, we need to model which subset of fluents is influenced by the action. We model this influence an action has on a fluent $F$ through the predicate $\psi^F_\alpha(\vec{x}, s)$. This predicate describes the condition under which action $\alpha$ influences fluent $F(\vec{x}, s)$ in situation $s$.
Instead of modeling an alternative action outcome we need to model the influence an action has on each fluent. One can argue that this does not resolve the problems we wanted to address. However, instead of describing in detail every possibility an action may fail we use the predicate to describe which fluent might be influenced. Thus, the description of the fault is fuzzy which allows an easier modeling. Furthermore, one does not need to enumerate all the different possibilities how an action fails instead by declaring an action to be faulty one covers all these possibilities.
One can follow different methods to derive the predicate $\psi^F_\alpha(\vec{x}, s)$.

- $\psi^F_\alpha(\vec{x}, s)$ is a precise formula as the possible faults can be estimated. This can be the case if one already has a description of the alternative action outcomes. If one can use such a formula, the precision of the diagnosis result is quite high.
- The formula can also be derived from the successor state axiom if the action only influences those fluents which are affected in the nominal case. The resulting formula is $\psi^F_\alpha(\vec{x}, s) \equiv \varphi^+(\alpha, \vec{x}, s) \vee \varphi^-(\alpha, \vec{x}, s)$. This formula also specifies a constraint which needs to be fulfilled by the formula. At least those fluents are influenced by an action which are influenced by the action through the successor state axioms. The approach presented in [117] can be represented using this kind of formulas.
- Sometimes the effect an action may have is not restricted to those fluent instances which are also affected by its nominal behavior. Instead, the influenced fluents are the same but the arguments of the grounded influenced fluent may differ to the nominal behavior. Thus, the formula becomes $\psi^F_\alpha(\vec{x}, s) \equiv \exists \vec{x}'.(\varphi^+(\alpha, \vec{x}', s) \vee \varphi^-(\alpha, \vec{x}', s))$.

Using the definition of the predicate $\psi^F_\alpha$ it remains to show how one links this predicate and the successor state axiom. We do so by following the idea of the so-called guarded successor state axioms [23]. Instead of using the successor state axiom as it is, the axioms are only applied if a guard condition holds. We use the predicate $\psi^F_\alpha$ as the guard condition of the successor state axiom. This is done with the following successor state axiom.

$$\begin{aligned}\forall \alpha \in \mathcal{A}.[\psi^F_\alpha(\vec{x},s) \rightarrow (\neg \mathit{fail}(\alpha,s) \wedge \Pi_\alpha(s))] \\ \rightarrow [F(\vec{x},do(\alpha,s)) \leftrightarrow \varphi^+(\alpha,\vec{x},s) \vee F(\vec{x},s) \wedge \neg\varphi^-(\alpha,\vec{x},s)]\end{aligned} \tag{4.2}$$

The definition implies that if the action influences the fluent the successor state axiom can only be applied if the action was not blamed for being faulty and the precondition holds. Because of this formula if an action fails all fluents' values which are influenced by this action are no longer determined by the successor state axiom. Thus, the agent does no longer know the truth value of the fluent after the action has failed. This allows the method to resolve a conflict between an observation and the effect an action normally has.
As fluent values are not only influence by successor state axioms but also through the sensing results we need to change how sensing results are used. Instead of asserting that sensing measurements need to be the same as expected sensing results for all time we condition this equality. The condition is the same as for the successor state axioms.

$$(\neg \mathit{fail}(\alpha,s) \wedge \Pi_\alpha(s)) \rightarrow (AO(\bar{s}) \leftrightarrow SF(\alpha,\bar{s})) \tag{4.3}$$

Thus, neither the sensing result of an action nor the effect it has due to the successor state axioms is applied if the action is faulty. This allows use to resolve any inconsistency between expected and measured sensing result. The only thing which needs to be done is to find a proper set of actions to be blamed as faulty.
As not every set of faulty actions may lead to a consistent situation we need to define the consistency of a history considering the effect a faulty action has. As we blame now actions to be faulty we slightly adapt the definition of a consistent situation as it was presented in Chapter 2.

**Definition 8.** *The consistency of a situation is defined with $Cons(\cdot)$ inductively defined as:*

1. $Cons(S_0) \doteq Invaria(S_0)$
2. $Cons(do(\alpha,s)) \doteq Cons(s) \wedge Invaria(do(\alpha,s)) \wedge$
   $((\neg \mathit{fail}(\alpha,s) \wedge \Pi_\alpha(s)) \rightarrow (AO(\bar{s}) \leftrightarrow SF(\alpha,\bar{s})))$

Thus, the definition of consistency has only changed slightly as we condition that the expected and the measured sensing result needs only to have the same truth value if the action is not faulty and the precondition of the action holds. This condition ensures that a faulty sensing action does not lead to a contradiction.
Using the definition of a consistent history we can now define what a diagnosis is as follows:

**Definition 9.** *A diagnosis $\delta$ for a history $s$ is set of faulty actions: $\delta = \{\langle\alpha,s\rangle | \mathit{fail}(\alpha,s)\}$. Furthermore, for a diagnosis $\delta$ it need to hold: $\mathcal{D}^* \setminus \mathcal{D}_{ssa} \cup \mathcal{D}_{ssa'} \setminus \{sensed\} \cup \{sensed\}' \cup \{\mathit{fail}(\alpha,s) | \langle\alpha,s\rangle \in \delta\} \cup \{\neg \mathit{fail}(\alpha,s) | \langle\alpha,s\rangle \notin \delta\} \models Cons(s)$ Where $\mathcal{D}_{ssa'}$ are the guarded successor state axioms presented in this chapter and $\{sensed\}'$ are the guarded sensing axioms presented in this chapter.*

Thus, a diagnosis is a set of actions blamed to be faulty which ensure consistency of the situation. Please note that the valuation of the fail predicates is defined through the diagnosis set and is added to the extended basic action theory.
Through the definition of a diagnosis, one ensures consistency with the help of a diagnosis. However, as in the case of classical history based diagnosis, not every diagnosis is very likely. To use only those diagnoses which are very likely we follow Occam's razor. We prefer those diagnoses which have a lower cardinality. This is stated in the following definition.

**Definition 10.** *A diagnosis $\delta^*$ for a history s is a preferred diagnosis if and only if the following holds. Let $\Delta$ be the set of all diagnoses then it holds that.* $\delta^* \in \Delta \wedge \nexists \delta . |\delta| < \delta^*$

After calculating the preferred diagnosis, the robot needs to use these diagnoses to perform the necessary reasoning task. This is done by considering the following action theory (CBAT):

$$\mathcal{D}^{cons}(\delta) = \mathcal{D}^* \setminus \mathcal{D}_{ssa} \cup \mathcal{D}_{ssa'} \setminus \{sensed\} \cup \{sensed\}' \cup \{fail(\alpha, s) | \langle \alpha, s \rangle \in \delta\} \cup \{\neg fail(\alpha, s) | \langle \alpha, s \rangle \notin \delta\} \tag{4.4}$$

Please note that the successor state axioms $\mathcal{D}_{ssa}$ are replaced by their modified version. Furthermore, the sensing axioms are replaced by their altered version. Thus, the robot can only draw those conclusions from $\mathcal{D}^{cons}(\delta)$ which are consistent with the specified diagnosis $\delta$.
If we would use just one diagnosis, the agent may only draw a conclusion considering only this diagnosis. This behavior may lead to a dangerous decision. To avoid such dangerous decisions, we define the belief of the agent according to all preferred diagnoses.

**Definition 11.** *Let $\Delta^*$ the set of preferred diagnoses for a situation s. A robot believes that a formula $\phi$ holds in s if and only if the following holds: $Belief(\phi, s) \doteq \forall \delta^* \in \Delta^* . \mathcal{D}^{cons}(\delta) \models \phi(s)$*

With the help of the definition of the belief, we ensure that the robot only believes those formulas to hold which can be concluded considering every diagnosis. Using the definitions so far, we can state some theoretical properties.
First, we make some assumptions which need to hold to state those properties.

**Assumption 1.** *Actions are only executable if their precondition is believed to hold. $Belief(\Pi_\alpha, s)$*

**Assumption 2.** *We assume that no action has a contradicting effect. $\mathcal{D}^{cons}(\{\}) \models \forall \alpha, s . Invaria(s) \wedge Invaria(do(\alpha, s))$*

The first assumption ensures that an action sequence is executable like the definition of an executable situation defined in [116]. We only assume that the robot believes as that the precondition of the action holds before it performs the action, thus later sensing results can indicate that the action precondition does not hold.
The second assumption only ensures that the actions are modeled properly. Otherwise, the robot could create an inconsistency history of performed actions just by executing actions without ever sensing the environment.
As a first theoretical result, we can state the following lemma.

**Lemma 1.** *If an action returns with true as result and it does not hold that $\mathcal{D}^{cons}(\{\}) \models \neg SF(\alpha, s)$ and if an action returns with false as result and it does not hold that $\mathcal{D}^{cons}(\{\}) \models SF(\alpha, s)$ it follows that the cardinality of the preferred diagnosis is 0.*

*Proof (Sketch).* A direct result of the definition of *Cons* and the fact that the sensing is not contradicting with the situation. □

The lemma states that if the sensing result is not expected otherwise, the cardinality is 0. Thus, the preferred diagnoses sets are empty till the expected, and the measured sensing results differ.
The second theoretical result we can state is the following lemma.

**Lemma 2.** *The minimum number of faults has as upper bound the number of sensing actions within the history.*

*Proof (Sketch).* This is a direct result of Assumption 2 and the definition of *Cons*. □

The lemma states that one diagnosis is to blame for all sensing actions performed so far. Thus, one can bound the maximum cardinality of the preferred diagnosis by the number of performed sensing actions.
Through Lemma 2 we can conclude the following lemma.

**Lemma 3.** *The size of the set of preferred diagnoses is bound by* $\binom{|s|}{\#\alpha\in s\wedge\alpha\in\mathcal{A}_{sensing}}$.

*Proof (Sketch).* This is a direct result of Lemma 2 □

The lemma states that the number of preferred diagnoses is exponentially bounded by the number of performed actions and the number of sensing actions performed so far. Thus, the number of diagnoses does not depend on the number of objects in the world. Instead, it depends only on the length of the history considered for the diagnoses.
This lemma can be strengthened further in the following Theorem.

**Theorem 1.** *The size of the set of preferred diagnoses is bound by* $\binom{|s|}{\#faulty\ sensing\ actions}$. *Where faulty sensing actions is defined as: if an action returns with true as result and* $Belief(\neg SF(\alpha),s)$ *holds or if an action returns with false as result and* $Belief(SF(\alpha),s)$ *holds.*

*Proof (Sketch).* Proof by induction
For the base case, we assume no fault has happened, thus through Lemma 1 the theorem holds.
For the induction step, let's assume we have a history with $n$ faulty sensing actions. The action $\alpha$ is performed, and the sensing results contradict with the belief of the agent. Thus, it follows that the minimal extension of any preferred diagnosis is due to a failing $\alpha$. Since we search for a minimal diagnosis, this bound must also hold for all new diagnoses thus the theorem follows. □

The theorem shows that the number of preferred diagnoses is further bounded. The number is correlated with the number of sensing actions which detected a discrepancy between the world and the robot's knowledge. As the number of those sensing actions is lower than the number of total performed sensing actions, the bound is drastically reduced.

## 4.3. Calculation of Diagnoses using Conflicts

The diagnoses can now be calculated in a trial and error fashion. Alternatively, a more focused search can be performed. One common method to enumerate all diagnoses is by using a conflict driven approach [113]. The idea is instead of just checking if the theory is consistent and blindly altering the assumptions made if the theory is inconsistent one uses the information of the inconsistency. One uses the information which is provided by a conflict set of the inconsistent theory. The conflict set contains a set of formulas which together can't be satisfied. One can expect that this set is significantly smaller than the complete theory. By using the conflict set, it is sufficient to choose the next assumption to withdraw from those assumptions present in the conflict set, as was shown in [113] for classical consistency based diagnosis.
Additionally, to focus search of the diagnoses one can use a minimal hitting set algorithm. Thus, instead of enumerating any diagnoses one uses a minimal hitting set algorithm for the diagnosis calculation. As was shown in [113] the minimal hitting set of the conflict sets are the minimal diagnoses of a system. Due to this focused search, one can speed up the computation of a diagnosis significantly. For a comparison of different diagnosis computation methods, we refer the interested reader to [97].

Due to these benefits of such a focused search, we are interested in carrying over such a search to our diagnosis problem. To do so, we could use a theorem-prover to calculate the conflicts for the complete theory. Unfortunately, as we discussed in Chapter 2 the usage of a theorem prover to reason about the action theory is very costly. This cost is mainly caused due to the inductive axiom to define a valid situation. Thus, instead of using a theorem prover directly, we use the structure of the problem itself for a conflict driven search.

Using the assumptions mentioned above that the definition of the action is well formed and that the initial situation is consistent then a conflict can only be caused by a sensing result. This imposes that if we have an inconsistency one part of the conflict needs to be a formula associated with the sensing action. Furthermore, we can observe that for any action which is part of the conflict set the precondition needs to hold to yield the conflict. Thus, any action which supported the precondition of an action within the conflict set needs to be itself a part of the conflict set.

Before we show how to calculate the conflict set for a general case, we will outline how the approach works for one sensing action which is in contradiction. Let's consider a simple example history. Let's assume we have performed the following history of actions which is consistent $s = [\alpha_1, \alpha_2, \alpha_1^S, \alpha_3]$ where $\alpha_1, \alpha_2, \alpha_3$ are primitive actions and $\alpha_1^S$ is a sensing action. Furthermore, we now perform the action $\alpha_2^S$ which is in contradiction with the history. As a sensing action asserts a sensing formula $SF$ to hold the contradiction is caused due to this formula. Thus, the following holds $\mathcal{D}^{cons}(\emptyset) \models \neg SF(\alpha_2^S, s)$. This result indicates that the robot expects the opposite sensing result. This indicates that $\alpha_2^S$ is one part of the conflict. Furthermore, there are only three cases such that the robot concludes that a formula holds. The first case is that the formula is assumed to hold through a successor state axiom. The second case is that a previous sensing formula imposes that the formula holds. The third case is that the formula holds because of the initial situation.

Let's consider the first possibility how the robot has drawn its conclusion, the successor state axiom. A conflict with a successor state axiom can occur if the truth value was carried over to the current state of the world or if the last action $\alpha$ changed the truth value. If the last action $\alpha$ has changed the value, this action is part of the conflict. One can reason that if the truth value would not have been modified the contradiction might not have happened. If the last action has not changed the truth value, the action can only be part of the conflict if it has an influence on the fluent in a faulty case thus $\psi_\alpha^F(\bar{x}, s)$ holds.

Considering the second possibility how the robot has drawn its conclusion, the sensing result of a previous action. This would be the case if the last action $\alpha$ was a sensing action the current sensing formula $SF(\alpha_2^S, s)$ conflicts with the formula $SF(\alpha, s)$. Such a conflict can be derived from the reasoning which formulas hold in the current state of the world. To resolve such a conflict one needs to add the sensing action $\alpha$ to the conflict set.

The final possibility how the robot has drawn its conclusion is through the initial situation. Through the assumption that the initial situation is not contradictory, no additional action can be added to the conflict set.

As the robot draws its conclusion from the complete history and not only from the last action the sensing formula which caused the conflict in the first place $SF(\alpha_2^S, s)$ also needs to be considered not only as the last action in mind. Instead, one can use the regression operator to regress the formula back in time. Thus, also actions are added to the conflict set which were performed before the last action but caused the robot to draw a conflicting conclusion.

We can generalize this approach by stepping through the history from the initial situation to the last situation and always perform the above-outlined calculation. Thus, we gather all conflict sets which are present in the current situation. This can be done with the help of Algorithm 1.

The algorithm steps through the complete history to find the next conflict set which is not covered

**Algorithm 1:** ComputeAllConflictSets

```
Data: $s$ ... a history
Data: $\delta$ ... a diagnosis to consider
Result: $\mathcal{C}$ ... a set of conflict sets
1  begin
2      $\mathcal{C} = \{\}$ ;
3      for $s' \leftarrow do(\alpha, s_0)$ to $s$ do
4          $s' = do(\alpha, s'')$ ;
5          if $\alpha \in \mathcal{A}_{Sensing}$ then
6              if $\mathcal{D}^{cons}(\delta) \cup SF(\alpha, s'') \models \perp$ then
7                  $\mathcal{C} = ComputeConflictSet(s', \alpha, SF(\alpha, s''), \delta) \cup ComputeConflictSet(s', \alpha, \neg\Pi_{\alpha}, \delta)$ ;
8                  break ;
9              end
10         end
11     end
12 end
```

by the current diagnosis. If the last action was a sensing action the algorithm checks if the theory is still consistent as only a sensing action can cause an inconsistency. If an inconsistency is caused Algorithm 2 is used to derive the new conflict set. The conflict search is called $O(|s|)$ times in order to compute all conflicts for a given situation $s$ with length $|s|$.
The algorithm first adds the action which was used to observe the conflict at first ($\alpha$) to the conflict set. Afterwards the algorithm iterates in a backward fashion through the history of performed actions till the initial situation is reached. In each iteration, the algorithm checks if the formula which was the reason for the conflict ($\phi$) still causes a contradiction. If the formula still causes a contradiction the conflict of this contradiction is inspected.
If the conflict is due to a successor state axiom which is influenced by the last performed action $\alpha'$ we add this action to the conflict set by a recursive call to the algorithm itself. Please, note that for the conflicting formula we use the negated precondition, as we also need to incorporate all those actions which supported the precondition of this action in the conflict set. Thus, every action which is in contradiction to the negated precondition of action $\alpha'$ has supported the precondition and is added by the algorithm.
If the conflict is due to the sensing result of the last action $\alpha'$ we add this action again by calling the algorithm recursively. Additionally, we use the negated precondition to find all those actions which supported the action $\alpha'$.
Finally, in Line 18 we regress back the formula. This allows us to check for further conflicts of the formula in the previous parts of the history. As the robot draws its conclusions from experience the formula needs to consider the complete history of performed actions, not only the last action. Furthermore, due to the usage of the regression operator, it is guaranteed that the formula is correctly transformed back in time. The overall complexity of the algorithm is $O(|s|)$ as either a successor stat axiom is in conflict (if $\alpha$ is a primitive action) or the sensing axiom is in conflict (if $\alpha$ is a sensing action).
To use the above algorithm in a minimal hitting set calculation to derive all minimal diagnoses we need to show that the conflicts arising from the algorithm are at least a superset of all minimal conflicts which would be derived if the complete theory would be used instead. We show this property with the

**Algorithm 2:** ComputeConflictSet

```
Data: $s$ ... a history
Data: $\alpha$ ... the action imposing $\phi$
Result: $\phi$ ... an inconsistent formula
Data: $\delta$ ... a diagnosis to consider
begin
    $\mathcal{C} = \{\alpha\}$ ;
    for $s' \leftarrow s$ to $s_0$ do
        if $s' = s_0$ then
            break ;
        end
        $\langle r, c \rangle = checkSat(\mathcal{D}^{cons}(\delta) \cup \phi)$ ;
        if $r = sat$ then
            break ;
        end
        $s' = do(\alpha', s'')$ ;
        if $successorStatAxiomInConflict(c, \alpha')$ then
            $\mathcal{C} = \mathcal{C} \cup ComputeConflictSet(s'', \alpha', \neg\Pi'_{\alpha}, \delta)$
        end
        if $sensingAxiomInConflict(c, \alpha')$ then
            $\mathcal{C} = \mathcal{C} \cup ComputeConflictSet(s'', \alpha', \neg\Pi'_{\alpha}, \delta)$
        end
        $\phi = \mathcal{R}(\phi, \alpha')$
    end
end
```

following theorem.

**Theorem 2.** *For every minimal conflict set $C$ the Algorithm 1 finds a conflict set $C'$ such that $C \subseteq C'$.*

*Proof (Sketch).* Proof by contradiction. Let's assume there exists a minimal conflict set $C$ where the algorithm produces no correct $C'$.

As shown in [132] only sensing action can indicate a fault. Thus, a sensing action needs to be part of $C$. Furthermore, after a sensing produces an inconsistent theory and the definition of a minimal conflict only the first sensing action leading to an inconsistency is of interest. Thus, it is sufficient only to consider $s'$, which is the situation till the first conflicting sensing action and adding the sensing action to the conflict set.

By definition of the consistency a sensing action can lead to an inconsistency iff, the precondition holds for the sensing action and the negation of the sensing action is part of the situation ($s'$) in which the action was performed. The precondition of an action holds in a situation iff adding the negation of the precondition yields to a contradiction in the theory. Thus, what remains to show is that there exists an action which is not part of the result of Algorithm 2 for a certain formula $\phi$.

We show this by induction. For the base case, the contradiction holds by the definition of the consistency, as it assumes that the initial situation is not contradicting. Thus, no further actions can be in the conflict set.

The inductive step let's assume for every formula $\phi$ with any situation $s'$ up to the length $N$ the contradiction holds. Furthermore, by the definition of regression Line 18 ensures that the formula

$\phi$ is correctly transformed closer to the initial situation. Thus, any conflicting action is found for $s$. Thus, what remains to show is that the action $\alpha$, such that $s = do(\alpha, s')$ holds, is the missing one in the conflict set. By definition of the CBAT, it holds that an action has only influence that a formula holds if it either is an effect of the action or if it is asserted through a sensing axiom. For the first case, the action would have been found as the conflict would have been with a successor state axiom. For the second case, the action would have been considered as the conflict would have been with the sensing axiom. Thus, there is no action missing in the conflict set. Thus, the theorem follows. □

With the help of the above algorithm, we can calculate all conflicts. Using these conflicts and a minimal hitting set algorithm we can calculate all minimal diagnoses. Afterwards, we can use these diagnoses to reason which formulas the robot believes to hold.

## 4.4. Using Default Logic to Model the Belief

The main concern of belief management is not the diagnoses it uses but the formulas which form the belief. Thus instead of using the diagnoses to form the belief, one can use a more direct approach. Instead of calculating the minimal diagnoses and using those to derive the belief one can use default logic as it was proposed in [112] to perform the reasoning on the belief.
To specify a theory with the help of this default logic one uses two parts. First, a set of axioms which are assumed to hold. These axioms are classical logical axioms as we have used them so far. The second part is a set of default rules which are applied if they do not yield a contradiction. We use these default rules to specify that an action is not faulty if it does not lead to a contradiction. The default rule $\theta$ is defined as follows:

$$\frac{: \neg \mathit{fault}(s)}{\neg \mathit{fault}(s)} \tag{4.5}$$

The reasoning in default logic now finds the maximum set of default rules which can be applied to still be consistent. Using this set of rules the default logic performs its reasoning. A formula holds if cautious reasoning is applied if it holds in every theory which is the result of maximizing the set of default rules. As we are not using a minimal set of not applied default rules, we need to add an additional axiom such that the number of not applied default rules is bounded to a specific number. The number of faulty actions is precisely bounded with the help of Theorem 1. Thus, we can generate an axiom $\Gamma$ which imposes that the number of faulty actions is the number of contradicting sensing actions.
To count the number of faulty actions we can use a simple fluent $\mathit{FaultyActions}(n, s)$. The fluent is set initially to hold only for $n = 0$. Thus, we add to the initial situation the following formula.

$$\forall n.\mathit{FaultyActions}(n, S_0) \leftrightarrow n = 0 \tag{4.6}$$

Furthermore, we specify as a successor state axiosm* that a fault increases this number.

$$\begin{aligned} \mathit{FaultyActions}(n, do(\alpha, s)) \equiv (n = n' + 1) \wedge & \mathit{FaultyActions}(n', s) \wedge \mathit{fail}(\alpha, s) \vee \\ & \neg \mathit{fail}(\alpha, s) \wedge \mathit{FaultyActions}(n, s) \end{aligned} \tag{4.7}$$

Finally, we can constrain this fluent at the end of the history $S$ to the number of faulty sensing action $k$ with the help of the following formula.

$$\forall n.\mathit{FaultyActions}(n, S) \leftrightarrow n = k \tag{4.8}$$

With the set $\Gamma$ consisting of all the definitions of the fluent *FaultyActions* we can specify the default logic theory $\mathcal{D}^D$. The theory is defining as follows:

$$\mathcal{D}^D = \langle \mathcal{D}^* \setminus \mathcal{D}_{ssa} \cup \mathcal{D}_{ssa'} \setminus \{sensed\} \cup \{sensed\}' \cup Cons \cup \Gamma; \theta \rangle \tag{4.9}$$

Where $\mathcal{D}_{ssa'}$ are the new successor state axioms and $\{sensed\}'$ are the new sensing axioms. Further *Cons* are the axioms of a consistent situation quantified over all situations.
With the help of this theory, we can now draw the conclusion what the robot knows directly. First, we can relate the minimal diagnosis to the set of maximum applied default rules. This set is called the maximum extension (according to [112]) of $\mathcal{D}^D$. We state this relation in the following Corollary.

**Corollary 1.** *Every maximal extension E of $\mathcal{D}^D$ corresponds to a minimal diagnosis $\delta$ where $\delta = \{s|fault(s) \in E\}$*

*Proof (Sketch).* Through the definition of default logic, every maximal extension is a subset minimal set of faulty actions. Through the cardinality constraint only cardinality minimal diagnoses are allowed. Furthermore, an extension needs to be consistent through the definition of *Cons* which is part of the theory. □

As we have a direct relation between maximal extensions and minimal diagnosis we can now relate the reasoning which is performed with this theory and the definition of *Belief*. This relation is stated formally in the following lemma.

**Lemma 4.** *A formula $\phi$ is believed to hold using the following query $\mathcal{D}^D \models_{cautious} \phi[s]$ iff $Belief(\phi, s)$ holds for the formula.*

*Proof (Sketch).* This directly follows from Corollary 1 and the cautious reasoning used to answer the query. □

## 4.5. Relation to History Based Diagnosis

We argued in the introduction that the classical history based diagnosis suffers from two drawbacks. Furthermore, we have argued above that we tackled these two drawbacks with the proposed method. However, the question arises if we can still draw the same conclusions with this approach (CHBD) as if we would use classical history based diagnosis (HBD).
As both methods CHBD and HBD have their separate definitions of action faults we need first to define a linkage between those to perform further comparisons. Let's assume for a given action $\alpha$ we have the fault modes $\alpha_{F_1}$ till $\alpha_{F_N}$. We can now define the predicate $\psi_\alpha^F$ as follows:

$$\psi_\alpha^F(\bar{x}, s) \equiv \bigvee_{i.1 \leq i \leq N} \exists \bar{y}.\phi_F^+(\bar{x}, \alpha_{F_i}(\bar{y}), s) \vee \exists \bar{y}.\phi_F^-(\bar{x}, \alpha_{F_i}(\bar{y}), s) \tag{4.10}$$

Thus, through this definition the fluents which are influenced by the action are those fluents which are influenced by any of the faulty modes of the actions.
With the help of this definition, we can draw already the first lemma relating the conclusion which can be drawn if both use the "same" diagnosis.

**Lemma 5.** *Given a situation s and a fault mode assignments of action $A_F$, which makes the situation consistent then it follows that $\mathcal{D}^{cons}(\delta) \models \phi(s) \Rightarrow \mathcal{D} \models \phi(s')$, where $\delta$ consist of those actions which have a fault mode assigned and $s'$ is the result of replacing all action in s through their fault mode.*

*Proof (Sketch).* Proof by contradiction. Let's assume there exists a $\phi$ which is entailed by $\mathcal{D}^{cons}(\delta)$ but not by $\mathcal{D}$ then it follows that there exists at least on fluent $F$ which has a truth assignment which differs in both reasoning systems. As the reasoning is the same for those actions which are not faulty, it follows that there exists an action $\alpha$, which is fault, which sets the truth value of the fluent differently. A faulty action $\alpha$ does not set a truth value to a fluent by definition of the successor state axiom. Thus, such a $\phi$ cannot exist. □

The lemma states that if we can draw a conclusion with the help of CHBD from a diagnosis, we can also draw this conclusion from HBD. Thus, CHBD does not allow to draw more conclusion. The contrary does not hold.
Let's consider a simple example to consider this problem. Let's assume we have one action $\alpha$ which sets the value of $F_1$ and $F_2$ to true. Furthermore, let's consider the sensing action $\alpha_S$ which senses the truth value of $F_1$. Thus, the action sequence $[\alpha, \langle\alpha_S, \perp\rangle]$ where the robot has performed first $\alpha$ and afterwards got the sensing result $\perp$ from $\alpha^S_{F_1}$ is in contradiction. Let's additionally assume one fault mode of the action is $\alpha'$ which sets $F_1$ to false instead of true. Additionally, we have a second fault mode $\alpha''$ which sets the fluent $F_2$ to false instead of true. Thus, using CHBD both fluents $F_1$ and $F_2$ would have an unknown truth value after a fault of action $\alpha$. Thus, if we use HBD and consider the alternative action sequence $[\alpha', \langle\alpha^S_{F_1}, \perp\rangle]$ which is consistent we would draw the conclusion that $F_2$ needs to hold. Instead of we consider CHBD and the action $\alpha$ to be fault we would not be able to draw any conclusion on the truth value of $F_2$.
Thus, the robot would draw a more cautious conclusion if the diagnosis is the same if CHDB is used. To further investigate how the robot can draw a conclusion we need to relate how the different diagnosis from HBD and CHDB relate to each other. As the set of diagnosis are used to draw the belief of the robot we need to check if we can draw the "same" diagnosis with both methods.

**Lemma 6.** *Given a situation $s$ and a fault mode assignments of action $A_F$, which makes the situation consistent then it follows that $\delta$ consisting of all actions which have a fault mode assigned is a diagnosis.*

*Proof (Sketch).* This directly follows from Lemma 5. Moreover, the definition of a diagnosis is a set of actions which blamed to be fault create a consistent situation. □

The lemma states that if we create a diagnosis with HBD, we can create the "same" diagnosis with CHBD. Unfortunately, the contrary does not hold. Which is a direct result of the fact that CHBD causes a more cautious reasoning. Thus, a contradiction may appear if HBD is used but may not appear if CHBD is used. This results can be extending to state that the minimal sets of HBD and CHBD may not be the same. As not every diagnosis of CHBD is a diagnosis of HBD, it can be the case that the minimal diagnosis of CHBD has a lower cardinality as those sets in HBD. Thus, the two-minimal set may not relate to each other at all.
This also implies that any reasoning performed with the help of the minimal diagnosis of these two sets will draw different conclusions. To show this impact let's consider the agent draws its belief either with the support of CHBD or it uses the minimal diagnosis of HBD to draw its conclusion. If the HBD is used, we assume the robot beliefs a formula if it holds in every altered situation which is a minimal diagnosis.
Let's consider the example from above and add an additional sensing action $\alpha^S_{F_2}$ which senses the truth value of $F_2$. Additionally, let's assume that for each sensing action there is a fault mode $\alpha^{\bar{S}}_{F_1}$ receptively $\alpha^{\bar{S}}_{F_2}$ which inverts the sensing result. The sequence $[\alpha; \langle\alpha^S_{F_1}, \perp\rangle; \langle\alpha^S_{F_2}, \perp\rangle]$ is now in contradiction. To resolve the problem, the set of minimal diagnoses for CHBD would be $\{\langle\alpha, S_0\rangle\}$. If HBD is used

instead the set of minimal diagnoses would be $\{[\alpha; \langle\alpha_{F_1}^{\bar{S}}, \bot\rangle; \langle\alpha_{F_2}^{\bar{S}}, \bot\rangle]\}$. Thus, CHBD would conclude that action $\alpha$ is faulty whereas HBD would conclude that both sensing actions are faulty. Thus, the robot would conclude $\neg F_1$ and $\neg F_2$ if CHBD would be used whereas HDB would conclude $F_1$ and $F_2$. Thus, the robot can draw completely different conclusion if HBD or CHBD is used. Thus, when using CHBD one should consider that it may lead to other conclusion then HDB if the influence an action may have is big.

## 4.6. Related Research

Before we conclude the chapter, we will discuss some related research specific to this chapter.
Our work is based on the consistency based diagnosis presented in [113]. Although, as also argued in the paper itself, the diagnosis method is not restricted to any system description, it was originally designed for static system. Thus, it was assumed that the formulas of the system do not change over time. This implied that one does not have to consider a sequence of actions. Instead, only a set of components was examined. Moreover, the diagnosis was a set of components which is consistent with the observations taken.
The idea of consistency based diagnosis was later extended in [85] and [5] to consider dynamic system. The system consists of components but can also perform actions to change some parameters. Additionally, observations can be made like sensing actions as we used it within this work. However, in contrast to our approach, not the actions were considered to be faulty or not. Instead of specific actions were used which could alter a component from working to faulty. Thus, one does not search for a set of actions to be faulty. Instead one searches for actions to insert to explain a certain observation. As such the method still considered a system to be diagnosed and not the changes of the system.
The method of diagnosing a system which can evolve was further investigated in [4]. The approach was extended to use a possible world semantic to capture the different possible explanations of the fault. Thus, one does not calculate the diagnosis but use each diagnosis as one possible world. This follows a similar idea as our definition of a belief of a robot. By using this possible world semantic one can reason which components are faulty in every diagnosis, in none of the diagnosis or in a subset of diagnosis. Due to this knowledge, one can plan to repair the system. Alternatively, one can use the information to plan for sensing action to rule out different diagnosis. We will discuss a similar planning method which rules out diagnosis for our problem domain in Chapter 7. The main difference to our approach is still that the diagnosis defines components to be faulty instead of actions.
To deal with the problem that sensing action needs to be forgotten a method was proposed in [108]. The method defines that one can forget a sensing action by removing this sensing action from the history. Thus, instead of ignoring the result of the sensing action one removes first the sensing action from the situation term and afterward uses the situation term for the reasoning. This contrasts with our approach which does not need to modify the situation term to forget the effect a sensing action has.
In [117] a method based on consistency based diagnosis was presented which diagnosed plans. The method uses a partially ordered plan and a set of partial observations about the world. If these observations are in contradiction with the plan a diagnosis is performed. The diagnosis is defined as a set of actions which are faulty, like our approach. If an action is faulty those truth values of fluent instances which are influenced by the nominal action are set to be no longer known by the robot. This is a special case of the formulas $\psi_\alpha^F$ of our approach as discussed above. This restriction causes that one cannot cover all cases an action may influence the world if it is executed fault. Let's consider a robot is stacking objects on top of each other on a table. If the robot now fails to stack an object, it might

influence not only the objects location but also all other object locations. This can be caused as the robot may collide with all objects on the table. Thus, we had generalized this approach and formalized the approach using the formal theory of the situation calculus. Additionally, we have shown how the agent can use the information on the diagnosis to draw a conclusion how the world looks like.
The diagnosis method of [117] was extended in [154]. The extension allowed to calculate the diagnosis in a distributed manner, thus, allowing the method to be applied to a multi-agent setting. Furthermore, the method was extended to find the underlying root cause, e.g. the environment causes the action to fail. This was used to assign different agents in the world to be guilty of certain faults. The method still has the same differences to our work as [117].
Finally, it is important to mention that the usage of a guarded successor state axiom is not new. In some earlier versions of the definition of the successor state axiom presented in [115] the precondition of an action was used as guard. This is very like our usage of the guard condition. However, instead, we guard the successor state axiom by the influence formula, the action precondition, and the predicate to indicate an action fault.
The usage of guarded theories was extended in [118] and [23] to specify guarded sensing results and action outcomes. This guarded conditions can be considered as the most general case whereas our guards are only used to propagate the fault an action may have.

## 4.7. Conclusion and Future work

In this chapter, we have shown a method to tackle two main drawbacks of the classical history based diagnosis. The first drawback is that the number of possible alternative situations may become infinite if one needs to consider an infinite number of objects. The second drawback is that it is not always feasible to describe how an action may fail. We addressed both drawbacks by using a fuzzy description of the effect an action has.
Due to the usage of this description, it is sufficient only to find a set of actions to be blamed faulty to restore a consistent history. As we are interested in reasonable explanations, we use those sets of faulty actions which are minimal in their cardinality. We have shown how such a set can be calculated efficiently by using a conflict driven search. Additionally, we have demonstrated how the robot can draw its conclusions from such set of faulty actions. Furthermore, we have shown how to draw the same conclusions if one uses default logic to avoid the explicit calculation of the set of faulty actions. Finally, we have shown how the classical history based diagnosis and this approach relate to each other on a theoretical basis.
It is left for future work to combine the classical approach of history based diagnosis with the approach outlined in this chapter. This would give rise to the possibility to be specific if possible but still find a diagnosis if no precise explanation can be given.

Chapter 5

# Reusing Common Sense Invariants

In the previous chapter, we have discussed how to easy the specification of the faults an action may encounter. However, in the definition of the history based diagnosis not only the action needs to be specified. Additionally, a set of invariant is used to define the commonsense knowledge about the world. This commonsense knowledge is used to draw an additional conclusion which allows detecting a faulty action outcome.

Let's consider the running example. Additionally, let's consider that object $O_1$ is in room $R_1$ and the robot is moving to room $R_2$. If now the robot scans the room for an object $O_1$ it detects the object and thus conclude that object $O_1$ is in room $R_2$. However, the object is also in room $R_1$ as this was the initial knowledge about the object location and the robot has not moved this object. If the initial knowledge base does not explicitly forbid the object $O_1$ to be in another room, then $R_1$ the robot is happy to have magically duplicated the object. However, instead, if we use common sense we can reason that no object can be in two places at once. Thus, the object can either be in $R_1$ or $R_2$ but not in both rooms at the same time. Thus, if the robot uses common sense, it can detect an inconsistency where otherwise none would have been detected.

As such it would be desirable to have as much common sense knowledge as possible about the world. However, the drawback is that one needs to define this commonsense knowledge. This definition can be a complex task as one needs not only to define a possible hug sets of axioms which hold but one also needs to ensure that these axioms are consistent. Additionally, this specification may repeatedly be done for different domains or environments.

Thus, it would be beneficial to reuse common sense knowledge if possible. This reuse becomes especially interested as one may be able to reuse big commonsense knowledge bases such as Cyc [77]. Often these knowledge bases are designed to allow a specific reasoning method. Thus, allowing a fast inference on those knowledge bases. This specific reasoning method often hinders the easy integration of such knowledge bases into the robotic system.

To address this problem, we follow the idea of ontology mapping, which can be used to construct large knowledge bases out of smaller ones. We refer the interested reader to [61] for a survey of ontology mapping methods. We map the knowledge of the robot into concepts used in the commonsense knowledge base to check if the knowledge is consistent. The mapping allows that the knowledge of the robot and the commonsense knowledge base use a different set of concepts. Furthermore, the commonsense knowledge base may also have a different reasoning method than those utilized by the robot for it is own knowledge base.

In this chapter, we will discuss how such a mapping can be defined. Additionally, how we can use

this mapping to derive a method to reuse commonsense knowledge from other sources. The method discussed in this chapter is based on the results presented in [132] and [93].
The remaining of the chapter is structured as follows. In the next section, we discuss some theoretical results on the history based diagnosis we use. These results are later exploited to create a proper implementation of the method. In Section 5.2 the theory how to in co-operating common sense from other sources is discussed. This section is followed by a section which shows how to implement this method by using Cyc [77] as an external source for common sense. Before we conclude the chapter, we discuss some related research peculiar to this chapter in Section 5.4. Finally, we conclude the chapter and point out some future work.

## 5.1. Some Theoretical Results on History Based Diagnosis

Before we show how we can map the knowledge of the robot into another knowledge base, we will first state some theoretical results on the history based diagnosis itself. This results will help us to derive a method which efficiently uses the external resource.
To state the first property, we make an assumption about the model of the actions. This assumption does not restrict the effect of the actions. Instead, it only ensures that actions are modeled in such a way that they do not introduce inconsistencies on there on.

**Assumption 3.** *A primitive action $\alpha$ never contradicts an invariant if executed in a consistent situation s:*
$D^* \models \forall s.(Cons(s) \wedge Invaria(do(\alpha,s)))$

Through this assumption, it is ensured that through the execution of primitive actions alone we do not produce an inconsistency. Thus, we ensure that the robot does not perform a diagnosis without any external evidence of a faulty action.
With the help of this assumption we can state the first lemma.

**Lemma 7.** *The situation $do(\alpha,s)$ resulting from the execution of a primitive action $\alpha$ in a consistent situation s is always consistent:*
$D^* \models \forall s.\alpha \in \mathcal{A}_{primitive} \Rightarrow (Cons(s) \Rightarrow Cons(do(\alpha,s)))$.

*Proof. (Sketch).* This follows trivially from the Assumption 3 and the observation that a primitive action is not a sensing action. As such $SF(\alpha,s) \leftrightarrow AO(\alpha,s)$ per definition of the action. □

Where $\mathcal{A}_{primitive}$ is the subset of all actions which are primitive actions. The lemma states a primitive action carries over the consistency of a situation. Thus, the robot will not create an inconsistency be just executing primitive actions if the situation before was consistent.
We can use this lemma to define further when a situation can become inconsistent. First, as no primitive action can cause an inconsistency, we can trivially conclude that only a sensing action can cause an inconsistency. That is what one may expect if one considers history based diagnosis. We can further specify when a sensing action leads to a contradiction with the following lemma.

**Lemma 8.** *A sensing action $\alpha$ is inconsistent with a consistent situation s iff :*
$SF(\alpha,s) \not\equiv AO(\alpha,s)$
*or*
$D^* \not\models Invaria(do(\alpha,s))$

*Proof (Sketch).* This follows trivially from the definition of consistency. □

The lemma states that a sensing action can only cause an inconsistency if either the sensing result is not as the robot expects it our if the commonsense of the robot yield an inconsistency after executing the sensing action.
We further assume that the initial situation is properly model. This is formally stated through the following assumption.

**Assumption 4.** *The initial situation $S_0$ never contradicts the invariant:*
$D_{S_0} \models Invaria(S_0)$

With the help of assuming that the initial situation is properly model we can further investigate what can cause an inconsistent situation. This is formally stated in the following theorem.

**Theorem 3.** *A history $s = do([\alpha_1, ..., \alpha], S_0)$ can only become inconsistent iff it contains a sensing action:*
$D^* \models \neg Cons(s) \Leftrightarrow \exists \alpha \in [\alpha_1, ..., \alpha_n].\alpha_i \in \mathcal{A}_{sensing}$

Where $\mathcal{A}_{sensing}$ is the subset of those actions which are sensing actions.

*Proof.* Through Assumption 4 we know that $Cons(S_0)$ is true. Through Lemma 7 we know that the consistency will not change if a primitive action is performed. Thus $Cons(s)$ remains true if $s$ comprises only primitive actions. Thus $Cons(s)$ can only become false through a sensing action. □

The theorem states that a situation becomes inconsistent only due to a sensing action. Thus, the robot will have a consistent history till a sensing action is performed which causes an inconsistency.
We can now combine the results from Lemma 8 and Theorem 3 into the following lemma.

**Lemma 9.** *A situation $s = do([\alpha_1, ..., \alpha_n], S_0)$ becomes inconsistent iff it contains a sensing action and either there is a contradicting sensing result or the invariant is violated:*
$D^* \models \neg Cons(s) \Leftrightarrow \exists \alpha_i \in [\alpha_1, ..., \alpha_n].\alpha \in \mathcal{A}_{sensing} \wedge [SF(\alpha, s) \not\equiv AO(\alpha, do(\alpha, s)) \vee \neg Invaria(do(\alpha, s))]$.

The lemma states that a situation only becomes inconsistent if a sensing action is performed and the expected sensing result does not coincide with the real sensing result or if the commonsense knowledge of the robot is in contradiction after the sensing action is executed. We will use this result later to restrict the calls to the external commonsense knowledge as far as possible.

## 5.2. Common Sense knowledge from another Sources

As we outlined above in the motivation, we want to check the situation to conform to the commonsense knowledge of an external source. To do so, we want to map the knowledge of the robot into external the commonsense knowledge base. Through this limited mapping, we do not need to map the complete theory of the situation calculus to the other knowledge base. Such a complete mapping was for example proposed in [140].
To define the mapping between the knowledge of the robot and the commonsense knowledge base we use two different languages. The language $\mathcal{L}_{SC}$ is used to define the language and its associated reasoning of the situation calculus. This reasoning is used to derive the knowledge the robot has. To represent the language and the reasoning of the commonsense knowledge base we use $\mathcal{L}_{KB}$. Using these two definitions, we now define the necessary mapping between the knowledge of the robot and the commonsense knowledge.

**Definition 12.** *Function $\mathcal{M}_{\mathcal{L}_{SC},\mathcal{L}_{\mathcal{KB}}} : \mathcal{L}_{SC} \times S \rightarrow \mathcal{L}_{\mathcal{KB}}$ represents the transformation of a situation and an extended basic action theory to a theory in $\mathcal{L}_{\mathcal{KB}}$.*

As one could now use an arbitrary mapping between the two different knowledge bases, we specify two properties which need to hold. These properties ensure that the result of the mapping can be used to decide if, in the current situation, the common sense is consistent.
The first property state that used constants has the same semantic meaning in both knowledge bases. Thus, it is only allowed to map a constant from one representation to another if the same semantics are used in both representations. For example, the object $O_1$ can only be mapped to an object in the commonsense knowledge base but, e.g., not to a room.
The second property states that the mapping must maintain the consistency of common sense. Thus, the mapping ensures that if we specified the knowledge of the commonsense knowledge base as part of the knowledge of the agent, we would have the same result regarding the consistency. This property is essential as otherwise, the mapping would not be useful at all. In case of an ontology mapping, a similar property is used which is called the query property [2].
To define the second property formally, we use the set of invariant $I$ which define the common sense knowledge we are interested in. Such an invariant could be for example that an object is only at one place at the same time. Additionally, to the invariant, we specify a certain implementation $\mathbb{I}$ of the invariant. For example, one can use first order to define the invariant that an object is not in two different rooms at the same time as follows $\forall o.\nexists r_1, r_2.isAt(o,r_1) \wedge isAt(o,r_2) \wedge r_1 \neq r_2$. With $\mathbb{I}_{SC}(I)$ we specify the implementation of the invariant in the situation calculus. Furthermore, we specify through $\mathbb{I}_{KB}(I)$ the implementation of the invariant in the commonsense knowledge base. It is important to mention that the implementation of the invariant ensures that the reasoning system associated with the invariant ensures a proper conclusion of the invariant. Thus, the interpretation for the implementation $\mathbb{I}(I)$ needs to be $I$ regardless of the concrete implementation. Due to the definition of the implementation of an invariant, we can now state the property which needs to hold for the knowledge mapping more formally as follows.

**Definition 13.** *The mapping $\mathcal{M}_{\mathcal{L}_{SC},\mathcal{L}_{\mathcal{KB}}}$ is proper if the transferred situation s together with the common sense $\mathbb{I}_{KB}(Invaria(s))$ lead to a contradiction if and only if the situation s together with the original implementation of the commonsense knowledge lead to a contradiction:*
$\mathbb{I}_{KB}(Invaria(s)) \cup \mathcal{M}_{\mathcal{L}_{SC},\mathcal{L}_{\mathcal{KB}}}(\mathcal{D}^*, s) \models_{\mathcal{L}_{\mathcal{KB}}} \bot \Leftrightarrow \mathcal{D}^* \cup \mathbb{I}_I(Invaria(s)) \models_{\mathcal{L}_{SC}} \bot$

The definition ensures that regardless which reasoning is used the invariant only lead to an inconsistency if also the other representation would have resulted in an inconsistency.
With the mapping defined we can now state the following lemma combining the mapping property and previously findings.

**Lemma 10.** *A sensing action $\alpha$ which does not lead to a contradiction in the expected and the real sensing result is inconsistent if the following holds:*
$\mathbb{I}_{KB}(BK) \cup \mathcal{M}_{\mathcal{L}_{SC},\mathcal{L}_{\mathcal{KB}}}(\mathcal{D}^*, s) \models_{\mathcal{L}_{\mathcal{KB}}} \bot$.

*Proof (Sketch).* This follows directly from Lemma 8 and the Definition 13 of a proper mapping. □

The lemma states to check if a sensing action does not lead to a contradiction we can use the external commonsense knowledge base together with the mapping. Thus, if we have a proper mapping we can reuse the commonsense knowledge of another source.
To perform an implementation in practice, we place some further restrictions to the mapping. First, we assume only a finite number objects and fluents in the knowledge base. This is necessary to

allow a complete mapping of the knowledge base of the robot. Additionally, we assume that the mapping maps all fluents to a certain truth value. This is done to avoid any different interpretations between a reasoning system with an open and a closed world assumption. Please note that these restrictions are not necessary from the theoretical point of view but should be considered to allow a proper implementation.

## 5.3. Implementation using Cyc

Following the above definition of a proper mapping, we implement such a mapping between the knowledge base of the robot and the commonsense knowledge base Cyc[77]. Cyc is a commonsense knowledge base which includes many other knowledge bases. The reasoning system of Cyc is a based on higher order logic to allow a complex reasoning. Through this higher order logic, the predicates and properties used in Cyc can be easily defined. One such property which can be defined for a predicate is partly uniqueness. This property ensures that if a predicate is specified through constants except for one variable the variable is uniquely bound to one constant. The predicate defining the location of the object has this property. Thus, if the object is fixed the room is uniquely defined.
In to perform the mapping so we assume a fixed number $n$ of constants $c_1, \dots, c_n$. Additionally, we assume a fixed number $m$ of relational fluent terms $F_1(\bar{x}_1, s), ..., F_m(\bar{x}_m, s)$. The set of variables for the $i^{th}$ fluent is represented with $\bar{x}_i$. The mapping function is defining with the help of two separate functions. The first function maps the constants from the extended basic action theory to Cyc constants. The second function maps the fluents from the extended basic action theory to Cyc predicates.

**Definition 14.** *For constant symbols $\{c_i\}$ we define a transformation functions $R_C : c_i \in \mathcal{L}_{SC} \to \mathcal{L}_{\mathcal{KB}}$.*

**Definition 15.** *For the fluent $F_i$ which is known to be true we define a transformation function $R_{F_i} : T^{|\bar{x}_i|} \to \mathcal{L}_{\mathcal{KB}}$.*
*For the fluent $F_i$ which is known to be false we define a transformation function $\bar{R}_{F_i} : T^{|\bar{x}_i|} \to \mathcal{L}_{\mathcal{KB}}$.*
*Where $T$ denotes a term.*

Using those two function we define a proper mapping form the extended basic action theory to Cyc as follows.

**Definition 16.** *The mapping $\mathcal{M}_{\mathcal{L}_{SC}, \mathcal{L}_{\mathcal{KB}}}(\mathcal{D}^*, s)$ consists of a set of transformed fluents and constants such that:*

$$\begin{aligned} \mathcal{M}_{\mathcal{L}_{SC}, \mathcal{L}_{\mathcal{KB}}}(\mathcal{D}^*, s) &\equiv \\ \{R_{F_i}(\bar{x}_i) | \forall i \forall \bar{x}_i.(\mathcal{D}^*, s) \models_{\mathcal{L}_{SC}} F_i(\bar{x}_i, s)\} &\bigcup_{x \in \bar{x}_i} R_C(x) \cup \\ \{\bar{R}_{F_i}(\bar{x}_i) | \forall i \forall \bar{x}_i.(\mathcal{D}^*, s) \models_{\mathcal{L}_{SC}} \neg F_i(\bar{x}_i, s)\} &\bigcup_{x \in \bar{x}_i} R_C(x)) \end{aligned}$$

Before we show the functions to map the constants and fluent expression it is important to notice that we are only interested in steady invariant [139]. These invariant holds for all the time in the environment. In contrast stabilizing invariant [139] hold after some time has elapsed, e.g., a ball will hit the ground after it has started to fall. As stabilizing invariant are per definition depending on change in the environment one would need to define how the environment can change to specify such invariant. Thus, also the dynamics of the environment would need to be mapped. Additionally, one needs to properly deal with the definition of consistency in such a setting, thus consistency hold after checking the invariant for 10 seconds, 10 minutes or forever.

The function to perform the mapping are defined as follows. We define now the mapping of the constants from the extended basic action theory to Cyc as specified in Listing 5.1[*].

Listing 5.1: Mapping of the Constants

$R_C(x) \rightarrow$
```
  (isa {x} PartiallyTangible)
```

The mapping of the fluents in the extended basic action theory to predicates in Cyc is specified in Listing 5.2.

Listing 5.2: Mapping of the Fluents

$R_{F_{isAt}}(x,y) \rightarrow$
```
  (isa storedAt_{x} AttachmentEvent)
  (objectAdheredTo storedAt_{x} {y})
  (objectAttached storedAt_{x} {x})
```
$\bar{R}_{F_{isAt}}(x,y) \rightarrow$
```
  (isa storedAt_{x} AttachmentEvent)
  (not(objectAdheredTo storedAt_{x} {y}))
  (objectAttached storedAt_{x} {x})
```
$R_{F_{holding}}(x) \rightarrow$
```
  (isa robotsHand HoldingAnObject)
```
$\bar{R}_{F_{holding}}(x) \rightarrow$
```
  (not (isa robotsHand HoldingAnObject))
```
$R_{F_{carryAnything}} \rightarrow$
```
  (isa robotsHand HoldingAnObject)
```
$\bar{R}_{F_{carryAnything}} \rightarrow$
```
  (not(isa robotsHand HoldingAnObject))
```

Due to the mapping, we do not longer need to define the invariant of the common sense knowledge. Instead this invariant is checked by Cyc. Let's consider a simple example we map the fluent *isAt* to the class *AttachmentEvent* for the predicate $storedAt\_\{x\}$. This class has the property that it only allows one object of the class *PartiallyTangible* to be attached to one location. Otherwise, Cyc reports an inconsistency.
We can use now the mapping to check if a given situation is consistent. This is done with the help of Algorithm 3.
The algorithm first checks if the action is a sensing action. If not the action returns true as only sensing action can lead to an inconsistency following Theorem 3. If the action is a sensing action, it is first checked if the expected sensing result is consistent with the reported sensing result. If this is not the case, inconsistency is reported. Otherwise, the algorithm performs the mapping to Cyc and use Cyc to check for consistency. Thus, by using Lemma 10 we ensure that the algorithm properly detects an inconsistency if there is one. Furthermore, as we explore Theorem 3 to save unnecessary mappings and calls to check the consistency.

## 5.4. Related Research

Before we conclude the chapter, we will discuss some related research which is specific to this chapter.

[*] {x} and {y} represent the unique string representation of the related constant in Cyc.

**Algorithm 3:** *checkConsistency*

```
input : $\alpha$ ... action to perform in the situation,
        $s$ ... situation before execution action $\alpha$
output: true iff situation is consistent
1  if $\alpha \in \mathcal{A}_{sensing}$ then
2      if $SF(\alpha, s) \not\equiv AO(\alpha, do(\alpha, s))$ then
3          return $\perp$
4      end
5      $s' = do(\alpha, s)$
6      $\mathcal{M} = \mathcal{M}_{\mathcal{L}_{SC}, \mathcal{L}_{\mathcal{KB}}}(\mathcal{D}^*, s')$
7      if $\neg checkCycConsistency(\mathcal{M})$ then
8          return $\perp$
9      end
10 end
11 return $\top$
```

[13] presented a method which makes it possible to reuse the reasoning capabilities from one logic in another logic. This contrasts with our approach as we do not want to transfer the reasoning capabilities. Instead, we want to use the use an external oracle which verifies if a given situation is valid. Thus, we can optimize the reasoning system depending on if it should reason about the actions or common sense.
The method proposed in this chapter does not rely on a certain knowledge representation. Thus, one could use another method to check the consistency in contrast to Cyc. On such system, which can perform such a check was proposed in [59] which was extended in [58]. The system uses a simulation to check physical common sense. With the help of such a simulation on can check if the state of the world is stable. It would also open the possibilities to check invariant which is not just steady but stabilizing. In such a setting one would need to define consistency properly.
Finally, we want to mention the system proposed in [64] which used a naive physical simulation to check if the interaction of a robot with its environment is faulty. This is done by transferring the knowledge of the current world in the simulation and afterward simulate the action to check if this action is successful. Although this approach has similarities to check if a situation is consistent, we have incorporated this check into a diagnosis system. Thus, we can not only detect a faulty execution but also perform reasoning to find the root cause of the fault. Additionally, one can use this information to reason how the environment looks like. This enables the robot to react properly.

## 5.5. Conclusion and Future work

History based diagnosis uses common sense knowledge to detect inconsistencies. The specification of such a common sense knowledge is a challenging and time-consuming task. The complexity of the task is caused as the commonsense knowledge base needs to capture all relevant facts without causing any inconsistencies on its own.
To address this problem, we have shown in this chapter how to use commonsense knowledge from external sources. We have defined how a mapping can be defined between different knowledge representations. Furthermore, we have shown which theoretical properties need to hold for this mapping

to allow the usage of this mapping for a consistency check. Additionally, we have shown an implementation which uses this mapping to test the consistency of a situation. This implementation uses several theoretical results of the history based diagnosis shown in this chapter to reduce the calls to the external commonsense knowledge base.
For future work, it is of interest to in co-operating other knowledge bases. One example is [11] which can be used to define physical systems and their properties. Thus, such an integration could allow reasoning about the physical system employed by the robot more easily.
Additionally, it is left for future work to perform the mapping in a lazy fashion. Thus, only those parts are mapped which are currently needed to check the consistency. This would result in a smaller run time of the mapping approach.

Chapter 6

# Belief-Aware execution of High Level Programs

In the previous Chapters, we discussed methods which ease the use of the history based diagnosis. The robot can use these methods to generate an explanation if a fault has happened. Thus, the robot can address the qualification problem with the help of history based diagnosis. However, with the creation of an explanation alone, the robot cannot perform the necessary reasoning to decide which action should be taken as next.

To address the problem which action should be performed next, the robot needs to use the diagnosed history of performed actions to generate a belief. This belief needs to represent what the robot knows for certain. Thus, allowing the robot to draw conclusions which are very likely to be true. With the help of a series of such conclusion, the robot can decide which action to perform next.

One method to decide which action should be taken next is with the help of the IndiGolog [20] transition semantic. This semantic uses the high-level program to decide which action to perform next. This program is interpreted using the current situation of the robot. Thus, several queries on the current situation are performed before the next action is chosen. These queries only take the current situation into account and not the current belief of the agent. Thus the robot may make a dangerous decision. Thus, one needs to link the belief created with the help of the diagnoses of the history together with the transition semantic provided by IndiGolog.

In this chapter, we will discuss how the robot can create a belief out of the diagnosis of the history. Furthermore, we will discuss how one needs to adopt the IndiGolog transition semantic to decide the next action with the help of the belief. The integration of the belief of the robot into its high-level control presented in this chapter is based on the work presented in [92].

The remaining of the chapter is structured as follows. The next section discusses how the robot can draw a belief from the result of the history based diagnosis. In Section 6.2 we will discuss how this belief can be queried efficiently to allow the robot to decide in reasonable time. Afterward, we will discuss how to change the original transition semantic of IndiGolog to integrate the belief of the robot proper. This section is followed by a section which discusses briefly the formal relation between the original transition semantic and the transition semantic proposed in this chapter. Finally, we conclude the paper and point out some future work.

## 6.1. Definition of Belief for an agent

As all decision, the robot makes to choose the next action it should perform should be anchored in the belief of the robot we first need to define what the belief of the robot is. To do so, we will present two definitions of the robot's belief. The first definition defines the belief if the robot using the classical history based diagnosis. The second definition is used if the robot uses the diagnosis approach presented in Chapter 4.
To define the belief of the robot if the classical history based diagnosis is used, we first need to define the basis which should be used to form the belief. If the robot uses the classical history based diagnosis, the robot alters the history to create a consistent history. Additionally, each altered history has a cost assigned to it. This cost represents the likelihood of the changes performed on the situation. The lower the costs, the more likely the changes are. As presented in Chapter 2 a diagnosis is defined as follows.

$$Diag(s', s, v) \tag{6.1}$$

Where $s'$ is the altered version of $s$. Additionally, $s'$ is consistent with the observations made so far. $v$ is the cost imposed by the changes. As we are interested in those diagnoses which are the most likely one we define the set of minimal diagnosis as those altered situations with the minimal costs.

$$minDiag(s) \doteq \{s' | Diag(s', s, v) \land \nexists s'', v''.Diag(s'', s, v'') \land v'' < v\} \tag{6.2}$$

With the help of the minimal diagnosis, we can define what the robot belief to hold. The robot belief those formulas to hold which hold in each minimal diagnosis.

$$Belief(\phi, s) \doteq \forall s' \in minDiag(s).\phi(s') \tag{6.3}$$

Through this definition, the robot is cautious as only those formulas are belief which holds in each altered situation. However, the robot is not too cautious as only those alternative situations are considered which are equally likely and are the most likely one.
Additionally, we can state that if no fault occurs the robot will draw the same conclusion from the belief if it would use the situation term only. We state this property more formally in the following corollary.

**Corollary 2.** *If the belief of the robot is formed according to Equation 6.3 and updated according to Equation 6.2 and no fault is observed then it holds that $Belief(\phi, s)$ if and only if $\mathcal{D}^* \models \phi(s)$.*

*Proof (Sketch).* This follow directly from Equation 6.3, Equation 6.2 and the definition of a minimal diagnosis in classical history based diagnosis. □

To define those formulas, the robot believes if the robot uses the consistency history based diagnosis presented in Chapter 4 one can follow a similar thought. The robot's belief should be formed by those formulas which hold considering the most likely explanations. The set of the most likely diagnosis is defined as $\Delta^*$, according to Chapter 4. Using those diagnoses together with the definition of the reasoning used for such a diagnosis $\delta$ $\mathcal{D}^{cons}(\delta)$ we can define the belief of the robot following the definition presented in Chapter 4 as follow.

$$Belief(\phi, s) \doteq \forall \delta^* \in \Delta^*.\mathcal{D}^{cons}(\delta) \models \phi(s) \tag{6.4}$$

As in the case of classical history based diagnosis the robot performs a cautious reasoning but is not to cautious.
Again, we can state that if no fault occurs the reasoning with the belief is the same as if it would use the situation term only. We state this property more formally in the following corollary.

**Corollary 3.** *If the belief of the robot is formed according to Equation 6.4 and no fault is observed then it holds that $\mathcal{D}^* \models Belief(\phi, s)$ if and only if $\mathcal{D}^{cons}(\emptyset) \models \phi(s)$.*

*Proof (Sketch).* This follow directly from Equation 6.4 and the definition of a minimal diagnosis in consistency history based diagnosis. □

Regardless of the used history based diagnosis the agent will draw the same conclusion if no fault has occurred. Thus, the belief of the robot will only be different than the classical reasoning if a fault is observed. This is stated more formally in the following lemma.

**Lemma 11.** *If the belief of the robot is defined through Equation 6.3 or Equation 6.4 and no fault has been observed the robot will draw the same conclusion as with the original situation calculus.*

*Proof.* This is a direct result of Corollary 2 and Corollary 3. □

## 6.2. Efficient Querying the Belief

To use the belief of the robot to take a decision which next action should be performed one needs to ensure that the queering of the belief can be performed efficiently. To address this problem two issues, need to be tackled. The first issue is the time spent to calculate the explanation itself. Thus, the faster the diagnosis can be calculated the faster the robot can derive the diagnosis used to answer the queries. We refer the interested reader to [109] for a discussion of different methods to improve the diagnosis calculation for classical history based diagnosis. The second issue to address, to perform an efficient reasoning with the belief, is how the reasoning is done if the explanation is known. This reasoning is of particular interest as this reasoning is often performed whereas the diagnosis calculation is only performed if a fault has occurred. Thus, any improvement in the reasoning if a diagnosis is known has a drastically impact on the overall performance on answering a query with the belief.
To address the issue of efficient answering a query if the explanation of a fault is known we use the following observations. First, most of the time one performs several queries using the current situation to make a decision. Second, the impact an action has is most of the time very limited. Whereas the queries to answer are often spanning over several fluents. Thus, if one needs to answer the queries, one needs to consider several action effects at the same time. We will exploit both observations by choosing a reasoning method which trades the costs of propagating the effect of an action against the costs of performing a query in the current situation.
To perform a reasoning using the current situation three options exists. Either one uses the logical theory to complete this reasoning. Such a reasoning is complex as one needs to consider the second order inductive axiom to create a valid situation term. To avoid the reasoning using this axiom one can use regression. As we have discussed in Chapter 2 regression rewrites the formula till the initial situation such that one can answer the query in the initial situation. To perform this rewriting, regression roles back the query using the situation term and the successor state axioms. Thus, the time spent to answer a query depends on the length of the situation. As such the method is very efficient for short situations, terms are considered but can cause long reasoning times if the situation term becomes long. Thus this kind of reasoning may become very inefficient if the robot performs many actions. The last method to perform reasoning is by using progression. As we have discussed in Chapter 2 progression updates the initial situation such that the effect of the performed action is reflected. Thus, one can always use the advanced initial situation to answer a query. This advancement implies that the performance on the query answer does no longer depend on the length of the situation. Instead one trades the cost of

advancing the initial situation with the costs of answering the query. As argued above such a tradeoff is preferable if one considers the two above mentioned observations made on a robotic system.
With the help of progression, one creates a knowledge base which represents the current state of the world and use this knowledge base to answer the query. Such a reasoning can be simply performed if no fault has occurred. However, if a fault has occurred, one needs to deal with different possible explanations. Each of this hypothesis represents one current state of the world and needs to be considered during reasoning. To use the hypothesis for the reasoning, we need to create a knowledge base per hypothesis. Each of this knowledge bases is advanced separately and afterward used in combination to answer the query. Using Algorithm 4 the advancement of the knowledge bases is performed. The

**Algorithm 4:** AdvanceKnowledgeBases

```
Data: $\alpha \dots$ the current executed action
Data: $s \dots$ the current situation
Data: $AO(\alpha, s) \dots$ the action outcome
Data: $\mathcal{D}_{S_0} \dots$ the knowledge base of the original initial situation
Data: $W \dots$ the current set of alternative worlds. Consisting of their hypothesis about the faults
      and the resulting state of the world.
Result: $W' \dots$ the new set of alternative worlds
begin
    $W' = \{\}$ ;
    if $\alpha \in \mathcal{A}_{sensing}$ then
        foreach $\langle h, kb \rangle \in W$ do
            if $checkIfConsistent(h, \alpha, AO(\alpha, s))$ then
                $h' = addToHypothesis(h, \alpha)$ ;
                $kb' = Progress(kb, \alpha, AO(\alpha, s))$ ;
                $W' = W' \cup \{\langle h', kb' \rangle\}$ ;
            end
        end
        if $W' = \emptyset$ then
            $H' = caculateHypothesis(s, \alpha, AO(\alpha, s))$ ;
            foreach $h' \in H'$ do
                $kb' = Progress(\mathcal{D}_{S_0}, do(\alpha, s), h')$ ;
                $W' = W' \cup \{\langle h', kb' \rangle\}$ ;
            end
        end
    else
        foreach $\langle h, kb \rangle \in W$ do
            $h' = addToHypothesis(h, \alpha)$ ;
            $kb' = Progress(kb, \alpha)$ ;
            $W' = W' \cup \{\langle h', kb' \rangle\}$ ;
        end
    end
end
```

algorithm first checks if the current action is a sensing action. If this is the case, the algorithm exploits the theoretical results presented in Chapter 5, which states that only a sensing action can lead to a

contradiction. The algorithm exploits this result by first checking which hypothesis of faulty actions is still consistent with the currently performed action and the sensing result. If the hypothesis is consistent, the algorithm advances the hypothesis by advancing it with the current action. Additionally, the knowledge base which is associated with the hypothesis is advanced with the current action and sensing outcome. The advanced hypothesis together with its advanced knowledge base is afterward added as the new alternative world.
If no hypothesis remains no hypothesis is consistent with the current action. Thus, the algorithm checks if the alternative world set is empty which is verified in Line 11. In such a case a new set of hypothesis needs to be generated. For each of this hypothesis, the initial knowledge base needs to be progressed till the current situation. This progression needs to consider the current hypothesis to create a new knowledge base which is associated with this new hypothesis. These new alternative worlds are gathered and returned by the algorithm.
If the current action is not a sensing action, one can exploit the theoretical results presented in Chapter 5. As only a sensing action can lead to inconsistency, the algorithm only advances the hypothesis with the current action. Additionally, the knowledge bases which are associated with the hypothesis are advanced. Finally, both the advanced hypothesis and the associated advanced knowledge bases are gathered as new alternative worlds. The algorithm terminates in $O(|H| \times |F|)$ steps, thus the number of hypothesis should be as small as possible to ensure a fast progression.
To use this algorithm one needs to define what a hypothesis is which defines the world. Additionally, depending on the hypothesis used the advancement of the hypothesis may look differently. The hypothesis we are considering is those which are either formed by the classical history based diagnosis or by the consistency history based diagnosis presented in this thesis.
If we consider the classical history based diagnosis the set of hypothesis $H$ for a current situation is define by the set of alternative situations which form the minimal diagnosis.

$$H = minDiag(s) \tag{6.5}$$

Due to this definition, every hypothesis $h$ is a situation term $s$. Thus, we can advance a hypothesis $h$ (represented by situation $s$) by the action $\alpha$ with standard means of the situation calculus as follows.

$$s' = do(\alpha, s) \tag{6.6}$$

Where $s'$ is the situation which represents the advanced hypothesis $h'$.
If we consider the consistency history based diagnosis approach presented in this thesis the set of hypothesis $H$ correspond to the set of minimal diagnosis $\Delta^*$.

$$H = \Delta^* \tag{6.7}$$

Due to this definition, every hypothesis $h$ is a diagnosis $\delta$ describing a minimal set of actions which are considered to be faulty to generate a consistent situation. To advance such a hypothesis $h$ nothing needs to be changed on the hypothesis $h$ as we consider the action $\alpha$ which should advance as not faulty. Thus, the diagnosis does not alter. Thus, the advancement results in no change.

$$h' = h \tag{6.8}$$

With the help of Algorithm 4 one can always use the current set of alternative worlds to perform the reasoning. This reasoning is done by using the different knowledge bases of the alternative worlds to decide if the agent believes a formula or not. The reasoning is done as follows.

$$Belief(\phi, s) \doteq \forall kb. (\exists h. \langle h, kb \rangle \in W) \rightarrow kb \models \phi(s) \tag{6.9}$$

The belief is those formulas which can be derived with all the knowledge bases which are within the current alternative worlds $W$. Thus, the answering of the query becomes a set of simple reasoning tasks on a knowledge base instead of a basic action theory.
The advancement of all alternatives can become a complex task, especially if one needs to consider that the number of hypotheses could be huge. Thus, one would prefer only to advance one knowledge base which represents the belief of the robot. To do so, we need to make the following assumption.

**Assumption 5.** *The number of hypothesis h is finite*

By exploiting the assumption, we can merge the knowledge bases of the robot. This is done by the following observations due to a finite set of hypothesis one can redefine the belief as follows.

$$Belief(\phi, s) \doteq \bigvee_{kb.\exists h.\langle h,kb\rangle \in W} \models \phi(s) \tag{6.10}$$

Thus, one can create a knowledge base $\overline{KB}$ which captures the belief of the robot by creating a disjunction of the different knowledge bases which define the alternative worlds.

$$\overline{KB} \doteq \bigvee_{kb.\exists h.\langle h,kb\rangle \in W} kb \tag{6.11}$$

As we are interested in using only this knowledge base we need to ensure that we can simple progress this knowledge base instead of each knowledge base separately. Using the assumption from above we can state the following theorem to show that one needs only to progress $\overline{KB}$.

**Theorem 4.** *If a knowledge base $\overline{KB}$ which consisting of n knowledge bases $\overline{KB} = kb_1 \vee \ldots kb_n$ is advanced by the action $\alpha$ to the knowledge base $\overline{KB}'$ then it holds under Assumption 5 and a causal completeness assumption of actions that $\overline{KB}' \models F \leftrightarrow \bigvee_{kb.\exists h'.\langle h',kb'\rangle \in W'} kb \models F$. Where $W'$ is the result of advancing each hypothesis and corresponding knowledge base separately.*

*Proof (Sketch).* Proof by contradiction.
Let's assume there exists a fluent $F$ such that $\overline{KB}' \models F$ but $\bigvee_{kb.\exists h'.\langle h',kb'\rangle \in W'} kb \not\models F$. We need to distinguished two cases of how one can reason that $F$ holds. Either $F$ was true before or caused due to progression to be true.
In the first case, the fluent was not changed. Thus it needs to holds that the separate progression has changed the value of the fluent such that $\neg F$ holds in at least one alternative. Thus, there is an effect axiom $\theta \rightarrow \neg F$. Furthermore at least in one alternative $kb$ needs to hold $\theta$. If it holds in all alternatives, it does also hold in $\overline{KB}$. Thus, the advancement of $\overline{KB}$ would cause $\neg F$, which causes that $\overline{KB}' \not\models F$. Thus, there needs to exist at least one alternative such that $\theta$ does not hold. Thus $\overline{KB}$ contains $\theta \vee \neg\theta$. This result implies that the effect axiom can be applied but also that the original value can be carried over. As such $F$ does not hold in the progressed knowledge base $\overline{KB}'$.
In the second case, the fluent was changed thus it needs to hold that there exists an effect axiom $\theta \rightarrow F$. As $F$ was caused due to progression it holds that $\overline{KB} \models \theta$. Thus, it further holds that each alternative knowledge base $kb \models \theta$. Thus, the advancement of each of this knowledge bases needs also to yield $F$.
Let's assume there exists a fluent $F$ such that $\overline{KB}' \not\models F$ but $\bigvee_{kb.\exists h'.\langle h',kb'\rangle \in W'} kb \models F$. We need to distinguished two cases of how one can reason that $F$ holds. Either $F$ was true before or caused due to progression to be true.
In the first case, there needs to exists an effect axiom of the following form axiom $\theta \rightarrow \neg F$ which cause the fluent $F$ to be false. This effect axiom either causes that $\neg F$ holds in $\overline{KB}'$ or that neither

$F$ nor $\neg F$ holds. If $\neg F$ holds that it follows that $\overline{KB} \models \theta$ holds. Thus $\neg F$ needs also to hold in the knowledge base which was created by separately progressing each knowledge base. If neither $F$ nor $\neg F$ holds then neither $\theta$ nor $\neg\theta$ holds in $\overline{KB}$. Thus, there exists at least one alternative $kb \not\models \neg\theta$. Thus, the progression of at least one alternative cannot conclude $F$.
In the second case, the fluent was changed thus it needs to hold that there exists an effect axiom $\theta \rightarrow F$. As $F$ was caused due to progression it needs to hold that for each alternative it holds that $kb \models \theta$. Thus, it further holds that $\overline{KB} \models \theta$. Thus $\overline{KB}' \models F$. This result is also a contradiction to the assumption. □

The theorem states that if one progresses the combined knowledge base one can draw the same conclusion as if one would progress the separated knowledge bases if Assumption 5 holds.
With the help of Theorem 4 we can now change Algorithm 4 into Algorithm 5 which uses the combined knowledge base for progression. The algorithm first checks if the current action is a sensing

**Algorithm 5:** AdvanceKnowledgeBases2

```
Data: $\alpha \ldots$ the current executed action
Data: $s \ldots$ the current situation
Data: $AO(\alpha, s) \ldots$ the action outcome
Data: $\mathcal{D}_{S_0} \ldots$ the knowledge base of the original initial situation
Data: $\overline{KB} \ldots$ the current knowledge base of the robot
Result: $\overline{KB}' \ldots$ the new knowledge base of the robot
begin
    if $\alpha \in \mathcal{A}_{sensing}$ then
        if $isConsistent(\overline{KB}, \alpha, AO(\alpha, s))$ then
            $\overline{KB}' = Progress(\overline{KB}, \alpha, AO(\alpha, s))$;
        else
            $H' = caculateHypothesis(s, \alpha, AO(\alpha, s))$ ;
            $\overline{KB}' = \perp$ ;
            foreach $h' \in H'$ do
                $kb' = Progress(\mathcal{D}_{S_0}, do(\alpha, s), h')$ ;
                $\overline{KB}' = \overline{KB}' \vee kb'$ ;
            end
        end
    else
        $\overline{KB}' = Progress(\overline{KB}, \alpha)$;
    end
end
```

action. If this is the case, the sensing action is checked if it is consistent with the belief of the agent. Please, not that a sensing action can only be inconsistent with the current belief of the agent if all hypothesis is inconsistent with the sensing result. This inconsistency is a direct consequence of Theorem 4. If the sensing action is consistent with the current knowledge base, the knowledge base is progressed. Otherwise, all hypothesis is calculated, and the knowledge base is created by a disjunction of all the progressed alternatives. If the action was not a sensing action, the knowledge base is simply progressed. The algorithm terminates in $O(|H| \times |F|)$ steps, thus the number of hypothesis should be as small as possible to ensure a fast progression.

As we also progress the merged knowledge base, not every progression method can be simply applied. Some of the progression methods need a certain structure of the initial knowledge to be applied. As the initial knowledge base is the only thing we change due to the merging of the knowledge bases, we will discuss two progression methods which need a certain structure of the initial knowledge bases. The progression method was part of the survey of progression methods presented in [147].

**Relative complete knowledge base** If one performs progression with a relative complete knowledge base, the initial knowledge base needs to have the following structure.

$$\forall \vec{x}.F(\vec{x}, S_0) \equiv \phi(\vec{x}) \tag{6.12}$$

In the above formula, $\phi$ needs to be situation independent. Through this form of initial knowledge, one can use the successor state axioms to perform the progression. The progression is done by replacing the right-hand side of the successor state axiom with the definition of the fluent of the current knowledge base. This replacement yields a formula with the same structure again.
Unfortunately, due to the disjunction of the knowledge bases, we disrupt this structure. As such the progression method, can't be simply applied to progress the combined knowledge base.

**Bounded unknowns** If one performs progression with bounded unknowns in the initial knowledge base the initial knowledge base has the following structure.

$$\exists \vec{w}.e(\vec{w}) \land \bigwedge_{F_i} .\forall \vec{x}.F_i(\vec{x}, S_0) \equiv \phi(\vec{x}, \vec{w}) \tag{6.13}$$

Where $e$ is situation independent and defines a constrain on the variables $\vec{w}$ which is used in the fluent terms.
As the core structure of the initial knowledge does not allow disjunctions as it was the case for the previous progression method, it is not possible to use this progression method with the above mentioned merged knowledge bases.

### 6.2.1. Progression for fault based history based diagnosis

Regardless of the algorithm, we use to create the belief of the robot, in either case, one needs to progress the initial knowledge base for each calculated hypothesis. If we consider the case of the classical history based diagnosis, we deal with an alternative situation. Thus, one can use standard progression method to advance the knowledge base as a situation term is known. We refer the interested reader to [147] for several progression methods.
The only thing which needs to be considered is that the progression operator only considers the effect an action has. Besides the effect, an action has we also need to deal with the sensing result of this action. The dealing with the sensing result can be simply done. As a sensing action only asserts a formula in the current situation we just need to add the following formula to the current knowledge base to properly progress the knowledge base in case of a sensing action.

$$AO(\alpha, S_0) \leftrightarrow SF(\alpha, S_0) \tag{6.14}$$

In the above formula, $\alpha$ is the currently performed sensing action. Additionally, we must add either $AO(\alpha, S_0)$ or $\neg AO(\alpha, S_0)$ to the knowledge base depending on the action outcome.
Unfortunately, not all sensing action can be combined with ever progression method. Some of the progression methods demand a certain structure of the initial knowledge base. As such we add the

effect of a sensing action to the initial knowledge base this structure can become disrupted. We will briefly discuss in the remaining of the subsection two progression method imposing constraints on the initial situation and their restrictions on sensing actions.

**Relative complete knowledge base** In case of progression using a relative complete knowledge base we have the following structure.

$$\forall \vec{x}.F(\vec{x},S_0) \equiv \phi(\vec{x}) \tag{6.15}$$

As such we can't use complex sensing results. Especially any kind of quantification cannot be represented. Thus, we restrict the sensing result to the following form.

$$SF(\alpha(\vec{x},s) \equiv F(\vec{y},s)\ \mu\ \psi(\vec{y}) \tag{6.16}$$

Where $\vec{y}$ is a subset of variables of $\vec{(x)}$. And there are no free variables in $\psi$. Furthermore $\mu$ is a binary logical operator. Thus, we either need to add $F(\vec{t},S_0)$ or $\neg F(\vec{t},S_0)$ to the knowledge base where $\vec{t}$ is a vector of constants. This allows us to simple add the sensing formula by either changing the initial knowledge base to.

$$\forall \vec{x}.F(\vec{x},S_0) \equiv \phi(\vec{x}) \wedge \vec{x} = \vec{t}\ \mu\ \psi(\vec{x}) \tag{6.17}$$

Respectively.

$$\forall \vec{x}.F(\vec{x},S_0) \equiv \phi(\vec{x}) \wedge \neg\left(\vec{x} = \vec{t}\ \mu\ \psi(\vec{x})\right) \tag{6.18}$$

**Bounded unknowns** If one allows progression with bounded unknowns the initial knowledge base has the following structure.

$$\exists \vec{w}.e(\vec{w}) \wedge \bigwedge_{F_i}.\forall \vec{x}.F_i(\vec{x},S_0) \equiv \phi(\vec{x},\vec{w}) \tag{6.19}$$

Through the quantification of the initial knowledge base we can use quantification in the sensing formulas. For simplicity, we will only present the version with all quantification.

$$SF(\alpha(\vec{x},s) \equiv \forall \vec{z}.F(\vec{y},\vec{z},s)\ \mu\ \psi(\vec{y},\vec{z}) \tag{6.20}$$

Where $\vec{y}$ is a subset of variables of $\vec{(x)}$. And there are no free variables in $\psi$. Furthermore $\mu$ is a binary logical operator. Thus, we either need to add $\forall \vec{z}.F(\vec{t},\vec{z},S_0)$ or $\exists \vec{z}.\neg F(\vec{t},S_0)$ to the knowledge base where $\vec{t}$ is a vector of constants. We can now change the initial knowledge base as follows considering the index fluent $F_k$ to be mentioned on the right-hand side of the sensing formula.

$$\exists \vec{w}.e(\vec{w}) \wedge \bigwedge_{i \neq k}.\forall \vec{x},\vec{z}.F_i(\vec{x},S_0) \equiv \phi(\vec{x},\vec{w}) \wedge \forall \vec{z}.F_k(\vec{x},S_0) \equiv \phi(\vec{x},\vec{w}) \wedge \vec{x} = \vec{t}\ \mu\ \psi(\vec{x},\vec{z},s) \tag{6.21}$$

Respectively.

$$\exists \vec{w},\vec{z}.e(\vec{w}) \wedge \bigwedge_{i \neq k}.\forall \vec{x}.F_i(\vec{x},S_0) \equiv \phi(\vec{x},\vec{w}) \wedge F_k(\vec{x},S_0) \equiv \phi(\vec{x},\vec{w}) \wedge \neg\left(\vec{x} = \vec{t}\ \mu\ \psi(\vec{x},\vec{z},s)\right) \tag{6.22}$$

### 6.2.2. Progression for consistency based history based diagnosis

If one considers the robot builds its belief with the help of consistency history based diagnosis one needs to consider how to incorporate the sensing results as well as how to deal with the modified successor state axioms.
To consider the sensing axioms two cases need to be distinguished. The first case is that the current action which is performed is deemed to be non-faulty. Thus, the sensing axiom applies as in the previous case of classical history based diagnosis. Moreover, such the same methods for the integration of sensing action as discussed in the previous subsection can be used. The second case is that the current action is considered to be faulty one does not need to consider the sensing axiom at all. Thus, one can just omit the integration of the sensing axiom.
To perform a progression with the modified successor state axiom of the consistency history based diagnosis we need to modify the successor state axioms to follow the common form of.

$$F(\vec{x},do(\alpha,s)) \leftrightarrow \varphi^{+}(\alpha,\vec{x},s) \vee F(\bar{x},s) \wedge \neg\varphi^{-}(\vec{x},s) \tag{6.23}$$

This common form is the basis for most of the progression methods to discuss the restriction which need to be applied to allow progression. As the successor state axioms of the consistency history based diagnosis does not follow this common form we need to modify these axioms into the following form.

$$\begin{aligned}F(\vec{x},do(\alpha,s)) \leftrightarrow\ & \varphi^{+}(\alpha,\vec{x},s) \wedge \left[\psi^{F}_{\alpha}(\vec{x},s) \rightarrow \left(\neg\left(\exists n.\bar{fail}(\alpha,n) \wedge curN(n,s)\right) \wedge \Pi_{\alpha}(s)\right)\right] \vee \\ & \left(\exists n.\bar{F}'(\vec{x},n) \wedge curN(n,s)\right) \wedge \neg\left[\psi^{F}_{\alpha}(\vec{x},s) \rightarrow \left(\neg\left(\exists n.\bar{fail}(\alpha,n) \wedge curN(n,s)\right) \wedge \Pi_{\alpha}(s)\right)\right] \vee \\ & F(\vec{x},s) \wedge \neg\varphi^{-}(\alpha,\vec{x},s) \wedge \left[\psi^{F}_{\alpha}(\vec{x},s) \rightarrow \left(\neg\left(\exists n.\bar{fail}(\alpha,n) \wedge curN(n,s)\right) \wedge \Pi_{\alpha}(s)\right)\right] \vee \\ & F(\vec{x},s) \wedge \neg\left(\exists n.\bar{F}'(\vec{x},n) \wedge curN(n,s)\right) \wedge \\ & \neg\left[\psi^{F}_{\alpha}(\vec{x},s) \rightarrow \left(\neg\left(\exists n.\bar{fail}(\alpha,n) \wedge curN(n,s)\right) \wedge \Pi_{\alpha}(s)\right)\right]\end{aligned} \tag{6.24}$$

The first case of the successor state axiom covers the case where no action has occurred and the fluent value is set to true due to some action. The second case of the successor state axiom covers the case where the action execution failed and the truth value is determined by $F'$. The predicate $F'$ is used to decouple the truth value of the fluent from its past. As such this predicate, does not have any successor state axiom. The last argument of the predicate $n$ is a running number which is used to distinguished the predicate instances depending on the time the robot has executed an action. The third case of the successor state axiom covers the case where the action was not faulty and the fluent is influenced by a negative effect axiom. The last case is by setting the fluent value to false if a fault has occurred and the truth value of the predicate $F'$ is false.
To make the predicate *fail*, indicating a fault of an action as well as the predicate $F'$ used to decouple the truth value of the fluent with its past situation independent we use a running number. This running number is defined with the fluent *curN*. In the initial knowledge base, we state that the value is set to 0.

$$\forall n.curN(n,S_0) \equiv n = 0 \tag{6.25}$$

Furthermore, we advance the current number always by one with the following successor state axiom[*].

$$\forall n,\alpha,s.curN(n,do(\alpha,s)) \equiv n = n' + 1 \wedge curN(n',s) \tag{6.26}$$

Due to this special form the fluent *curN* can be simply progressed by increasing the value $n$ for each action. Thus, we will not consider this fluent in the remaining of our discussion.

[*]We assume a proper axiomatization for the natural numbers.

The successor state axiom stated in Equation 6.24 can now be used to analyze which restrictions need to apply to use progression. In the remaining of the subsection, we will discuss several progression methods presented in [147] which pose some constraints on the successor state axioms.

**Strong context-free actions** If progression with strong context-free actions is performed one considered that $\varphi^+$ and $\varphi^-$ to be restricted to only contain equality constraints between $\vec{x}$ and a vector of constants. This restriction also implies that no situation depending formulas are allowed. As at least the fluent *curN* is situation depended on this progression method cannot be used.
Instead one would need to "compile" the successor state axiom such that one replaces the fluent *curN* by its only value. Thus, if all the actions effect, the action influence and the precondition of the action are defined only with the help of equality constraints the method can be applied.

**Context-free actions** If progression with context-free actions is performed one considered that $\varphi^+$ and $\varphi^-$ to be restricted to mention only situation independent predicates. Thus the same problems occur as with the previous method as at least *curN* is situation depended. Again, one can use the trick to "compile" the successor state axioms before applying them to use the method.

**Local-effect actions** If progression with local-effect actions is performed the variable vector $\vec{x}$ of the fluent needs to be contained in any variable vector $\vec{y}$ of any action mentioned at the right-hand side. Additionally, all predicates use only those variables referred to in action. Thus, if the action is executed all variables are either bound, or the expression becomes trivially false due to the unique name assumption of actions. Thus, this method can be applied if all conditions of the action effect, the action influence and the precondition of the action using only variables mentioned in $\vec{y}$. Please note $\bar{fail}$ is restricted to have only variables of $\vec{y}$ as an action term is used. Additionally, the predicate $F'$ does only mention the variables $\vec{x}$ thus this predicates does comply with the restrictions.

**Normal actions** If progression with normal actions is performed the successor state axioms either have the local-effect property or they use only fluents which have the local effect property for the current situation. As in the previous case of local-effect actions the progression method can be applied if all conditions of the action effect, the action influence and the precondition of the action comply with the restrictions.

## 6.3. Integration into IndiGolog execution

Besides the definition of the belief of the robot and how to perform an efficient reasoning with this belief one needs to consider how to integrate this belief into the high-level control of the robot. As the integration depends on the high-level control of the robot, we focus on one high-level control. We focus our attention to IndiGolog. As we have discussed in Chapter 2 IndiGolog allows to mix declarative and imperative programming. As such it allows different programs ranging from pure imperative to pure declarative. Thus, covering a wide range of possible high-level control approaches. To integrate the belief of the robot into IndiGolog on could just use the existing transition semantic of IndiGolog and replace the queries to the robotic system with queries to the belief. Unfortunately, this does often not lead to the desired result. Let's consider a simple example to point out the problem we are facing. Let's consider the following program $\delta =$ **if** $\phi$ **then** *nill* **else** $\alpha$ **endif** where *nil* is the empty program and $\alpha$ is some action. If we would modify the transition semantic of IndiGolog just to use the belief of the agent instead of the robot itself the program would have the decision tree depicted in

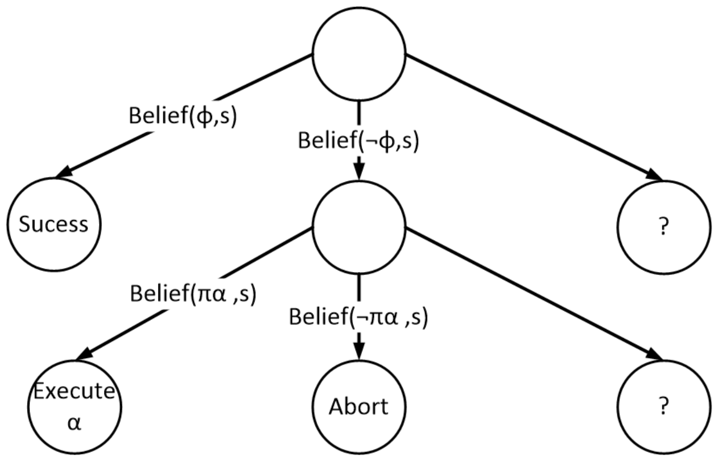

Figure 6.1.: Decision tree of a simple high-level control program.

Figure 6.1. If the robot currently believes the condition $\phi$ to hold the if branch is taken and program finish with success. If the robot currently believes $\neg\phi$ the else branch of the program is taken. It the robot neither belief $\phi$ nor $\neg\phi$ the robot is not able to decide what to do. One can use a closed world assumption thus if $\phi$ does not hold $\neg\phi$ is true. However, this would cause the robot to execute the else branch even in a case when it does not know if the condition is true or false. Furthermore, if the robot executes the else branch, it must decide according to the precondition of the action $\alpha$. The robot can execute $\alpha$ if the robot believes that the precondition $\pi\alpha$ holds. If the robot belief that the precondition does not hold $\neg\pi\alpha$ the robot cannot execute the action $\alpha$ and thus needs to abort its mission. However, if the robot cannot decide if the precondition or its negation hold the robot needs to take a decision what to do. It either can perform the action on chance, which may be dangerous. Alternatively, it can abort the mission even the action may have been possible to execute. To circumvent this problem, we want to distinguish the cases where the robot belief that something is not possible in those cases where the robot has not enough knowledge about the world to take a decision.
To distinguish these two cases, we use two pairs of predicates. The first pair *KTrans* and *KFinal* decide if the robot has enough knowledge about the world to decide if it can take a transition or needs to terminate. The second pair $\overline{Trans}$ and $\overline{Final}$ specify which transition the robot should take. These two predicates correspond to the original transition semantic where queries to the robot where replaced to the belief of the robot.
To easy the representation we use the following macro which defines if the robot belief a formula $\phi$ or its negation.

$$BWhether(\phi(s)) \doteq Belief(\phi, s) \vee Belief(\neg\phi, s) \tag{6.27}$$

Furthermore, we will only discuss the transition semantic for the non-concurrent part. As the diagnosis method, which is used to generate the belief does not deal with concurrency explicit, the diagnosis may be not adequate to capture concurrent programs.
We define now the predicate *KTrans* as follows.

- $KTrans(nil, s) \equiv \top$. The robot can always decide that the empty program cannot be used for a transition.
- $KTrans(\alpha, s) \equiv BWhether(\Pi_\alpha, s)$. The robot can decide if an action can be performed or the mission needs to be aborted if it belief the precondition or the negated precondition.

- $KTrans(\phi?, s) \equiv BWhether(\phi, s)$. The robot can decide if a test action is fulfilled or should lead to an abortion of the mission if it belief the test condition or the negated test condition.
- $KTrans(\delta_1; \delta_2, s) \equiv KTrans(\delta_1, s) \vee \overline{Final}(\delta_1, s) \wedge KTrans(\delta_2, s)$. The robot can decide about a transition of a sequence if either it can decide for the first part of the sequence if it can perform a transition or the first part is finished and it can decide for the second part the transition.
- $KTrans(\delta_1 \mid \delta_2, s) \equiv KTrans(\delta_1, s) \vee KTrans(\delta_2, s)$. The robot can decide how to proceed on a non-deterministic branch if one of the branches can be decided.
- $KTrans(\pi x.\delta, s) \equiv \exists t.KTrans(\delta|^x_t, s)$. The robot can decide how to proceed on a non-deterministic argument pick if a variable can be found such that the robot can decide if it can perform a transition.
- $KTrans(\delta^*, s) \equiv KTrans(\delta, s)$. The robot can decide if a transition is possible on a non-deterministic iteration if the robot can decide if a transition is possible for the program within the loop.
- $KTrans(\textbf{if } \phi \textbf{ then } \delta_1 \textbf{ else } \delta_2 \textbf{ endif}, s) \equiv BWhether(\phi, s) \wedge [Belief(\phi, s) \wedge KTrans(\delta_1, s) \vee Belief(\neg\phi, s) \wedge KTrans(\delta_2, s)]$. The robot can decide a transition on a branch if it belief either the condition or its negation. Furthermore, depending on the condition the branch needs also to be decidable to take a transition.
- $KTrans(\textbf{while } \phi \textbf{ do } \delta \textbf{ endWhile}, s) \equiv BWhether(\phi, s) \wedge [Belief(\phi, s) \wedge KTrans(\delta, s) \vee Belief(\neg\phi, s)]$. The robot can decide a transition on a while loop if the robot belief the loop condition or its negation. If the robot belief the loop conditions the program within the loop needs to be decidable to take a transition as well.

To decide if the robot has enough knowledge about the world to determine if it can terminate with a success we use the predicate *KFinal* which is defined as follow.

- $KFinal(nil, s) \equiv \top$. The robot can always decide termination for the empty program.
- $KFinal(\alpha, s) \equiv \top$. The robot can always decide termination for one action.
- $KFinal(\phi?, s) \equiv \top$. The robot can always decide termination for a test condition.
- $KFinal(\delta_1; \delta_2, s) \equiv KFinal(\delta_1, s) \wedge KFinal(\delta_2, s)$. The robot can decide termination of a sequence if it can decide termination of both parts of the sequence.
- $KFinal(\delta_1|\delta_2, s) \equiv KFinal(\delta_1, s) \vee KFinal(\delta_2, s)$. The robot can decide termination of a non-deterministic branching if one branch can be decided.
- $KFinal(\pi v.\delta, s) \equiv \exists x KFinal(\delta^v_x, s)$. The robot can decide termination of a non-deterministic choice of variables if there exists a variable such that the robot can decide termination of the program.
- $KFinal(\delta^*, s) \equiv \top$. The robot can always decide termination for a non-deterministic iteration.
- $KFinal(\textbf{if } \phi \textbf{ then } \delta_1 \textbf{ else } \delta_2 \textbf{ endif}, s) \equiv BWhether(\phi, s) \wedge [Belief(\phi, s) \wedge KFinal(\delta_1, s) \vee Belief(\neg\phi, s) \wedge KFinal(\delta_2, s)]$. The robot can decide termination of a condition if the robot belief the condition or belief the negation of the condition and depending on the condition the robot can decide termination of the branch.

- $\mathit{KFinal}(\textbf{while}\ \phi\ \textbf{do}\ \delta\ \textbf{endWhile}, s) \equiv \mathit{BWhether}(\phi, s) \land [\mathit{Belief}(\phi, s) \land \mathit{KFinal}(\delta, s) \lor \mathit{Belief}(\neg\phi, s)]$. The robot can decide termination of a while loop if the robot belief the loop condition or belief the negation of the loop condition. If the loop condition holds the robot furthermore needs to be able to decide for termination of the program within the loop.

With the above defined predicates the robot can decide if it can take a decision. If the robot can take a decision the robot can progress further to take the decision. To take a decision for a program transition the robot uses the predicate $\overline{\mathit{Trans}}$ which is defined as follows.

- $\overline{\mathit{Trans}}(\mathit{nil}, s, \delta', s') \equiv \bot$. The empty program cannot lead to any further execution.
- $\overline{\mathit{Trans}}(\alpha, s, \mathit{nil}, \mathit{do}(\alpha, s)) \equiv \mathit{Belief}(\Pi_\alpha, s)$. If the action precondition is believed the action can be executed and yield an empty remaining program. Furthermore, the resulting situation is created by extending the current situation by the executed action.
- $\overline{\mathit{Trans}}(\phi?, s, \mathit{nil}, s) \equiv \mathit{Belief}(\phi, s)$. If the condition is believed the program can proceed to the empty program.
- $\overline{\mathit{Trans}}(\delta_1; \delta_2, s, \delta', s') \equiv \exists\delta''.\delta' = \delta''; \delta_2 \land \overline{\mathit{Trans}}(\delta_1, s, \delta'', s') \lor \overline{\mathit{Final}}(\delta_1, s) \land \overline{\mathit{Trans}}(\delta_2, s, \delta', s')$. The program either performs a transition in $\delta_1$ if this is possible. If $\delta_1$ can terminate the program performs a transition according to $\delta_2$.
- $\overline{\mathit{Trans}}(\delta_1 \mid \delta_2, s, \delta', s') \equiv \overline{\mathit{Trans}}(\delta_1, s, \delta', s') \lor \overline{\mathit{Trans}}(\delta_2, s, \delta', s')$. The program takes either the transition of $\delta_1$ or the transition of $\delta_2$.
- $\overline{\mathit{Trans}}(\pi x.\delta, s, \delta', s') \equiv \exists t.\overline{\mathit{Trans}}(\delta|^x_t, s, \delta', s')$. The program performs a transition if a valid instance for the parameter $x$ can be found.
- $\overline{\mathit{Trans}}(\delta^*, s, \delta', s') \equiv \exists\delta''.\delta' = \delta''; \delta^* \land \overline{\mathit{Trans}}(\delta, s, \delta'', s')$. The program performs a transition in $\delta$ to yield $\delta''$. This program is further executed till it is finished. Afterwards the loop can be started again. Thus, the loop is unrolled and executed.
- $\overline{\mathit{Trans}}(\textbf{if}\ \phi\ \textbf{then}\ \delta_1\ \textbf{else}\ \delta_2\ \textbf{endif}, s, \delta', s') \equiv \mathit{Belief}(\phi, s) \land \overline{\mathit{Trans}}(\delta_1, s, \delta', s') \lor \mathit{Belief}(\neg\phi, s) \land \overline{\mathit{Trans}}(\delta_2, s, \delta', s')$. If the condition is believed $\delta_1$ is considered for the transition. If the negation of the condition believed $\delta_2$ is considered for the transition.
- $\overline{\mathit{Trans}}(\textbf{while}\ \phi\ \textbf{do}\ \delta\ \textbf{endWhile}, s, \delta', s') \equiv \exists\gamma : \delta' = \gamma; \textbf{while}\ \phi\ \textbf{do}\ \delta\ \textbf{endWhile} \land \mathit{Belief}(\phi, s) \land \mathit{Trans}(\delta, s, \gamma, s')$. If the condition is believed the transition is process further with the inner program. This program is executed till it finishes. Afterwards the loop can be started again. Thus, like the non-deterministic iteration the loop is unrolled.

Additionally, to take a decision to perform a transition the robot can take a decision to terminate safely. This is done with the predicate $\overline{\mathit{Final}}$ which is defined as follows.

- $\overline{\mathit{Final}}(\mathit{nil}, s) \equiv \top$. The empty program is a legal termination.
- $\overline{\mathit{Final}}(\alpha, s) \equiv \bot$. An action is never considered as a legal termination.
- $\overline{\mathit{Final}}(\phi?, s) \equiv \bot$. A condition is never considered as a legal termination.
- $\overline{\mathit{Final}}(\delta_1; \delta_2, s) \equiv \overline{\mathit{Final}}(\delta_1, s) \land \overline{\mathit{Final}}(\delta_2, s)$. The sequence is a legal termination if both parts of a sequence can terminate legally.

- $\overline{Final}(\delta_1 \mid \delta_2, s) \equiv \overline{Final}(\delta_1, s) \vee \overline{Final}(\delta_2, s)$. If either of the program can terminate the non-determinism branch can terminate legally.
- $\overline{Final}(\pi x.\delta, s) \equiv \exists t.\overline{Final}(\delta|_t^x, s)$. The program can terminate if a valid instance for the parameter $x$ can be found. Pleas not that $\delta|_t^x$ substitutes ever occurrence of $x$ with $t$ in the program $\delta$.
- $\overline{Final}(\delta^*, s) \equiv \top$. A legal termination is all way possible. Thus, the non-deterministic loop can instead of iterate terminate.
- $\overline{Final}(\textbf{if } \phi \textbf{ then } \delta_1 \textbf{ else } \delta_2 \textbf{ endif}, s) \equiv Belief(\phi, s) \wedge \overline{Final}(\delta_1, s) \vee Belief(\neg\phi, s) \wedge \overline{Final}(\delta_2, s)$. If the condition is believed the program can terminate if $\delta_1$ can terminate. If the negation of the condition believed $\delta_2$ is to decide legal termination.
- $\overline{Final}(\textbf{while } \phi \textbf{ do } \delta \textbf{ endWhile}, s, \delta', s') \equiv Belief(\neg\phi, s) \vee \overline{Final}(\delta, s)$. The loop stops if either the negation of the condition is believed or the body of the loop can terminate.

Through the changed transition semantic the robot decides a transition or a termination according to the belief it has. There is one difference between the transition semantic specified through the two predicates $\overline{Trans}$ and $\overline{Final}$ compared to the original transition semantic of IndiGolog. This difference is that the robot does not use the situation term directly to determine the truth value of a query but use the belief of the robot instead. Thus, as long the agent does not encounter any problem due to a faulty action the transition semantic will behave as the original transition semantic of IndiGolog.
To use the both parts of the proposed transition semantic, all four predicates need to be properly combined. This combination should ensure that the robot takes a transition if possible, terminates if it is allowed or necessary and distinguished those cases where the robot is lacking knowledge to those where it is certain. This combination is achieved by the following decision tree which is depicted in Figure 6.2.
If the robot can decide if it is finished it first checks if it can terminate legally. If this is not the case the and the robot can decide for transition the robot searches for a transition. If no transition is possible the robot aborts is mission. If a transition is possible the transition is taken. If either the robot is lacking knowledge about the environment for either the termination or the transition, then the robot needs to gather knowledge. We can define this behavior more formally as follows.

**Definition 17.** *The on-line execution semantics for a situation s and a remaining program $\delta$ behaves as follows:*

1. *Terminate with a success if*
$$\mathcal{D}^* \models KFinal(\delta, s) \wedge \overline{Final}(\delta, s)$$
2. *Perform a transition to the new program $\delta'$ without executing an action if*
$$\mathcal{D}^* \models KTrans(\delta, s) \wedge \overline{Trans}(\delta, s, \delta', s)$$
3. *Perform a transition to the new program $\delta'$ and execute the action $\alpha$ if*
$$\mathcal{D}^* \models KTrans(\delta, s) \wedge \overline{Trans}(\delta, s, \delta', do(\alpha, s))$$
4. *Create a program to gather knowledge about the environment $\delta'$ and execute the program $\delta'$ if one of the following conditions holds:*

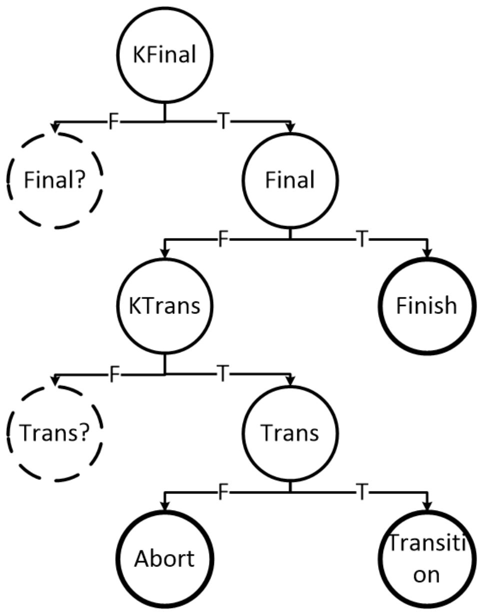

Figure 6.2.: Decision tree for one step transition of a high-level program. Bold leaves depict states where the agent possesses enough knowledge to decide program progression safely. Dashed states are those where no safe program progression can be done.

- $\mathcal{D}^* \not\models KFinal(\delta, s)$
- $\mathcal{D}^* \not\models KTrans(\delta, s)$

We will discuss in Chapter 7 how one can create a program to gather knowledge about the environment.
With the above usage of the transition semantic the robot can decide which action to take next. As it is sometimes advantageous to plan ahead one needs to extend this transition semantics. This can be achieved in IndiGolog with the help of the search operator $\Sigma$. To use the search operator within our transition semantic we first define how the robot can search for a plan. This is done by using the following predicate *DoSearch* which defines if a program can terminate in the current situation and how the final situation looks like.

**Definition 18.**

$$DoSearch(\delta, s, s') = \exists \delta' TransB^*(\delta, s, \delta', s') \land KFinal(\delta', s') \land \overline{Final}(\delta', s')$$

*where $TransB^*$ is, the reflexive transitive closure of*

$$TransB : TransB^*(\delta, s, \delta', s') = \forall T[\ldots \supset T(\delta, s, \delta', s')]$$

*where ... stands for the conjunction of the universal closure of*

$$T(\delta, s, \delta, s)$$
$$KTrans(\delta, s) \land \overline{Trans}(\delta, s, \delta', s') \land T(\delta', s', \delta'', s'') \supset T(\delta, s, \delta'', s'')$$

With the help of the definition when a program leads to a legal final termination we can finally define the transition of the search operator as follows.

- $$\begin{aligned}\overline{Trans}(\Sigma(\delta),s,\delta',s') = \exists \delta'',s''.\delta' = \Sigma(\delta'') \wedge KTrans(\delta,s) \wedge \\ \overline{Trans}(\delta,s,\delta'',s') \wedge DoSearch(\delta'',s',s'')\end{aligned}$$

The operator searches for the next transition which allows the robot to finally terminate in an accepting situation.
Beside the actual search the robot needs to decide if a search can be safely performed at all. Thus, we need to specify the predicate *KTrans* for the search operator. We define the predicate as follows.

$$\begin{aligned}KTrans(\Sigma(\delta),s) = & KTrans(\delta,s) \bigwedge \\ & [(\exists \delta'',s''.\overline{Trans}(\delta,s,\delta'',s') \wedge DoSearch(\delta'',s',s'')) \bigvee \\ & (\forall \delta'',s''.\neg\overline{Trans}(\delta,s,\delta'',s') \vee \overline{Trans}(\delta,s,\delta'',s') \wedge \neg DoSearch(\delta'',s',s''))]\end{aligned}$$

Through the definition of the predicate, the robot can decide if it can take a transition with the search operator if either the robot finds a valid plan or if it can determine that there exists no plan. The robot can determine that there exists no plan if every transition which is possible leads to a dead end. Thus, the program does not terminate with success instead it leads to a program which needs to be aborted.
The search operator is applied as follows. First, it is checked if it the search can terminate already. This check is performed by first checking if termination can be decided. The robot can decide on the termination of the search operator if the program which is used within the search can be decided to be terminated.

$$KFinal(\Sigma(\delta),s) \equiv KFinal(\delta,s)$$

Additionally, the robot can terminate with the search operator if the program which is used within the search is ready for termination.

$$\overline{Final}(\Sigma(\delta),s) \equiv \overline{Final}(\delta,s)$$

## 6.4. Formal Relation of Proposed and Original IndiGolog execution

With the above definitions, we have defined how a robot can decide the next action to perform and how to plan ahead. This transition allows the robot to safely operate in an environment without getting stuck in situation where the robot is lacking knowledge. We can state this more formally as we can relate the original IndiGolog semantic with the proposed semantic in this Chapter. The predicates $\overline{Trans}$ and $\overline{Final}$ are the original predicates of IndiGolog where the queries of the formulas are replaced to queries to the belief. This replacement ensures that as long as no fault occurs, the two predicates correspond to the original transition semantic. However, as we first need to decide if we can take a decision we need to ensure that the robot will never be imaging a situation where knowledge is lacking, but a transition is possible.
The first part of the transition semantic is the predicate to decide the transition. We can state the following corollary which shows that if a transition exists the robot has enough knowledge about possible transitions.

**Corollary 4.** $\forall \delta,s,\delta',s' : \overline{Trans}(\delta,s,\delta',s') \rightarrow KTrans(\delta,s)$

*Proof (Sketch).* Proof by cases over the statements of the program. Most of the cases are the defined in the same way for $\overline{Trans}$ and $KTrans$ thus the corolla holds trivially for those cases. In those cases, which are defined differently $\overline{Trans}$ is a part of a disjunction with an additional formula in the definition of $KTrans$. Thus, the corolla also holds. □

Like the above corollary, we can state the following corollary which specifies that if the robot can terminate the robot has enough knowledge to decide this termination.

**Corollary 5.** $\forall \delta, s : \overline{Final}(\delta, s) \rightarrow KFinal(\delta, s)$

*Proof (Sketch).* Proof by cases over the statements of the program. Most of the cases are the defined in the same way for $\overline{Final}$ and $KFinal$ thus the corolla holds trivially for those cases. In those cases, which are defined differently $\overline{Final}$ is a part of a disjunction with an additional formula in the definition of $KFinal$. Thus, the corolla also holds. □

With the help of Corollary 4 and Corollary 5 we can state that the robot will perform a transition or termination if it would do in case of the original IndiGolog transition semantic. Thus, with the help of the new transition semantic the robot will only perform additional transition but not less transitions. Additionally, the robot will only perform a transition if is safe to do so. We can state this property more formally with the following tow lemmas.

**Lemma 12.** *If no fault has been observed the robot will decide that a transition is possible for a program* $\delta$ *and* $s$ $KTrans(\delta, s) \wedge \exists \delta', s' \overline{Trans}(\delta, s, \delta', s')$ *if and only if* $\exists \delta', s' Trans(\delta, s, \delta', s')$

*Proof (Sketch).* This is direct result of Corollary 4 and Lemma 11 that the robot draws the same conclusion from the belief as the original situation calculus if no fault occurred. □

**Lemma 13.** *If no fault has been observed the robot will decide that it can terminate for a program* $\delta$ *and* $s$ $KFinal(\delta, s) \wedge \overline{Final}(\delta, s)$ *if and only if* $Final(\delta, s)$

*Proof (Sketch).* This is direct result of Corollary 5 and Lemma 11 that the robot draws the same conclusion from the belief as the original situation calculus if no fault occurred. □

To point out the difference between proposed transition semantic and an unsafe transition semantic as it was used in the original history based diagnosis and a transition semantic which does not distinguish the cases where it lacks knowledge, let's consider a simple example. The robot needs to bring an object $O$ from room $R_1$ to room $R_2$, paint the object and afterward bring the object to room $R_3$. To achieve the task, the robot performs the following sequence of actions $[goto(R_1), pick(O), goto(R_2), put(O), senseObjectAt(O), pick(O), goto(L_3), put(O)]$. Now let's further consider the sensing action $senseObjectAt(O)$ determines that the object is not in room $R_2$. Thus, the history of performed actions is now in contradiction. One can consider three simple explanations. First, the robot failed to pick up the object, the robot failed to put down the object, or the robot performed a wrong sensing.
If the robot would now, consider the first case it would conclude that the object is in room $R_1$ and drive back to fetch the object. Afterwards the robot would drive to room $R_2$ and deliver the object. At the end of the sequence the robot will have deliver the object but will have spent time to repair a failing action which may have not been failed at all. In contrast if the robot considers the third possibility the robot assumes that the object is correctly delivered and would progress as nothing has happened. The result would be that the robot may fail to achieve its task as the object was finally not delivered.
If the robot would only use the transition semantic consisting of $\overline{Trans}$ and $\overline{Final}$ the robot would neither belief that the object is in $R_1$ nor in $R_2$ as there are two different possibilities. As such the

robot, cannot decide if it can proceed. To be more precis the robot cannot decide which transition it can take. Thus, the robot is lacking knowledge.
In contrast if the robot performs a step to gather knowledge about the world the robot the robot would first gather knowledge. After gathering the knowledge about the environment, the robot can decide if the object is in room $R_1$ or in room $R_2$. Thus, the robot can decide which transition to take next.
Thus, with the help of the proposed transition semantic the robot would neither take unnecessary actions, perform dangerous action nor will it terminate if is lacking knowledge. Instead the robot will work towards its goal in a safe and efficient manner.

## 6.5. Conclusion and Future Work

A robot needs not only be capable to reason if a fault has occurred or what the fault was but it also need to incorporate this information in its reasoning to decide which action should be performed.
To do so we have defined how the robot can create a belief with the information it has gathered if a fault has happened. The belief of the robot is formed by those formulas which hold in every alternative which is considered the world may look like. These alternatives are created through the most likely explanations of the fault. Thus, the robot draw only conclusion which are safe as but still likely.
To use the belief in the reasoning system of the robot the time to answer a query needs to be short. Thus, we have proposed a method to allow the robot to progress its belief with the execution of action. Progression allows the robot to only use one knowledge base to answer a query. This knowledge base is updated every time an action occurred. As the robot needs only to use one knowledge base to answer the queries the robot can draw a conclusion in a fast manner.
After defining how the belief can be queried in an efficient manner we have shown how the belief can be used to decide which action to take next. The method outline in this chapter uses the IndiGolog transitions semantic as a base. The transition semantic was extended in such a way that only a transition is taken if the robot belief the transition is safe. Additionally, we extended the transition semantic in such a way that the robot can decide when it is lacking knowledge about the environment. In such a case the robot can gather knowledge about the world to allow further execution.
The new transition semantic is restricted to non-concurrent parts only. Thus, it is left for future work to extend the transition semantic to also incorporate the concurrency parts of IndiGolog. Additionally, it is left for future work to derive which properties need to hold for the active knowledge gathering step to be of use.

Chapter 7

# Active gathering of Knowledge

In the previous chapter, we have presented how a robot can decide which action to take next even in the presence of a faulty action execution. The method uses the belief the robot has about the world to take this decision. As the belief is formed by those formulas which hold in all alternative worlds which are very likely the robot is very cautious. This cautious behavior can cause a situation where the robot cannot decide as it is not certain about a fact. Instead, the fact is true in some alternatives but is false in others.
To address the problem of lacking knowledge the robot needs to gather knowledge actively. This gathering of knowledge is done by choosing sensing action which should be performed. With the help of sensing actions, the robot can rule out those alternatives which do not fit with the sensing result derived. Thus, the robot reduces the possible alternatives and thus gain knowledge about the real world. If the robot has ruled out enough of the alternatives the robot can decide the fact which is needed to take a decision which transition to perform.
As not every sensing action can reduce the uncertainty about the world one needs to choose a sensing action cleverly. The choice of the sensing action needs to consider that the sensing can be safely executed. Furthermore, the sensing action should maximize the information gathered about the environment. Additionally, the information gain is not allowed to be depending on the expected sensing outcome as one cannot assume a certain outcome. Instead one needs to choose a sensing action which gains knowledge about the environment regardless of the sensing result which will be reported.
In this chapter, we will present such an approach. The sensing action will be chosen in such a manner that these actions are safe to execute and maximize the information gained. Additionally, we will show under which condition this method will result in a situation where the robot is certain how the world looks like. Thus, allowing the robot to take a clear decision which action to execute next. The content presented in this chapter is based on the work presented in [91].
The remaining of the chapter is structured as follows. In the next section, we formally define the problem at hand to find the optimal sensing action. We call this issue following the idea originating from the diagnosis community the active diagnosis problem for robots with belief. After this definition, we will show how to select the optimal sensing action. This selection will consider the two things simultaneously. The first thing considered is which action can be performed. The second point considered is which knowledge is gained regardless of the sensing outcome. As this definition of the optimal sensing action is purely theoretical, we will describe an algorithm to derive this optimal sensing action in Section 7.3. This first algorithm relies on a finite number of objects and thus cannot be used in every setting. To circumvent this problem, we propose an alternative algorithm in Section

7.4. After discussing how the optimal sensing action can be found we discuss under which condition the active diagnosis lead to a single remaining alternative world. Thus, maximizing the knowledge of the robot. Finally, we conclude the chapter and point out some future work.

## 7.1. Definition of the Active Diagnosis Problem for Robots with Belief

Before we can define what, the optimal sensing action would be to gather knowledge we first need to state the problem formally which needs to be addressed by this chapter. To keep the representation simple, we will only define the problem for the classical history based diagnosis. Please, note that one can define the problem in an analog fashion for the consistency history based diagnosis.
As the sensing action, should be selected according to the belief we need to recap two definitions. The first definition defines the minimal diagnosis which are used to create the belief. The minimal diagnosis is defined as follow.

$$minDiag(s) \doteq \{s' | Diag(s', s, v) \land \nexists s'', v''.Diag(s'', s, v'') \land v'' < v\} \tag{7.1}$$

Please note that the minimal diagnosis is the set of those alternative situations which have the smallest cost for these changes. Furthermore, note that every situation in the minimal diagnosis set is consistent with the current observations. Thus, the set represents those alternatives of the current world which are consistent and very likely. We use this definition in order state which formulas the robot beliefs to hold.

$$Belief(\phi, s) \doteq \forall s' \in minDiag(s).\phi(s') \tag{7.2}$$

To define which sensing action to take to increase the knowledge, about the world the robot has, we first will state which impact a sensing action has on the set of minimal diagnosis. As such a change, can only occur if a faulty version of an action is considered to be less likely as the real action outcome we state the following assumption which is used in the remaining of the chapter.

**Assumption 6.** *The cost for a faulty action is greater than 0.*

$$\forall \alpha, \alpha', s.val(Varia(\alpha', \alpha, s)) > 0 \tag{7.3}$$

*Additionally, the cost for every insertion is great then 0.*

$$\forall \alpha', s.val(Inser(\alpha', s)) > 0 \tag{7.4}$$

The set of minimal diagnosis is reduced if a sensing action does not contradict at least one situation in the set. This is formally stated in the following lemma.

**Lemma 14.** *If a sensing action $\alpha$ is performed in situation s such that the sensing result is not in contradiction with at least one situation in the set $minDiag(s)$ or more formally $\exists s' \in minDiag(s).\mathcal{D}^* \models Cons(do(\alpha, s'))$ then the following holds. $minDiag(do(\alpha, s)) \subseteq \{do(\alpha, s') | s' \in minDiag(s)\}$.*

*Proof (Sketch).* This is a direct result of the definition of $minDiag(s)$ to be the set of those situations which are consistent and Assumption 6. □

The lemma states that the set of minimal diagnosis does not increase if the result of the sensing action is not in contradiction with at least one alternative world. Thus, the lemma states that a sensing action

will not increase the number of alternatives we need to consider if at least one alternative is not in contradiction with the sensing outcome.
Through this lemma, we can conclude that there is a set of sensing action which can be performed which do not increase the set of alternatives. Furthermore, we can strengthen the lemma into the following theorem.

**Theorem 5.** *If a sensing action $\alpha$ is performed in situation $s$ such that the sensing result is not in contradiction with at least one situation in the set $minDiag(s)$ but in contradiction with at least one situation in the set $minDiag(s)$ or more formally $\exists s' \in minDiag(s).\mathcal{D}^* \models Cons(do(\alpha,s')) \wedge \exists s'' \in minDiag(s).\mathcal{D}^* \not\models Cons(do(\alpha,s'))$ then the following holds. $minDiag(do(\alpha,s)) \subset \{do(\alpha,s')|s' \in minDiag(s)\}$.*

*Proof (Sketch).* This is a direct result of the definition of $minDiag(s)$ to be the set of those situations which are consistent and Assumption 6. □

The theorem states that the possible alternatives are reduced if we use a sensing action which is consistent with at least one alternative but renders at least one alternative inconsistent. Thus, the problem we need to address is to find such a sensing action to increase the knowledge about the environment.
But one can raise the question if the reduction of alternatives increases the knowledge of the robot. Basically, each reduced alternative reduces the uncertainty about how the world looks like. But how does the performed action impact the belief of the robot. We can link Lemma 14 directly to the belief by using the definition of the belief. If the agent does not belief the negation of a formula, then there exists a minimal diagnosis which is consistent with this formula. We can state this property for a sensing action more formally in the following two corollaries.

**Corollary 6.** *If the robot performs an action $\alpha$ in situation $s$ with sensing outcome $\top$ and the robot does not belief $\neg SF(\alpha,s)$ or more formally $\neg Belief(\neg SF(\alpha,s),s)$ then there exists a situation $s'$ in $minDiag(s)$ which is consistent with the sensing action or more formally $\exists s' \in minDiag(s).\mathcal{D}^* \models Cons(do(\alpha,s'))$.*

*Proof (Sketch).* This is a direct result of the definition of the belief. □

**Corollary 7.** *If the robot performs an action $\alpha$ in situation $s$ with sensing outcome $\bot$ and the robot does not belief $SF(\alpha,s)$ or more formally $\neg Belief(SF(\alpha,s),s)$ then there exists a situation $s'$ in $minDiag(s)$ which is consistent with the sensing action or more formally $\exists s' \in minDiag(s).\mathcal{D}^* \models Cons(do(\alpha,s'))$.*

*Proof (Sketch).* This is a direct result of the definition of the belief. □

By combining the above corollaries with Lemma 14 we can state the following lemma defining that through a non-contradicting sensing the robot will not lose knowledge about the world.

**Lemma 15.** *If a robot performs an action $\alpha$ in situation $s$ and the sensing outcome is $\top$ and it holds that $\neg Belief(\neg SF(\alpha,s),s)$ or if the sensing outcome is $\bot$ and it holds that $\neg Belief(SF(\alpha,s),s)$ then the following holds. $\forall\phi.Belief(\phi,s) \rightarrow Belief(\phi,do(\alpha,s))$*

*Proof (Sketch).* This is a direct result of Lemma 14, Corollary 6 and Corollary 7. □

The lemma states that the robot belief all formulas it has believed before executing a sensing action which does not contradict the belief. Thus, the robot will not forget knowledge about the world if the sensing action is not contradicting to the current belief of the robot.
We can further extend this lemma through the following theorem. The theorem states if the robot neither belief the sensing nor its negation the robot will gain additional knowledge about the environment.

**Theorem 6.** *If a robot performs an action* $\alpha$ *in situation* $s$ *and it holds that* $\neg Belief(\neg SF(\alpha,s),s)$ *and it holds that* $\neg Belief(SF(\alpha,s),s)$ *then the following holds.* $\forall\phi.Belief(\phi,s) \rightarrow Belief(\phi,do(\alpha,s))$ *and* $\exists\phi.\neg Belief(\phi,s) \wedge Belief(\phi,do(\alpha,s))$

*Proof (Sketch).* This is a direct result of the Lemma 15 and the fact that the robot belief after executing the sensing action at least the formula imposed by the sensing action which was not believed before. □

The theorem states that if the robot does neither belief the sensing action outcome to be true nor to be false the robot gains knowledge about the environment if it executes this sensing action. Thus, if we choose a sensing action according to Theorem 5 the condition of the Theorem 6 are also met and the robot will gain knowledge through this action.
As the robot, should perform actions only if they are safe we need to restrict the set of sensing actions the robot can choose from to the set of sensing actions which are safe to perform this is defined as follows.

**Definition 19.** *The set of safe sensing actions* $A_{SS}(s)$ *is the set of grounded sensing actions which have a precondition which is believed to hold:* $A_{SS}(s) = \{\alpha | \alpha \in \mathcal{A}_{sensing} \wedge Belief(\Pi_\alpha(s),s)\}$

Using the definition of a safe sensing action we can now define the knowledge gathering step as active diagnosis for belief management as follows.

**Definition 20.** *The active diagnosis for belief management (ADBM) is defined as: Given a minimal diagnosis of a situation s* $minDiag(s)$ *find a safe sensing action* $\alpha \in A_{SS}(s)$ *such that the set* $minDiag(s)$ *is reduced by at least one element regardless of the actual sensing outcome.*

## 7.2. Selecting the Optimal Sensing Action

To find a solution to the ADBM problem we need to fine a sensing action which reduce the minimal diagnosis set regardless of the sensing outcome. To find such a sensing action we first define two sets which capture those situations which become inconsistent if the result of the sensing action is false respectively true.

**Definition 21.** *The set* $\Omega_\alpha(s) \subseteq minDiag(s)$ *contains all possible situation which would be removed from the minimal diagnosis set if false would be measured due to their inconsistency:* $\Omega_\alpha(s) = \{\omega \in minDiag(s) | \mathcal{D}^* \not\models Cons(do(\alpha,\omega))\}$.
*The set* $\Lambda_\alpha(s) \subseteq minDiag(s)$ *contains all possible situation which would be removed from the minimal diagnosis set if true would be measured due to their inconsistency:* $\Lambda_\alpha(s) = \{\lambda \in minDiag(s) | \mathcal{D}^* \not\models Cons(do(\alpha,\lambda))\}$.

With the two sets $\Omega_\alpha$ and $\Lambda_\alpha$ we can choose a sensing action such that these sets are not empty. To be more precise we search for a sensing action $\alpha$ such that these two sets are maximized in such a

way that half of the situations are in $\Omega_\alpha$ and the other half is in $\Lambda_\alpha$. As one may want to consider not every situation to be equally likely and as a sensing action may be executed faulty we need a proper criterion to find the best sensing action.
To define such a criterion, we follow the idea of [26] and [63]. The idea is to use the mutual information as a criterion to choose the next best sensing action. The mutual information chooses the sensing action which is very likely to discriminate between different alternatives by considering the likely hood of a wrong measurement. Thus, it fits our need for a criterion to select the sensing action.
We define the mutual information which we use as a criterion which should be maximized as follows.

$$I(\alpha; minDiag(s)) = H(\alpha) - H(\alpha | minDiag(s)) \tag{7.5}$$

The first part of the equation is the entropy of the action. Thus, it measures how well the sensing action divides the set of situations into $\Omega_\alpha$ and $\Lambda_\alpha$. This measurement is done according to the probability mass. Thus, if not all situations are considered to be equally likely the set may differ in size to balance their probability. The second part of the equation defines the impact of the noise the sensing action has. Thus, the noisier the sensing action, the lower the chance to be selected. Furthermore, this part of the equation is used to prefer those sensing actions which have a tiny set of situations which are neither in $\Omega_\alpha$ nor $\Lambda_\alpha$.
To define the necessary probabilities, we assume for each situation in $minDiag(s)$ a probability $p_s$. With the definition of this probability we can define the overall probability of the set $\Omega_\alpha$ and $\Lambda_\alpha$ which is used for the entropy calculation. The probability of the set $\Omega_\alpha$ is defined through Equation 7.6 and the probability of the set $\Lambda_\alpha$ is defined through Equation 7.7.

$$p_{\Omega_\alpha} = \sum_{\omega \in \Omega_\alpha} p_\omega \tag{7.6}$$

$$p_{\Lambda_\alpha} = \sum_{\lambda \in \Lambda_\alpha} p_\lambda \tag{7.7}$$

Both equations accumulate the probabilities of their associated situations. Using the probabilities of the two sets we can define how the impact of a sensing action is through its entropy. The entropy $H(\alpha)$ is defined by the following equation.

$$H(\alpha) = -\left[p_{\Omega_\alpha} * ld(p_{\Omega_\alpha}) + p_{\Lambda_\alpha} * ld(p_{\Lambda_\alpha})\right] \tag{7.8}$$

As the entropy is used to check that the chosen sensing action balance the impact it has on the two sets $\Omega_\alpha$ and $\Lambda_\alpha$ we can state the following two simple corollaries which prove this balance at least for extreme cases.

**Corollary 8.** *If a sensing action $\alpha$ causes that all situations are either in the set $\Omega_\alpha$ or $\Lambda_\alpha$ the entropy defined through Equation 7.8 is equal to 0.*

*Proof (Sketch).* If the sensing has all situations in $\Omega_\alpha$ or $\Lambda_\alpha$ the probability of $p_{\Omega_\alpha}$ is 1 and the probability of is $p_{\Lambda_\alpha}$ 0 or vice versa. Thus Equation 7.8 is 0. $\square$

**Corollary 9.** *If all situations are equally likely $\exists c. \forall s' \in minDiag(s). p_{s'} = c$ then the entropy has its maximum if $(|\Omega_\alpha| = |\Lambda_\alpha|) \wedge (\Omega_\alpha \cup \Lambda_\alpha = minDiag(s))$*

*Proof (Sketch).* The maximum of the entropy is reached if $p_{\Omega_\alpha} = \frac{1}{2}$ and $p_{\Lambda_\alpha} = \frac{1}{2}$. If equal likely situations are assumed this is the case if $(|\Omega_\alpha| = |\Lambda_\alpha|) \wedge (\Omega_\alpha \cup \Lambda_\alpha = minDiag(s))$, as both sets have half of the probability mass. $\square$

The first corollary states that if the impact of the sensing action effects only one sensing action then the entropy is low. Thus, such a sensing action would not very likely be chosen. The second corollary states that if the probabilities of all situations are equal the maximum of the entropy will be reached if the two sets $\Omega_\alpha$ and $\Lambda_\alpha$ have the same size. Thus, the entropy meets our expectations in at least the extreme cases.
To specify the impact of the sensor accuracy we assume that for every sensing action $\alpha$ there is a probability assigned to it $p_\alpha$ which specifies if the action yield the true result. With the help of this probability we can define the probability of the sensing action to yield $\top$ as sensing result in the current situation. This probability is defined through Equation 7.9. Furthermore, we can define the probability to yield $\perp$ as sensing result with Equation 7.10. Please note that we assume that the chance to yield either $\top$ or $\perp$ in an alternative world which does not specify the sensing result is equally high.

$$p(\alpha,\top|s) = \left\{ \begin{array}{cl} \frac{1}{2}, & s \notin \Omega_\alpha \cup \Lambda_\alpha \\ p_\alpha, & s \in \Omega_\alpha \\ 1-p_\alpha, & s \in \Lambda_\alpha \end{array} \right\} \tag{7.9}$$

$$p(\alpha,\perp|s) = \left\{ \begin{array}{cl} \frac{1}{2}, & s \notin \Omega_\alpha \cup \Lambda_\alpha \\ 1-p_\alpha, & s \in \Omega_\alpha \\ p_\alpha, & s \in \Lambda_\alpha \end{array} \right\} \tag{7.10}$$

With the above probabilities, we can define the conditional entropy of the sensing action for a given situation. This conditional entropy specifies how likely it is that the sensing action will yield a false result. The conditional entropy for a given situation is defined as follow.

$$H(\alpha|s) = -\left[p(\alpha,\top|s) * ld(p(\alpha,\top|s)) + p(\alpha,\perp|s) * ld(p(\alpha,\perp|s))\right] \tag{7.11}$$

Finally, we can use Equation 7.11 to calculate the conditional entropy of the sensing action $\alpha$ with all situations specified in *minDiag*$(s)$ as follows.

$$H(\alpha|minDiag(s)) = \sum_{s' \in minDiag(s)} (p_s * H(\alpha|s)) \tag{7.12}$$

Using the result of Equation 7.8 and Equation 7.12 we can calculate the mutual information $I(\alpha; minDiag(s))$. Please note that the calculation of the mutual information depends on the sets $\Omega_\alpha$ and $\Lambda_\alpha$ if the probability of the sensing action failure is fixed for each sensing action regardless of its arguments. We will show later how this property can be exploited.

## 7.3. Using Grounded Sensing Actions to Find the Optimal Action

With the above criteria, we have defined which sensing action should be performed. But it is still missing how one can find this sensing action. If we can consider only a small set of sensing actions which are possible we can simply use Algorithm 6.
The algorithm first computes all possible sensing actions. Afterward, it iterates through the sensing action and calculates the mutual information for each of the sensing actions. The sensing action which yields the highest value of the mutual information is returned. Thus, this algorithm is a brute force method to find the sensing action which maximizes the mutual information. This brute force method also yields to the result that the algorithm is sound and complete as stated in the following theorem.

**Theorem 7.** *Algorithm computeNextSensingAction is sound and complete.*

**Algorithm 6:** *computeNextSensingAction*

```
input  : $minDiag(s)$ ... a set of minimal diagnosis for the situation $s$
output: a grounded sensing action to perform next or NULL if no grounded sensing action exists
begin
    if $A_{SS} = \emptyset$ then
        return NULL
    end
    $v_{max} = 0$;
    $a_{max} = NULL$;
    for $\alpha \in A_{SS}$ do
        $\langle \Omega_\alpha, \Lambda_\alpha \rangle = calculateSplitting(\alpha, minDiag(s))$ ;
        $v = I(\alpha; minDiag(s))$ ;
        if $v > v_{max}$ then
            $v_{max} = v$ ;
            $a_{max} = \alpha$
        end
    end
    return $a_{max}$
end
```

*Proof.* Due to the calculation of the set $A_{SS}$ we know that 1) each element in the set $A_{SS}$ is a safe sensing action, and 2) all safe sensing actions are in the set $A_{SS}$. Because we only return a sensing action which is part of set $A_{SS}$ and has the highest mutual information we can state that the algorithm is sound and complete. □

The run time of the algorithm depends on the number of sensing actions and the number of possible alternatives to consider. We can state this more formally as $O(|A_{SS}| \times |minDiag(s)|$. Thus, the algorithm is efficient if the number of sensing actions is tiny.

## 7.4. Using the Alternative Worlds to Find the Optimal Action

Unfortunately, the Algorithm 6 cannot be used if there is an infinite number of possible sensing actions. If the number of sensing action is finite but huge, a brute force approach may not yield a result either. Thus Algorithm 6 cannot be applied in all situations.
Instead of using the set of all possible situation to calculate the best sensing we can exploit the fact that the mutual information depends on the two sets $\Omega_\alpha$ and $\Lambda_\alpha$ if we fix the probability that a sensing action fail regardless of its arguments. Thus, instead of calculating the mutual information for each grounded action term we try to find a grounded action such that a certain splitting is caused. Algorithm 7 performs such a calculation.
The algorithm first calculates all possible splitting together with all possible sensing action with open arguments. The possible splitting's, the action with open arguments and the value of the resulting mutual information is gathered in the set **T**. In this set the sensing actions with open arguments $\overline{\mathcal{A}_{sensing}}$ are used. Additionally, we exploit the fact that we can calculate the mutual information if we have specified the sets $\Omega_\alpha$ and $\Lambda_\alpha$. After creating this set, the algorithm creates a queue **Q** which is created from the set **T** where the entries are sorted according to the mutual information. Thus, the

**Algorithm 7:** *computeNextSensingAction2*

```
input : $minDiag(s)$ ... the minimal diagnosis set of a given situation
output: return a grounded sensing action to perform next or $NULL$ if no grounded sensing action
        exists
1  begin
2      $\mathbf{T} = \{\langle\Omega,\Lambda,\alpha,v\rangle | (\Omega \subseteq minDiag(s)) \wedge (|\Omega| > 0) \wedge (\Lambda \subseteq minDiag(s)) \wedge (|\Lambda| > 0) \wedge (\Omega \cap \Lambda = \{\}) \wedge (\alpha \in \overline{\mathcal{A}_{sensing}}) \wedge (v = I(\alpha; minDiag(s)))\}$;
3      $\mathbf{Q} = Insert(\mathbf{T})$;
4      while $\mathbf{Q} \neq \emptyset$ do
5          $\langle\Omega,\Lambda,\alpha,v\rangle = pop(\mathbf{Q})$ ;
6          $\alpha' = findGrounding(\langle\Omega,\Lambda,\alpha\rangle)$ ;
7          if $\alpha' \neq NULL$ then
8              return $\alpha'$
9          end
10     end
11     return $NULL$
12 end
```

high the value of the mutual information the close the entry is to the begin of the queue. After creating the queue, the algorithm iterates through the queue and searches for a grounding for the action $\alpha$ with the function *findGrounding*.
To allow the algorithm to find a proper sensing action the function *findGrounding* needs to fulfill certain properties. These properties ensure that the found grounded version of the sensing action actual yield the desired splitting and can be executed. The properties which need to be fulfilled by the function *findGrounding* are the following.

1. $Belief(\Pi_\alpha(s), s)$. Use only grounded sensing actions which are safe.
2. $\forall\omega \in \Omega_\alpha : \mathcal{D}^* \cup AO(\alpha,\omega) \not\models Cons(do(\alpha,\omega))$. The grounding results in the specified splitting for the set $\Omega_\alpha$.
3. $\forall\lambda \in \Lambda_\alpha : \mathcal{D}^* \cup \neg AO(\alpha,\lambda) \not\models Cons(do(\alpha,\lambda))$. The grounding results in the specified splitting for the set $\Lambda_\alpha$.

To increase the efficiency of the function *findGrounding* one can collapse the last two condition into one call to a theorem prover. Thus, one needs only to call first a theorem prover to get a grounding which can be executed and afterwards check if this grounding causes the desired splitting. The two last conditions can be collapsed into the following condition.

$$\mathcal{D}^* \bigcup_{\omega\in\Omega_\alpha} AO(\alpha,\omega) \bigcup_{\lambda\in\Lambda_\alpha} \neg AO(\alpha,\lambda) \not\models \bigvee_{\omega\in\Omega_\alpha} Cons(do(\alpha,\omega)) \bigvee_{\lambda\in\Lambda_\alpha} Cons(do(\alpha,\lambda)) \qquad (7.13)$$

Assuming a proper implementation of *findGrounding* we can prove that the algorithm is sound and complete with the following theorem.

**Theorem 8.** *The algorithm computeNextSensingAction2 is sound and complete if the search step is sound and complete.*

*Proof.* The algorithm will only return a grounded sensing action if the search step realized through *findGrounding* return a grounded sensing action. Thus, if the search step is sound the algorithm is sound. As the algorithm will iterate over every possible combination of splitting in the pool and all possible sensing actions the algorithm is complete if the search step realized through *findGrounding* is complete. □

Thus, the algorithm allows us to calculate a sensing action which maximizes the mutual information even if there is a big set of possible sensing actions. To ensure the algorithm to be fast one needs a proper implementation of the function *findGrounding*. The implementation needs to derive a grounding for a sensing action fast or can decide that there is no such grounding. Please note that through the sorting of the mutual information it may often be the case that only a few calls to the function *findGrounding* are necessary to find a sensing action. As a final remark, we want to state that in the worst case the algorithm performs even worse that Algorithm 6 as one needs first to calculate all splitting's which is exponential in the number of situation in $minDiag(s)$.

## 7.5. Conditions for Active Diagnosis to Yield One Remaining Alternative

In this chapter, we have presented two different algorithms to solve the ADBM problem in an online fashion. As we address the problem online instead of creating a conditional plan, the question arises if this online execution always results in a single remaining situation. The quick answer is no as there is also not always a conditional plan to yield only one remaining situation. Thus, we present the conditions which need to hold such that the question can be answered with yes.
We focus our attention to those situations which have a minimal diagnosis set which can be reduced to one situation if a conditional plan consisting of sensing actions is used. We call this property complete active diagnosable. This property is formally defined as follows.

**Definition 22.** *We define a minimal diagnosis set $minDiag(s)$ as complete active diagnosable ($CAD(minDiag(s))$) recursively:*

- $CAD(minDiag(s))$ *if* $|minDiag(s)| = 1$
- $\exists \alpha : [(|\Omega_\alpha| > 0) \vee (|\Lambda_\alpha| > 0)] \bigwedge CAD(minDiag(s) \setminus \Omega_\alpha) \bigwedge CAD(minDiag(s) \setminus \Lambda_\alpha)$

The definition basically states that if there is a sensing action which yield in one case of the sensing outcome at least a smaller set of remaining diagnosis the complete set is complete active diagnosable if the subsets are. Through this definition, it is ensured that there exists at least on conditional plan which leads to one remaining situation in any case. But as we choose the sensing action online we cannot perform a backtracking which may be necessary to generate a plan. Instead we generate a conditional plan in a greedy fashion. In general, such a greedy approach may not yield a plan even if there is a plan. To discuss the difference which is caused due to a greedy generation we state a condition which needs to be fulfilled for a greedy generation to work. The condition is called strong complete active diagnosable. We define a set of minimal diagnosis for a situation as strong complete active diagnosable as follow.

**Definition 23.** *We define a minimal diagnosis set $minDiag(s)$ as strong complete active diagnosable ($SCAD(minDiag(s))$) recursively:*

- $SCAD(minDiag(s))$ *if* $|W(s)| = 1$

- $$[\exists\alpha : (|\Omega_\alpha| > 0) \wedge (|\Lambda_\alpha| > 0)] \bigwedge$$
  $$[\forall\alpha.((|\Omega_\alpha| > 0) \wedge (|\Lambda_\alpha| > 0)) \rightarrow SCAD(minDiag(s) \setminus \Omega_\alpha) \wedge SCAD(minDiag(s) \setminus \Lambda_\alpha)]$$

The definition of strong complete active diagnosis ensures that regardless of the choice of the sensing action one will reach a set with only one remaining situation. Thus, a greedy algorithm will yield a set containing only one situation if the set is a strong complete active diagnosis. However, as the definition of strong complete active diagnosis seems to be very restricted compared to the definition of complete active diagnosis, we will show how these two restrictions relate. This relation will allow us to show under which condition the online execution yield a single remaining situation.
To state this relation, we first need to specify how the set of allowed sensing actions change during the execution of a sensing action. We can derive the following lemma for this change[*].

**Lemma 16.** *Each set $minDiag'(s)$ of a minimal diagnosis which is the result of executing a sensing action $\alpha$ which is not in contradiction to the belief result in a super set of possible safe sensing actions $A_{SS}(minDiag'(s)) \supseteq A_{SS}(minDiag(s))$.*

*Proof.* This is a direct result of Theorem 6. □

The lemma states after executing a sensing action the set of possible sensing action can only increase. Thus, we can execute a sensing action without worrying that a plan can no longer be executed. We can use this result to state the following lemma.

**Lemma 17.** *If a set of minimal diagnosis $minDiag(s)$ fulfill the $CAD(minDiag(s))$ property also every $minDiag(s)'$ which is the result of executing action $\alpha$ which is not contradicting with the belief fulfill the property $CAD(minDiag(s)')$.*

*Proof.* As the pool fulfill the $CAD(minDiag(s))$ property we know there exists a conditional plan which reduce the pool to cardinality 1. Through Lemma 16 we know that after the execution of a sensing action which is not in contradiction to the belief at least the same sensing actions are safe to perform. Thus, we can use the same conditional plan which was retrieved for the set $minDiag(s)$ for the set $minDiag(s)'$. Thus, the set $minDiag(s)'$ also fulfills the property $CAD(minDiag(s)')$. □

The lemma states that the *CAD* property is preserved if we execute a sensing action which does not contradict the belief of the robot. We can use this lemma to state that there exists a minimal plan which can be used to yield only one situation remaining.

**Lemma 18.** *If a minimal set of diagnosis $minDiag(s)$ is $CAD(minDiag(s))$, then there exists a minimal plan (concerning the number of transitions) using only sensing action which eliminates in every case of sensing result at least one possible situation from the pool.*

*Proof.* We prove this lemma by contradiction. Assume there exists a minimal plan which uses a sensing action $\alpha_1$ which does not eliminate one situation in any case of the sensor result. Thus, there exists one sensor result which does not change the set. Thus, the set stays the same after the sensing action is performed and the sensor result was retrieved. As the resulting set needs to be $CAD(minDiag(s))$, there must exist a sensing action which changes the pool regardless of the sensing

[*]We use the expression of a sensing action $\alpha$ which is not in contradiction to the belief to specify a sensing action which yields a sensing result which is neither believed nor is the negation of the sensing result is believed

outcome. Thus, we can remove the sensing action $\alpha_1$ without effect on the minimal set of diagnosis and would have still a conditional plan. This result is in contradiction to the assumption that this plan is minimal. □

Finally, by combining the above lemmas we can state that the two restrictions are equivalent. This is formally stated in the following theorem.

**Theorem 9.** *A minimal set of diagnosis $minDiag(s)$ is complete active diagnosable $CAD(minDiag(s))$ iff the set is strong complete active diagnosable $SCAD(minDiag(s))$.*

*Proof.* The if direction holds since the condition of the $CAD(minDiag(s))$ is always true if the condition of the $SCAD(minDiag(s))$ is true.
To prove the only if direction we use Lemma 17 and Lemma 18. Through Lemma 18 we know that if the minimal diagnosis set is $CAD(minDiag(s))$ there exists a conditional plan which uses sensing action fulfilling the requirements in $SCAD(minDiag(s))$. Thus, we need to show if we choose a sensing action according the requirements for $SCAD(minDiag(s))$ we will preserve that the set is $CAD(minDiag(s))$. To show that $CAD(minDiag(s))$ is preserved we need to analyze what happens in one transition of the conditional plan. During one transition of the conditional plan at least one possible situation from $minDiag(s)$ is eliminated and the remaining possible situations build the set $minDiag(s)'$. Thus, the chosen sensing action does not contradict the belief of the robot. Due to Lemma 17 we know that the $CAD(minDiag(s))$ property is preserved and thus the Theorem holds. □

The theorem states that if a minimal diagnosis fulfills the $SCAD(minDiag(s))$ property it also fulfills the $CAD(minDiag(s))$ property. We can use this result to finally show that the online execution yields also only one remaining situation if a conditional plan would yield only one remaining situation.

**Theorem 10.** *Solving the ADBM online will result in a minimal diagnosis of cardinality* 1 *if the minimal diagnosis fulfills the $CAD(minDiag(s))$ property.*

*Proof.* The proof is simple as an online execution is a sub tree of a conditional plan which was produced greedy. A conditional plan which was produced greedy fulfills the $SCAD(minDiag(s))$ property. Due to Theorem 9 we also known that this property only holds if the $CAD(minDiag(s))$ property hold. Thus, this theorem holds. □

The theorem states that we can solve the ADBM problem online without sacrificing the chance that only one situation remains. Thus, one can always find the next best sensing action till either the robot has enough knowledge or only one situation is remaining. If only one situation is remaining the robot can only gain additional knowledge by performing sensing actions which regardless of the sensing outcome are not believed by the robot. However, no further guidance can be given. Thus, one may consider aborting the execution of the plan as the uncertainty of the current situation is not caused due to the different possible faults.

## 7.6. Conclusion and Future work

If a robot uses its belief to decide which action to perform next, it can face the problem that it lacks the knowledge to make a decision. Thus, the robot needs to perform an active knowledge gathering step. In this chapter, we have shown a method to perform such an active knowledge gathering step.
First, we have shown that the robot gains knowledge about the environment if it performs sensing action which has an outcome that is not predicted by the robot's belief. We demonstrated that this sensing action could further be used to find those sensing actions used for active knowledge gathering.

Using the mutual information of the sensing action and its impact on the robot's belief we showed how the robot could decide which sensing action to perform next. Additionally, we have shown two algorithms to find the sensing action which maximizes the mutual information. The algorithm differs on how they use the set of sensing actions which can be performed. The first algorithm uses a brute force search over all sensing action. Thus, the algorithm can only be applied if there is a small number of sensing actions. The second algorithm first estimates the value of the mutual information and uses this value to search for a sensing instance which produces this value. Thus, the algorithm can deal with a significant number of possible sensing actions but has a higher worst case run time than the first algorithm. Finally, we have shown on a theoretical basis that the online execution of choosing a sensing action the algorithm yields the same result as if one would create a conditional plan consisting of sensing actions only.
To not only just gather knowledge about the environment but instead to gather that knowledge which allows the robot to decide further transition is left for future work. Such a guided sensing could reduce the time the robot is gathering knowledge. Thus, the robot would faster react to a situation where it lacks knowledge. Such a method could consider the query which failed due to the lack of knowledge as a goal which needs to be reached through a conditional plan.

Chapter 8

# Creating Reactive Programs

In the previous chapters, we have discussed how the robot can use the information of the history based diagnosis to form a belief about the environment. This belief is used to allow the robot to take a decision which action should be performed next. Furthermore, we have discussed how the robot can react to situations where it is missing knowledge about the environment and thus needs to gather knowledge. All these methods assumed that the high-level program of the robot could react to abrupt changes of the underlying belief.

Unfortunately, most of the programs which are written are not designed to react to changes in the belief. Thus, they lack the possibility to specify how the robot should react in a certain situation to achieve the overall goal. Instead, high level programs are often simple imperative programs describing how to achieve a task with a set of actions which are executed flawlessly. Such programs only cover one possible trace how to reach from one situation to the goal situation. The creation of such programs as simple as one can often come up easily with a sequence of actions to achieve a certain task. Sometimes it also possible to generate these programs from descriptions found on the internet [137]. Due to their sequential nature, the robot cannot react to unforeseen changes. The reason for this restriction is simply caused by the fact that these programs specify how the robot can achieve a task if everything works as expected. During the creation of this program, one does not, or even cannot consider all cases an action might fail. Thus, the cases of a failing action are not considered in the program. In contrast to a simple sequence of action a specification of a reactive behavior allows the robot to reach the goal from a different situation which may result in a different sequence of action as it would be specified otherwise.

Besides the lack of proper high-level control programs which can react there is also a software development point of view which should be considered. It is often easy to describe which actions the robot should perform to achieve a task. Furthermore, one can test such a high-level program rigorously to ensure that all cases of valid initial scenarios are covered. This testing may be time intensive, but it is feasible if one needs only to consider actions which execute as expected. It is also possible to prove that the program is correct by using a method for formal verification [21]. However, again, this is only feasible if one considers non-faulty action outcomes only. This restriction allows to neglect that there may be different beliefs after an action has been executed, depending on the success of the execution. Thus, it would be preferable if one would only need to specify an imperative program, but the robot would still use this program in a reactive manner.

As one specifies only an imperative program but the robot should use this program in a reactive manner the robot needs first to alter the program such that the reactive behavior is possible. During this change,

it is important that the program not behave differently as if the imperative program would be executed if no fault occurred. Thus, the robot should create a reactive program out of the imperative program which behaves the same if no action fault occurred. In this chapter of the thesis, we will address this problem. We will use the program structure to infer a reactive behavior. The method presented in this chapter ensures that the reactive behavior behaves the same as the sequential program if no fault occurs.
To allow the robot to react to a fault the robot needs not only a reactive high-level program but it needs to detect the fault in the first place. As we have discussed in the previous chapters, the robot can only detect a fault if a sensing action is performed. Thus, the program needs to check the outcome of each action with the help of sensing actions. As this check depends on the available sensing actions on the robot a general specification may become complex. Additionally, it is a cumbersome task to add all the sensing action to the program. Furthermore, through the adding of sensing actions, the program gets more complex. Last but not least the addition of sensing actions does not change if an action is executed successfully. Thus, it would be preferable if one can only specify the program without supervision sensing actions, but the robot performs this sensing actions on its own to check if the execution is carried out as expected.
In this chapter, we will show an approach which allows the robot to generate the supervision sensing actions automatically. The method uses the definition of the actions and the sensing action to derive the super vision actions. Furthermore, the creation of this supervision sensing actions only relies on standard means of the situation calculus thus allow an easy integration.
The remaining of the chapter is structured as follows. In the next section, we discuss how to generate a reactive program from a sequential one. The proceeding section discusses how to produce supervision sensing actions. In Section 8.3 we discuss some related research which is unique to this chapter. Finally, we conclude the chapter and point out some future work.

## 8.1. Generation of Reactive Programs

To create a reactive program, from a given imperative program the robot needs first to ”understand“ the program, before creating a new reactive program. The first step is that the robot parses the program to retrieve the structure of the program. Furthermore, through the usage of the semantic of the program and its structure the robot can reason how the different parts of the program depend on each other. Afterward, the robot can use these dependencies to generate a reactive program.
In the remaining of the chapter, we assume the high-level program is given as an IndiGolog program. The result of parsing the program is an abstract syntax tree (AST [1]). The AST represents the program structure. Thus, it relates the different statements of the program to each other according to the program syntax. We define the AST formally as follows.

**Definition 24.** *An Abstract Syntax Tree (AST) consists of a set of nodes $\mathcal{N}$ and a set of edges $\mathcal{E} \subseteq \mathcal{N} \times \mathcal{N}$. The nodes in the AST represent program statements, and the edges in the AST represent a parent child relation. The AST has a unique root node N which has no parent but may have children. Furthermore, all nodes in $\mathcal{N}$ are connected through the edges in such a way that the form one tree. Additional it holds that each node n in $\mathcal{N}$ is a tuple consisting of the following elements:*

- *$T$. A type of the statement, e.g., While, If*
- *Con. A condition of the node.*
- *x. A variable of the node*

- $\alpha$. *An action of the node*

To show how an AST of a program looks like let's consider a simple example. Let's consider the following program: **if** $\phi$ **then** $\alpha_1$ **else** $\alpha_2$ ; $\alpha_3$ **endIf**. The program is a simple if statement which causes the execution depending on the formula $\phi$ to execute either action $\alpha_1$ or the action sequence $\alpha_2$ and $\alpha_3$. The resulting AST is depicted in Figure 8.1.

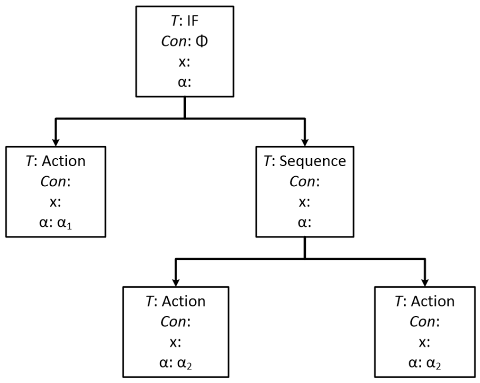

Figure 8.1.: The AST of the simple example program **if** $\phi$ **then** $\alpha_1$ **else** $\alpha_2$ ; $\alpha_3$ **endIf**.

To use the AST for further changes on the program we assume a function *child* : $\mathcal{N} \leftrightarrow \mathcal{N}^1$. The function returns all the children of a node in a list. Thus, allowing a simple iteration over all children of this node. Furthermore, we assume that the function preserves the order of the children. Thus, in the case of a sequence, the first child will always be the first element of the sequence.
The AST of the program only relates the different statements of the program with each other. However, as the robot needs to "understand" the program, we generate a new graph called the requisite graph (*RG*) using the information provided by the AST. The graph represents the conditions which need to hold to execute a certain action. Thus, one can use the graph to execute the complete program. One just needs to check the condition on the edges of the graph and afterward execute the action mentioned in the node of the graph. We define the requisite graph formally as follows.

**Definition 25.** *A requisite graph RG consists of a set of nodes* $\overline{\mathcal{N}}$ *and a set of edges* $\overline{\mathcal{E}}$. *Each node* $n \in \overline{\mathcal{N}}$ *can be of one of the following six types:*

1. *an action node which defines an action to be executed.*
2. *A nil node which defines an empty statement*
3. *A variable binding node which defines a variable binding for a specific scope*
4. *A variable releasing node which defines the end of a scope for a variable*
5. *A start node which defines the beginning of a program*
6. *An end node which defines the end of a program*

*Each edge* $e \in \overline{\mathcal{E}}$ *is defined as a tuple consisting of the following elements:*

- *A start node* $n_s \in \overline{\mathcal{N}}$

- *A goal node $n_g \in \overline{\mathcal{N}}$*
- *A condition C which needs to hold to use the edge during interpretation.*

As we use the *RG* to specify the execution of the program we will define the execution of a requisite graph formally with the help of two predicates. The first predicate is $transRG(\overline{\mathcal{N}}, \overline{\mathcal{E}}, n, \tau, s, b, n', s', b')$ which specifies given a requisite graph the current node $n$, $\tau$ specifies the type of the current node, the current situation $s$ and the current variable binding $b$ what the next node in the graph $n'$ will be and how the situation changes to $s'$ additionally the variable binding can be altered to a new binding $b'$. The second predicate $finalRG(\overline{\mathcal{N}}, \overline{\mathcal{E}}, n, \tau, s, b)$ specifies if the program can legally terminate with the given requisite graph in the current node $n$ with the type $\tau$, in the current situation $s$ with the current variable binding $b$.
To specify these two predicates, we assume that the variable binding consists of a set of tuples $\langle v,t \rangle$, such that $v$ is a variable and $t$ is a grounded atom which is a valid substitution for the variable $v$. Additionally, we write $\phi|_b$ to define that each variable $v$ is substituted by $t$ in formula $\phi$.
We define the transition of the requisite graph through the predicate *transRG* as follows:

1. $transRG(\overline{\mathcal{N}}, \overline{\mathcal{E}}, n, NIL\_NODE, s, b, n', s', b') \equiv \exists c. \langle n, n', c \rangle \in \overline{\mathcal{E}}. c|_{b'}[s'] \wedge s = s' \wedge b = b'$. Check the condition for an outgoing edge. If the condition of the edge is true process further with the next node.

2. $transRG(\overline{\mathcal{N}}, \overline{\mathcal{E}}, n, ACTION\_NODE, s, b, n', s', b') \equiv \exists c. \langle n, n', c \rangle \in \overline{\mathcal{E}}. c|_{b'}[s'] \wedge s' = do(\alpha, s) \wedge b = b'$. Execute the associated action $\alpha$. Add the action to the history of performed actions. Add the sensing outcome of this action to the knowledge base. Choose non-deterministic one outgoing edge which has a condition which holds for the new situation.

3. $transRG(\overline{\mathcal{N}}, \overline{\mathcal{E}}, n, VAR\_BINDING\_NODE, s, b, n', s', b') \equiv \exists c. \langle n, n', c \rangle \in \overline{\mathcal{E}}. \exists t. c|_b|^v_t[s] \wedge s = s' \wedge b' = b \cup \langle v,t \rangle$. Search for a variable binding $t$ such that the condition of one outgoing edge is fulfilled and the specified condition for the variable $v$ choice is also fulfilled. If such a bind can be found use the binding to execute the program further. This is done be choosing non-deterministic one outgoing edge which has a condition which holds for the current situation.

4. $transRG(\overline{\mathcal{N}}, \overline{\mathcal{E}}, n, VAR\_RELEASING\_NODE, s, b, n', s', b') \equiv \exists c. \langle n, n', c \rangle \in \overline{\mathcal{E}}. c|_{b'}[s'] \wedge s = s' \wedge b' = b \setminus \{\langle v,t \rangle | \langle v,t \rangle \in b\}$. Remove any binding $t$ of the specified variable $v$. Afterwards proceed by choosing non-deterministic one outgoing edge which has a condition which holds for the current situation.

5. $transRG(\overline{\mathcal{N}}, \overline{\mathcal{E}}, n, START\_NODE, s, b, n', s', b') \equiv \exists c. \langle n, n', c \rangle \in \overline{\mathcal{E}}. c|_{b'}[s'] \wedge s' = S_0 \wedge b' = \emptyset$. Set the current situation to be the initial situation. Afterwards choose non-deterministic one outgoing edge which has a condition which holds for the current situation.

6. $transRG(\overline{\mathcal{N}}, \overline{\mathcal{E}}, n, END\_NODE, s, b, n', s', b') \equiv \bot$. Terminate the program with a success.

We define the termination condition of the requisite graph through the predicate *finalRG* as follows:

1. $finalRG(\overline{\mathcal{N}}, \overline{\mathcal{E}}, n, NIL\_NODE, s, b) \equiv \bot$, no termination possible.

2. $finalRG(\overline{\mathcal{N}}, \overline{\mathcal{E}}, n, ACTION\_NODE, s, b) \equiv \bot$, no termination possible.

3. $finalRG(\overline{\mathcal{N}}, \overline{\mathcal{E}}, n, VAR\_BINDING\_NODE, s, b) \equiv \bot$, no termination possible.

4. $\mathit{finalRG}(\overline{\mathcal{N}}, \overline{\mathcal{E}}, n, \mathit{VAR\_RELEASING\_NODE}, s, b) \equiv \bot$, no termination possible.
5. $\mathit{finalRG}(\overline{\mathcal{N}}, \overline{\mathcal{E}}, n, \mathit{START\_NODE}, s, b) \equiv \bot$, no termination possible.
6. $\mathit{finalRG}(\overline{\mathcal{N}}, \overline{\mathcal{E}}, n, \mathit{END\_NODE}, s, b) \equiv \top$, end of program reached terminate with success.

Using the two predicates to specify the execution of the requisite graph $RG$. One can interpret the program as follows:

- Terminate with a success if the following holds $\exists \tau.\tau = \mathit{type}(n) \wedge \mathit{finalRG}(\overline{\mathcal{N}}, \overline{\mathcal{E}}, n, \tau, s, b)$, where $n$ is the current node of the program, $s$ is the current situation, and $b$ is the current variable binding
- Proceed with the next node $n'$, the new situation $s'$ and the new variable binding $b'$ if the following holds $\exists \tau.\tau = \mathit{type}(n) \wedge \mathit{transRG}(\overline{\mathcal{N}}, \overline{\mathcal{E}}, n, \tau, s, b, n', s', b')$, where $n$ is the current node of the program, $s$ is the current situation, and $b$ is the current variable binding
- If neither of the above holds terminate indicating a fault.

Through the execution semantic of the requisite graph above, we have defined how to interpret a requisite graph. As the high-level program is given as AST, we need to define a transformation between the AST and a requisite graph. This transformation should ensure that the interpretation of the requisite graph corresponds to the interpretation of the AST itself.
We define the transformation for an AST given through the tuple $\langle \mathcal{N}, \mathcal{E} \rangle$ the current node of the AST $n$ which is considered, $\tau$ the type of the current node and the requisite graph represented through the tuple $\langle \overline{\mathcal{N}}, \overline{\mathcal{E}}$ with the current start node $\overline{n_s}$, the current end node $\overline{n_e}$ and the current condition to execute the start node $c$. The transformation will ensure that for the AST of the full program the start node will correspond to a start node in the requisite graph. Furthermore, the end node in the requisite graph will correspond to an end node in the requisite graph. Additionally, the condition $c$ will be true for the AST of the full program. The transformation is defined with the help of the predicate *TransASTToRG* as follows:

1. $\mathit{TransASTToRG}(\mathcal{N}, \mathcal{E}, n, \mathit{ROOT}, \overline{\mathcal{N}}, \overline{\mathcal{E}}, \overline{n_s}, \overline{n_e}, c) \equiv \exists n', \overline{\mathcal{N}}', \overline{\mathcal{E}}', \overline{n_s}', \overline{n_e}', c', \tau.\langle n, n' \rangle \in \mathcal{E} \wedge \tau = \mathit{type}(n') \wedge \mathit{TransASTToRG}(\mathcal{N}, \mathcal{E}, n', \tau, \overline{\mathcal{N}}', \overline{\mathcal{E}}', \overline{n_s}', \overline{n_e}', c') \wedge \overline{\mathcal{N}} = \overline{\mathcal{N}}' \cup \{\overline{n_s}, \overline{n_e}\} \wedge \overline{\mathcal{E}} = \overline{\mathcal{E}}' \cup \{\langle \overline{n_s}, \overline{n_s}', c' \rangle, \langle \overline{n_e}', \overline{n_e}, \top \rangle\} \wedge \mathit{type}(\overline{n_s}) = \mathit{START\_NODE} \wedge \mathit{type}(\overline{n_e}) = \mathit{END\_NODE}$
2. $\mathit{TransASTToRG}(\mathcal{N}, \mathcal{E}, n, \mathit{ACTION}, \overline{\mathcal{N}}, \overline{\mathcal{E}}, \overline{n_s}, \overline{n_e}, c) \equiv \overline{\mathcal{N}} = \{\overline{n_s}, \overline{n_e}\} \wedge \overline{\mathcal{E}} = \{\langle \overline{n_s}, \overline{n_e}, c \rangle\} \wedge c = \pi_\alpha \wedge \mathit{action}(\overline{n_e}) = \alpha \wedge \mathit{type}(\overline{n_e}) = \mathit{ACTION\_NODE} \wedge \mathit{type}(\overline{n_s}) = \mathit{NIL\_NODE}$
3. $\mathit{TransASTToRG}(\mathcal{N}, \mathcal{E}, n, \mathit{TEST}, \overline{\mathcal{N}}, \overline{\mathcal{E}}, \overline{n_s}, \overline{n_e}, c) \equiv \overline{\mathcal{N}} = \{\overline{n_s}, \overline{n_e}\} \wedge \overline{\mathcal{E}} = \{\langle \overline{n_s}, \overline{n_e}, c \rangle\} \wedge c = \mathit{COND} \wedge \mathit{type}(\overline{n_e}) = \mathit{NIL\_NODE} \wedge \mathit{type}(\overline{n_s}) = \mathit{NIL\_NODE}$
4. $\mathit{TransASTToRG}(\mathcal{N}, \mathcal{E}, n, \mathit{NIL}, \overline{\mathcal{N}}, \overline{\mathcal{E}}, \overline{n_s}, \overline{n_e}, c) \equiv \overline{\mathcal{N}} = \{\overline{n_s}, \overline{n_e}\} \wedge \overline{\mathcal{E}} = \{\langle \overline{n_s}, \overline{n_e}, c \rangle\} \wedge c = \top \wedge \mathit{type}(\overline{n_e}) = \mathit{NIL\_NODE} \wedge \mathit{type}(\overline{n_s}) = \mathit{NIL\_NODE}$
5. $\mathit{TransASTToRG}(\mathcal{N}, \mathcal{E}, n, \mathit{IF}, \overline{\mathcal{N}}, \overline{\mathcal{E}}, \overline{n_s}, \overline{n_e}, c) \equiv \exists n_1, n_2.n_1 = \mathit{child}(n)[1] \wedge n_2 = \mathit{child}(n)[2] \wedge \exists \overline{\mathcal{N}_1}, \overline{\mathcal{E}_1}, \overline{n_{s1}}, \overline{n_{e1}}, \tau_1, c_1, \overline{\mathcal{N}_2}, \overline{\mathcal{E}_2}, \overline{n_{s2}}, \overline{n_{e2}}, \tau_2, c_2.\tau_1 = \mathit{type}(n_1) \wedge \tau_2 = \mathit{type}(n_2) \wedge \mathit{TransASTToRG}(\mathcal{N}, \mathcal{E}, n_1, \tau_1, \overline{\mathcal{N}_1}, \overline{\mathcal{E}_1}, \overline{n_{s1}}, \overline{n_{e1}}, c_1) \wedge \mathit{TransASTToRG}(\mathcal{N}, \mathcal{E}, n_2, \tau_2, \overline{\mathcal{N}_2}, \overline{\mathcal{E}_2}, \overline{n_{s2}}, \overline{n_{e2}}, c_2) \wedge \overline{\mathcal{N}} = \overline{\mathcal{N}_1} \cup \overline{\mathcal{N}_2} \cup \{\overline{n_s}, \overline{n_e}\} \wedge \overline{\mathcal{E}} = \overline{\mathcal{E}_1} \cup \overline{\mathcal{E}_2} \cup \{\langle \overline{n_s}, \overline{n_{s1}}, \mathit{COND} \wedge c_1 \rangle, \langle \overline{n_s}, \overline{n_{s2}}, \neg \mathit{COND} \wedge c_2 \rangle, \langle \overline{n_{e1}}, \overline{n_e}, \top \rangle, \langle \overline{n_{e2}}, \overline{n_e}, \top \rangle\} \wedge c = (\mathit{COND} \wedge c_1) \vee (\neg \mathit{COND} \wedge c_2) \wedge \mathit{type}(\overline{n_e}) = \mathit{NIL\_NODE} \wedge \mathit{type}(\overline{n_s}) = \mathit{NIL\_NODE}$

6. $\mathit{TransASTToRG}(\mathcal{N},\mathcal{E},n,\mathit{WHILE},\overline{\mathcal{N}},\overline{\mathcal{E}},\overline{n_s},\overline{n_e},c) \equiv \exists n'.n' = \mathit{child}(n)[1] \wedge \exists\overline{\mathcal{N}}',\overline{\mathcal{E}}',\overline{n_s}',\overline{n_e}',\tau',c'.\tau' = \mathit{type}(n') \wedge \mathit{TransASTToRG}(\mathcal{N},\mathcal{E},n',\tau',\overline{\mathcal{N}}',\overline{\mathcal{E}}',\overline{n_s}',\overline{n_e}',c') \wedge \overline{\mathcal{N}} = \overline{\mathcal{N}}' \cup \{\overline{n_s},\overline{n_e}\} \wedge \overline{\mathcal{E}} = \overline{\mathcal{E}}' \cup \{\langle\overline{n_s},\overline{n_s}',\mathit{COND} \wedge c'\rangle,\langle\overline{n_e},\overline{n_s},\mathit{COND} \wedge c'\rangle,\langle\overline{n_s},\overline{n_e},\neg\mathit{COND}\rangle,\langle\overline{n_e}',\overline{n_e},\neg\mathit{COND}\rangle\} \wedge c = (\mathit{COND} \wedge c') \vee \neg\mathit{COND} \wedge \mathit{type}(\overline{n_e}) = \mathit{NIL\_NODE} \wedge \mathit{type}(\overline{n_s}) = \mathit{NIL\_NODE}$

7. $\mathit{TransASTToRG}(\mathcal{N},\mathcal{E},n,\mathit{SEQUENCE},\overline{\mathcal{N}},\overline{\mathcal{E}},\overline{n_s},\overline{n_e},c) \equiv \exists n_1,n_2.n_1 = \mathit{child}(n)[1] \wedge n_2 = \mathit{child}(n)[2] \wedge \exists\overline{\mathcal{N}_1},\overline{\mathcal{E}_1},\overline{n_{s1}},\overline{n_{e1}},\tau_1,c_1,\overline{\mathcal{N}_2},\overline{\mathcal{E}_2},\overline{n_{s2}},\overline{n_{e2}},\tau_2,c_2.\tau_1 = \mathit{type}(n_1) \wedge \tau_2 = \mathit{type}(n_2) \wedge \mathit{TransASTToRG}(\mathcal{N},\mathcal{E},n_1,\tau_1,\overline{\mathcal{N}_1},\overline{\mathcal{E}_1},\overline{n_{s1}},\overline{n_{e1}},c_1) \wedge \mathit{TransASTToRG}(\mathcal{N},\mathcal{E},n_2,\tau_2,\overline{\mathcal{N}_2},\overline{\mathcal{E}_2},\overline{n_{s2}},\overline{n_{e2}},c_2) \wedge \overline{\mathcal{N}} = \overline{\mathcal{N}_1} \cup \overline{\mathcal{N}_2} \wedge \overline{\mathcal{E}} = \overline{\mathcal{E}_1} \cup \overline{\mathcal{E}_2} \cup \{\langle\overline{n_{e1}},\overline{n_{s2}},c_2\rangle\} \wedge c = c_1 \wedge \overline{n_s} = \overline{n_{s1}} \wedge \overline{n_e} = \overline{n_{e2}}$. Without loss of generality we assume that a sequence only consists of two elements.

8. $\mathit{TransASTToRG}(\mathcal{N},\mathcal{E},n,\mathit{CHOOSE},\overline{\mathcal{N}},\overline{\mathcal{E}},\overline{n_s},\overline{n_e},c) \equiv \exists n_1,n_2.n_1 = \mathit{child}(n)[1] \wedge n_2 = \mathit{child}(n)[2] \wedge \exists\overline{\mathcal{N}_1},\overline{\mathcal{E}_1},\overline{n_{s1}},\overline{n_{e1}},\tau_1,c_1,\overline{\mathcal{N}_2},\overline{\mathcal{E}_2},\overline{n_{s2}},\overline{n_{e2}},\tau_2,c_2.\tau_1 = \mathit{type}(n_1) \wedge \tau_2 = \mathit{type}(n_2) \wedge \mathit{TransASTToRG}(\mathcal{N},\mathcal{E},n_1,\tau_1,\overline{\mathcal{N}_1},\overline{\mathcal{E}_1},\overline{n_{s1}},\overline{n_{e1}},c_1) \wedge \mathit{TransASTToRG}(\mathcal{N},\mathcal{E},n_2,\tau_2,\overline{\mathcal{N}_2},\overline{\mathcal{E}_2},\overline{n_{s2}},\overline{n_{e2}},c_2) \wedge \overline{\mathcal{N}} = \overline{\mathcal{N}_1} \cup \overline{\mathcal{N}_2} \cup \{\overline{n_s},\overline{n_e}\} \wedge \overline{\mathcal{E}} = \overline{\mathcal{E}_1} \cup \overline{\mathcal{E}_2} \cup \{\langle\overline{n_s},\overline{n_{s1}},c_1\rangle,\langle\overline{n_s},\overline{n_{s2}},c_2\rangle,\langle\overline{n_{e1}},\overline{n_e},\top\rangle,\langle\overline{n_{e2}},\overline{n_e},\top\rangle\} \wedge c = c_1 \vee c_2 \wedge \mathit{type}(\overline{n_s}) = \mathit{NIL\_NODE} \wedge \mathit{type}(\overline{n_e}) = \mathit{NIL\_NODE}$. Without loss of generality we assume that a choice is made between two elements.

9. $\mathit{TransASTToRG}(\mathcal{N},\mathcal{E},n,\mathit{PICK},\overline{\mathcal{N}},\overline{\mathcal{E}},\overline{n_s},\overline{n_e},c) \equiv \exists n'.n' = \mathit{child}(n)[1] \wedge \exists\overline{\mathcal{N}}',\overline{\mathcal{E}}',\overline{n_s}',\overline{n_e}',\tau',c'.\tau' = \mathit{type}(n') \wedge \mathit{TransASTToRG}(\mathcal{N},\mathcal{E},n',\tau',\overline{\mathcal{N}}',\overline{\mathcal{E}}',\overline{n_s}',\overline{n_e}',c') \wedge \overline{\mathcal{N}} = \overline{\mathcal{N}}' \cup \{\overline{n_s},\overline{n_e}\} \wedge \overline{\mathcal{E}} = \overline{\mathcal{E}}' \cup \{\langle\overline{n_s},\overline{n_s}',c'\rangle,\langle\overline{n_s},\overline{n_e},\top\rangle\} \wedge [c = \exists \mathit{variable}(n).(\mathit{COND} \wedge c') \vee \neg\mathit{COND}] \wedge \mathit{type}(\overline{n_e}) = \mathit{VAR\_BINDING\_NODE} \wedge \mathit{type}(\overline{n_s}) = \mathit{VAR\_RELEASING\_NODE} \wedge \mathit{variable}(n_s) = \mathit{variable}(n) \wedge \mathit{variable}(n_e) = \mathit{variable}(n)$

With the help of the above transformation, the AST of a program can be transformed into a requisite graph. This transformation ensures that one can either use the program and executed it according to the IndiGolog semantics or one can apply the transformation and use the transition semantics of the requisite graph. We state this property formally in the following theorem.

**Theorem 11.** *Given a IndiGolog program $\delta$ and an initial situation $S_0$ any sequence of actions $[\alpha_1,\dots\alpha_n]$ which can be executed by the according to $\delta$ can also be executed according to the requisite graph (RG) which was generated from program $\delta$ according to the above transformation.*

*Proof (Sketch).* Proof by induction.
For the base case, we only need to consider a simple action. In the case of IndiGolog, this action would be executed if the precondition of the action holds. The *RG* consists of three nodes the start node one action node and the end node. The action node can only be reached if the precondition holds. The condition of the edge between action node and the end node is true. Thus, if the condition of the action holds in the initial situation the action is executed and the finale the end node is reached according to the interpreter defined above.
For the induction step lets's assume the theorem holds for a program $\delta$ with size $n$. Further, assume that the program is now a statement which uses in its body a program of size at least $n$.
Let's consider this statement is an if. The if branch if branch is executed if the condition holds per definition. Additionally, the else branch is executed if the negation of the condition holds per definition. The conversion of the program creates a sub graph representing the if part, which can only

be executed if the condition holds. Additionally, the sub graph representing the else branch can only be executed if the negation of the condition holds thus the theorem holds for an if statement.
Let's consider this statement is a while. Per definition of IndiGolog if the condition does not hold the program terminates. If the condition holds the body is executed once and afterward the loop condition is rechecked. After conversion of the program, the sub graph representing the loop body is only executed if the condition holds. Additionally, after the execution of the loop body, the interpreter is again on the same node as at the beginning of the loop. Thus, performing the same checks for the loop again. In case the negation of the condition holds the program terminates. Thus, the theorem holds for a while statement.
Let's consider this statement is a test. The program can only proceed if the test is passed per definition. Due to the conversion, the condition is part of the edge leading to the next node and thus needs to hold for any future execution of the program. Thus, the theorem holds for a test statement.
Let's consider this statement is a sequence. The first part of the sequence needs to be executed completely before the second part is executed per definition. Due to the conversion, the second part is appended to the completion of the first part. Thus, the theorem holds for a sequence statement.
Let's consider this statement is a nondeterministic choice of branches. One of the branches, which is possible to be executed, is executed per definition. Due to the conversion one creates several branches of the start state to the different branches of the statement. Due to the interpretation, only one of the branches is executed which also can be executed. Thus, the theorem holds for a nondeterministic choice of branches.
Let's consider this statement is a nondeterministic choice of arguments. Per definition of IndiGolog one executes the body with a variable binding which allows the execution. Due to the conversion, a node is created which searches for a variable binding. Due to the interpretation, the variable bind ensures that further execution is possible. Thus, the theorem holds for a nondeterministic choice of arguments. □

Before we discuss how we use the requisite graph to construct a reactive program we will shortly discuss the requisite graph of the example AST. The requisite graph of the example AST is depicted in Figure 8.2. The graph shows that there are two branches form the start. The first branch leads to the action $\alpha_1$. We can take this branch if the condition of the if holds and the precondition of the action is fulfilled. The other branch leads to action $\alpha_2$ which imposes that the negation of the condition holds together with the precondition of the action. Additionally, we have connection between $\alpha_2$ and $\alpha_3$. Which can only be taken if the precondition of $\alpha_3$ hold. Finally, the program can terminate after either action $\alpha_2$ or action $\alpha_3$ was executed.
In Figure 8.2 we have shown a requisite graph of the program pruned to remove *NIL* nodes. This pruning allows to handle a smaller graph as well as to ignore *NIL* nodes in further transformation process. To use these benefits for the transformation from an imperative to a reactive program we define a reduction method which removes *NIL* nodes from the graph but preserves its execution behavior. We specify this transformation through the predicate *Compress* which uses a given requisite graph defined through the tuple $\overline{\mathcal{N}}$ and $\overline{\mathcal{E}}$ and results in a new requisite graph defined through the tuple $\overline{\mathcal{N}}'$

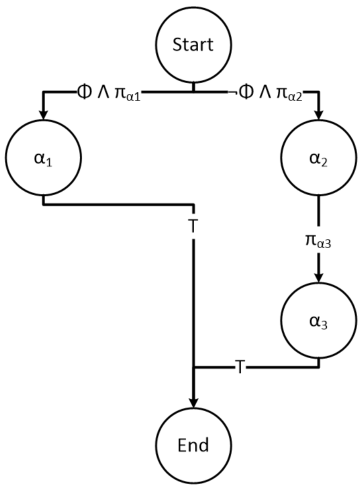

Figure 8.2.: The requisite graph of the simple example program **if** $\phi$ **then** $\alpha_1$ **else** $\alpha_2$ ; $\alpha_3$ **endIf**.

and $\overline{\mathcal{E}}'$. The predicate is now defined as follows.

$$\begin{aligned}
Compress(\overline{\mathcal{N}},\overline{\mathcal{E}},\overline{\mathcal{N}}',\overline{\mathcal{E}}') \equiv & [\nexists n_s,n_e,n,c_1,c_2.type(n) = NIL\_NODE \wedge \langle n_s,n,c_1\rangle \in \overline{\mathcal{E}} \wedge \\
& \langle n,n_e,c_2\rangle \in \overline{\mathcal{E}} \rightarrow \overline{\mathcal{N}}' = \overline{\mathcal{N}} \wedge \overline{\mathcal{E}}' = \overline{\mathcal{E}}] \bigvee \\
& [\exists n_s,n_e,n,c_1,c_2.type(n) = NIL\_NODE \wedge \langle n_s,n,c_1\rangle \in \overline{\mathcal{E}} \wedge \\
& \langle n,n_e,c_2\rangle \in \overline{\mathcal{E}} \rightarrow \overline{\mathcal{E}}' = \overline{\mathcal{E}} \setminus \{\langle n_s,n,c_1\rangle,\langle n,n_e,c_2\rangle\} \cup \langle n_s,n_e,c_1 \wedge c_2\rangle \wedge \\
& \overline{\mathcal{N}}' = \left\{n | n \in \overline{\mathcal{N}} \wedge \exists n',c'.\langle n,n',c'\rangle \in \overline{\mathcal{E}} \vee \langle n',n,c'\rangle \in \overline{\mathcal{E}}\right\}]
\end{aligned}$$

With the help of the above predicate, the requisite graph is reduced such that no *NIL* node remains in the graph. As we want to use such a reduced graph for the transformation into a reactive program, we need to ensure that this transformation does not yield a requisite graph which behaves differently. The following lemma states formally that the execution behavior remains the same regardless if the requisite graph is compressed or not.

**Lemma 19.** *Any action sequence* $[\alpha_1,\ldots\alpha_n]$ *which is the result of the execution of a requisite graph given through* $\overline{\mathcal{N}}$ *and* $\overline{\mathcal{E}}$ *can also be the result of the execution of the compressed requisite graph given through* $\overline{\mathcal{N}}'$ *and* $\overline{\mathcal{E}}'$*, where it holds that* $Compress(\overline{\mathcal{N}},\overline{\mathcal{E}},\overline{\mathcal{N}}',\overline{\mathcal{E}}')$.

*Proof (Sketch).* This holds as the definition of *Compress* ensures that both conditions to enter a *NIL* node and to leave the *NIL* node to execute any next node after this *NIL* node. Thus, the same traces can be executed in the graph if the *NIL* nodes are removed from that trace. □

Above we have formally defined how to generate a requisite graph from a given AST. This transformation definition defines a proper compilation from an AST to a requisite graph. We perform this compilation only considering those parts of the IndiGolog semantic which are used for non-concurrent program execution. Additionally, we consider the program does not contain the search operator. To perform this compilation Algorithm 8 can be used. The algorithm is called with the root node of the AST and with all other sets empty and the condition to prepend to be true. Thus, the initial call is as follows $ComputeRG(N, \{n_{progStart}\}, \top, \{n_{progEnd}\}, \emptyset, N_g, C_{newP}, \overline{\mathcal{N}}, \overline{\mathcal{E}})$. The algorithm performs the transformation per *TransASTToRG* and performs the reduction of the graph per *Compress*. Thus, the result of the algorithm is a requisite graph from the given Program which is already reduced not to contain *NIL* nodes.

**Algorithm 8:** *ComputeRG*

```
Data: $n_{AST} \ldots$ the current node of the AST
Data: $N_s \ldots$ a set of start nodes
Data: $C_P \ldots$ a condition to prepend
Data: $\overline{\mathcal{N}} \ldots$ a set of nodes defining the current requisite graph
Data: $\overline{\mathcal{E}} \ldots$ a set of edges within the current requisite graph
Result: $N_g \ldots$ the current set of goal nodes
Result: $C_{newP} \ldots$ a new condition to prepend
Result: $\overline{\mathcal{N}}' \ldots$ a set of nodes defining the new requisite graph
Result: $\overline{\mathcal{E}}' \ldots$ a set of edges within the new requisite graph
1  begin
2      $\langle T, Co, x, \alpha \rangle = n_{AST}$ ;
3      $Ch = child(n_{AST})$ ;
4      switch $T$ do
5          case If do
6              $CRGIF(Ch, Co, N_s, C_P, \overline{\mathcal{N}}, \overline{\mathcal{E}}, N_g, C_{newP}, \overline{\mathcal{N}}', \overline{\mathcal{E}}')$ ;
7          end
8          case While do
9              $CRGWhile(Ch, Co, N_s, C_P, \overline{\mathcal{N}}, \overline{\mathcal{E}}, N_g, C_{newP}, \overline{\mathcal{N}}', \overline{\mathcal{E}}')$ ;
10         end
11         case Sequence do
12             $CRGSequence(Ch, N_s, C_P, \overline{\mathcal{N}}, \overline{\mathcal{E}}, N_g, C_{newP}, \overline{\mathcal{N}}', \overline{\mathcal{E}}')$ ;
13         end
14         case Choose do
15             $CRGChoose(Ch, N_s, C_P, \overline{\mathcal{N}}, \overline{\mathcal{E}}, N_g, C_{newP}, \overline{\mathcal{N}}', \overline{\mathcal{E}}')$ ;
16         end
17         case Pick do
18             $CRGPick(Ch, Co, x, N_s, C_P, \overline{\mathcal{N}}, \overline{\mathcal{E}}, N_g, C_{newP}, \overline{\mathcal{N}}', \overline{\mathcal{E}}')$ ;
19         end
20         case Test do
21             $CRGTest(Co, N_s, C_P, \overline{\mathcal{N}}, \overline{\mathcal{E}}, N_g, C_{newP}, \overline{\mathcal{N}}', \overline{\mathcal{E}}')$ ;
22         end
23         case Action do
24             $CRGAction(\alpha, N_s, C_P, \overline{\mathcal{N}}, \overline{\mathcal{E}}, N_g, C_{newP}, \overline{\mathcal{N}}', \overline{\mathcal{E}}')$ ;
25         end
26     end
27 end
```

To perform the compilation from the AST to the requisite graph the Algorithm 8 uses the typo of the statement to dispatch the compilation to one of the following Algorithms 9, 10, 11, 12, 13, 14 and

15. These algorithm calls Algorithm 8 to resolve child nodes in the AST. The recursion stops with the depth of the AST. Thus algorithm visits each node in the AST once and terminates in $O(|AST|)$.

**Algorithm 9:** *CRGIF*

```
Data: Children... children of the current node of the AST
Data: Cond.... condition of the current node of the AST
Data: $N_s$... a set of start nodes
Data: $C_P$... a condition to prepend
Data: $\overline{\mathcal{N}}$... a set of nodes defining the current requisite graph
Data: $\overline{\mathcal{E}}$... a set of edges within the current requisite graph
Result: $N_g$... the current set of goal nodes
Result: $C_{newP}$... a new condition to prepend
Result: $\overline{\mathcal{N}}'$... a set of nodes defining the new requisite graph
Result: $\overline{\mathcal{E}}'$... a set of edges within the new requisite graph
1 begin
2     $C'_P = C_P \wedge Cond.$ ;
3     $ComputeRG(Children[0], N_s, C'_P, \overline{\mathcal{N}}, \overline{\mathcal{E}}, N'_g, C'_{newP}, \overline{\mathcal{N}}'', \overline{\mathcal{E}}'')$ ;
4     $C''_P = C_P \wedge \neg Cond.$ ;
5     $ComputeRG(Children[1], N_s, C''_P, \overline{\mathcal{N}}'', \overline{\mathcal{E}}'', N''_g, C''_{newP},, \overline{\mathcal{N}}', \overline{\mathcal{E}}')$ ;
6     $N_g = N'_g \cup N''_g$ ;
7     $C_{newP} = C'_{newP} \vee C''_{newP}$ ;
8 end
```

Algorithm 9 handles an if. This is done by creates two sub graphs. The first sub graph represents the case when the condition holds. Thus, the condition of the if is imposed on this sub graph. The second sub graph represents the else branch of the if. Thus, the negation of the condition of the if is imposed on this sub graph. Finally, the set of both final nodes is composed as the robot is in one of these nodes during the execution.
Algorithm 10 handles a while. A while is created by creating a sub graph which contains the transformation of the loop body. This sub graph is connected by imposing the loop condition as a prerequisite. To allow to continue the execution after the loop is finished the end of the loop body relates to the beginning of the while. Additionally, the start nodes of the while loop are the only nodes which allow to end the loop if the condition becomes false.
Algorithm 11 handles a test in the program. It simply adds the test to the condition which needs to hold to execute the next action.
Algorithm 12 handles a sequence in the program. The algorithm simply concatenates the different program parts of the sequence one after the other.
Algorithm 13 handles the non-deterministic chose of branches. The algorithm creates a sub graph for each of the different program branches. Afterwards the termination nodes of the program branches are combined. Furthermore, all branches use the same start node. Thus, the robot can choose one branch and afterwards execute this branch. Regardless of the executed branch the robot will end up in the same set of goal nodes.
Algorithm 14 handles a non-deterministic chose of arguments. The algorithm creates a node indicating the new creation of a variable $x$. This node is connected to the sub graph created by the body of the pick statement. Finally, the goal nodes of the body are connected to a new created node which releases

```
Algorithm 10: CRGWhile
Data: Children ... children of the current node of the AST
Data: Cond.... condition of the current node of the AST
Data: $N_s$ ... a set of start nodes
Data: $C_P$ ... a condition to prepend
Data: $\overline{\mathcal{N}}$ ... a set of nodes defining the current requisite graph
Data: $\overline{\mathcal{E}}$ ... a set of edges within the current requisite graph
Result: $N_g$ ... the current set of goal nodes
Result: $C_{newP}$ ... a new condition to prepend
Result: $\overline{\mathcal{N}}'$ ... a set of nodes defining the new requisite graph
Result: $\overline{\mathcal{E}}'$ ... a set of edges within the new requisite graph
begin
    $C'_P = C_P \wedge Cond.$ ;
    $ComputeRG(Children[0], N_s, C'_P, \overline{\mathcal{N}}, \overline{\mathcal{E}}, N'_g, C'_{newP}, \overline{\mathcal{N}}', \overline{\mathcal{E}}')$ ;
    $N_g = N_s$ ;
    $C_{newP} = C_P \wedge \neg Cond.$ ;
    foreach $n_g \in N'_g$ do
        foreach $n_s \in N_s$ do
            $\overline{\mathcal{E}}' = \overline{\mathcal{E}}' \cup \{\langle n_g, n_s, C'_{newP} \rangle\}$
        end
    end
end
```

```
Algorithm 11: CRGTest
Data: Cond.... condition of the current node of the AST
Data: $N_s$ ... a set of start nodes
Data: $C_P$ ... a condition to prepend
Data: $\overline{\mathcal{N}}$ ... a set of nodes defining the current requisite graph
Data: $\overline{\mathcal{E}}$ ... a set of edges within the current requisite graph
Result: $N_g$ ... the current set of goal nodes
Result: $C_{newP}$ ... a new condition to prepend
Result: $\overline{\mathcal{N}}'$ ... a set of nodes defining the new requisite graph
Result: $\overline{\mathcal{E}}'$ ... a set of edges within the new requisite graph
begin
    $C_{newP} = C_P \wedge Cond.$ ;
    $N_g = N_s$ ;
end
```

the variable $x$.
Finally, Algorithm 15 handles an action. To execute the action, the precondition is added to the edge condition. Afterwards a node is created which performs the action execution. All start nodes are afterwards connected to this newly created node. The goal node is the node which representing the action execution.
The above algorithms convert an AST into a requisite graph. This transformation is done properly

**Algorithm 12:** *CRGSequence*

```
Data: $Children \dots$ children of the current node of the AST
Data: $N_s \dots$ a set of start nodes
Data: $C_P \dots$ a condition to prepend
Data: $\overline{\mathcal{N}} \dots$ a set of nodes defining the current requisite graph
Data: $\overline{\mathcal{E}} \dots$ a set of edges within the current requisite graph
Result: $N_g \dots$ the current set of goal nodes
Result: $C_{newP} \dots$ a new condition to prepend
Result: $\overline{\mathcal{N}}' \dots$ a set of nodes defining the new requisite graph
Result: $\overline{\mathcal{E}}' \dots$ a set of edges within the new requisite graph
begin
    $\overline{\mathcal{N}}_{tmpIn} = \overline{\mathcal{N}}$ ;
    $\overline{\mathcal{E}}_{tmpIn} = \overline{\mathcal{E}}$ ;
    for $i \leftarrow 0$ to $sizeOf(Children)$ do
        $ComputeRG(Children[i], N_s, C_P, \overline{\mathcal{N}}_{tmpIn}, \overline{\mathcal{E}}_{tmpIn}, N_g, C_{newP}, \overline{\mathcal{N}}_{tmpOut}, \overline{\mathcal{E}}_{tmpOut})$ ;
        $C_P = C_{newP}$ ;
        $N_s = N_g$ ;
        $\overline{\mathcal{N}}_{tmpIn} = \overline{\mathcal{N}}$ ;
        $\overline{\mathcal{E}}_{tmpIn} = \overline{\mathcal{E}}$ ;
    end
    $\overline{\mathcal{N}}' = \overline{\mathcal{N}}_{tmpOut}$ ;
    $\overline{\mathcal{E}}' = \overline{\mathcal{E}}_{tmpOut}$ ;
end
```

which is state more formally in the following Proposition.

**Proposition 1.** *For any given program, defined through* $\mathcal{N}$ *and* $\mathcal{E}$*, the result of Algorithm 8 is* $\overline{\mathcal{N}}'$ *and* $\overline{\mathcal{E}}'$*. Furthermore it holds that* $\exists n, \tau, \overline{\mathcal{N}}, \overline{\mathcal{E}}, \overline{n_s}, \overline{n_e}, c.n = rootOf(\mathcal{N}, \mathcal{E}) \wedge \tau = ROOT \wedge TransASTToRG(\mathcal{N}, \mathcal{E}, n, \tau, \overline{\mathcal{N}}, \overline{\mathcal{E}}, \overline{n_s}, \overline{n_e}, c) \wedge Compress(\overline{\mathcal{N}}, \overline{\mathcal{E}}, \overline{\mathcal{N}}', \overline{\mathcal{E}}')$

With the help of the above algorithm the AST is transformed into a requisite graph. This graph can be afterwards used to derive a reactive program. To define a conversion of the program into a reactive program we first need to define a reactive program.

**Definition 26.** *A reactive program* **R** *consists of a goal condition GC and a list of tuples of the following form:*

- *C. A condition which needs to hold to execute the action.*
- *A. An action sequence to execute.*

*The reactive program is executed till the goal condition holds. In each iteration of the program the action sequence with the lowest index is executed of those action sequence where the condition is fulfilled.*

We could now convert the complete program into a reactive version. However, this would cause the robot to reason about the complete reactive program each time an action is executed. Thus, causing a

**Algorithm 13:** *CRGChoose*

```
   Data: $Children\ldots$ children of the current node of the AST
   Data: $N_s\ldots$ a set of start nodes
   Data: $C_P\ldots$ a condition to prepend
   Data: $\overline{\mathcal{N}}\ldots$ a set of nodes defining the current requisite graph
   Data: $\overline{\mathcal{E}}\ldots$ a set of edges within the current requisite graph
   Result: $N_g\ldots$ the current set of goal nodes
   Result: $C_{newP}\ldots$ a new condition to prepend
   Result: $\overline{\mathcal{N}}'\ldots$ a set of nodes defining the new requisite graph
   Result: $\overline{\mathcal{E}}'\ldots$ a set of edges within the new requisite graph
1  begin
2      $C_{newP} = \perp$ ;
3      $N_g = \{\}$ ;
4      $\overline{\mathcal{N}}_{tmpIn} = \overline{\mathcal{N}}$ ;
5      $\overline{\mathcal{E}}_{tmpIn} = \overline{\mathcal{E}}$ ;
6      foreach $c \in Children$ do
7          $ComputeRG(c, N_s, C_P, \overline{\mathcal{N}}_{tmpIn}, \overline{\mathcal{E}}_{tmpIn}, N'_g, C'_{newP}, \overline{\mathcal{N}}_{tmpOut}, \overline{\mathcal{E}}_{tmpOut})$ ;
8          $C_{newP} = C_{newP} \vee C'_{newP}$ ;
9          $N_g = N_g \cup N'_g$ ;
10         $\overline{\mathcal{N}}_{tmpIn} = \overline{\mathcal{N}}$ ;
11         $\overline{\mathcal{E}}_{tmpIn} = \overline{\mathcal{E}}$ ;
12     end
13     $\overline{\mathcal{N}}' = \overline{\mathcal{N}}_{tmpOut}$ ;
14     $\overline{\mathcal{E}}' = \overline{\mathcal{E}}_{tmpOut}$ ;
15 end
```

high run time. This great run time is due to a reactive program consisting of $L$ statements would need to check $L$ statements every time we want to execute the next actions thus we would perform $O(L^2)$ checks just to carry out a simple sequential program.

Instead of converting the complete program we will create a reactive program for each action in the program together with a local recovery. This local recovery is bounded by a fixed number $k$ allowing only $k$ actions to step back to perform this recovery. Beside the improvement of a large program, the local recovery follows a simple idea. If a program is written to achieve a certain task on performs several actions to prepare an action which is executed next. Thus, it is often the case that the actions before the current actions can be used to perform a repair. Performing the local recovery, we exploit this fact.

As a reactive program needs a termination condition to stop its iteration, we need to take care that the reactive program only terminates if the action we are interested in was executed successfully. To ensure this, we create a new fluent *FinishedAction*. The fluent is true if the action is successfully executed. Please note we assume that every fault mode imposes that this fluent is set to false. Additionally, the fluent is set to false after any action execution, which is not the currently focused action. To set the fluent to be false we use the action *SetFinishedActionFalse*.

With the above-helping action and the created requisite graph we can use Algorithm 16 to create a reactive program for a certain action $\alpha$ which is defined in the graph by the node $n$.

**Algorithm 14:** *CRGPick*

```
Data: $Children \ldots$ children of the current node of the AST
Data: $Cond. \ldots$ condition of the current node of the AST
Data: $x \ldots$ variable to bind
Data: $N_s \ldots$ a set of start nodes
Data: $C_P \ldots$ a condition to prepend
Data: $\overline{\mathcal{N}} \ldots$ a set of nodes defining the current requisite graph
Data: $\overline{\mathcal{E}} \ldots$ a set of edges within the current requisite graph
Result: $N_g \ldots$ the current set of goal nodes
Result: $C_{newP} \ldots$ a new condition to prepend
Result: $\overline{\mathcal{N}}' \ldots$ a set of nodes defining the new requisite graph
Result: $\overline{\mathcal{E}}' \ldots$ a set of edges within the new requisite graph
1  begin
2      $\overline{\mathcal{N}}_{tmpIn} = \overline{\mathcal{N}}$ ;
3      $\overline{\mathcal{E}}_{tmpIn} = \overline{\mathcal{E}}$ ;
4      $n_{bs} = createBindVariableNode(x, Cond.)$ ;
5      $\overline{\mathcal{N}}_{tmpIn} = \overline{\mathcal{N}}_{tmpIn} \cup n_{bs}$ ;
6      foreach $n \in N_s$ do
7          $\overline{\mathcal{E}}_{tmpIn} = \overline{\mathcal{E}}_{tmpIn} \cup \{\langle n, n_{bs}, C_P \rangle\}$ ;
8      end
9      $N_s = \{n_{bs}\}$ ;
10     $ComputeRG(Children[0], N_s, \top, \overline{\mathcal{N}}_{tmpIn}, \overline{\mathcal{E}}_{tmpIn}, N_g', C_{newP}', \overline{\mathcal{N}}_{tmpOut}, \overline{\mathcal{E}}_{tmpOut})$ ;
11     $\overline{\mathcal{N}}' = \overline{\mathcal{N}}_{tmpOut}$ ;
12     $\overline{\mathcal{E}}' = \overline{\mathcal{E}}_{tmpOut}$ ;
13     $n_{es} = createReleaseVariableNode(x)$ ;
14     $\overline{\mathcal{N}}' = \overline{\mathcal{N}}' \cup n_{es}$ ;
15     foreach $n \in N_g'$ do
16         $\overline{\mathcal{E}}' = \overline{\mathcal{E}}' \cup \{\langle n, n_{es}, C_{newP}' \rangle\}$ ;
17     end
18     $C_{newP} = \top$ ;
19     $N_g = \{n_{es}\}$ ;
20 end
```

The first line sets the termination condition to be the fluent *FinishedAction*. Afterward, the algorithm creates a queue $Q$ of nodes from the requisite graph which need to be integrated into the reactive program. Initially, this queue only contains the action of interested. In each iteration, one element of the queue is removed.
If this element has an index high, then $K$ it indicates that the backtracking goes too far into the past. Thus, no further expansion is performed. Additionally, if the node is not an action no further expansion is performed. There is two reason for this restriction to action nodes. First, this is the case because only actions can be added to the reactive program. Second, any other node type does not allow a simple backward reasoning. In the event of a start node, such a reasoning does not make sense as no node can be before a start node. The end node can never be checked as it cannot be reached by

**Algorithm 15:** *CRGAction*

```
Data: $\alpha\ldots$ the action to perform
Data: $N_s\ldots$ a set of start nodes
Data: $C_P\ldots$ a condition to prepend
Data: $\overline{\mathcal{N}}\ldots$ a set of nodes defining the current requisite graph
Data: $\overline{\mathcal{E}}\ldots$ a set of edges within the current requisite graph
Result: $N_g\ldots$ the current set of goal nodes
Result: $C_{newP}\ldots$ a new condition to prepend
Result: $\overline{\mathcal{N}}'\ldots$ a set of nodes defining the new requisite graph
Result: $\overline{\mathcal{E}}'\ldots$ a set of edges within the new requisite graph
1  begin
2      $\overline{\mathcal{N}}' = \overline{\mathcal{N}}$ ;
3      $\overline{\mathcal{E}}' = \overline{\mathcal{E}}$ ;
4      $C'_P = C_P \wedge \Pi_\alpha$ ;
5      $n = createActionNode(\alpha)$ ;
6      $\overline{\mathcal{N}}' = \overline{\mathcal{N}}' \cup n$ ;
7      foreach $n_s \in N_s$ do
8          $\overline{\mathcal{E}}' = \overline{\mathcal{E}}' \cup \{\langle n_s, n, C'_P\rangle\}$ ;
9      end
10     $C_{newP} = \top$ ;
11     $N_g = \{n\}$ ;
12 end
```

backward reasoning. The unbinding of a variable cannot be undone as the value of the variable cannot be inferred reversely. The last case is the creation of a variable binding node. If such a node would be allowed in the reactive program, the robot may change the variable binding during the execution of the reactive program. Thus, break the initial intention of the program that a certain set of actions is executed with the same variable binding to perform a local recovery.

If the removed element should be processed the action is used to regress the condition which was gathered so far backward in time. The condition is used to specify which condition should be met after executing the action. Thus, through the regression, we have the condition which needs to hold to allow a local recover using the action.

Afterward, the algorithm iterates over all ingoing edges of the node. Thus, the algorithm covers every possibility the action should be invoiced. Within each iteration, the regressed condition is combined with the condition of the edge. Thus, adding the condition of the action which should be invoiced. If this condition is not a tautology, the parent node which is connected to this edge is added to the queue together with its condition. Thus, one creates a new rule to resolve the missing local recoveries for the focused action. After updating the queue, a rule is added to the reactive program. The rule is composed of the generated condition together with the action specified in the node. Please note that also the actions to set the finished fluent to false is used if the action is not the action we are focusing on.

Let's consider our example to show how the reactive program looks like which was generated. Let's consider we focus our attention to the action $\alpha_3$ the resulting reactive program would be as depicted in Listing 8.1. Where $\pi'_{\alpha_3}$ is the regressed formula of $\pi_{\alpha_3}$ using action $\alpha_2$. The listing shows that the

**Algorithm 16:** *GenerateReactive*

```
Data: $\overline{\mathcal{N}}$... a set of nodes defining the requisite graph
Data: $\overline{\mathcal{E}}$... a set of edges within the requisite graph
Data: $n$... the node of seed action in the graph
Data: $K$... the maximum number of back steps to consider
Result: $R$... a list of rules for a reactive program
Result: $GC$... the termination condition of the reactive program
begin
    $GC = FinishedAction$ ;
    $Q = \{\langle\top,\top,n,0\rangle\}$ ;
    while $Q \neq \emptyset$ do
        $\langle C, First, n, i\rangle = pop(Q)$ ;
        if $(i > K) \vee \neg(n$ isA $action)$ then
            continue;
        end
        $\alpha = actionOf(n)$ ;
        $\bar{C} = regress(\alpha, C)$ ;
        $i' = i + 1$ ;
        $ingoingEdge = \{\langle n', c\rangle | \langle n', n, c\} \in \mathcal{E}\}$ ;
        foreach $\langle n', c\rangle \in ingoingEdge$ do
            $C' = \bar{C} \wedge c$;
            if $\bar{C} \neq \top$ then
                $Q = add(Q, \langle C', \bot, n', i'\rangle)$ ;
            end
            if $First$ then
                $\delta = \alpha$
            else
                $\delta = \alpha; SetFinishedActionFalse$
            end
            $R = add(R, \langle C', \delta\rangle)$ ;
        end
    end
end
```

program would react to a fault of action $\alpha_3$ by executing either $\alpha_3$ again if the precondition holds or the robot would back step to action $\alpha_2$ if the condition of this action is still fulfilled.

Listing 8.1: Reactive program generated by Algorithm 16 using the requisite graph depicted in Figure 8.2 for action $\alpha_3$

$\pi_{\alpha_3} \rightarrow \alpha_3$
$\pi'_{\alpha_3} \wedge \neg\phi \wedge \pi_{\alpha_1} \rightarrow \alpha_1;\mathit{SetFinishedActionFalse}$

By replacing every action in the original program by the reactive program generated by Algorithm 16 we create a program which allows a local recovery. Thus, the robot can react to faults during the action execution. However, as we argued above the program should behave the same if no fault occurred. The same behavior should be ensured as this program behavior is well tested or even verified to be correct. Through the construction of the reactive program using the requisite graph, we can state that the program does not alter its behavior if it is executed without an action fault. This result is formally stated in the following lemma.

**Lemma 20.** *If an IndiGolog program $\delta$ allows a sequence of actions to be executed after each other than the same program, where the execution of an action is replaced by the reactive program generated through Algorithm 16, can perform the same action sequence.*

*Proof (Sketch).* This lemma holds as a direct result of Theorem 4 and the fact that the first rule in the reactive program is the action which should be executed. □

## 8.2. Generation of Supervision Sensing Actions

Besides a reactive program which allows the robot to react to faults during the action execution, the robot needs to perform proper sensing actions to detect this faults in the first place. As we argued above this specification can be cumbersome. Thus, instead of a manual specification of these sensing actions, we show a method which allows deriving automatically proper sensing actions.
To derive the proper sensing action, one can either use only the specifications which are provided in the basic action theory. Alternatively, one can use additionally the specification which is provided in the history based diagnosis. We will show how to derive sensing actions for both cases.
If only be specifications of the basic action theory are used we have only the definition of the precondition of an action, the effect the action has and the formula which is imposed by a sensing action. We can use this information to derive that a sensing action $\alpha_s$ can supervise another action $\alpha$ as follows. First we construct a new basic action theory $\mathcal{D}^S = \mathcal{D}^* \setminus \mathcal{D}_{S_0} \cup \mathcal{D}'_{S_0} \cup \exists \bar{x}.\Pi_{\alpha_s}(do(\alpha(\bar{x}), S_0))$, where $\mathcal{D}'_{S_0} = \Pi_\alpha(S_0)$. Thus, the basic action theory is used to ensure that the action $\alpha$ can be executed and afterwards also the sensing action $\alpha_s$. If this basic action theory is inconsistent $\mathcal{D}^S \models \perp$ then the sensing action cannot be executed after the action $\alpha$. If the theory is consistent we can perform the following checks to determine if the sensing action $\alpha_s$ is of use. We can use the sensing action if one of the following conditions hold:

- $$\mathcal{D}^S \models \exists \bar{x}, \bar{y}.\Pi_{\alpha_s}(do(\alpha(\bar{x}), s)) \wedge SF(\alpha_s(\bar{y}), do(\alpha(\bar{x}), S_0)) \tag{8.1}$$

- $$\mathcal{D}^S \models \exists \bar{x}, \bar{y}.\Pi_{\alpha_s}(do(\alpha(\bar{x}), s)) \wedge \neg SF(\alpha_s(\bar{y}), do(\alpha(\bar{x}), S_0)) \tag{8.2}$$

Through this definition, the action $\alpha$ implies a certain truth value of $\alpha_s$ as sensing result. Thus, one can use the sensing action as supervision action as there is a defined value the sensing result should have.
Although this method allows deriving supervision actions, it may be too demanding. As it only uses an action if the truth value of the sensing outcome is predicted by the action only. Thus, the action $\alpha$ needs to change the fluents utilized in the sensing formula of $\alpha_s$ without any assumptions made for the initial situation. This condition strongly restricts the set of sensing actions to be used as supervision actions.
To allow more sensing actions to be used we can remove some of these demands. Instead of imposing that the action determines the truth value of the sensing outcome we check if the action changes the truth value of the sensing outcome. This can be done with the help of the following basic action theory. $\mathcal{D}^{S_2} = \mathcal{D}^* \setminus \mathcal{D}_{S_0} \cup \mathcal{D}'_{S_0} \cup \exists \bar{x}.\Pi_\alpha(do(\alpha_s(\bar{x}), S_0)) \cup \exists \bar{x}, \bar{y}.\Pi_{\alpha_s}(do(\alpha(\bar{y}), do(\alpha_s(\bar{x}), S_0)))$, where $\mathcal{D}'_{S_0} = \Pi_{\alpha_s}(S_0)$. Thus, the basic action theory asserts that the sensing action can be performed. Additionally, the action can be performed after the sensing action. And the sensing action can be executed again after the action. As in the previous case if the basic action theory is inconsistent $\mathcal{D}^{S_2} \models \bot$ the sensing action cannot be used. To check if the sensing action can now be used both following needs to hold:

-
$$\begin{aligned}\mathcal{D}^{S_2} \cup \exists \bar{x}.SF(\alpha_s(\bar{x}), S_0) \models \exists \bar{x}, \bar{y}, \bar{z}.\Pi_{\alpha_s}(do(\alpha(\bar{y}), do(\alpha_s(\bar{x}), s))) \wedge SF(\alpha_s(\bar{x}), S_0) \wedge \\ \neg SF(\alpha_s(\bar{z}), do(\alpha(\bar{y}), do(\alpha_s(\bar{x}), S_0)))\end{aligned}$$

-
$$\begin{aligned}\mathcal{D}^{S_2} \cup \exists \bar{x}.\neg SF(\alpha_s(\bar{x}), S_0) \models \exists \bar{x}, \bar{y}, \bar{z}.\Pi_{\alpha_s}(do(\alpha(\bar{y}), do(\alpha_s(\bar{x}), s))) \wedge \neg SF(\alpha_s(\bar{x}), S_0) \wedge \\ SF(\alpha_s(\bar{z}), do(\alpha(\bar{y}), do(\alpha_s(\bar{x}), S_0)))\end{aligned}$$

The condition states that the action $\alpha$ changes the expected truth value of the sensing action $\alpha_s$. Thus, if we perform the sensing action before and after the action was executed one can determine if the action was executed successfully. This allows more sensing actions to be used to supervise the action executions.
As the supervision actions are used to detect if a fault has occurred the above methods may perform sensing actions which are not necessary to detect a fault. However, as we have assumed that no information is provided about the faults an action can have we were not able to restrict the sensing actions further to those which are useful. If we have a specification of how the action can fail instead, we can rule out unnecessary sensing actions. We can use the information provided in the history based diagnosis to specify the action fault. In history based diagnosis we use the predicate $Var(\alpha, \alpha', \theta)$ to state that action $\alpha'$ is a variation of action $\alpha$ under the condition $\theta$.
We can use the information provided to determine the sensing actions to be used more closely with the help of the following basic action theory $\mathcal{D}^{S_3} = \mathcal{D}^* \setminus \mathcal{D}_{S_0} \cup \mathcal{D}'_{S_0} \cup \exists \bar{x}.\Pi_{\alpha_s}(do(\alpha(\bar{x}), S_0)) \wedge \Pi_{\alpha_s}(do(\alpha'(\bar{x}), S_0))$, where $\mathcal{D}'_{S_0} = \Pi_\alpha(S_0) \wedge \Pi'_\alpha(S_0) \wedge \theta(S_0)$. The basic action theory states that both actions $\alpha$ as well as the variation $\alpha'$ can be executed. Additionally, the condition holds which allow to replace action $\alpha$ with action $\alpha'$. Finally, the sensing action $\alpha_s$ can be executed regardless if action $\alpha$ or action $\alpha'$ was executed. If the basic action theory is inconsistent $\mathcal{D}^{S_3} \models \bot$ the sensing action cannot be used. We can now determine if a sensing action can be used to detect the fault if one of the following conditions hold:

- $$\begin{aligned}\mathcal{D}^{S_3} \models \exists \bar{x}, \bar{y}. \Pi_{\alpha_s}(do(\alpha(\bar{x}), S_0)) \land \Pi_{\alpha_s}(do(\alpha'(\bar{x}), S_0)) \land \theta(S_O) \land \\ SF(\alpha_s(\bar{y}), do(\alpha(\bar{x}), S_0)) \land \neg SF(\alpha_s(\bar{y}), do(\alpha'(\bar{x}), S_0))\end{aligned}$$

- $$\begin{aligned}\mathcal{D}^{S_3} \models \exists \bar{x}, \bar{y}. \Pi_{\alpha_s}(do(\alpha(\bar{x}), S_0)) \land \Pi_{\alpha_s}(do(\alpha'(\bar{x}), S_0)) \land \theta(S_O) \land \\ \neg SF(\alpha_s(\bar{y}), do(\alpha(\bar{x}), S_0)) \land SF(\alpha_s(\bar{y}), do(\alpha'(\bar{x}), S_0))\end{aligned}$$

The condition states that if the truth value of the sensing result is determined to be different depending on if the fault has happened or not the sensing action is of use. This condition allows that through the sensing action one can decide if the fault has occurred or not. Such a sensing action is perfectly suitable to perform a supervision. Unfortunately, we place a high demand on the relation between action and sensing. The action needs regardless of the initial situation determine the truth value of a sensing outcome. Thus, one may not expect to find a sensing action for each fault.
We can reduce the demand if we only expect that the change of the expected sensing outcome differs between the action and a fault. We can find such a sensing action with the help of the following basic action theory $\mathcal{D}^{S_4} = \mathcal{D}^* \setminus \mathcal{D}_{S_0} \cup \mathcal{D}'_{S_0} \cup \exists \bar{x}. \Pi_\alpha(do(\alpha_s(\bar{x}), S_0)) \land \Pi_{\alpha'}(do(\alpha_s(\bar{x}), S_0)) \land \theta(do(\alpha_s(\bar{x}), S_0)) \cup \exists \bar{x}, \bar{y}. \Pi_{\alpha_s}(do(\alpha(\bar{y}), do(\alpha_s(\bar{x}), S_0))) \land \Pi_{\alpha_s}(do(\alpha'(\bar{y}), do(\alpha_s(\bar{x}), S_0))) \land \theta(do(\alpha_s(\bar{x}), S_0))$, where $\mathcal{D}'_{S_0} = \Pi_{\alpha_s}(S_0)$, where $\mathcal{D}'_{S_0} = \Pi_{\alpha_s}(S_0)$. The basic action theory states the sensing action can be performed in the initial situation and after executing the sensing action the action of interest or its faulty variant can be executed. Additionally, the condition to exchange these two actions is fulfilled as well as the sensing action can be executed regardless of the performed actions before. As in the previous cases if the basic action theory is inconsistent $\mathcal{D}^{S_4} \models \perp$ the sensing action cannot be used. If the basic action theory is consistent we can determine that the sensing action can be used if both following conditions hold:

- $$\begin{aligned}\mathcal{D}^{S_4} \cup \exists \bar{x}. SF(\alpha_s(\bar{x}), S_0) \models \exists \bar{x}, \bar{y}, \bar{z}. \Pi_{\alpha_s}(do(\alpha(\bar{y}), do(\alpha_s(\bar{x}), s))) \land \Pi_{\alpha_s}(do(\alpha'(\bar{y}), do(\alpha_s(\bar{x}), s))) \land \\ \theta(S_0) \land SF(\alpha_s(\bar{x}), S_0) \land \\ \neg SF(\alpha_s(\bar{z}), do(\alpha(\bar{y}), do(\alpha_s(\bar{x}), S_0))) \land \\ SF(\alpha_s(\bar{z}), do(\alpha'(\bar{y}), do(\alpha_s(\bar{x}), S_0)))\end{aligned}$$

- $$\begin{aligned}\mathcal{D}^{S_4} \cup \exists \bar{x}. \neg SF(\alpha_s(\bar{x}), S_0) \models \exists \bar{x}, \bar{y}, \bar{z}. \Pi_{\alpha_s}(do(\alpha(\bar{y}), do(\alpha_s(\bar{x}), s))) \land \Pi_{\alpha_s}(do(\alpha'(\bar{y}), do(\alpha_s(\bar{x}), s))) \land \\ \theta(S_0) \land \neg SF(\alpha_s(\bar{x}), S_0) \land \\ SF(\alpha_s(\bar{z}), do(\alpha(\bar{y}), do(\alpha_s(\bar{x}), S_0))) \land \\ \neg SF(\alpha_s(\bar{z}), do(\alpha'(\bar{y}), do(\alpha_s(\bar{x}), S_0)))\end{aligned}$$

The condition states that the action changes the expected sensing outcome whereas the faulty version of the action does not. Thus, one can detect the fault if one performs the sensing action $\alpha_s$ before and after the action $\alpha$. This method allows us to use more sensing actions as the previous method but still allow a detailed detection.
The above condition used the fact that the action has changed the expected sensing outcome but the faulty version does not. One can also use the opposite direction of imposed change to determine if a sensing action can be of use. A sensing action can detect a faulty if the both following conditions hold:

- $$\begin{aligned}\mathcal{D}^{S_4} \cup \exists \bar{x}.SF(\alpha_s(\bar{x}),S_0) \models \exists \bar{x},\bar{y},\bar{z}.&\Pi_{\alpha_s}(do(\alpha(\bar{y}),do(\alpha_s(\bar{x}),s))) \wedge \Pi_{\alpha_s}(do(\alpha'(\bar{y}),do(\alpha_s(\bar{x}),s)))\wedge\\ &\theta(S_0) \wedge SF(\alpha_s(\bar{x}),S_0)\wedge\\ &SF(\alpha_s(\bar{z}),do(\alpha(\bar{y}),do(\alpha_s(\bar{x}),S_0)))\wedge\\ &\neg SF(\alpha_s(\bar{z}),do(\alpha'(\bar{y}),do(\alpha_s(\bar{x}),S_0)))\end{aligned}$$

- $$\begin{aligned}\mathcal{D}^{S_4} \cup \exists \bar{x}.\neg SF(\alpha_s(\bar{x}),S_0) \models \exists \bar{x},\bar{y},\bar{z}.&\Pi_{\alpha_s}(do(\alpha(\bar{y}),do(\alpha_s(\bar{x}),s))) \wedge \Pi_{\alpha_s}(do(\alpha'(\bar{y}),do(\alpha_s(\bar{x}),s)))\wedge\\ &\theta(S_0) \wedge \neg SF(\alpha_s(\bar{x}),S_0)\wedge\\ &\neg SF(\alpha_s(\bar{z}),do(\alpha(\bar{y}),do(\alpha_s(\bar{x}),S_0)))\wedge\\ &SF(\alpha_s(\bar{z}),do(\alpha'(\bar{y}),do(\alpha_s(\bar{x}),S_0)))\end{aligned}$$

Thus, this condition makes it possible to use sensing action which detects a faulty outcome directly. Furthermore, the expected sensing value is only changed if a fault occurs thus this sensing action would not have been chosen with any other method presented so far. To use the sensing action one performs the action $\alpha_s$ before and after the execution of $\alpha$. Allowing to detect if the sensing outcome has changed.

## 8.3. Related Research

Before we conclude the chapter, we will discuss related research which is specific to this chapter.
To supervise the execution of a generated plan through STRIPS [43] a method was proposed in [42] which used a so-called triangle table. The table is used to specify for each part of the plan its precondition to achieve the remaining of the plan as well as the effect of this section of the plan. This table can be used for supervision of the plan. However, it can be furthermore used to react to a plan failure. The table is constructed in such a way that the columns are used to state the execution of an action and the rows specify the effect each action has over the execution time. One can now use a rectangular of the i-th row and column to specify which condition needs to hold such that the execution of the i-th action will lead to the goal. During the execution, one checks each of this rectangular to determine which action should be executed next. The order to choose an action for execution is to start checking backward from the last to the first action. Thus, the robot works towards the goal even in case the knowledge has changed. One can consider this method as a basis of our method. Through the use of regression, we have generalized which action can be employed. Thus, we do not restrict the actions to be STRIPS actions. Furthermore, we have extended the approach to allow conditions within the programs thus we allow a reactive behavior about a program instead of just a linear plan.
A similar approach to model the program as a graph was presented in [15]. The method use *"characteristic graphs"*[15] to represent the program in such a way that it allows a label propagation algorithm to determine if a temporal logical formula holds. The graph consists of nodes which have no additional information and edges which are defined through the following pattern $\pi\vec{x} : \alpha/\phi$. Which is interpreted as follows: find a binding for the variables $\vec{x}$ such that $\phi$ holds and perform $\alpha$ afterward. Thus, these graphs are a compact variant to represent a program compared to our requisite graph. We use this more verbose version of representation as it simplifies the algorithm to create a reactive program, as regression and checking for a condition can be separated. Furthermore, we can easily determine when a variable is bound or not. This determination also allows a straightforward interpretation of the requisite graphs.

## 8.4. Conclusion and Future Work

Robots acting in a real environment need to deal with faulty action execution. As such the robot needs to incorporate the effect a fault has in its belief. Furthermore, the robot needs to take the decision which action to perform according to its belief. However, besides the incorporate of the effect a fault has the robot needs also a reactive program which allows reacting to changes in the belief.
In this chapter, we have presented a method which allows generating a reactive program out of a sequential program. This generation allows using already defined sequential programs for the robot control without losing the ability of the robot to react to changes. The method is general enough that the program can contain conditional branching as well as loops. Also, the programmer can use nondeterministic branching and non-deterministic choice of arguments in the program. Such construct can often be found if one programs a robot with the help of IndiGolog or a similar high-level control language.
Besides a reactive program, the robot also needs to detect the fault in the first place. To perform such a detection, the robot needs to perform a sensing action which checks if the previously performed action was successfully executed. In this chapter, we have shown that such sensing actions can be automatically derived by using the basic action theory. If additionally, the information how an action fails is reused from the history based diagnosis more accurate sensing actions can be derived.
The presented method to create a reactive program is restricted to sequential execution only. Thus, it would be of interest to extend the approach to concurrent execution. Additionally, the approach has not defined how to treat planning within the high-level program. To prolong the approach to deal with planning would thus be also of interest for future work.
Finally, the derivation of the sensing actions is done without considering the program around the performed action. If one would use this information one may reduce the sensing actions carried out to detect a fault. It could also lead to the use of other sensing actions as some sensing actions only detect a fault if a certain sequence of actions was performed before.

Chapter 9

# Practical Realization

In the previous chapters, we have shown how to create a high-level control for a robotic system which considers faulty action executions. The extension of the high-level control of the robot in the previous chapters focused on theoretical definitions and their resulting theoretical properties. Before we discuss the effect of the proposed methods for the high-level control of the robot in the next chapter using an evaluation, we will discuss in this chapter some considerations for a practical implementation of the proposed methods.
The theoretical foundations of the high-level control are based on several definitions using first and second order logic. Although this definition allows a direct realization of a reasoning system, such as Prolog [16]*, One can exploit the structure of the definitions to gain efficiency. To do so, we split the high-level control into several parts which interact with each other through well-defined interfaces. Each of this parts tackles one specific problem for the high-level control and exploits the structure of the specific issue.

## 9.1. Software Architecture

We split the implementation of the high-level control of a robotic system into four main parts. The different components and how the different parts are used is depicted in Figure 9.1. The *Execution* is concerned with the execution of the high-level program given the current situation. This part of the system implements the transition semantics described in Chapter 6. To answer queries for the execution the *Belief* of the robot is used. The *Belief* interface allows answering a query for a given situation. It hides the details how the reasoning is performed to respond to the query for the rest of the system. A more advanced method to form the belief of the robot is by using a *Belief management* component. The *Belief management* implements the proposed methods in this thesis to manage the belief of the robot to deal with faulty action execution. As the semantics of the belief management uses a possible world semantics the *Belief management* uses itself a *Belief* component to answer a query for each situation which is a possible world. Thus, one can exploit the advances for a belief also in the belief management without additional overhead. Finally, to answer a query the *Belief* uses an automatic *Theorem prover*. The *Belief* defines how the logical theory is composed. Whereas the *Theorem prover* is used to derive the answer for the query after constructing the logical theory. Due to this splitting, one can simply exchange the automatic *Theorem prover* with another implementation

*With some limitations as no full first and second order formulas can be achieved.

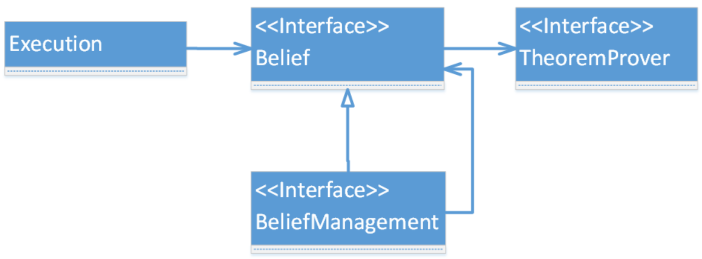

Figure 9.1.: Architecture of the implementation for a robotic system with advanced belief management.

to exploit advances in automatic theorem proving.

## 9.2. Theorem Proving

The first part we want to discuss in the practical realization is the automatic theorem prover. The interface allows specifying a logical theory consisting of first order formulas, as well as, inductive definitions of data types. The latter is used to define a situation term. It is important to notice that we only ask queries with a fixed situation term. Thus, one can either use the inductive definition of situation data type or perform an unrolling of the definition. Additionally, to the specification of a theory, one can use the automatic theorem prover to check if a query logically follows from the specified theory or not.
For the practical realization, three aspects need to be considered for the query answering. First, the queries are first order thus the formulas may contain quantifier which needs to be treated by the solver efficiently. The second one uses the theory to check if a formula follows. Thus, one is not interested in the consistency of a theory, but one is interested in the formulas one can logically derive from a given theory. The focus on logically derived formulas imposes that the solver should consider to answer the query instead of checking for consistency of the complete, probably big, theory. The third aspect which needs to be considered is the inductive definition of a situation. This definition imposes that first one can define the situation properly, which is not possible in first order, and additionally one needs to be able to perform an efficient reasoning on this inductive definition.
The first theorem prover we consider is based on Prolog [16]. Prolog allows answering a query by unification and the use of a closed world assumption. Through the unification process, one can simply represent an inductive definition. However, Prolog does not allow to use first order formula directly. Instead one can use a quantification which is restricted to a finite domain. This restriction is sufficient for the most practical application. Although the specification of the theory and the query is simply in Prolog, it pointed out that for a practical implementation the treatment of quantified variables often imposes a limit. This limitation was evident if a formula does not hold the implementation based on Prolog was often slow compared to other methods.
As Prolog has a problem with quantification, one can use a theorem prover designed for first order. Thus, the second theorem prover we consider is SPASS [153]. SPASS is a first order theorem prover. Thus, the theorem prover is designed to deal with formulas which use quantification. As the inductive definition of the situation term cannot be simply defined in first order, one needs to unroll the inductive definition before the theorem prover is used. Although the theorem prover can natively handle quantified formulas, it pointed out that the query answering is slow compared to another method. The reason is that due to the unrolling of the inductive definition the theory which needs to be considered

grows significantly and thus makes the reasoning process complex.
To address the problem of quantified formulas as well as an inductive definition the third prover we consider is IDP3 [18]. IDP3 is a first order theorem prover which allows the native definition of inductive data types. Thus, the queries, as well as the inductive definition, don't need special treatment. Although the theorem prover offers all features which are required it pointed out that the time to answer a query is slow compared to the last approach.
The last theorem prover we consider is Z3 [28]. Z3 is a sat solver which allows specifying quantified formulas as well as inductive data types. To answer a query with the help of Z3 one needs to convert the query problem into a sat problem, and checks afterward for satisfiability to respond if the query holds. As the theory may contain several parts which are not related to the query but would be verified during the sat solving one can reduce the theory first only to contain those formulas which specify predicates which are used within the query or formulas which specify predicates which are used for other formulas linked to the query indirectly. Thus, one first finds the minimal set of formulas which contains all predicates defined in the query. Additionally, the set contains all formulas which specify a predicate within the set. Thus, the reduced theory is equivalent in answering a query if one considers that the theory is consistent. Although Z3 allows inductive definitions of data types, it pointed out that the unrolling of the inductive definitions yield better performance. Thus, one first unrolls the definitions afterward reduce the theory and finally convert the query to a sat problem to answer the query.

## 9.3. Belief of a Robot

With the help of the theorem prover, one can answer a query for a given theory. To create the theory properly the implementation of the belief is used. The first logical representation of the belief used uses the definition of the situation calculus to define the theory. Although this theory can be simply created it pointed out that it is slower than the other methods to form a belief.
To address the problem of the situation term, one can replace the situation term through a time point. Furthermore, one defines for each time point which action is performed. This replacement of the situation term with a time point is a simple mapping from the situation calculus to a specific case of the event calculus. Due to this mapping one uses an integer instead of the situation term. Although this does not have a theoretical benefit, it turned out that the automatic theorem prover can better handle the integer then the situation term. Thus, through this mapping, the time to answer a query can be reduced.
The situation calculus using the situation term as well as time points has the problem that the reasoning process needs to reason backward in time to answer a query. This backward reasoning imposes that the reasoning process gets slower if the history of performed actions gets longer. To avoid this relation, one can use progression to have a knowledge base representing the current state of the world. It is important to notice that for the belief management one still uses a history of performed actions to answer a query. However, instead of using the history of performed actions to respond the query one uses the history as an index for the knowledge bases describing the state of the world specified through the history. If no state is associated with the history used in the query one first steps back in the history till a saved world state is retrieved. Afterward, one performs progression in step wise manner for each action and saves the result for future use. This method also allows reusing the knowledge bases which are created during planning within the execution of the program. To perform the progression a progression with a relative complete knowledge base is implemented. This progression method allows using arbitrary successor state axioms. The progression is done by replacing each fluent term at the

right-hand side by definition in the knowledge base. This replacement creates a new knowledge base which can be used for the query answering. The important property of a relative complete knowledge base is that it only contains equality constraints to define a fluent. Thus, by replacing the fluents within a query and resolving equality constraints, often the query can be simply answered without any call to the theorem prover. If the theorem prover needs to be invoked one needs only to consider the query as the theory is already used for the replacement process. This approach allows answering queries in a timely fashion.
Although through the usage of Z3 and progression acceptable timings could be achieved it is left for future work to find a better representation to achieve faster query times for the belief. Especially the inductive definition of a situation may give rise to a more rapid implementation. Also, the structure of the action effects could be exploited to lead to a faster implementation.

## 9.4. high-level Execution

The final part we consider in the practical realization is the execution of the high-level programs. The semantics of the execution is defined through the first order and second order formulas defined in Chapter 6. The definitions use four predicates to define if a transition is possible or if the program can be safely terminated. Instead of using these four predicates and perform a reasoning we use the AST [1] of the program for interpretation. Like the method presented in [75], we execute a program by traversing its AST. This traversal allows to efficiently decide which program query needs to be executed to decide for a change or termination. Additionally, many program constructs such as an if or while statement can be realized by native statement of the programming language thus allow a fast and sound interpretation.

Chapter 10

# Empirical Evaluation

In the previous chapters, we have discussed different methods to improve the original belief management of a robot. Ranging from methods which make it easier to use the belief management, to the integration of the belief management into the high-level control of the robot. Throughout these chapters, we have motivated why the presented method improve the overall belief management. However, for most of the chapters, we have not confirmed this improvement through an evaluation.
To address this, lack of conformance we will show the results of different evaluations indicating the benefits of the improvements presented in this thesis. Furthermore, as the methods present in this thesis extends the basic belief management, we will first show the positive impact a belief management has on a robotic system.
To allow to compare different approaches in the same environment we evaluated a simulation. This comparison also enabled us to evaluate many different approaches which would be otherwise too time-consuming. All the performed evaluation uses the running example of the thesis for the task the robot must perform. Some of the assessment results presented in this chapter were discussed in [95], [132] and [93].
The remaining of the chapter is structured as follows in the next section we will show an evaluation comparing a robotic system using the classical belief management with a robotic system which does not use a belief management approach. As this belief management is the basis for our extensions, this evaluation shows the basic improvement a robot has through the belief management. All other assessments in this chapter will show the further improve the robotic system through the methods presented in this thesis. In Section 10.2 we evaluate the difference between the classical belief management and the advanced belief management. The advanced belief management uses the integration of the belief into the high-level program execution as it was discussed in Chapter 6 together with the method proposed in Chapter 7 to gather knowledge. As we have argued in Chapter 6 that progression is more efficient in a robotic system than direct reasoning we perform an evaluation which shows this improvement in Section 10.3. Next, we evaluate the difference between the use of classical history based diagnosis to consistency based diagnosis for the belief of the robot. This evaluation is discussed in detail in Section 10.4. The last evaluation shows the impact the generation of a reactive program has. This evaluation uses the method described in 8 to create a reactive program. Afterward, we discuss the impact of the use of common sense in Section 10.6. Finally, we will conclude the chapter.

## 10.1. Comparison of a Robot with and without Belief Management

The first statement we want to check with the help of evaluation is if the use of a belief management approach does improve the robot's performance. As we argued in the introduction and throughout the thesis, a robot which is aware of the faults during the action execution is more successful. Thus, a robot which uses a belief management should, therefore, finish a task with a higher success rate than those which don't use a belief management. With the evaluation presented in this section, we want to confirm this hypothesis. The assessment presented in this section is based on the results presented in [95].
To perform the evaluation, we first state the hypothesis which should be checked through this assessment.

**Hypothesis 1.** *A robotic system using a belief management has a higher success rate in executing a task than a robotic system which does not use a belief management.*

As the evaluation should compare a robotic system with a belief management and one without a belief management we need to different systems. As a robotic system, which uses a belief management we use the method presented in [51] which was also used to evaluate a real robotic system [52]. For the robotic system, which does not use a belief management we use the Cognitive Robot Abstract Machine (CRAM) [7]. We chose CRAM as it has shown to be capable of performing a non-trivial task in household environment [6]. Thus, the system has the capabilities to perform reasoning and planning. Additionally, CRAM uses KnowRob [136] to specify the knowledge of the robot. KnowRob allows diverse knowledge to be combined into one knowledge base. It also features the possibility to perform different reasoning methods to derive plans, objects fitting a description or even most likely locations of objects. Additionally, KnowRob and therefore also CRAM may become a significant standard method through its integration into the RoboEarth project [151].

### 10.1.1. Task and Environment

With the hypothesis, we want to check and the two approaches we want to compare outlined we need to define which task the robot must perform the evaluation. As we use the running example of the thesis (see 1.4 for further details) the robot carries out a simple delivery task. The robot needs to serve a set of delivery requests. Each application defines an object which should be delivered and a destination where the object should be delivered to. As the robot first needs to pick the object, we assume the robot has complete knowledge of the initial situation. Thus, the robot knows at the beginning where each object is located. Please note that the knowledge the robot has about the environment can get inconsistent as the action execution is possibly faulty.
To confirm the hypothesis, we perform an evaluation consisting of three different scenarios. The number of requests is different for the different scenarios and is either 3 or 9. The robot has finished the task with success if it has delivered all objects to their destination. Additionally, the number of rooms and objects differ in each scenario. Through this difference in the scenarios, the evaluation also gives an inside about how well the different approaches scale. The various scenarios with the number of rooms used and the number of objects to deliver are depicted in Table 10.1. The environment the robot is performing the delivery is illustrated in Figure 10.1. Please note that the same nine objects are always present in the environment, but only a subset of these nine objects may need to be delivered.
As we simulate the robot to not always succeed to execute an action each action has a certain probability to fail. The probabilities for the different faults which may occur are depicted in Table 10.2. The probabilities are the same for all three scenarios. Thus, the chance that the robot performs a task

| scenario | # rooms | # objects |
|---|---|---|
| S1 | 20 | 3 |
| S2 | 29 | 3 |
| S3 | 20 | 9 |

Table 10.1.: Number of rooms and objects for the different scenarios of the first evaluation.

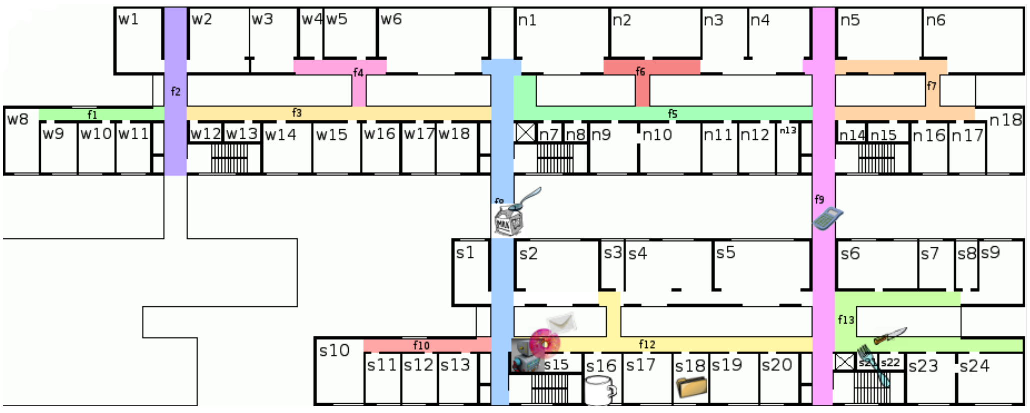

Figure 10.1.: Domain for scenario S2 of the first evaluation. The layout resembles the institute's floor plan.

for scenario S3 without any fault is highly unlikely. A similar set of faults were also considered in [87] and in [52] thus one can consider this faults reassemble the behavior of a real robot.

| fault | probability |
|---|---|
| goto to wrong location | 0.05 |
| pick up wrong object | 0.2 |
| pick fails | 0.2 |
| put fails | 0.3 |
| random movement of object | 0.02 |

Table 10.2.: Probabilities for fault of actions and the exogenous event.

### 10.1.2. The compared approaches

As we have discussed above briefly, we compare two robot systems performing the delivery task. The first system does not use a belief management. This system will use CRAM to command the robot. The second system uses the classic version of the belief management as it was presented in 2.2. In the following, we will discuss different high-level programs for the first system and afterward the second system which is used in the evaluation.

#### Cognitive Robot Abstract Machine (CRAM)

The first robotic system uses CRAM to control the robot. CRAM uses the CRAM Plan Language (CPL) to define the high-level control program of a robot. CPL allows describing a task through a

given goal and its parameters. These parameters can be refined for a given task. This refinement is performed through reasoning on the used knowledge base of the robot. Besides the description of a task, CPL also allows specifying how a task can be achieved through a program allowing sequential a parallel execution. Additionally, one can use a dedicated monitoring method [87] to check for the successful task execution. However, it is important to notice that this monitoring does not perform a diagnosis to reason about what went wrong thus it can only be used to check if something is wrong.
Due to the flexibility, CPL offers one can use different control programs to achieve the task. Thus, we will discuss three different control programs. Each differs in the way it deals with the faults which might occur. Please note that the fault handling was hard coded thus it has only a limited flexibility to react to a fault. Furthermore, one can assume that it is not feasible for more complex tasks to perform a fault handling in such a way.
The first program (CRAM 1) we use for the execution with CRAM is depicted in Listing 10.1. The program performs the main function a simple loop which iterates till all requests are served. For each request, the subroutine *handleRequest* is used. The subroutine *handleRequest* performs a simple sequential program to deliver the object. The program first moves the robot to the location of the object. Afterward, it picks the object moves the robot to the destination of the delivery. Finally, it puts the object down. Thus, this program can be considered as the solution which solves the delivery problem in the simplest way possible. One can easily check that this program will deliver the objects successfully if no fault occurs. However, one can consider that the program will fail if an action is executed in a faulty way. Thus, we consider this program a base line.

Listing 10.1: Simple sequential program (CRAM 1)

```
main() {
  while(!requests.isEmpty()) {
    handleRequest(requests.pop())
  }
}
handleRequest(req) {
  goto(place(req.object))
  pick(req.object)
  goto(req.destination_place)
  put(req.object)
}
```

The second program (CRAM 2) is depicted in Listing 10.2. It incorporates a first method to deal with faults. The program performs in the main routine a loop which iterates till all requests are served. In each iteration, one request is picked, and the robot tries to serve this request. To serve the request the robot uses a modified version of the subroutine *handleRequest*. The subroutine *handleRequest* performs the same sequential execution to deliver the object as it was done in the case of CRAM 1. However, after the object was delivered the subroutine performs several sensing actions to confirm that the delivery was successful. The result of the sensing can indicate a fault which is reported back to the main loop. The main loop handles the fault by first trying to put the object down and afterward performs a search for the object to find the object. This search should ensure that if the object were not picked up in the first place, the robot would find the object. After performing the search, the robot has found the object and performs another try to serve the same delivery.

Listing 10.2: Program with error detection and recovery (CRAM 2)

```
main() {
  while(!requests.isEmpty()) {
    request = requests.pop()
    do{
        clearError()
        handleRequest(request)
        if(error) {
          put(request.object)
          searchForObject(request.object)
        }
      }while(error)
  }
}
handleRequest(req) {
  goto(place(req.object))
  pick(req.object)
  goto(req.destination_place)
  put(req.object)
  scanCarryAnything()
  scanForObject()
  if(carry_anything || object_not_found)
    setError()
}
```

The third program (CRAM 3) is depicted in Listing 10.3. The program performs the same main routine as CRAM 2. This routine also includes the same error recovery method. The difference between CRAM 2 and CRAM 3 is the subroutine *handleRequest*. The subroutine in CRAM 3 uses a Teleo Reactive program [98] to serve the delivery. The program performs an iteration of the set of rules till the top most rule was executed. In each iteration, the program checks the rule in a top down fashion. If the condition of the rule is satisfied the rule is triggered, and the associated action is executed. The program checks with the highest priority if the object was successfully delivered. This check is done if the object should be at the destination and object is not held. If the object is held, the robot puts down the object through the next rule. The following rule checks if the robot is still not at its destination and triggers that the robot moves to the destination place. The forth rule ensures that the robot picks up the object if it is in the same room and does not hold the object already. The last rule moves the robot to the object thus allowing all other rules to be applied. To perform the checks needed to decide if a rule can tiger the robot uses a simple knowledge base. It is updated according to the performed action. Furthermore, the sensing results override the action outcomes. If the sensing result is not consistent with the current knowledge, the error flag is set.

Listing 10.3: Teleo-Reactive subroutine to handle requests (CRAM 3)

```
main() {
  while(!requests.isEmpty()) {
    request = requests.pop()
    do{
        clearError()
        handleRequest(request)
        if(error) {
          put(request.object)
```

```
            searchForObject(request.object)
          }
        }while(error)
    }
}
handleRequest(req)
{
    [(place(robot) == req.destination_place) &&
        (place(object) == req.destination_place) &&
        not carry-anything] --> scanForObject()
    [(place(robot) == req.destination_place) &&
        (place(object) == req.destination_place)]
        --> { putdown(req.object)
            scanCarryAnything() }
    [(place(robot) == place(req.object) && carry_anything]
        --> goto(req.destination_place)
    [(place(robot) == place(req.object)] --> pickup(req.object)
    [true] --> goto(place(req.object))
}
```

**Classical Belief Management (HBD)**

As the second robotic system used in the evaluation, we use the classical belief management. The belief management uses the classical history based diagnosis to create the belief of the agent. One of the possible explanations which are very likely is used for further reasoning. To perform the execution IndiGolog is used which has an extended main loop to make the diagnosis after the execution of an action. We refer the interested reader to [109] for further details.
The program we use for the classical belief management is (HBD) is depicted in Listing 10.4. The program uses in the main routine a simple loop which iterates till all requests are served. In the loop, a request is picked through a nondeterministic choice of arguments from IndiGolog. After picking a request, the request is served by the same Teleo Reactive program as for CRAM 3. This program circumvents the problem which arises when the robot needs to adapt its belief due to a fault. After correcting the belief after a faulty action execution, the program loops again and detects those conditions which are true and execute the action. This reaction allows the robot to react to changes and still work towards the goal of delivering the object.

Listing 10.4: Main loop of the HBD program

```
main() {
    while($\exists r : request(r)$) {
        $\pi r : request(r)$
        $handleRequest(r)$
    }
}
handleRequest(req)
{
    [(place(robot) == req.destination_place) &&
        (place(object) == req.destination_place) &&
```

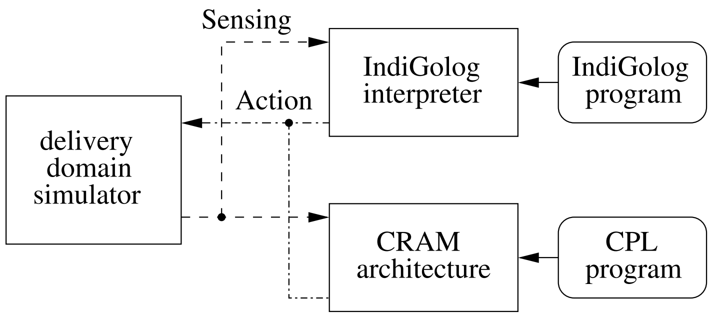

Figure 10.2.: System overview of the evaluation. Both control approaches use the same ROS interfaces for sending (dashed lines) and executing actions (chain dotted lines).

```
      not carry-anything] --> scanForObject()
   [(place(robot) == req.destination_place) &&
      (place(object) == req.destination_place)]
      --> { putdown(req.object)
         scanCarryAnything() }
   [(place(robot) == place(req.object) && carry_anything]
      --> goto(req.destination_place)
   [(place(robot) == place(req.object)] --> pickup(req.object)
   [true] --> goto(place(req.object))
}
```

#### Comparison framework

To allow a fair comparison both robotic systems were attached to the same simulation of the robot. An overview of the used evaluation system is depicted in Figure 10.2. The system consists of one ROS [107] node which simulates the robot. The simulation takes care of the effect an action has. Furthermore, the simulation performs a faulty action execution according to the specified probability. The node offers a service interface which is used by both robotic systems. This interface allows commanding a high-level action to be performed such as $goto(R_1)$. Additionally, the high-level sensing action can be conducted to retrieve their sensing result. A similar interface was used in [52] for the control of a real robot. To execute the program HDB we used the interpreter presented in [48] extended as discussed in [109]. To execute the programs, CRAM 1, CRAM 2 and CRAM 3 we used the public CRAM[*].

### 10.1.3. Experiments

We evaluated all three different scenarios with all the four presented high-level control programs[†]. For each scenario 30 different delivery tasks were randomly created. Each delivery task was tried five times with different seed values for the random number generated to trigger the faults. The robot had 3 minutes to execute all deliveries. During the task execution, the success of delivering all objects, the run time and if the robot execution ran into a timeout was recorded.

[*]We use the ROS-based open-source implementation. For details, we refer the interested reader to `http://www.ros.org/wiki/cram_core`.

The results of the first scenario S1 is depicted in Table 10.3. The robot had to deliver 3 objects within an environment consisting of 20 rooms. The table depicts the percentage of successful task executions. The run time is given in seconds additionally the percentage of executions which run into a timeout are given. The best three high level program are marked bold. These programs will be used for the further scenarios.

| program | success-rate/% | timeouts/% | average runtime/s | |
|---|---|---|---|---|
| | | | mean | std |
| CRAM 1 | 17.33 | 0 | 0.23 | 0.03 |
| **CRAM 2** | **70.67** | **0** | **3.32** | **9.60** |
| **CRAM 3** | **68.46** | **12.75** | **23.37** | **60.08** |
| **HBD** | **78.00** | **6.67** | **16.85** | **43.92** |

Table 10.3.: Performance of the different programs for the delivery task with 20 rooms and 3 objects (scenario 1).

The results of the first scenario indicate that the best three high-level programs use some error detection and recovery. These three approaches have a significantly higher success rate than CRAM 1 which does not perform a mistake recovery. The usage of CRAM 1 has the advantage that the execution is fast. Thus CRAM 1 would be the best choice if no fault would occur. This observation also indicates that a robot which performs an error recovery will spend more time to execute a task. This result is caused partly due to the additional sensing action. However, mostly due to the necessary reasoning to check for an error and to recover from this mistake.
Regarding the success rate, the best performing program was HBD. Thus, we get a first indication that through the usage of a belief management the success rate can be increased. The run time of HBD was higher than CRAM 2 but lower than CRAM 3 indicating that the performed reasoning of a TR program increases the run time significantly. However, it also shows that due to the focused changed of the belief of HBD the recovery after an error is faster done as through a TR program.
Finally, one can observe that CRAM 3 has a lower success rate than CRAM 2. This observation is interesting as both programs use the same error recovery the main reason for this discrepancy is due to the high run time of CRAM 3. Thus, the additional reasoning performed through a TR program causes the significant higher run time which is also indicated by the higher time out percentage. This higher time out percental indicates that a reactive program does not increase the success rate of a robot if an error recovery is performed.
The results of the second scenario S2 are depicted in Table 10.4. The robot had to deliver 3 objects within an environment consisting of 29 rooms. The table depicts the percentage of successful task executions. The run time is given in seconds additionally the percentage of executions which run into a timeout are given. The best high level program is marked bold. Please note that only the best three performing programs of S1 are used in the comparison.
The evaluation of the second scenario confirms that HBD is the best performing high-level control program. Thus, also this scenario indicates that the use of a belief management is preferable. The increase in run time is moderately stating that the approach scales well with the increasing environment. As in the first scenario also the second scenario shows that the run time of CRAM 3 is very like those of HBD. Thus, indicating the higher run time is caused through the reactive program. Finally, it is important to mention that the same conclusion can be drawn that a TR program alone does not increase the success rate if the program performs a fault recovery.
The results of the third scenario S3 are depicted in Table 10.5. The robot had to deliver 9 objects

| Program | success-rate/% | timeouts/% | average runtime/s | |
|---|---|---|---|---|
| | | | mean | std |
| CRAM 2 | 86.00 | 0 | 1.90 | 6.35 |
| CRAM 3 | 82.43 | 6.76 | 24.90 | 90.51 |
| **HBD** | **88.67** | **2.67** | **18.84** | **60.26** |

Table 10.4.: Performance of the different programs in a delivery task with 29 rooms and 3 objects (scenario 2).

within an environment consisting of 20 rooms. The table depicts the percentage of successful task executions. The run time is given in seconds additionally the percentage of executions which run into a timeout are given. The best high level program is marked bold. Please note that only the best three performing programs of S1 are used in the comparison.

| Program | success-rate/% | timeout/% | average runtime/s | |
|---|---|---|---|---|
| | | | mean | std |
| CRAM 2 | 32.00 | 0 | 15.55 | 27.55 |
| CRAM 3 | 20.57 | 51.77 | 187.13 | 179.75 |
| **HBD** | **49.33** | **27.33** | **113.97** | **151.84** |

Table 10.5.: Performance of the different programs in a delivery task with 20 rooms and 9 objects (scenario 3).

The third scenario shows that the HBD outperforms the other approaches. Thus, also in this scenario, a belief management is preferable. This observation is of particular significance as the HBD had a high time out. Thus, one could expect that the robot would perform even better if it had more time to recover from a fault. The drastically increased time out ratio is also observable for CRAM 3. Thus, indicating that a TR program may take significantly longer if faults occur more often. Also, the reasoning takes longer due to the long history of performed actions. This increase in the reasoning time is caused as the answering of a query takes longer on a long history. However, it is also caused as the calculation of the diagnosis takes more time if the action history is longer.
The increase in run time of all three approaches is very interesting. One might expect that the run time increase by a factor of 3 (3 times more objects to deliver) however, the evaluations show that the increase is higher. The time increase for CRAM 2 by a factor of 4.68, CRAM 3 has an increase factor of 8 and for HBD the time increases by a factor of 6.76. Thus, the time increase indicates that a robot needs to spend significantly more time for the execution of a longer task if one performs the task in an environment with faults. It is also interesting to observe that the CRAM 3 has a high increase in run time than HBD although both programs use a TR program for the delivery the focused belief change of HBD shows a beneficial effect for the run time.
Regardless of the scenario HBD performed best regarding the success rate. Even though it imposes a higher run time than a hand coded error recovery, it outperforms this error recovery. Thus, we can confirm Hypothesis 1. A robotic system using a belief management performs better than a robotic system without belief management. Thus, any further increase in the belief management will result in a robotic system which is more successful.

## 10.2. The difference between Belief Management and Advanced Belief Management

The second statement we want to check is if the advanced belief management improves the overall success rate of the robotic system. As we argued in the Chapter 6 an execution which considers the belief of the robot in its execution has a higher chance to succeed. Especially if an active knowledge gathering step is performed which was discussed in Chapter 7. The evaluation presented in this section should confirm this hypothesis. We stated the hypothesis we want to check through this evaluation as follows.

**Hypothesis 2.** *A robotic system using the advanced believe management has a higher success rate in executing a task on a robotic system which uses only the classical belief management.*

The task the robot is performing in the evaluation is the task of the running example. The robot must deliver 3 objects within 20 rooms. The environment is the same as in the previous evaluation and is depicted in Figure 10.1. Several of the actions which are executed may fail. The probabilities for an action failure is depicted in Table 10.6.

| fault | probability |
|---|---|
| pick up wrong object | 0.2 |
| pick fails | 0.2 |
| put fails | 0.3 |
| sense carry anything fails | 0.05 |

Table 10.6.: Probabilities for fault of actions.

To successfully perform the task, the robot needs a proper high-level program for the delivery of an object. We use the high-level program depicted in Listing 10.5 to execute the deliveries. The high-level program picks non-deterministic one delivery to perform. Afterward, it uses a reactive program to do the delivery. To speed up the execution time we use progression instead of a direct reasoning. This method restricts the sensing action to a sensing which checks if the robot is carrying anything.

Listing 10.5: Main loop of the delivery program

```
main() {
  while($\exists r : request(r)$) {
    $\pi r : request(r)$
    $handleRequest(r)$
  }
}
handleRequest(req)
{
  [(place(robot) == req.destination_place) &&
     (place(object) == req.destination_place)]
     --> { putdown(req.object)
       scanCarryAnything() }
  [(place(robot) == place(req.object) && carry_anything]
     --> goto(req.destination_place)
  [(place(robot) == place(req.object)] --> pickup(req.object)
  [true] --> goto(place(req.object))
}
```

### 10.2.1. Experiments

To verify the proposed hypothesis, we executed 100 different delivery tasks. For each delivery task 10 retries with different seed values for the random number generator where used. The robot had 3 minutes to execute all three delivery tasks. For the execution, we compared the classical belief management (CB), proposed in [51] with the advanced belief management (AB) proposed in this thesis. During the evaluations, the successful execution of the task, the percentage how often the program ran into a timeout and the run time of the program was recorded. The evaluation results are depicted in Table 10.7.

| program | success-rate/% | timeouts/% | average runtime/s | |
|---|---|---|---|---|
| | | | mean | std |
| CB | 87.8 | 0 | 6.8 | 3.65 |
| **AB** | **92.1** | **0** | **6.1** | **3.1** |

Table 10.7.: Performance for the delivery task with 20 rooms and 3 objects with classical and advanced belief management.

The result of the evaluation shows that the advanced belief management outperforms the classical belief management, concerning the success rate. The reason for the higher success-rate is due to the usage of the belief instead of one diagnosis. If the robot uses only one diagnosis as it was proposed in the classical belief management the robot might use a situation which is not the current situation. This focus can cause that the robot decided that delivery is fulfilled even though it is not satisfied in practices. Furthermore, the choice of the situation can yield to the execution of an action which is not possible and thus do not yield the expected result. The detection of a not expected result may take some time. This late detection is indicated by the slightly higher run-time of the classical belief management. As a consequence of the evaluation, one can state that Hypothesis 2 can be confirmed. Thus, the usage of the advanced belief management has an advantage compared the classical belief management.

## 10.3. Comparing Belief Management with and without Progression

The previous evaluation used progression instead of a direct reasoning. This method was chosen as progression has a better performance than a direct reasoning. We argued already in Chapter 6 that progression has a better performance. However, it remains to show that this is the case. In this section, we perform an evaluation which confirms this argument. The hypothesis we want to confirm through this assessment is stated as follows.

**Hypothesis 3.** *A robotic system using progression performs a faster reasoning as a robotic system which uses a direct reasoning approach.*

The task the robot is performing in the evaluation is the same deliver task as in the previous section. The robot must deliver three objects in 20 rooms in the environment depicted in Figure 10.1. The probabilities for the different actions is illustrated in Table 10.6. For the execution, the robot uses the program depicted in Listing 10.5.

### 10.3.1. Experiments

To verify the proposed hypothesis, we executed 100 different delivery tasks. For each delivery task 10 retries with different seed values for the random number generator where used. The robot had 6 minutes to execute all three delivery tasks. For the execution, we compared the belief management with progression (WP),and the belief management without progression (NP). During the evaluations, the successful execution of the task, the percentage of the program executions which ran into a timeout and the run time of the program was recorded. The evaluation results are depicted in Table 10.8.

| program | success-rate/% | timeouts/% | average runtime/s | |
|---|---|---|---|---|
| | | | mean | std |
| NP | 49.97 | 42.90 | 210.96 | 87.16 |
| **WP** | **84.70** | **0** | **6.60** | **4.06** |

Table 10.8.: Performance for the delivery task with 20 rooms and 3 objects with and without progression.

The evaluation results indicate that the usage of progression results in a faster execution of the program. This observation is indicated by the lower execution time as well as, the lower rate of executions which ran into a timeout. The reason for this faster execution is caused by the effort for progression is only spent once whereas the answering for a query is done in a fast manner. Additionally, the time for answering a query stays nearly the same regardless of the history length. This constant time behavior contrasts with the direct reasoning which causes the time to respond a query to increase with the length of the history.

## 10.4. The difference between Consistency and Classical History Based Diagnosis

To simplify the specification of the belief management we proposed in Chapter 4 a method which uses the action influence to calculate the belief of the robot. We have argued that the simpler specification comes with the costs of a not so fine gained diagnosis. This lake of precision can cause that the robot has not enough knowledge left to proceed its execution. Thus, we assume that the method proposes in Chapter 4 has a lower success-rate during its execution. In this section, we perform an evaluation showing evidence for this statement. We state the hypothesis we check with the help of the evaluation as follows.

**Hypothesis 4.** *A robotic system using consistency history based diagnosis performs not as successfully in finishing a task as a robotic system which uses the classical history based diagnosis.*

The task the robot is performing in the evaluation is the same deliver task as in the previous sections. The robot must deliver three objects in 20 rooms in the environment depicted in Figure 10.1. The probabilities for the different actions is illustrated in Table 10.6. For the execution, the robot uses the program depicted in Listing 10.5.

### 10.4.1. Experiments

To verify the proposed hypothesis, we executed 100 different delivery tasks. For each delivery task 10 retries with different seed values for the random number generator where used. The robot had 3 minutes to execute all three delivery tasks. For the execution, we compared the belief management with the classical history base diagnosis (HBD),and the belief management with the consistency history based diagnosis (CHBD). During the evaluations, the successful execution of the task, the percentage of the program executions which ran into a timeout and the run time of the program was recorded. The evaluation results are depicted in Table 10.9.

| program | success-rate/% | timeouts/% | average runtime/s | |
|---|---|---|---|---|
| | | | mean | std |
| **HBD** | **87.5** | **0** | **6.5** | **2.76** |
| CHBD | 55.3 | 39.7 | 10.45 | 7.78 |

Table 10.9.: Performance for the delivery task with 20 rooms and 3 objects with the classical history based diagnosis and the consistency history based diagnosis.

The evaluation results show that the classical history based diagnosis performs better than the consistency history based diagnosis. This result is caused as the consistency history based diagnosis causes the robot often to lose too much knowledge. As only actions are blamed for being faulty the influence is used to specify the effect on faulty action has the knowledge of the robot gets reduced with each diagnosis. If multiple diagnoses can yield an explanation of the current discrepancy, the knowledge of the robot may be reduced drastically. As the robot can only perform one sensing action which checks if it is holding something, it may not be able to recover from this lack of knowledge. If one action influence removes the knowledge the robot has about the position of an object, the robot has no sensing action which can retrieve this information again. This issue also shows that if consistency history based diagnosis is used one needs to ensure that the sensing actions the robot has can derive all knowledge which can be lost due to the action influence. Otherwise, the robot may get stuck in the execution as it lacks knowledge. Another direct result of this evaluation is that we can confirm Hypothesis 4 which states that the classical history based diagnosis has a higher success rate than with consistency history based diagnosis.

## 10.5. Comparing Linear with Reactive Programs

The next evaluation we performed discusses the benefit of the generation of a reactive program as it was proposed in Chapter 8. We argued that a reactive program be more likely to succeed in executing a task than a linear program. Thus, we generated a reactive program from a linear program. With the evaluation, which is performed in this section we will show that a reactive program increases the success of achieving a goal in case of a faulty action execution. We state this hypothesis as follows.

**Hypothesis 5.** *A robotic system using the reactive program has a higher success rate in executing a task on a robotic system which uses a linear program.*

The task the robot is performing in the evaluation is the same deliver task as in the previous sections. The robot must deliver three objects in 20 rooms in the environment depicted in Figure 10.1. The probabilities for the different actions is illustrated in Table 10.6. For the execution, the robot uses either the linear program depicted in Listing 10.6, the reactive program depicted in Listing 10.5 or uses the generated program which was derived from method discussed in Chapter 8 from the linear program.

Listing 10.6: Main loop of the delivery program

```
main() {
   while ($\exists r : request(r)$) {
      $\pi r : request(r)$
      $handleRequest(r)$
   }
}
handleRequest(req) {
   goto(place(req.object))
   pick(req.object)
   goto(req.destination_place)
   put(req.object)
}
```

### 10.5.1. Experiments

To verify the proposed hypothesis, we executed 100 different delivery tasks. For each delivery task 10 retries with different seed values for the random number generator where used. The robot had 3 minutes to execute all three delivery tasks. For the execution, we compared the linear program (L), the hand written teleo-reactive program (TR) and the program which was generated through the method proposed in Chapter 8 (G). This generation was done with a back-stepping limit of four steps. During the evaluations, the successful execution of the task, the number of the program executions which ran into a timeout and the run time of the program was recorded. The evaluation results are depicted in Table 10.10.

| program | success-rate/% | timeouts/% | average runtime/s | |
|---|---|---|---|---|
| | | | mean | std |
| L | 45.6 | 0 | 2.13 | 1.35 |
| **TR** | **85.6** | 0 | 6.49 | 3.28 |
| **G** | 81.9 | 0 | **4.93** | **2.51** |

Table 10.10.: Performance for the delivery task with 20 rooms and 3 objects with a linear, teleo-reactive and a generated reactive program.

The evaluations show that the reactive programs outperform the linear program. The hand written reactive program is slightly better than the generated program. This discrepancy is partly caused due to the carefully crafted program which allows triggering at least one action regardless of the fault. The evaluation shows additionally that the generated program is slightly faster than the hand written reactive program. This observation is caused as the generated program does not force the execution to perform a reasoning over several rules if no fault has occurred. Overall one can state that the evaluation confirms the Hypothesis 5. Thus, the usage of a reactive program should be preferred over a linear program. Due to the method proposed in Chapter 8 one can specify a linear program for the task and afterward generate a reactive program which ensures a high success rate to finish the task.

## 10.6. Impact of common sense

With the previous evaluations, we have shown which impact the belief management, in general, has on the success rate of the robot. Furthermore, we have shown that the usage of a reactive program

increases the success rate. In this evaluation, we will evaluate the impact the use of common sense has. Especially we will assess the impact of common sense from an external knowledge base which was mapped according to the method described in Chapter 5.
We perform two evaluations to measure the impact of the usage of external common sense. The first assessment focus on the implications the domain size has on the approach. The second assessment concentrate on the impact of the given timeout has on the success of solving a task.
Both evaluations used the implementation of the history based diagnosis approach as it was proposed in [51] extended with the mapping method described in Chapter 5. The task the robot is performing in the evaluation is the same deliver task as in the previous sections. The environment depicted in Figure 10.1 was used for the robot as evaluation setting. The probabilities for the different actions is illustrated in Table 10.6.
Each experimental setup[‡] was carried out with 20 different object settings (different initial locations and goals for object). Additionally, ever of these 20 runs was tried with five different seeds for the random number generator. Due to these variations, any advantage of object setting or occurrence of fault should be ruled out.
During the evaluation, three different methods were used. The first method (HBD 0) does not apply common sense. Thus, it only checks if a sensing action results in contradiction with the predicted result. The second method (HBD 1) use common sense which was encoded as invariant within the extended basic action theory itself. Thus, no mapping was required to perform the consistency check. The third method (HBD 2) use the proposed mapping to Cyc to check for consistency of the state with the common sense knowledge.
All the different methods used the same program to perform the delivery task. The program used a Teleo-Reactive (TR) approach [98] for the deliveries. The same reactive program as in the first evaluation was used and is depicted in Listing 10.4. Through the usage of a TR program, the robot can react to changes in the knowledge about the world. This reaction allows the robot to work towards the goal even in case a fault has occurred.
During the execution of a test the run-time of the execution, the number of successfully delivered objects as well as if the test was finished before the timeout where recorded.
With the help of this recorded data different hypothesis are checked. In both evaluation setups, we wanted to check if the following hypothesis holds.

**Hypothesis 6.** *The programs which use the background knowledge have a higher success-rate compared to the program not using background knowledge.*

### 10.6.1. Impact of the domain size

The first evaluation setup evaluated the impact of the domain size. To test the impact of the domain size different evaluations which differ in the number of rooms and objects were used. For all assessments, a time out of 12 minutes was used.
The first evaluation performed for this evaluation setup was to deliver three objects in an environment with 20 rooms. The success rate of delivery all three objects, the percentage of runs resulting in a timeout (TIR) as well as the run time of the execution is depicted in Table 10.11. Please note that the run time does omit those cases where the program ran into the timeout. The result of the first assessment shows that the use of common sense increases drastically the chance to deliver the objects. This increase is the case as the robot detects a problem which would otherwise be missed. It is also

[‡]The tests were performed on an Intel quad-core CPU with 2.40 GHz and 4 GB of RAM running 32-bit Ubuntu 12.04. The ROS version Groovy was used for the evaluation.

| | success rate/% | TIR/% | average run-time/s | |
|---|---|---|---|---|
| | | | mean | stdv. |
| HDB 0 | 54.11 | 18.81 | 3.93 | 4.03 |
| HDB 1 | 95.55 | 1.98 | 5.95 | 7.10 |
| HDB 2 | 79.80 | 36.63 | 399.18 | 153.09 |

Table 10.11.: Success rate, timeout rate (TIR) and the average run-time of the approaches to map common sense for the first evaluation in the first evaluation setup.

interesting to notice that the method which does not use common sense has a high time out probability. This high time out probability is caused by the fact that the robot detects after some time that the object was not correctly delivered due to a sensing mismatch. However, due to this late detection, the agent needs to perform a more complex reasoning and repair to deliver the object correctly. As a last result, it is to mention that the variant which uses the mapping of Cyc has a drastically higher run time. This result is caused due to the complete transfer of the knowledge into Cyc. Furthermore, the consistency check in Cyc is more complex as the uses common sense knowledge incorporates more than just simple invariant about object locations as they are used in the extended basic action theory.
The second evaluation performed in the first evaluation setup was to deliver three objects in an environment consisting of 29 rooms. The success rate of delivery all three objects, the percentage of runs resulting in a timeout (TIR) as well as the run time of the execution is depicted in Table 10.12. Please note that the run time does omit those cases where the program ran into the timeout. The results again

| | success rate/% | TIR/% | average run-time/s | |
|---|---|---|---|---|
| | | | mean | stdv. |
| HDB 0 | 57.89 | 14.0 | 3.64 | 228.52 |
| HDB 1 | 95.44 | 1.0 | 13.14 | 65.80 |
| HDB 2 | 72.98 | 50.0 | 560.00 | 123.99 |

Table 10.12.: Success rate, timeout rate (TIR) and the average run-time of the approaches to map common sense for the second evaluation in the first evaluation setup.

indicate that the use of common sense increases the success rate. What is more interesting to observe is that the time spent to deliver all objects has grown in case of HDB 1 by a factor of 2.33 and in case of HDB 2 the factor was 1.43. The slower increase of the run time for the method using CYC is caused as more runs caused to run into a timeout. The growth of HDB is fascinating as it shows that the checking of the invariant has not scaled in the same way as the environment size. As the environment size only increases by a factor of 1.45. Thus, if invariant is used to check the consistency of the current situation one needs to take care that the approach scales well with the domain size.
The final evaluation performed for the first evaluation setup was to deliver nine objects in an environment consisting of 20 rooms. The success rate of delivery all nine objects, the percentage of runs resulting in a timeout (TIR) as well as the run time of the execution is depicted in Table 10.13. Please note that the run time does omit those cases where the program ran into the timeout. In this assessment, it is important to notice that the mapping to Cyc always run into a timeout. Thus, the robot was not able to deliver nine objects within 12 minutes. If instead the invariant in the extended basic action theory were used the robot had only encountered 29% of the cases a timeout. This increase in the time out of Cyc was to expect as the timeout ratio was 36.63% if only three objects had to deliver. Thus, an increase by a factor of 3 explains this thigh timeout ratio. It is also interesting to notice that the

| | success rate/% | TIR/% | average run-time/s | |
|---|---|---|---|---|
| | | | mean | stdv. |
| HDB 0 | 22.01 | 32.00 | 216.30 | 305.99 |
| HDB 1 | 73.49 | 29.00 | 206.41 | 291.54 |
| HDB 2 | 18.23 | 100 | - | - |

Table 10.13.: Success rate, timeout rate (TIR) and the average run-time of the approaches to map common sense for the third evaluation in the first evaluation setup.

run time as well as the ratio to run into a timeout is higher if no commonsense knowledge is used. It is to expect that due to the usage of the commonsense knowledge the faults are discovered early thus allowing a faster recovery of this fault.
As a final observation to all the evaluation one can state that Hypothesis 6 holds. Thus, the usage of common sense increases the chance to find a faulty execution. Moreover, thus, directly increase the success rate to finish a task.

## 10.6.2. Impact of timeout

The second evaluation evaluated if a higher timeout may positively influence the success rate. During the assessment, the robot had to deliver three objects. The environment consists of 20 different rooms. With the help of various time outs, the following two hypotheses should be checked.

**Hypothesis 7.** *With an increasing timeout, the success-rate increases too.*

**Hypothesis 8.** *With an increasing timeout, the program which uses the common sense represented in Cyc shows the same success rate as the program which uses the hand-coded background knowledge.*

In this evaluation setup, three different evaluations were performed. Each evaluation had a different timeout. The first evaluation used a timeout of 660 seconds. The second evaluation used a timeout of 6600 seconds. The third evaluation used a timeout of 66000 seconds. The results of the first assessment are depicted in Table 10.14. Table 10.15 depicts the result of the second evaluation. Finally, the results of the third evaluation are depicted in Table 10.14. All their assessment results show the success rate of delivery all three objects, the percentage of runs resulting in a timeout (TIR) as well as the run time of the execution is depicted in Table 10.13. Please note that the run time does omit those cases where the program ran into the timeout.

| | success-rate | TIR | average runtime/s | |
|---|---|---|---|---|
| | | | Mean | Std |
| HBD 0 | 48.47 | 15.00 | 9.32 | 46.20 |
| HBD 1 | 89.49 | 7.00 | 11.79 | 47.18 |
| HBD 2 | 66.44 | 44.00 | 423.29 | 141.50 |

Table 10.14.: Success-rate, timeout-rate (TIR) and runtime for a timeout of 660 seconds for mapping common sense.

The evaluations show that through the increase of the timeout the run time increases as the robot can spent more time to deliver the objects. Furthermore, the success rate increases. This can also be seen in Figure 10.6.2. The figure shows the relation between the timeout time and the success rate. It is

| | success-rate | TIR | average runtime/s | |
|---|---|---|---|---|
| | | | Mean | Std |
| HBD 0 | 54.92 | 6.00 | 132.78 | 750.37 |
| HBD 1 | 91.53 | 2.00 | 216.85 | 1188.02 |
| HBD 2 | 77.29 | 22.00 | 903.02 | 1118.48 |

Table 10.15.: Success-rate, timeout-rate (TIR) and runtime for a timeout of 6600 seconds for mapping common sense.

| | success-rate | TIR | average runtime/s | |
|---|---|---|---|---|
| | | | Mean | Std |
| HBD 0 | 46.78 | 0.0 | 704.21 | 2064.03 |
| HBD 1 | 89.83 | 0.0 | 351.36 | 1513.69 |
| HBD 2 | 78.31 | 17.00 | 2340.75 | 4789.59 |

Table 10.16.: Success-rate, timeout-rate (TIR) and runtime for a timeout of 66000 seconds for mapping common sense.

important to notice that the success rate of the method (HDB 2) using the mapping to Cyc increase significantly.
In contrast to the other approach, the method which uses the definition of invariant in the extended basic action theory (HDB 2) has a small fluctuation around 90 %. Thus, indicating that this is the maximum value of the success rate which is possible. This result also shows that some deliveries cannot be successfully executed as the fault either is not detected or the robot has no proper repair. A similar fluctuation can be seen from the method (HDB 0) which does not use common sense fluctuate around 50 %. Thus, without the use of common sense, only 50% of the deliveries can be performed success fully.

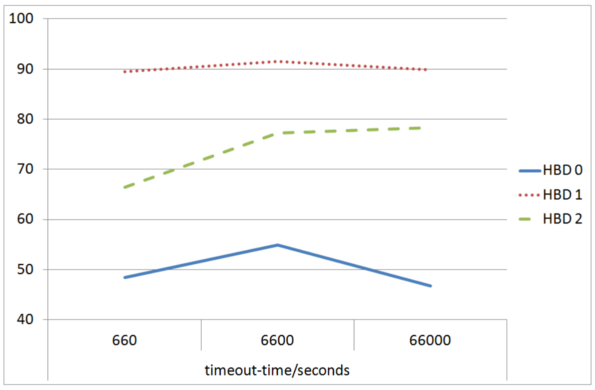

Figure 10.3.: Success-rate of the different approach in relation to the timeout.

As a final remark to the performed evaluation one can state that we can verify Hypothesis 8. As through an increase in the timeout, the method using the mapping (HBD 2) starts to reach the method which uses the native common sense (HBD 1). However, it also shows that the effort of the mapping is very high. This high effort is caused as the complete knowledge needs to be mapped. Thus, if we

consider $m$ fluents, $n$ objects and the maximum cardinality of a fluent to be $k$, then the mapping needs to map $O(m \times n^k)$ fluent instances.
Hypothesis 7 cannot be verified. As an increase in the timeout time does not directly increase the success rate. This observation is caused by the reasoning which, if fast enough to cause no time outs, does not allow to increase the success rate further.

## 10.7. Conclusion and Future Work

In the previous chapters of this thesis, we have proposed several improvements in the classical belief management. In these chapters, we have argued why several improvements increase the usability of the belief management or increase the success rate of a robot using the advanced belief management. In this chapter, we have shown several evaluations which confirm this statements.
First, we have shown that the usage of a belief management outperforms a robot which does not use any belief management. The evaluation was performed on a simple delivery task, with different numbers of objects to deliver and rooms to consider for the delivery. Regardless of the size of the environment or the number of deliveries robots using a belief management outperforms robots which don't use a belief management.
Afterward, we have shown that the incorporation of the belief into the high-level execution as it was proposed in Chapter 6 together with the knowledge gathering approach presented in Chapter 7 increase the success rate of a robot. The evaluation shows that the advanced belief management as it was presented in this thesis outperforms the classical belief management, concerning the success of executing a task. This improvement was achieved without causing any increase in run-time. Thus, the improvements proposed in this thesis can be considered as advancement of the classical belief management.
The next evaluation showed that the usage of progression results in a faster execution than if classical reasoning is used. This evaluation also indicates that the progression method proposed in Chapter 6 should be preferred over a direct reasoning approach.
The proceeding evaluation related the classical history based diagnosis with the consistency history based diagnosis as it was proposed in Chapter 4. The evaluation shows that the usage of the consistency history based diagnosis results in a robotic system which aborts its mission more often than the classical approach. This higher rate of abortion is caused by the fact that the knowledge of the robot is reduced more than it can be expanded by sensing actions. Thus, the evaluation shows that the usage of the consistency history based diagnosis should be performed carefully as the robot should be able to perform sensing action to gather enough knowledge regardless of the diagnosis.
The next evaluation showed that the reactive program generated with the method described in Chapter 8 outperforms a simple linear program. Thus, the method allows to specify the behavior of the robot in a simple linear program, but yield a good chance of success due to the generation of a reactive version of this program.
Finally, we have performed two different evaluation settings to show the impact of the usage of common sense. Both evaluations settings confirmed that the use of common sense increase the success rate to perform a task. It further has shown that the run time of the mapping is much higher than a native use of the common sense knowledge. Please note that this mapping is still of uses as a bigger commonsense knowledge base can be integrated into the robotic system.

Chapter 11

# Conclusion

Robots acting in an environment use their belief about the environment to take decisions which actions should be performed next to achieve a task at hand. If the robot acts in a not idealized environment the robot may fail to execute a certain action successfully. Thus, an action may not lead to the expected result, or the action may have other effects. To deal with these faults, the robot needs to manage its belief to ensure that the belief about the environment coincides with the real environment. Such a belief management is necessary as it is the only way of ensuring that the robot takes a reasonable decision about the next action to perform. In this thesis, we have presented several methods for improving an existing belief management approach. In this chapter, we will first give a summary of the presented methods and afterwards discuss some future research in the area of belief management for robots.

## 11.1. Summary

The ease of use of the belief management and its efficiency is one of the main concerns which need to be addressed to allow widespread use of belief management for robots.
The first method presented in this thesis addresses the problem of the ease of use as well as the efficiency of current belief management approaches. The method cost of using the belief management and thus allows its widespread use. Current approaches use a detailed description of how an action may fail to derive an explanation what fault has happened. The specification how the action is failing is used to derive an alternative history of performed actions such that the observations made so far are consistent with this history. The specification of the faults an action can have in such detail is a complex task, and sometimes it is merely not possible to specify all faults in advance as it is necessary. Additionally, considering every possibility how an action can fail one may have the problem that there may be an infinite number of possible explanations. To address this issue, we proposed in this thesis to use the influence of actions instead. The action influence only specifies which parts of the environment might be influenced due to an action failure. This specification eases defining an envelope of the influence an action has, thus simplifying the definition of a faulty action. Furthermore, the method ensures that one only needs to consider finitely many alternatives.
The second method presented in this thesis further simplifies the use of the belief management. To detect a fault earlier, the robot can use common sense knowledge about the environment. The specification of this common sense can become a complex task as one needs to specify a big set of properties which hold in the environment. Furthermore, this specification needs to be consistent. Thus, to eases

this specification, it is desirable to reuse the common sense knowledge which is already specified in other knowledge bases. The method presented in this thesis can be used to perform such reuse. The method allows using any arbitrary external source for the common sense of the robot. The external source is invoked to check the consistency of the situation only when needed. Furthermore, the method is not restricted to a certain logic of the external resource. Thus, it increases the possibility to reuse common sense drastically.
The third method presented in this thesis tackles the problem of integrating the belief of the robot into the decision cycle of the robot. This integration ensures that the robot uses its current belief to decide which action to perform next. Furthermore, the approach allows the robot to draw the conclusion that it lacks knowledge about the environment to make a safe decision. Thus, the agent can trigger a gathering of knowledge in such a case. Additionally, we have shown that a robot using this method will only choose actions which the robot can safely assume to be executable. Furthermore, we demonstrated that the robot never terminates its execution if there is a chance that it can proceed with its mission. Instead, the method allows the robot to safely but continuously move towards the achievement of its task.
The fourth method presented in this thesis allows the robot to gather additional knowledge about its environment if necessary. The method uses the information which explanations are currently used to describe the state of the environment to choose a proper sensing action. Due to this chosen sensing action, the robot gains knowledge about the environment. Furthermore, the sensing action is selected in such a way that the robot can rule out explanations which are not consistent with the real environment. Additionally, we have shown that even if the method is applied in an online fashion, it yields the same result as if one would construct a conditional plan. This result allows the robot to gather knowledge without the need of a very complex planning step.
The fifth method presented in this thesis tackles the problem of specifying a high-level program which allows the robot to react to faults during the action execution. Often only simple sequential programs are known to achieve a certain task. Unfortunately, such a simple sequential program often fails to react to faults. Although the robot can update the belief accordingly, a consistent continuation of the execution of the program fails as no proper action can be chosen due to the restrictions of the sequential program. To address this problem, we presented a method which automatically derives a reactive program from a sequential one. This reactive program allows the robot to react to faults and achieve its task. The presented method is general enough to deal with branches and even loops in the specified program. This generality allows an easy conversion of already existing high-level control programs into reactive programs.
Finally, the thesis has shown in several evaluations that the use of a belief management improves the overall success rate of a robot to achieve its task. Additionally, several evaluations were performed which show that the proposed extension to the belief management improve the success rate of the robot to finish its task further. The assessment presented allows concluding that a robot using the methods proposed in this thesis will achieve its task with a higher probability than a robot which does not use the proposed methods.

## 11.2. Further research

For future research one major point of interest is to improve the efficiency of the proposed belief management, although different methods have been shown within this thesis to enhance the effectiveness. Firstly, the calculation of explanations which action failed could be improved. A good starting point would be a generalization of the method presented in [111] which allows performing some of the

calculations offline. Thus, the robot can perform some precomputation to allow a faster generation of the explanations. Furthermore, one could exploit the structure of the history of performed actions and the observed discrepancy to improve the efficiency.
Secondly one could restrict the number of situations in which an explanation needs to be calculated. A similar approach was presented in [106] for another belief management method. One could extend this method to ensure that one only checks for those inconsistencies which are relevant for the robot during its future execution. Also, one may restrict the generation of explanations to those which are different with regard to properties of interest. Thus, one could reduce the generated explanations.
Thirdly, the presented belief management was defined in such a way that any effect an action may have can be used. Due to this general definition, some of the calculations need to be performed very generally and thus cause a high complexity. Thus, it would be of interest to adapt the approach to a certain fragment of allowed action effects. This restriction could make it possible to easy some definitions and thus allow faster methods to be used for the reasoning. One such restricted reasoning was used to specify the high-level programming language defined in [39].
Besides the improvements in efficiency, one can also extend the approach to be used on a broader basis. The presented approach focuses on sequential execution of a program. However, many high-level control programs use parallel execution to simplify the program. Thus, an extension towards a parallel execution of the actions as well as towards parallel programs would be of interest.
Finally, all the evaluations were performed in simulation. This restriction allowed us to perform several different assessments, as well as to perform a high number of assessments. It is left for future work to evaluate a real robot. Furthermore, all evaluations were conducted on a simple delivery task. It is left for future work to expand the evaluation to other tasks. The usage of other tasks may yield an insight how the proposed approach depends on the structure of the belief. This insight could yield to new research questions concerning the effectiveness of the proposed approach.

# Bibliography

[1] AHO, A. V., SETHI, R., AND ULLMAN, J. D. 1986. *Compilers, Principles, Techniques*. Addison wesley. (Cited on pages 94 and 118.)

[2] ARENAS, M., BOTOEVA, E., CALVANESE, D., AND RYZHIKOV, V. 2013. Exchanging OWL 2 QL Knowledge Bases. *The Computing Research Repository (CoRR)* April 2013. (Cited on page 56.)

[3] BACCHUS, F., HALPERN, J. Y., AND LEVESQUE, H. J. 1999. Reasoning about noisy sensors and effectors in the situation calculus. *Artificial Intelligence 111,* 1, 171–208. (Cited on pages 31 and 34.)

[4] BAIER, J., MOMBOURQUETTE, B., AND MCILRAITH, S. A. 2014. Diagnostic problem solving via planning with ontic and epistemic goals. In *Proceedings of the 14th International Conference on Principles of Knowledge Representation and Reasoning (KR)*. (Cited on page 51.)

[5] BARAL, C., MCILRAITH, S., AND SON, T. C. 2000. Formulating diagnostic problem solving using an action language with narratives and sensing. In *Proceedings of the Seventh International Conference on Principles of Knowledge Representation and Reasoning*. 311–322. (Cited on pages 28, 29, and 51.)

[6] BEETZ, M., JAIN, D., MÖSENLECHNER, L., AND TENORTH, M. 2010. Towards Performing Everyday Manipulation Activities. *Robotics and Autonomous Systems 58,* 9, 1085–1095. (Cited on page 120.)

[7] BEETZ, M., MÖSENLECHNER, L., AND TENORTH, M. 2010. CRAM - A Cognitive Robot Abstract Machine for everyday manipulation in human environments. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2010*. IEEE, 1012–1017. (Cited on page 120.)

[8] BELLE, V. AND LEVESQUE, H. J. 2013. Reasoning about continuous uncertainty in the situation calculus. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence (IJCAI)*. (Cited on page 31.)

[9] BELLE, V. AND LEVESQUE, H. J. 2014. Prego: An action language for belief-based cognitive robotics in continuous domains. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence (AAAI)*. 989–995. (Cited on page 31.)

[10] BELLE, V. AND LEVESQUE, H. J. 2015. Allegro: Belief-based programming in stochastic dynamical domains. In *Proceedings of the 24th International Conference on Artificial Intelligence*. AAAI Press, 2762–2769. (Cited on pages 31 and 32.)

[11] BORST, P., AKKERMANS, J., POS, A., AND TOP, J. 1995. The physsys ontology for physical systems. In *Working Papers of the Ninth International Workshop on Qualitative Reasoning QR*. 11–21. (Cited on page 60.)

[12] Burgard, W. and Stachniss, C. 2013. Introducing ... Obelix. *German Research 35,* 3, 6–10. (Cited on page 1.)

[13] Cerioli, M. and Meseguer, J. 1993. May i borrow your logic? In *Mathematical Foundations of Computer Science 1993*. Springer, 342–351. (Cited on page 59.)

[14] Classen, J. and Lakemeyer, G. 2009. Tractable first-order golog with disjunctive knowledge bases. *Proc. Commonsense*, 27–33. (Cited on pages 34 and 35.)

[15] Classen, J. and Lakemeyer, G. 2010. On the verification of very expressive temporal properties of non-terminating golog programs. In *European Conference on Artificial Intelligence (ECAI)*. 887–892. (Cited on page 112.)

[16] Clocksin, W. and Mellish, C. S. 2003. *Programming in PROLOG.* Springer Science & Business Media. (Cited on pages 115 and 116.)

[17] Console, L., Dupré, D. T., and Torasso, P. 1991. On the relationship between abduction and deduction. *Journal of Logic and Computation 1,* 5, 661–690. (Cited on pages 17 and 40.)

[18] De Cat, B., Jansen, J., and Janssens, G. 2013. Idp3: Combining symbolic and ground reasoning for model generation. In *2nd Workshop on Grounding and Transformations for Theories With Variables*. 17–24. (Cited on page 117.)

[19] De Giacomo, G., Lespérance, Y., and Levesque, H. J. 2000. Congolog, a concurrent programming language based on the situation calculus. *Artificial Intelligence 121,* 1, 109–169. (Cited on page 21.)

[20] De Giacomo, G., Lespérance, Y., Levesque, H. J., and Sardina, S. 2009. Indigolog: A high-level programming language for embedded reasoning agents. In *Multi-Agent Programming:*. Springer, 31–72. (Cited on pages 10, 21, 22, 23, 24, and 61.)

[21] De Giacomo, G., Lespérance, Y., Patrizi, F., and Sardina, S. 2016. Verifying congolog programs on bounded situation calculus theories. In *Proc. of the 30th AAAI Conf. on Artificial Intelligence (AAAI)*. (Cited on page 93.)

[22] De Giacomo, G. and Levesque, H. J. 1999. An incremental interpreter for high-level programs with sensing. In *Logical foundations for cognitive agents*. Springer, 86–102. (Cited on page 15.)

[23] De Giacomo, G., Levesque, H. J., and Sardina, S. 2001. Incremental execution of guarded theories. *ACM Transactions on Computational Logic (TOCL) 2,* 4, 495–525. (Cited on pages 41 and 52.)

[24] De Giacomo, G., Reiter, R., and Soutchanski, M. 1998. Execution monitoring of high-level robot programs. In *Principles of Knowledge Representation and Reasoning-International Conference*. Morgan Kaufmann Publishers, 453–465. (Cited on page 37.)

[25] De Kleer, J. and Williams, B. C. 1987a. Diagnosing multiple faults. *Artificial intelligence 32,* 1, 97–130. (Cited on pages 20 and 40.)

[26] De Kleer, J. and Williams, B. C. 1987b. Diagnosing multiple faults. *Artificial intelligence 32,* 1, 97–130. (Cited on page 85.)

[27] De Leoni, M., Mecella, M., and De Giacomo, G. 2007. Highly dynamic adaptation in process management systems through execution monitoring. In *International Conference on Business Process Management*. Springer, 182–197. (Cited on page 37.)

[28] DE MOURA, L. AND BJØRNER, N. 2008. Z3: An efficient smt solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 337–340. (Cited on page 117.)

[29] DELGRANDE, J. P. AND LEVESQUE, H. J. 2012. Belief revision with sensing and fallible actions. In *International Conference on Principles of Knowledge Representation and Reasoning (KR)*. Citeseer. (Cited on pages 32 and 33.)

[30] DELGRANDE, J. P. AND LEVESQUE, H. J. 2013. A formal account of nondeterministic and failed actions. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence (IJCAI)*. (Cited on page 33.)

[31] DEMOLOMBE, R. AND PARRA, M. D. P. P. 2000. A simple and tractable extension of situation calculus to epistemic logic. In *International Symposium on Methodologies for Intelligent Systems*. Springer, 515–524. (Cited on page 35.)

[32] DEMOLOMBE, R. AND PARRA, P. P. 2006. Belief revision in the situation calculus without plausibility levels. In *International Symposium on Methodologies for Intelligent Systems*. Springer, 504–513. (Cited on page 32.)

[33] EITER, T., ERDEM, E., AND FABER, W. 2004. Diagnosing plan execution discrepancies in a logic-based action framework. Tech. Rep. 1843-04-03, Technical Report INFSYS RR-1843-04-03, Vienna University of Technology. (Cited on pages 28 and 29.)

[34] EITER, T., FINK, M., AND SENKO, J. 2005. Kmonitor–a tool for monitoring plan execution in action theories. In *International Conference on Logic Programming and Nonmonotonic Reasoning*. Springer, 416–421. (Cited on page 28.)

[35] FAN, Y., CAI, M., LI, N., AND LIU, Y. 2012. A first-order interpreter for knowledge-based golog with sensing based on exact progression and limited reasoning. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence (AAAI)*. (Cited on page 35.)

[36] FANG, L. AND LIU, Y. 2013. Multiagent knowledge and belief change in the situation calculus. In *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence*. (Cited on page 33.)

[37] FANG, L., LIU, Y., AND WEN, X. 2015. On the progression of knowledge and belief for nondeterministic actions in the situation calculus. In *Proceedings of the 24th International Conference on Artificial Intelligence*. AAAI Press, 2955–2963. (Cited on page 33.)

[38] FERREIN, A. 2008. Robot controllers for highly dynamic environments with real-time constraints. Ph.D. thesis, Knowledge-based Systems Group, RWTH Aachen University, Aachen Germany. (Cited on page 37.)

[39] FERREIN, A., MAIER, C., MÜHLBACHER, C., NIEMUELLER, T., STEINBAUER, G., AND VASSOS, S. 2015. Controlling logistics robots with the action-based language yagi. In *Workshop on Task Planning for Intelligent Robots in Service and Manufacturing*. 13. (Cited on page 141.)

[40] FERRI, G., MANZI, A., SALVINI, P., MAZZOLAI, B., LASCHI, C., AND DARIO, P. 2011. Dustcart, an autonomous robot for door-to-door garbage collection: From dustbot project to the experimentation in the small town of peccioli. In *IEEE International Conference on Robotics and Automation (ICRA), 2011*. IEEE, 655–660. (Cited on page 1.)

[41] FICHTNER, M., GROSSMANN, A., AND THIELSCHER, M. 2003. Intelligent execution monitoring in dynamic environments. *Fundamenta Informaticae 57,* 2-4, 371–392. (Cited on page 30.)

[42] FIKES, R. E., HART, P. E., AND NILSSON, N. J. 1972. Learning and executing generalized robot plans. *Artificial intelligence 3*, 251–288. (Cited on page 112.)

[43] FIKES, R. E. AND NILSSON, N. J. 1971. Strips: A new approach to the application of theorem proving to problem solving. *Artificial intelligence 2,* 3-4, 189–208. (Cited on pages 14 and 112.)

[44] FRITZ, C. 2009. Monitoring the generation and execution of optimal plans. Ph.D. thesis, University of Toronto. (Cited on pages 37 and 38.)

[45] GABALDON, A. AND LAKEMEYER, G. 2007. Esp: A logic of only-knowing, noisy sensing and acting. In *PROCEEDINGS OF THE NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE*. Vol. 22. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 974. (Cited on page 34.)

[46] GAT, E. ET AL. 1998a. On three-layer architectures. *Artificial Intelligence and Mobile Robots 195*, 210. (Cited on page 2.)

[47] GAT, E. ET AL. 1998b. On three-layer architectures. *Artificial intelligence and mobile robots 195*, 210. (Cited on page 2.)

[48] GIACOMO, G. D., LESPÉRANCE, Y., LEVESQUE, H. J., AND SARDINA, S. 2009. *Multi-Agent Programming: Languages, Tools and Applications*. Springer, Chapter IndiGolog: A High-Level Programming Language for Embedded Reasoning Agents, 31–72. (Cited on page 125.)

[49] GROSSKREUTZ, H. AND LAKEMEYER, G. 2000. Turning high-level plans into robot programs in uncertain domains. In *Proceedings of the 14th European Conference on Artificial Intelligence (ECAI)*. 548–552. (Cited on page 36.)

[50] GROSSKREUTZ, H. AND LAKEMEYER, G. 2001. Belief update in the pgolog framework. In *Annual Conference on Artificial Intelligence*. Springer, 213–228. (Cited on page 36.)

[51] GSPANDL, S., PILL, I., REIP, M., STEINBAUER, G., AND FERREIN, A. 2011. Belief Management for High-Level Robot Programs. In *International Joint Conference on Artificial Intelligence (IJCAI)*. Vol. 22. 900–905. (Cited on pages 4, 5, 15, 17, 18, 19, 20, 31, 120, 129, and 133.)

[52] GSPANDL, S., PODESSER, S., REIP, M., STEINBAUER, G., AND WOLFRAM, M. 2012. A dependable perception-decision-execution cycle for autonomous robots. In *IEEE International Conference on Robotics and Automation (ICRA), 2012*. IEEE, 2992–2998. (Cited on pages 7, 120, 121, and 125.)

[53] GUIZZO, E. 2008. Three Engineers, Hundreds of Robots, One Warehouse. *Spectrum, IEEE 45,* 7, 26–34. (Cited on pages 1 and 7.)

[54] HAAS, A. R. 1987. The case for domain-specific frame axioms. In *The frame problem in artificial intelligence. Proceedings of the 1987 workshop*, F. Brown, Ed. Morgan Kaufmann Publishers, San Francisco, 343–348. (Cited on page 10.)

[55] IAGNEMMA, K. AND BUEHLER, M. 2006. Editorial Special Issue on the DARPA Grand Challenge. *J. Field Robot. 23,* 9, 655–656. (Cited on page 1.)

[56] IWAN, G. 2002. History-based diagnosis templates in the framework of the situation calculus. *AI Communications 15,* 1, 31–45. (Cited on pages 17 and 18.)

[57] JIN, Y. AND THIELSCHER, M. 2004. Representing beliefs in the fluent calculus. In *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI)*. Vol. 16. 823. (Cited on page 30.)

[58] JOHNSTON, B. AND WILLIAMS, M.-A. 2009a. Autonomous learning of commonsense simulations. In *International Symposium on Logical Formalizations of Commonsense Reasoning*. 73–78. (Cited on page 59.)

[59] JOHNSTON, B. AND WILLIAMS, M.-A. 2009b. Conservative and reward-driven behavior selection in a commonsense reasoning framework. In *AAAI Symposium: Multirepresentational Architectures for Human-Level Intelligence*. 14–19. (Cited on page 59.)

[60] JONES, J. L. 2006. Robots at the tipping point: the road to iRobot Roomba. *Robotics Automation Magazine, IEEE 13,* 1, 76–78. (Cited on page 1.)

[61] KALFOGLOU, Y. AND SCHORLEMMER, M. 2003. Ontology mapping: the state of the art. *The knowledge engineering review 18,* 1, 1–31. (Cited on page 53.)

[62] KRUIJFF, G.-J. M., PIRRI, F., GIANNI, M., PAPADAKIS, P., PIZZOLI, M., SINHA, A., TRETYAKOV, V., LINDER, T., PIANESE, E., CORRAO, S., ET AL. 2012. Rescue robots at earthquake-hit Mirandola, Italy: A field report. In *IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR), 2012*. IEEE, 1–8. (Cited on page 1.)

[63] KUHN, L., PRICE, B., DO, M., LIU, J., ZHOU, R., SCHMIDT, T., AND DE KLEER, J. 2010. Pervasive diagnosis. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans 40,* 5, 932–944. (Cited on page 85.)

[64] KÜSTENMACHER, A., AKHTAR, N., PLÖGER, P. G., AND LAKEMEYER, G. 2013. Unexpected Situations in Service Robot Environment: Classification and Reasoning Using Naive Physics. In *17th annual RoboCup International Symposium*. Eindhoven, Netherlands. (Cited on page 59.)

[65] LAKEMEYER, G. AND LEVESQUE, H. J. 2004. Situations, si! situation terms, no! In *International Conference on Principles of Knowledge Representation and Reasoning (KR)*. 516–526. (Cited on pages 33 and 34.)

[66] LAKEMEYER, G. AND LEVESQUE, H. J. 2005. Semantics for a useful fragment of the situation calculus. In *Proceedings of the 19th international joint conference on Artificial intelligence (IJCAI)*. 490–496. (Cited on page 33.)

[67] LEVESQUE, H. J., REITER, R., LESPERANCE, Y., LIN, F., AND SCHERL, R. B. 1997. Golog: A logic programming language for dynamic domains. *The Journal of Logic Programming 31,* 1, 59–83. (Cited on page 21.)

[68] LIN, F. AND REITER, R. 1997. How to progress a database. *Artificial Intelligence 92,* 1, 131 – 167. (Cited on pages 14 and 15.)

[69] LIU, Y. AND LAKEMEYER, G. 2009. On first-order definability and computability of progression for local-effect actions and beyond. In *Twenty-First International Joint Conference on Artificial Intelligence (IJCAI-09)*. 860–866. (Cited on page 15.)

[70] LIU, Y., LAKEMEYER, G., AND LEVESQUE, H. J. 2004. A logic of limited belief for reasoning with disjunctive information. In *International Conference on Principles of Knowledge Representation and Reasoning (KR)*. 587–597. (Cited on pages 34 and 35.)

[71] LOIGGE, S. 2016. Unified and dependable robot control architecture based on ros. M.S. thesis, Faculty of Computer Science and Biomedical Engineering, Graz University of Technology. (Cited on page 2.)

[72] LUSSIER, B., CHATILA, R., INGRAND, F., KILLIJIAN, M.-O., AND POWELL, D. 2004. On Fault Tolerance and Robustness in Autonomous Systems. In 3$^{rd}$ *IARP - IEEE/RAS - EURON Joint Workshop on Technical Challenges for Dependable Robots in Human Environments*. Manchester, UK. (Cited on page 1.)

[73] LUSSIER, B., LAMPE, A., CHATILA, R., GUIOCHET, J., INGRAND, F., KILLIJIAN, M.-O., AND POWELL, D. 2005. Fault Tolerance in Autonomous Systems: How and How Much? In 4$^{th}$ *IARP - IEEE/RAS - EURON Joint Workshop on Technical Challenges for Dependable Robots in Human Environments*. Nagoya, Japan. (Cited on page 1.)

[74] MA, J., LIU, W., AND MILLER, P. 2012. Belief change with noisy sensing in the situation calculus. *arXiv preprint arXiv:1202.3743*. (Cited on page 32.)

[75] MAIER, C. 2015. YAGI - An Easy and Light-Weighted Action-Programming Language for Education and Research in Artificial Intelligence and Robotics. M.S. thesis, Faculty of Computer Science, Graz University of Technology. (Cited on page 118.)

[76] MARTIN, Y. AND THIELSCHER, M. 2001. Addressing the qualification problem in flux. In *KI 2001: Advances in Artificial Intelligence*. Springer, 290–304. (Cited on page 29.)

[77] MATUSZEK, C., CABRAL, J., WITBROCK, M., AND DEOLIVEIRA, J. 2006. An introduction to the syntax and content of Cyc. In *Proceedings of the 2006 AAAI Spring Symposium on Formalizing and Compiling Background Knowledge and Its Applications to Knowledge Representation and Question Answering*. 44–49. (Cited on pages 53, 54, and 57.)

[78] MCCARTHY, J. 1963. Situations, actions, and causal laws. Tech. Rep. AD0785031, STANFORD UNIV CA DEPT OF COMPUTER SCIENCE. 7. (Cited on pages 9 and 10.)

[79] MCCARTHY, J. 1980. Circumscription - a form of non-monotonic reasoning. *Artificial intelligence 13,* 1, 27–39. (Cited on page 9.)

[80] MCCARTHY, J. 1986. Applications of circumscription to formalizing common-sense knowledge. *Artificial Intelligence 28,* 1, 89–116. (Cited on page 9.)

[81] MCCARTHY, J. 1987. Epistemological problems of artificial intelligence. *Readings in Artificial Intelligence*, 459. (Cited on page 9.)

[82] MCCARTHY, J. AND HAYES, P. J. 1969. Some philosophical problems from the standpoint of artificial intelligence. *Readings in artificial intelligence*, 431–450. (Cited on page 9.)

[83] MCILRAITH, S. A. 1999a. Explanatory diagnosis: Conjecturing actions to explain observations. In *Logical Foundations for Cognitive Agents: Papers in Honour of Ray Reiter*. Artificial Intelligence. Springer, Berlin, Heidelberg, 155–172. (Cited on page 17.)

[84] MCILRAITH, S. A. 1999b. Model-based programming using golog and the situation calculus. In *Proceedings of the Tenth International Workshop on Principles of Diagnosis (DX 99)*. 184–192. (Cited on page 28.)

[85] MCLLRAITH, S. A. 1999. Explanatory diagnosis: Conjecturing actions to explain observations. In *Logical Foundations for Cognitive Agents*. Springer, 155–172. (Cited on pages 28, 29, and 51.)

[86] MOORE, R. C. 1984. A formal theory of knowledge and action. Tech. Rep. ADA458917, SRI INTERNATIONAL MENLO PARK CA ARTIFICIAL INTELLIGENCE CENTER. (Cited on pages 16 and 35.)

[87] MÖSENLECHNER, L. AND BEETZ, M. 2009. Using physics- and sensor-based simulation for high-fidelity temporal projection of realistic robot behavior. In *19th International Conference on Automated Planning and Scheduling*. (Cited on pages 121 and 122.)

[88] MÜHLBACHER, C., GSPANDL, S., REIP, M., AND STEINBAUER, G. 2016. Improving dependability of industrial transport robots using model-based techniques. In *IEEE International Conference on Robotics and Automation (ICRA), 2016*. IEEE, 3133–3140. (Cited on page 2.)

[89] MÜHLBACHER, C., GSPANDL, S., REIP, M., AND STEINBAUER, G. 2017a. Adapting edge weights for optimal paths in a navigation graph. In *International Conference on Robotics in Alpe-Adria Danube Region*. Springer, Cham, 372–380. (Cited on page 2.)

[90] MÜHLBACHER, C., GSPANDL, S., REIP, M., AND STEINBAUER, G. 2017b. Estimation of the traversal time for a fleet of industrial transport robots. In *International Conference on Robotics in Alpe-Adria Danube Region*. Springer, Cham, 363–371. (Cited on page 2.)

[91] MÜHLBACHER, C. AND STEINBAUER, G. 2014a. Active diagnosis for agents with belief management. In *25th International Workshop on Principles of Diagnosis (DX)*. [online; accessed 13.9.2017]. (Cited on pages 7 and 81.)

[92] MÜHLBACHER, C. AND STEINBAUER, G. 2014b. Knowledge-aware execution of programs in indigolog. In *The 9th International Workshop on Cognitive Robotics*. [online; accessed 13.9.2017]. (Cited on pages 7 and 61.)

[93] MÜHLBACHER, C. AND STEINBAUER, G. 2014c. Using common sense invariants in belief management for autonomous agents. In *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*. Springer International Publishing, 49–59. (Cited on pages 6, 54, and 119.)

[94] MÜHLBACHER, C. AND STEINBAUER, G. 2016a. Belief management using the action history and consistency-based-diagnosis. In *27th International Workshop on Principles of Diagnosis (DX)*. [online; accessed 13.9.2017]. (Cited on pages 6 and 40.)

[95] MÜHLBACHER, C. AND STEINBAUER, G. 2016b. Diagnosis makes the difference for a successful execution of high-level robot control programs. In *Intelligent Autonomous Systems 13*. Springer International Publishing, 1119–1132. (Cited on pages 7, 119, and 120.)

[96] MÜHLBACHER, C., STEINBAUER, G., GSPANDL, S., AND REIP, M. 2017. Model-based testing of an industrial multi-robot navigation system. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, 1652–1654. (Cited on page 2.)

[97] NICA, I., PILL, I., QUARITSCH, T., AND WOTAWA, F. 2013. The route to success-a performance comparison of diagnosis algorithms. In *Twenty-Third International Joint Conference on Artificial Intelligence (IJCAI)*. Vol. 13. 1039–1045. (Cited on page 44.)

[98] NILSSON, N. J. 1994. Teleo-reactive programs for agent control. *Journal of Artificial Intelligence Research 1,* 1, 139–158. (Cited on pages 123 and 133.)

[99] PAGNUCCO, M., RAJARATNAM, D., STRASS, H., AND THIELSCHER, M. 2013. Implementing belief change in the situation calculus and an application. In *International Conference on Logic Programming and Nonmonotonic Reasoning*. Springer, 439–451. (Cited on page 31.)

[100] PEDNAULT, E. P. 1989. ADL: Exploring the middle ground between strips and the situation calculus. In *Proceedings of the First International Conference on Principles of Knowledge*

*Representation and Reasoning (KR'89)*. Morgan Kaufmann Publishers, Inc, 324–332. (Cited on page 10.)

[101] PETRICK, R. P. 2008. Cartesian situations and knowledge decomposition in the situation calculus. In *Proceedings of the Eleventh International Conference on Principles of Knowledge Representation and Reasoning*. 629–639. (Cited on page 35.)

[102] PETRICK, R. P. AND BACCHUS, F. 2002. A knowledge-based approach to planning with incomplete information and sensing. In *Proceedings of the Sixth International Conference on Artificial Intelligence Planning Systems (AIPS)*. 212–222. (Cited on page 36.)

[103] PETRICK, R. P. AND BACCHUS, F. 2004. Extending the knowledge-based approach to planning with incomplete information and sensing. In *International Conference on Principles of Knowledge Representation and Reasoning (KR)*. 613–622. (Cited on page 36.)

[104] PETRICK, R. P. AND LEVESQUE, H. J. 2002. Knowledge equivalence in combined action theories. In *International Conference on Principles of Knowledge Representation and Reasoning*. 303–314. (Cited on page 35.)

[105] PIRRI, F. AND REITER, R. 1999. Some contributions to the metatheory of the situation calculus. *Journal of the ACM (JACM) 46,* 3, 325–361. (Cited on pages 13 and 14.)

[106] PODESSER, S., STEINBAUER, G., AND WOTAWA, F. 2012. Selective belief management for high-level robot programs. In *Proceedings of the International Workshop on Principles of Diagnosis (DX-12)*. (Cited on page 141.)

[107] QUIGLEY, M., CONLEY, K., GERKEY, B. P., FAUST, J., FOOTE, T., LEIBS, J., WHEELER, R., AND NG, A. Y. 2009. ROS: an open-source Robot Operating System. In *ICRA Workshop on Open Source Software*. (Cited on page 125.)

[108] RAJARATNAM, D., LEVESQUE, H. J., PAGNUCCO, M., AND THIELSCHER, M. 2014. Forgetting in action. *International Conference on Principles of Knowledge Representation and Reasoning (KR)*. (Cited on page 51.)

[109] REIP, M. 2016. Dependable belief management for high-level robot programs. Ph.D. thesis, Institute for Software Technology, Graz University of Technology, Graz Austria. (Cited on pages 20, 31, 63, 124, and 125.)

[110] REIP, M., STEINBAUER, G., AND FERREIN, A. 2012a. Improving belief management for high-level robot programs by using diagnosis templates. In *International Workshop on Principles of Diagnosis (DX), Great Malvern, UK*. (Cited on page 31.)

[111] REIP, M., STEINBAUER, G., AND FERREIN, A. 2012b. Improving belief management for high-level robot programs by using diagnosis templates. In *Proceedings of the International Workshop on Principles of Diagnosis (DX-12)*. (Cited on page 140.)

[112] REITER, R. 1980. A logic for default reasoning. *Artificial intelligence 13,* 1, 81–132. (Cited on pages 48 and 49.)

[113] REITER, R. 1987. A theory of diagnosis from first principles. *Artificial Intelligence 32,* 1 (Apr.), 57–95. (Cited on pages 6, 17, 40, 44, and 51.)

[114] REITER, R. 1991a. Artificial intelligence and mathematical theory of computation. Academic Press Professional, Inc., San Diego, CA, USA, Chapter The frame problem in situation the calculus: a simple solution (sometimes) and a completeness result for goal regression, 359–380. (Cited on pages 9, 10, and 13.)

[115] REITER, R. 1991b. The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. *Artificial intelligence and mathematical theory of computation: papers in honor of John McCarthy 27*, 359–380. (Cited on page 52.)

[116] REITER, R. 2001. *Knowledge in Action. Logical Foundations for Specifying and Implementing Dynamical Systems*. MIT Press, Cambridge, Massachusetts. (Cited on pages 11, 13, 15, 16, 18, and 43.)

[117] ROOS, N. AND WITTEVEEN, C. 2005. Diagnosis of plans and agents. In *Multi-Agent Systems and Applications IV*. Springer, 357–366. (Cited on pages 41, 51, and 52.)

[118] SARDINA, S. 2005. Deliberation in agent programming languages. Ph.D. thesis, University of Toronto. (Cited on pages 21, 24, and 52.)

[119] SCHERL, R. B. AND LEVESQUE, H. J. 1993. The frame problem and knowledge-producing actions. In *In Proceedings of the Eleventh National Conference on Artificial Intelligence*. AAAI Press/The MIT Press, 689–695. (Cited on pages 15 and 31.)

[120] SCHERL, R. B. AND LEVESQUE, H. J. 2003. Knowledge, action, and the frame problem. *Artificial Intelligence 144,* 1–2 (Mar.), 1–39. (Cited on pages xiii, 15, and 16.)

[121] SCHIFFEL, S. AND THIELSCHER, M. 2005. Interpreting golog programs in flux. In *7th International Symposium On Logical Formalizations of Commonsense Reasoning, The*. (Cited on page 29.)

[122] SCHUBERT, L. K. 1990. Monotonic solution of the frame problem in the situation calculus: An efficient method for worlds with fully specified actions. In *Knowledge Representation and Defeasible Reasoning*, H. E. Kyburg, R. P. Loui, and G. N. Carlson, Eds. Vol. Volume 5. Kluwer Academic Publishers, Dordrecht / Boston / London, 23–67. (Cited on page 10.)

[123] SCHWERING, C. 2016. Conditional beliefs in action. Ph.D. thesis, Ph. D. dissertation, RWTH Aachen University. (Cited on page 34.)

[124] SCHWERING, C. AND LAKEMEYER, G. 2014. A semantic account of iterated belief revision in the situation calculus. In *Proceedings of the Twenty-first European Conference on Artificial Intelligence (ECAI)*. 801–806. (Cited on pages 33 and 34.)

[125] SCHWERING, C. AND LAKEMEYER, G. 2015. Projection in the epistemic situation calculus with belief conditionals. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI)*. 1583–1589. (Cited on page 34.)

[126] SCHWERING, C., LAKEMEYER, G., AND PAGNUCCO, M. 2015. Belief revision and progression of knowledge bases in the epistemic situation calculus. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI)*. (Cited on page 34.)

[127] SHAPIRO, S. 2006. Belief change with noisy sensing and introspection. In *Belief Change in Rational Agents: Perspectives from Artificial Intelligence, Philosophy, and Economics*, J. Delgrande, J. Lang, H. Rott, and J.-M. Tallon, Eds. Number 05321 in Dagstuhl Seminar Proceedings. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, Dagstuhl, Germany. (Cited on page 32.)

[128] SHAPIRO, S. AND PAGNUCCO, M. 2004. Iterated belief change and exogenous actions in the situation calculus. In *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI)*. Vol. 16. 878. (Cited on page 32.)

[129] SHAPIRO, S., PAGNUCCO, M., LESPÉRANCE, Y., AND LEVESQUE, H. J. 2000. Iterated belief change in the situation calculus. In *International Conference on Principles of Knowledge Representation and Reasoning (KR)*. 527–538. (Cited on pages 31, 32, 33, and 34.)

[130] SOUTCHANSKI, M. 2001. An on-line decision-theoretic golog interpreter. In *International Joint Conference on Artificial Intelligence (IJCAI)*. 19–26. (Cited on page 36.)

[131] STEINBAUER, G., LOIGGE, S., AND MÜHLBACHER, C. 2016. Supervision of hardware, software and behavior of autonomous industrial transport robots. In *IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C), 2016*. IEEE, 298–300. (Cited on page 2.)

[132] STEINBAUER, G. AND MÜHLBACHER, C. 2014. Using common sense invariants in belief management for autonomous agents. In *2014 AAAI Spring Symposium Series*. Association for the Advancement of Artificial Intelligence (www.aaai.org), 63–70. (Cited on pages 6, 47, 54, and 119.)

[133] STEINBAUER, G. AND MÜHLBACHER, C. 2016. Hands off - a holistic model-based approach for long-term autonomy. In *ICRA 2016 Workshop: AI for Long-term Autonomy*. [online; accessed 13.9.2017]. (Cited on page 2.)

[134] STEINBAUER, G. AND WOTAWA, F. 2010. On the way to automated belief repair for autonomous robots. In *21st International Workshop on Principles of Diagnosis (DX-10)*. (Cited on pages 30 and 31.)

[135] TECHNICAL COMMITTEE ISO. 2011. ISO 26262 Compliance Achieving Functional Safety for Automotive E/E Systems. [online; accessed 13.9.2017]. (Cited on page 1.)

[136] TENORTH, M. AND BEETZ, M. 2009. KnowRob - Knowledge Processing for Autonomous Personal Robots. In *IEEE International Conference on Intelligent Robots and Systems*. (Cited on page 120.)

[137] TENORTH, M. AND BEETZ, M. 2013. Knowrob: A knowledge processing infrastructure for cognition-enabled robots. *The International Journal of Robotics Research 32,* 5, 566–590. (Cited on page 93.)

[138] THIELSCHER, M. 1998a. Introduction to the fluent calculus. (Cited on pages 29 and 30.)

[139] THIELSCHER, M. 1998b. Reasoning about actions: Steady versus stabilizing state constraints. *Artificial Intelligence 104,* 1, 339–355. (Cited on page 57.)

[140] THIELSCHER, M. 1999. From situation calculus to fluent calculus: State update axioms as a solution to the inferential frame problem. *Artificial intelligence 111,* 1, 277–299. (Cited on pages 29 and 55.)

[141] THIELSCHER, M. 2000. Representing the knowledge of a robot. In *International Conference on Principles of Knowledge Representation and Reasoning (KR)*. 109–120. (Cited on page 30.)

[142] THIELSCHER, M. 2001. The qualification problem: A solution to the problem of anomalous models. *Artificial Intelligence 131,* 1, 1–37. (Cited on pages 29 and 30.)

[143] THIELSCHER, M. 2005. Flux: A logic programming method for reasoning agents. *Theory and Practice of Logic Programming 5,* 4-5, 533–565. (Cited on pages 29 and 30.)

[144] THRUN, S., BENNEWITZ, M., BURGARD, W., CREMERS, A. B., DELLAERT, F., FOX, D., HAHNEL, D., ROSENBERG, C., ROY, N., SCHULTE, J., ET AL. 1999. Minerva: A second-generation museum tour-guide robot. In *IEEE International Conference on Robotics and Automation (ICRA)*. Vol. 3. IEEE. (Cited on page 1.)

[145] TREBI-OLLENNU, A. 2006. Special Issue on Robots on the Red Planet. *IEEE Robotics & Automation Magazine 13,* 2. (Cited on page 1.)

[146] URMSON, C., BAKER, C., DOLAN, J., RYBSKI, P., SALESKY, B., WHITTAKER, W., FERGUSON, D., AND DARMS, M. 2009. Autonomous driving in traffic: Boss and the urban challenge. *AI Magazine 30,* 2, 17. (Cited on page 1.)

[147] VASSOS, S. 2009. A reasoning module for long-lived cognitive agents. Ph.D. thesis, University of Toronto, Toronto, Canada. (Cited on pages 15, 68, and 71.)

[148] VASSOS, S., LAKEMEYER, G., AND LEVESQUE, H. J. 2008. First-order strong progression for local-effect basic action theories. In *Eleventh International Conference on Principles of Knowledge Representation and Reasoning*. 662–672. (Cited on page 15.)

[149] VASSOS, S. AND LEVESQUE, H. J. 2013. How to progress a database iii. *Artificial Intelligence 195*, 203–221. (Cited on page 14.)

[150] VASSOS, S. AND PATRIZI, F. 2013. A classification of first-order progressable action theories in situation calculus. In *Twenty-Third International Joint Conference on Artificial Intelligence (IJCAI)*. (Cited on page 15.)

[151] WAIBEL, M., BEETZ, M., CIVERA, J., D'ANDREA, R., ELFRING, J., GALVEZ-LOPEZ, D., HAUSSERMANN, K., JANSSEN, R., MONTIEL, J., PERZYLO, A., SCHIESSLE, B., TENORTH, M., ZWEIGLE, O., AND VAN DE MOLENGRAFT, R. 2011. RoboEarth — A World Wide Web for Robots. *Robotics Automation Magazine, IEEE 18,* 2, 69–82. (Cited on page 120.)

[152] WALDINGER, R. 1981. Achieving several goals simultaneously. *Readings in artificial intelligence*, 250–271. (Cited on page 13.)

[153] WEIDENBACH, C., BRAHM, U., HILLENBRAND, T., KEEN, E., THEOBALD, C., AND TOPIĆ, D. 2002. Spass version 2.0. In *International Conference on Automated Deduction*. Springer, 275–279. (Cited on page 116.)

[154] WITTEVEEN, C., ROOS, N., VAN DER KROGT, R., AND DE WEERDT, M. 2005. Diagnosis of single and multi-agent plans. In *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*. ACM, 805–812. (Cited on page 52.)

[155] WURMAN, P. R., D'ANDREA, R., AND MOUNTZ, M. 2007. Coordinating hundreds of co-operative, autonomous vehicles in warehouses. In *Proceedings of the 19th national conference on Innovative applications of artificial intelligence - Volume 2*. IAAI'07. AAAI Press, 1752–1759. (Cited on pages 1 and 7.)