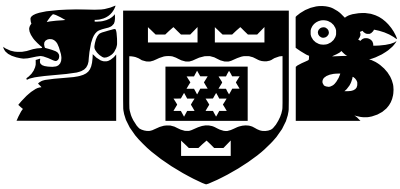
VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wānanga o te Ūpoko o te Ika a Māui



School of Engineering and Computer Science
Te Kura Mātai Pūkaha, Pūrorohiko

PO Box 600
Wellington
New Zealand

Tel: +64 4 463 5341
Fax: +64 4 463 5045
Internet: office@ecs.vuw.ac.nz

Autonomously Learning About Meaningful Actions from Exploratory Behaviour

Heidi Newton

Supervisor: Peter Andreae

Submitted in partial fulfilment of the requirements for
Master of Computer Science.

Abstract

The thesis addresses the problem of creating an autonomous agent that is able to learn about and use meaningful hand motor actions in a simulated world with realistic physics, in a similar way to human infants learning to control their hand. A recent thesis by Mugan presented one approach to this problem using qualitative representations, but suffered from several important limitations. This thesis presents an alternative design that breaks the learning problem down into several distinct learning tasks. It presents a new method for learning rules about actions based on the Apriori algorithm. It also presents a planner inspired by infants that can use these rules to solve a range of tasks. Experiments showed that the agent was able to learn meaningful rules and was then able to successfully use them to achieve a range of simple planning tasks.

Acknowledgments

This thesis is dedicated to my father who would have been so proud I have completed it.

Thankyou my wonderful cat, Java, for all the purrrrrrrrrrrrrrrrrs, meows, cuddles, and endless support.

Also, thanks to Pondy for being an awesome and supportive thesis supervisor and having many useful discussions with me about the different ways the agent could be implemented.

Contents

Chapter 1

Introduction

An important problem in artificial intelligence (AI) is building systems that are able to learn autonomously without the help of a teacher. When trying to design such a system, one source of inspiration is human infants. In their first 6 - 12 months of life, they learn their first knowledge of the world largely by themselves with little direct help from their parents or other teacher. In a short period, they go from waving their arms around in a seemingly meaningless fashion to being able to successfully reach for, grasp, and bring in an object to put in their mouth. Psychologists have carried out various observational experiments with infants in order to come up with hypotheses to explain this learning resulting in some very useful findings for AI researchers. For example, they have found that infants approach the problem of reaching for an object as a problem solving task with large variance between different infants [28], and that infants seem to understand events qualitatively before they do quantitatively [3]. It is of course difficult, if not impossible, to draw certain conclusions about how infants are actually learning for at least two reasons. Firstly, infants are unable to communicate with the researchers to explain why they are doing what they are. Secondly, knowledge of how even the adult human brain learns is very limited. Because of this, the results of observations are at best open for interpretation and psychologists don't yet have a good explanatory theory of how infants learn to control their hands.

In addition to psychologists, artificial intelligence researchers have also explored how infants learn by building computer systems that attempt to learn very simple tasks like what an infant would. [14] and [11] give an overview of some of these systems. Despite the lack of solid conclusions about how infants learn, the psychology research is still valuable as inspiration for these systems. Approaches can be based on possible interpretations of observations, and observations of what infants can and cannot do can be taken into account when determining what one of these computer systems should and should not be able to do. The resulting computational models from the AI researchers' work provide further insight into what is and is not computationally possible, which may be able to help psychologists to further understand their observations.

One of the more recent attempts by AI researchers to make a system that can learn meaningful actions autonomously from random behaviour is the work of Mugan and Kuipers [22]. Their objective was to build a simulated agent that could autonomously learn to achieve simple goals given only a set of continuous motor and world variables as input. Their system, QLAP, is able to autonomously learn to reach for an object, among other simple tasks, learning from random behaviour. A key feature of QLAP is that it uses a qualitative representation, converting the set of continuous motor and sensor variables describing the world into qualitative variables using "landmark" values to separate the continuous values into meaningful qualitatively distinct ranges. It learns actions in the form of contingencies specifying when one change to a variable, is likely to be followed by another change

to a variable.

However, QLAP has many serious limitations and would be very difficult to further build on. While using a qualitative representation for learning about higher level concepts such as actions and events has many advantages, a purely quantitative approach far better models the complex relationships between variables in a realistic motor control simulation. QLAP is only able to reason about one variable at a time, and landmarks can only be learnt on single variables. Therefore relationships between variables cannot be identified. One consequence is that the agent could not be extended to work with round objects as opposed to square objects, as round objects have a complex relationship between the surface and the dimensions of the object that the landmark learning on single variables cannot represent. Another consequence is that QLAP would not work with a realistic model of an arm because a real human infant's arm has many muscles that must be coordinated with one another in order to carry out arm movement actions; QLAP could not handle these coordinations. A further limitation is that mapping motor control directly onto higher level actions with contingencies does not allow understanding and reasoning about movement and events during an action. Infants learn to master simple motor actions such as moving to an object, grasping it, and bringing it back to their mouth. These motor actions and others are also used to explore and learn about the environment and objects within it. These motor actions are not directly modelled in Mugan's system, limiting the ability for the system to focus on exploring the environment without constantly relearning the motor control to do so.

The goal of the thesis is to explore how an agent can learn to achieve goals in a similar realistic world to QLAP but avoid some of the serious limitations of QLAP.

The thesis outlines an approach to the learning problem, breaking it down into several interacting components, unlike QLAP which attempted to do too much as a single learning component. The main focus of the thesis is on learning rules to describe the various effects of meaningful motor actions and then using the rules to plan for simple goals. A brief exploration on how motor control could be learnt from initially random behaviour was also carried out and is presented in the thesis. A visual system that converts continuous world states into simple qualitative representations was hard-coded. A motor control policy for the rule learner and planner to use was also hard-coded. The development of a visual system learner and motor control policy learner that will provide suitable input for the rule learner and planner is left to future work, as this work is for a master's thesis and therefore it was not realistic to implement an entire agent in the time frame.

The Action Rule learner learns with a qualitative representation of the world, whereas the vision system and motor control learner are assumed to learn quantitatively. This is unlike Mugan's system which attempted to learn everything qualitatively. The learner learns Action Rules with multiple effects using a frequent itemset generation algorithm. This simplifies planning, because a means-end analysis planner is then often able to achieve a goal set with multiple facts using a single Action Rule. The first version of the Action Rule learner had significant limitations. By introducing explicit places and constraints on places into the Action Rules overcame these limitations and enabled the planner to achieve a wider range of goals.

Chapter 2 explains Mugan's system and its limitations, and then gives an overview of the key ideas in psychology that were used, and concludes with key algorithms that were used in the implementation for this thesis. Chapter 3 gives an overview of the agent implemented for the thesis and outlines the overall system architecture. Chapter 4 presents an initial exploration on learning motor actions from random behaviour. Chapter 5 presents the Action Rule learner, the key work of the thesis. Chapter 6 presents the Action Rule planner, which uses the learnt Action Rules to achieve simple goals. Chapter 7 describes improvements to the Action Rule learner, in particular the addition of places and constraints. Chapter 8 eval-

uates the Action Rule learner and Action Rule planner. Finally, Chapter 9 concludes with the main findings and future work.

Chapter 2

Background

2.1 QLAP - Qualitative Learner and Planner

This thesis was inspired by Mugan's QLAP system [15, 22, 21, 20, 18, 19, 17, 16]. QLAP was able to learn action rules for a simulated arm argent in a world, and then carry out actions such as grasping or hitting an object. QLAP is different to most systems attempting to learn like an infant in that it uses a qualitative representation of the world, and explored in order to identify contingencies in events which were the basis of rules. However, QLAP made some strong and unrealistic assumptions about the world which lead to strong limitations.

The QLAP agent assumes a set of continuous motor and world (referred to as "magnitude" variables by Mugan) as input. From these variables, QLAP's objective is to learn to achieve simple tasks such as grasping and hitting objects. This was achieved by applying forces to motor variables and learning how these in turn cause changes in world variables. The QLAP agent describes its experience using a qualitative representation, and identifies "events" as changes in the qualitative values of the values. QLAP then learns rules for causing each possible event (each variable into each of its qualitative regions) so that it could chain events together in order to control world variables. This was achieved by learning rules that described a contingent pair of events such that if the first event occurred, the consequent event would soon occur.. These rules were represented as Dynamic Bayesian Networks (DBN) and then converted into plans which were represented as Markov Decision Process policies which the QLAP agent could then execute. The remainder of this section goes into further details of the algorithm and then finishes with a discussion of the limitations.

2.1.1 Representing the world qualitatively

QLAP's input is a set of continuous motor and magnitude (world) variables. Each variable has a corresponding direction of change variable which can have a value of zero, positive, or negative. Motor and magnitude variables are broken up into qualitative regions by defining *land marks*, values on the continuous variables that QLAP believes to be significant. All landmarks are qualitative zones, as are the regions around landmarks. For example, a variable with one landmark has three qualitative regions (the landmark and the two regions either side of it) and a variable with three landmarks has seven qualitative regions (the three landmarks and four regions surrounding them)

2.1.2 Learning DBN's for rules

QLAP learns Dynamic Bayesian Networks (DBN's) for predicting events on direction of change variables ("change DBN's") and magnitude variables ("magnitude DBN's"). Change DBN's are learnt by identifying contingencies between events and checking for additional context variables, and magnitude DBN's are describe using the corresponding direction of change variable to a magnitude variable in order to cause an events on the magnitude variable. These DBN's are the first step in building QLAP's representation of action rules.

An event X $\rightarrow$ x is defined as being when a qualitative variable reaches a certain value. As an example, if a variable denoting the hand position went to touching the right wall, the event that occurred would be represented as "hand position" $\rightarrow$ "right wall". A contingency occurs when an event has a significantly higher probability of occurring "soon" after another than it does otherwise. The initial event is known as the antecedent event, and the following event is known as the consequent event. To be "soon", the consequent event must occur within k time steps of the antecedent event, where k is a constant used for all contingencies.

In order to identify suitable contingencies for change DBN's, QLAP stores and uses a pairwise search on statistics for possible pairs of events. As change DBN's are only concerned with how to control direction of change variables, the consequent event must be on a direction of change variable. Once a contingency has been identified through the pairwise search, a DBN is formed for the contingency. The antecedent event becomes the parent node of the DBN, and the consequent event becomes a single child node off the parent node.

QLAP also builds a second type of DBN for the magnitude variables: for every possible event on every qualitative magnitude variable, QLAP creates two DBN's that have the qualitative magnitude variable becoming the event value as the consequent event, and with the corresponding direction of change variable as the antecedent event. One of the DBN's has the direction of change variable set to negative, and the other set to positive.. These DBN's are built automatically and are not learnt and are used in the planning process in the same way in as the change DBN's.

2.1.3 Improving the reliability of the DBN's

QLAP has two mechanisms for improving the reliability of a DBN: the addition of context variables that act as preconditions, and the addition of landmarks to variables characterise the quantitative space. When deciding whether or not adding a context variable to a DBN or a landmark to a variable offers a sufficient improvement in the DBN, QLAP uses several heuristics. For the modification to be accepted, it must improve the reliability of the DBN by either increasing the best reliability or decreasing the overall entropy of the DBN. The amount of improvement must be above a certain threshold.

Best reliability is given by the probability of success of the DBN if it starts in the most reliable context (see below for an explanation of context). QLAP attempts to improve the best reliability until it is above a given threshold before attempting to reduce the overall entropy (Mugan used a threshold of 0.75 in his experiments). When the DBN is used for planning, the planner will attempt to make the context of the best reliability of a DBN true before using the DBN. Decreasing the overall entropy of the DBN increases how predictable the DBN is.

Additionally, a context is learnt to better allow for predicting when the consequent event will follow the antecedent event. To do this, QLAP tries adding each variable in the world to the DBN's context, and chooses the one that best improves the DBN and is above the threshold for sufficient improvement (if any). If there are no suitable context variables to be added, then the DBN stays how it was.

Context variables are represented in the DBN using a Conditional Probability Table (CPT). A CPT holds the probabilities of the consequent event for each combination of variable values. As a result, the amount of space a CPT takes up is exponential in the number of context variables of the DBN. For this reason, the number of context variables a DBN can handle is very restricted. Mugan set a limit of two context variables for his experiments.

QLAP's other DBN improvement mechanism is learning new landmarks on the quantitative variables to make the way they are discretized into qualitative variables more meaningful to the context of the world the QLAP agent is in.

2.1.4 Building plans from DBN's for accomplishing actions

In QLAP, an action is defined as making an event occur. QLAP defines a set of actions; one for every possible event (each qualitative value of every variable). An action does not contain a specification of how to accomplish the action; for this, QLAP has plans. Plans are generated from reliable DBN's; a DBN forms a plan for the action that is the consequent event. For magnitude variable actions, there will always be two plans, one for accomplishing the action when on the left side of the desired value, and one for when on the right side of the desired value. For change variable actions, there can be zero, one, or many plans, depending on what contingencies QLAP was able to learn for achieving that action (and whether or not they were reliable). Motor variables do not require plans to accomplish them as the QLAP agent is able to directly control them.

Plans are implemented as options, containing a Markov Decision Process (MDP) to represent the possible states within the plan (consisting of the antecedent event variable, consequent event variable, and context variables of the DBN the MDP is based on). A policy is learnt for the MDP, which tells the QLAP agent what to do in each state to have the best chance of accomplishing the goal with the minimum number of transitions required to get from the current state to the antecedent event, as in theory the consequent event should follow, thus successfully completing the action.

Once plans have been formed, QLAP can be given an action to accomplish. To accomplish the action, QLAP looks for a suitable plan for that action. If no such plan exists, the action fails. If a plan is found, QLAP uses that plan to accomplish the action. Often, a plan will contain other actions that need to be accomplished to accomplish the current plan. For these new actions, QLAP searches for a suitable plan. This continues in a recursive nature, building up a hierarchy of plans, until the ends of all paths through the hierarchy contain a motor action. QLAP can then directly carry out each of the motor actions (changing motor variable values), and if all plans work as expected, the action should be successfully completed.

2.1.5 Limitations of QLAP

QLAP makes some strong assumptions about the world which are not realistic, and therefore lead to some strong limitations.

QLAP assumes the world is represented as a set of low level motor and world variables and that it should learn qualitative abstractions on single variables rather than combining the variables. As a consequence, events also can only involve a single variable, and QLAP learns rules with a single effect. Even though the preconditions can contain more than one variable, the planner has to plan to satisfy each of the preconditions separately and cannot look for ways to achieve multiple events or preconditions using a single motor action.

The precondition learning mechanism used is restrictive because it has to consider all known variables and their qualitative values in order to determine whether or not they

should be included. This makes it intractable. Mugan's experiments had to limit the agent to learning two context variables per rule.

A surprising limitation of treating variables independently and not learning relationships between them is that QLAP cannot learn with spherical objects. Determining whether or not a point is touching a sphere requires using a formula that considers the relative positions in each dimension. QLAP is of course unable to learn about such relationships as they require combining variables.

Another implication of QLAP doing all learning qualitatively and treating variables independently is that its motor actions consist only of applying a positive or negative force to a single motor variable until a world variables reaches a qualitative value. QLAP cannot characterise the movement during the action with this action representation. Therefore, QLAP cannot understand and distinguish between different actions such as moving and hitting.

In order for QLAP to reach for an object, it has to consider its relative position to the object and plan in each dimension separately. This is not realistic to a real infant who coordinates many muscles with carefully controlled amounts of force on each one in order to achieve motor actions that are meaningful and direct for interacting with the world. QLAP would have to move each muscle one at a time, which is impossible.

In order to address this limitation, it is necessary to separate learning motor control for actions and learning effects and preconditions for action. While using a qualitative abstraction is invaluable for learning effects and preconditions for actions, it is unsuitable for learning motor control. Therefore, the agent should initially learn how to perform actions using a quantitative process that learns to coordinate muscles in order to achieve meaningful higher level actions such as moving and hitting and then it should learn the effects and preconditions for the actions using qualitative abstractions.

The system presented in the thesis addresses these limitations in three ways:

1. It assumes an underlying visual system was learnt that provides a qualitative abstraction on top of the low level world variables, which can express relationships between variables where necessary.
2. It assumes a motor control policy is learnt that combines lower level motor variables in order to provide the agent with meaningful motor actions such as moving and hitting. This separates knowledge of how to perform actions from knowledge of what effects they have on the world.
3. It learns rules that have coordinated effects, uses a precondition learner that does not suffer from intractability, and a planner that utilises multi-effect rules to plan for multiple goals or unsatisfied preconditions.

2.2 Developmental Psychology

2.2.1 Visual perception of the world

The psychology literature suggests newborn infants already have a visual system that is able to identify and distinguish between simple shapes such as squares, circles, triangles, and crosses [26, 10], and within the first few months distinguish objects and recognise spatial relationships between them such as above and below, [24, 8]. A little later, they are able to recognise more complicated relationships such occlusion [2], containment [12] and support. [5]

It is therefore reasonable for this thesis to assume the existence of a visual system that is able to construct a simple qualitative description of the world. The thesis does not consider

the problem of learning visual perception, so a hard coded, non-learning visual system was used. Assumptions that seem to be supported by the literature were made, for example the agent is able to recognise round objects and walls as separate entities it can interact with, and it can recognise simple relationships between them and its hand such as behind, touching, and moving. However, it is clear from the psychology literature that a richer world with a visual system able to recognise relationships such as occlusion, containment, and support would be realistic. Because the focus of the thesis was learning rules for action, this level of complexity was not necessary.

2.2.2 Learning about and understanding the physical world

A key area of the psychology literature for this thesis is how infants view, understand, and represent knowledge of the physical world and objects within it.

Baillaregeon has done many studies [4, 3] on infant understanding of the world. Because infants are believed to have difficulty with planning and executing actions, her studies involve observing infants watching events with their eyes rather than carrying them out for themselves. Because infants are known to look at events that are surprising for longer than events that are expected, this allows a mechanism for probing at the understanding infants have. How infants could apply this knowledge to their own deliberation actions and planning is addressed in the next section.

When learning rules to explain the world, Baillaregeon believes, infants initially recognise interesting concepts and variables in an all or nothing fashion. For example, if a cloth has a protrusion underneath it, an infant is not surprised when the cloth is lifted to reveal an object, but is surprised if no object is revealed, or a cloth without a protrusion is lifted to reveal an object. They do not seem to understand that the size of the protrusion in the cloth is related to the size of the object under the cloth. Similar observations have also been made for object support: infants seem to understand that for an object to not fall, at least part of its base needs to be in contact with the supporting object; it is only later that they seem to understand that the amount of the base in contact with the supporting object is significant.

After learning about an initial concept, Baillaregeon believes infants refine it by reasoning about related variables qualitatively and then quantitatively. For example, if infants can see the size of the object before a cover from it is lifted (by placing an identical object next to it), they are surprised if the size is larger than implied by the size of the protrusion. Whereas, if they have no object to compare to, only older infants are unsurprised. Baillaregeon claims this finding suggests having an object to compare to allow a qualitative comparison, whereas not having a comparison object requires a quantitative comparison, which comes at a later age. It isn't clear that this finding does provide evidence for infant ability at quantitative reasoning, the finding may be explainable with improved visual memory allowing the infant to picture the protrusion and then revealed object in their memory and carry out a qualitative comparison between the two.

It is not clear how Baillagreon believes infants determine which variables and concepts in the world are worth attending to, or how they represent such ideas. As this problem is likely to be at least partially a visual system problem, the thesis did not address it and instead assumed the infant begins by viewing the world as time states with a pool of interesting qualitative facts describing information such as whether or not sound or touch is sensed, if the hand is touching or near an object, or whether objects are touching or aligned with one another.

The approach taken in the thesis for learning rules to describe the world is related to Baillaregeon's model. The agent initially learns sets of qualitative changes (i.e. facts that become true or false from the pool of facts) that commonly occur together as effect sets.

This is related to the identifying of all or nothing concepts and variables. The agent then refines its knowledge of the set of effects by learning other qualitative facts as preconditions to predict when the set of effects will occur. This is related to the refining described by Baillaregeon, as the facts used to represent the world are qualitative relationships between the hand and object and between objects.. Learning quantitative preconditions was beyond the scope of the thesis and was not attempted, although could be a good next step.

Gibson's research found that infants learn about events in the physical world and affordances of objects through exploration with their motor capabilities [9]. As infants improve their motor control and learn additional motor skills such as crawling and walking, they are able to further explore and therefore improve their understanding of the world. The Action Rule learning algorithm presented in the thesis takes a related approach in that motor control for a set of straight forward motor actions is assumed to be known, and this motor control is used to interact and explore the world in a non-deliberate fashion.

Needham [23] found that by equipping prereaching infants with "sticky mittens" (which allowed the infants to grab items without having to grasp them), they were able to explore objects in a similar way to how older infants who were already able to reach did. This finding supports the idea that infant understanding of the world improves as new motor skills are obtained, but that similar reasoning is used to interpret the results of exploration.

In addition, Gibson also noted infants are more interested in change in the world than static scenes. Again, this is closely related to the algorithm presented in the thesis: learning of new Action Rules is initiated by the agent identifying sets of qualitative changes occurring in the world. The thesis also proposes that changing sensory data could be used as a motivation to learn meaningful motor actions.

Elsner and Aschersleben [6] found that infants are only able to learn about the effects of actions by observation of a teacher once they are around 12 months old. The thesis focusses on infants under 12 months for inspiration, and therefore assumes the agent is autonomous and learns without the aid of a teacher. Further work looking at how to extend such an agent to learning like an infant over 12 months would need to learn from observation, as this is likely an important learning mechanism for infants in that age group.

2.2.3 Deliberate actions and planning

One limitation of Baillagreon's work is that it only considers infant understanding of observed events, and not of events resulting from their own actions. Infants have been found to struggle with planning tasks although the exact reason for this is unclear and multiple theories exist. Willatts [29, 30] put forward the theory that infants aren't able to effectively plan until they are able to effectively learn about the effects their own motor actions have on the world. His experiments showed that infants of around 7 months are able to plan deliberately to solve problems such as removing a cloth from on top of an object in order to then be able to retrieve the object. The infant's ability to succeed at the task seemed to be dependent on their ability to remove the particular kind of cloth covering the object. This suggests being able to learn the effects of actions and plan requires a certain level of mastery of motor control.

The agent in the thesis follows Willatt's theory in that it learns about the effects of actions it carries out itself rather than effects of actions it observes. This was necessary for planning, as the agent needs to learn what effects its own motor actions can lead to, so that it can then chain motor actions together in a plan that it can execute. Consistent with Willatt, important that the motor control policy used to enable exploring the world and planning had an adequate level of consistency and reliability. The agent's ability to plan with less predictable actions such as hitting was noticeably less than its ability to plan with more

predictable actions such as moving to and grasping.

No psychology literature on the finer details of the planning mechanisms used by infants was found, other than the claim they use Means-End Analysis and can use this for two step plans [29]. According to Fabricius, infants don't use forward search planning, because this seems to be a much later development, first appearing at around 5 years of age. Forward search planning seems to be a much later development, first appearing at around 5 years of age. [7]. Because infants have limited working memory that at best can remember 3 - 4 items at one year of age [25], working memory usage needs to be limited.

The agent in the thesis uses a Means-End Analysis planner that at each step of the planner considers the alternatives and chooses the seemingly best one to commit to. There is no backtracking or searching more than one step back as infant working memory does not allow for this, although the agent is allowed multiple attempts at goal sets or sub goals sets that were not successfully achieved. The planning stack also has a depth limit. This results in a planner that takes an explorative approach to planning, where failures are considered acceptable and normal. This fits in with the exploratory nature of the approach infants seem to take with other learning mechanisms. If anything, the agent's planner is more sophisticated than what is realistic for a human infant.

2.3 Algorithms

2.3.1 The Apriori algorithm

The Action Rule learning component of the thesis uses the Apriori algorithm [27]. The Apriori algorithm is a standard algorithm used in frequent itemset mining.Given a minimum support threshold, s, and a set of transactions, each of which contains a set of items, the algorithm identifies all sets of items with sufficient support; existing in at least s transactions. This is achieved by exploiting the Apriori Principle, which states that for an itemset to have sufficient support, all its subsets must also have sufficient support. The Apriori algorithm exploits the Apriori Principle by building up a lattice with all possible itemsets and links between itemsets and their immediate (one size smaller) subsets. The algorithm then checks each singleton itemset for sufficient support. If the itemset does not have sufficient support, it and all its supersets (of all sizes) are pruned from the lattice. This process is then repeated at each layer of the lattice.

The thesis used the Apriori algorithm for identifying sets of effects that occurred together frequently. Because learning examples arrived to the learner online, the Apriori algorithm also had to work online. This was implemented in the thesis by building up an explicit lattice that generated new effect sets when at least two of their immediate subsets had sufficient support.

Effect sets were never pruned from the lattice, even if they never gained support. This was because of a tradeoff between computational speed and space. Experimenting with this tradeoff is for future work.

2.3.2 The STRIPS planner

The Action Rule learning component of the thesis learns STRIPS rules [1], referred to as Action Rules in the thesis. A STRIPS based planner was then used to solve simple planning problems with the learnt Action Rules.

A STRIPS rule consists of three parts: an action, preconditions, and effects (commonly referred to as postconditions). Variables are used to generalise rules. When the action is

carried out in the world, if the preconditions were all true then the effects are expected to occur.

The STRIPS planner uses Means-End Analysis to plan backwards from the goal to the initial state. Given a goal described as a set of conditions, a rule whose effects are the conditions is selected. The preconditions of the selected rule are then checked against the current state of the world. If they are satisfied, the action is carried out. Otherwise, they are treated as a new goal and the process is repeated recursively. Normally STRIPS will look ahead to find the shortest possible plan.

The thesis needed to ensure the planner used had some realism to real infants. Therefore, only the immediate goal was considered when choosing an Action Rule. This of course meant the planner was not optimal and failed at plans a sophisticated planner would have succeeded at, but the objective of the thesis is to learn like a human infant.

Chapter 3

Overview of System

As abstraction was identified as an important consideration in the system design, the learning problem was split into five major components, each consisting of one or more subcomponents.

Learning World Simulation: The simulation models a simple learning world. The simulation has an input interface which takes values for its force variables, and an output interface which outputs quantitative states of the world at frequent intervals. A tabletop simulation was used in the thesis, although the other components are designed to work with any simulation that has a similar interface.

Vision and Sensation: The vision and sensation provides a layer of abstraction of the raw quantitative state output of the simulation, by generating a simple qualitative description of each state. This conversion was hardcoded rather than learnt, as learning vision and sensation was outside the scope of the thesis.

Motor Control: The motor control provides a layer of abstraction over the raw force variables on the simulation input, in order to allow the agent to use and learn higher level motor actions that can be applied in different contexts.

Action Rule Learner: The Action Rule learner learns Action Rules describing the various sets of effects each motor action in the motor action policy can result in and preconditions attempting to predict when a given set of effects will occur.

Action Rule Planner: The planner uses the Action Rules learnt by the Action Rule learner to achieve simple tasks. The motor control is used to carry out the Action Rules in the simulation with the help of the vision and sensation. The planner is also able to refine the Action Rules so acts as a simple learner.

Figure 3.1 shows how all the system components fit together. As the presented system is for a Master's thesis the scope of what could be implemented was limited. The motor control policy learner and the vision and sensation system were outside the scope, so hardcoded components for these were implemented to work with the tabletop simulation. All remaining components were implemented to be general and independent of the other components. The main focus of the thesis was on the Action Rule learner and the Action Rule planner. An exploration of how examples could be generated for a motor control policy learner was also carried out.

There are several advantages to using a system design like this. Firstly, it allows meaningful abstractions to be built over the raw simulation input and output rather than attempting to bridge directly from the simulation to the rule learning like Mugan did. Secondly, it allows the agent to separate knowledge of how to physically perform an action from knowledge of what the action can be used to achieve, allowing realistic motor control as realistic motor actions can be modelled. And thirdly, it allows each component to learn at its own

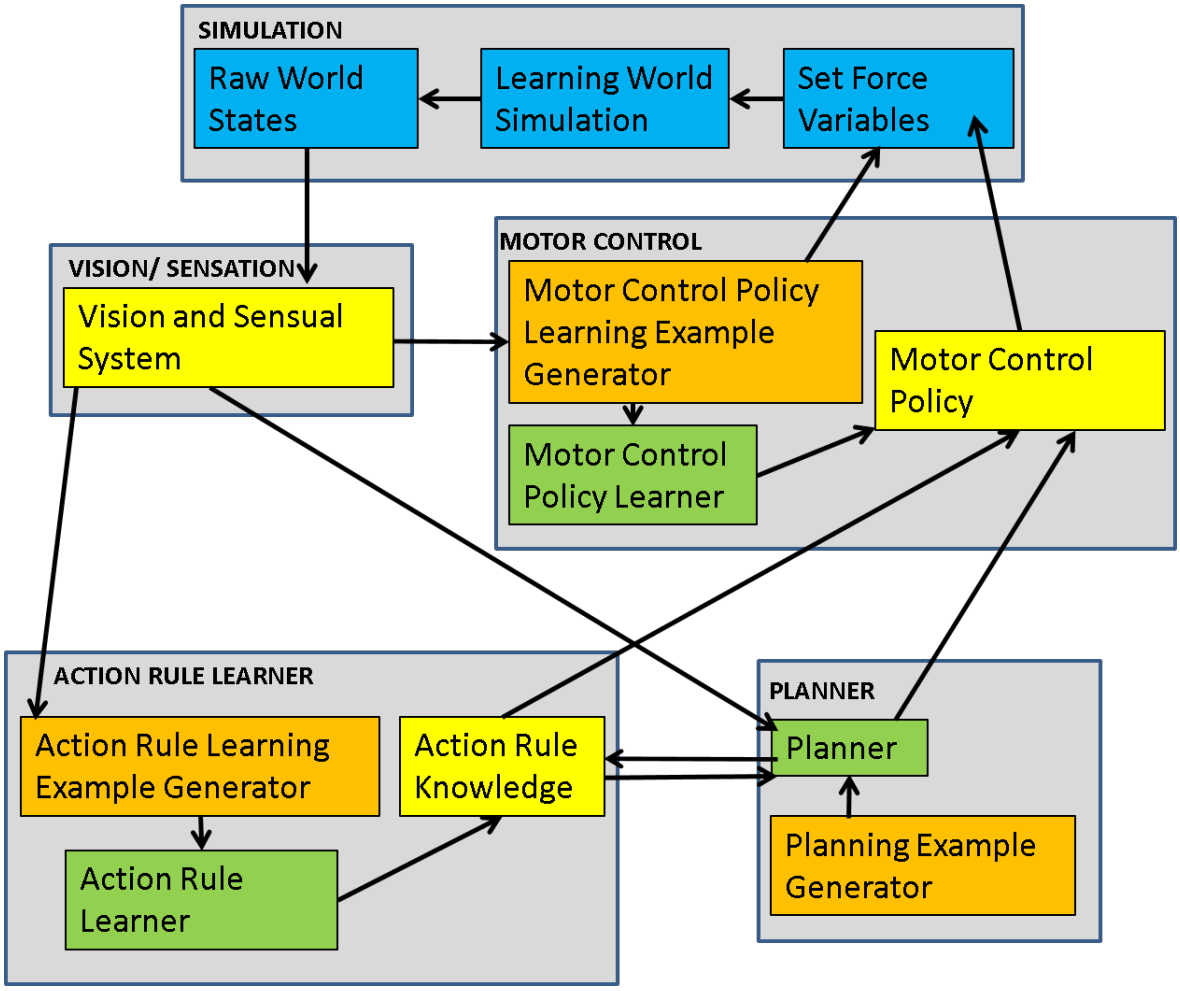


Figure 3.1: The components of the system presented in the thesis

pace, much like a real infant who learns about vision and then motor control before using actions and planning.

The remainder of this chapter outlines the implementation of the simulation, vision and sensation, and the hardcoded motor control policy. The exploration on generating motor control learning examples is discussed in chapter 4, the Action Rule learner in Chapter 5, the Action Rule planner in Chapter 6, and an improved version of the Action Rule learner and planner in Chapter 7.

3.1 The Simulation

A two dimensional simulation modelling an infant, referred to as *the agent*, was used in the thesis. The agent is sitting at a table with balls on the tabletop and its hand stretched out to interact with the balls. Only the area of the tabletop and the infant's hand is modelled; the agent's body and the area outside the tabletop is ignored. The tabletop is rectangular with walls on the four sides preventing the hand leaving the tabletop. One wall has a slot below it large enough for balls to "fall" from the tabletop. Figure 3.2 illustrates the simulation. While the arm has been drawn for clarity, it plays no part in the simulation and balls can "roll" through it.

The hand moves around the table top when forces are applied to it. When the hand hits

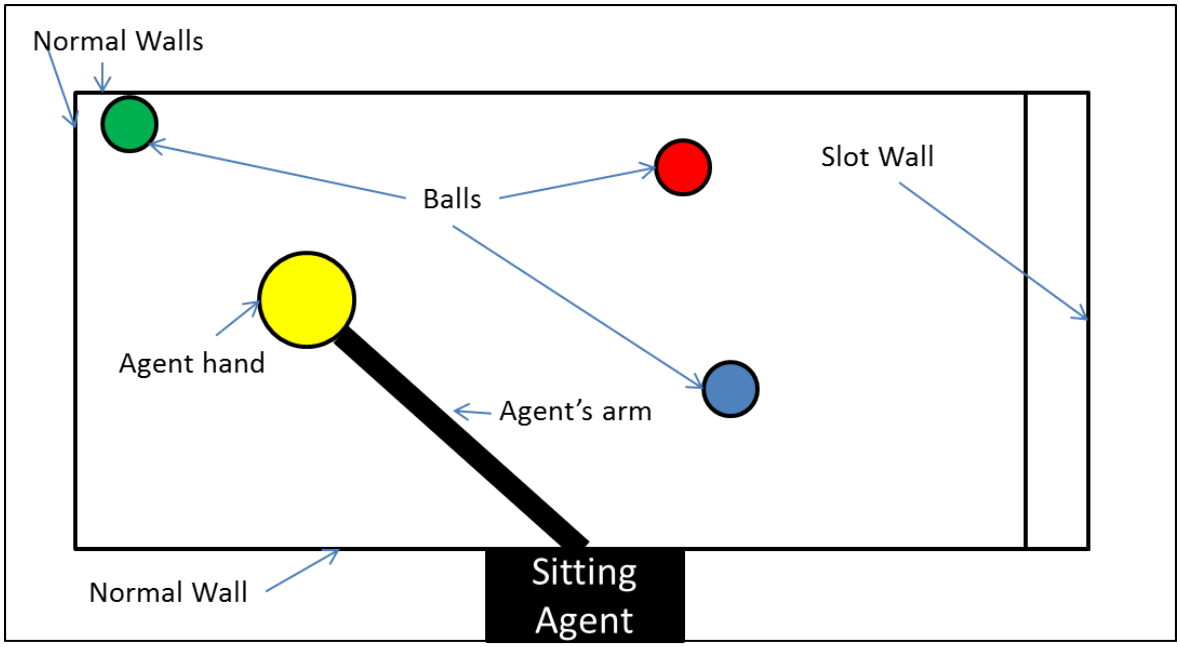


Figure 3.2: The simulated learning world used for the thesis

a ball, the ball moves using realistic physics built into the simulation. Balls can be hit into other balls, and can bounce off walls. Balls that roll onto the slot fall from the table and are no longer in the simulation. When there are no more balls in the world new balls are randomly placed on the table top.

The hand also has fingers which open and close along a one dimensional path. When the fingers close near a ball, the ball becomes grasped and moves around with the hand. The ball is no longer grasped when the fingers reopen.

The simulation makes sounds the agent can hear when the hand hits a ball or wall, or balls bang against one another or a wall.

The simulation was implemented in python using realistic physics. Some artefacts were present in the simulation, although debugging these was considered as unimportant as they weren't frequent enough to be problematic, and the real world is also noisy.

3.2 Vision and Sensation

The vision and sensation provides an abstraction over the raw simulation output by converting the current state of the tabletop into a qualitative representation describing various relationships between objects (balls, walls, and the slot wall) and the hand. In addition to the vision system, the agent has touch sensors which tell it when it senses something against its hand (i.e. a ball or wall), or when an object is gripped, and a sound sensor that activates when the simulation makes a sound. The state of each sensor is added to the qualitative representation. All this information is represented as a set of *qualitative facts*

A qualitative fact consists of a predicate and optional parameters. For example, the Green and Red balls touching is represented as Touching(Green, Red). When the hand is included in a fact, it is written as a part of the predicate. For example the hand touching the Green ball is represented as Hand_Touching(Green). Learning the types of objects (for example ball or wall) was outside the scope of the thesis, so object types were built in. Walls are named after their position; left_wall, near_wall, far_wall, and right_wall. The right_wall

is referred to as the *slot wall* and is of a different type to the other three walls. Balls are named after their colours. Figure 3.3 shows an example of a qualitative representation for a state of the world. Other predicates were also experiemented with, such as alignment of objects relative to the hand.

The exact set of predicates used in the qualitative representation generated by the vision system and sensors is unimportant to the Action Rule learning algorithm, which is designed to learn in any world in which states can be described as a set of qualitative facts with predicates and typed parameters.

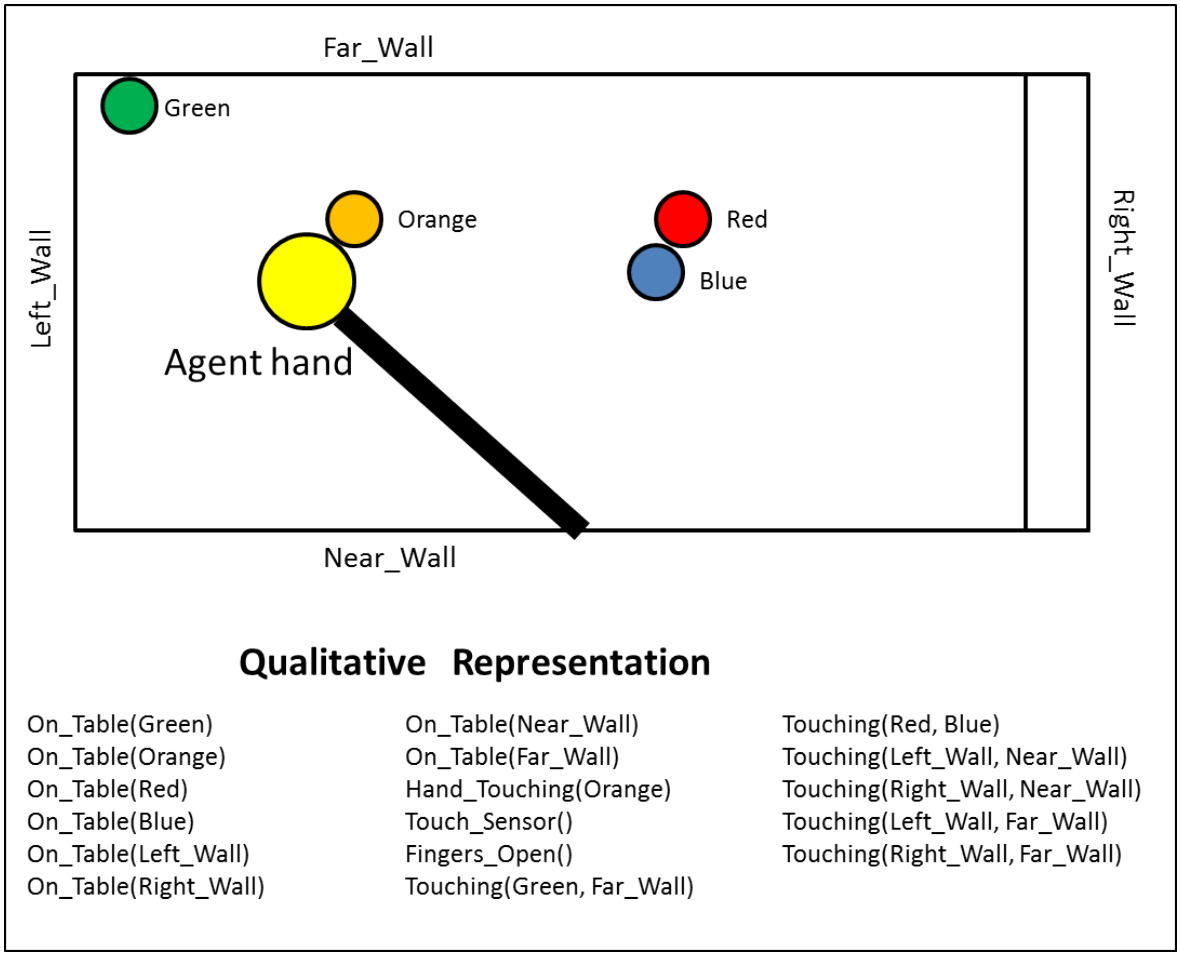


Figure 3.3: A state of the world with a corresponding qualitative representation

3.3 Motor Control Policy

The motor control policy provides an abstraction over the raw simulation input by using policies to set the simulation's force variables in order to model higher level actions such as moving and hitting. While a possible approach to the learning example policy learning example generator is discussed in Chapter 4, a hardcoded motor control policy was implemented to enable the Action Rule learner and Action Rule planner which are dependent on a complete motor control policy. The following actions were included in the policy.

Move_To(target): move towards target and slow down once near so as to touch target.
Hit(target): move fast towards target so as to hit it.

Grasp: close the fingers, stopping when either an object becomes grasped or the fingers are fully closed.

Ungrasp open the fingers, stopping once the fingers are fully open.

Chapter 4

Generating Learning Examples for Motor Control

While learning a motor control policy is outside the scope of the thesis, an initial exploration on identifying learning examples for learning a motor control policy was carried out. We assume that the agent needs to learn its motor control policy autonomously, so the agent must be able to generate its own examples of motor actions to drive the learner, and must do this without knowing what the actions are. Forces were set on the agent's motor variables by a random movement generator that had a tendency to apply a force in a single direction for periods of time rather than constantly changing its forces. By using a simple heuristic procedure to analyse its own random movements generated, the agent was able to identify meaningful actions examples that happened by chance. The heuristics were based on very general measures of simplicity and interestingness. This process involved firstly using regression on the hand and finger positions in order to identify segments that could be explained with a simple path, and then filtering those that were *interesting* to the agent, i.e. caused qualitative sensor input or qualitative visual events and interactions with objects. In addition, having qualitative events at the ends of the segments will help the policy learner to identify when it is learning an action successfully.

4.1 Identifying Segments with Regression

Given a time sequence of states, the agent breaks the sequence into segments in which the hand positions could be modelled with linear or circular (an arc) regression. This was done because smooth and direct paths are simpler and can be described easily, and thus are more likely to be of interest to a human infant to learn about.

Segmentation was implemented by iterating through the sequence of states attempting to add states onto the segment currently being built up. When adding a state resulted in the regression error going above a threshold, the segment was completed and a new segment was started. This algorithm was greedy and did not attempt to go back and optimise the segmentation. Figure 3.1 (1) shows the hand movement for around 8000 sequential states of the world. (2) shows the result of applying the segmentation algorithm to this sequence of states. The colours show the different segments that were identified.

The agent also identified segments among finger movement. Because finger movement in the simulation is modelled on a one dimensional path of opening and closing, only direction of movement was used to segment finger movement. This identified finger closing actions where the finger stopped moving halfway because of an object or closed all the way making a fist, and finger opening actions where either the hand fully opened from a fist or

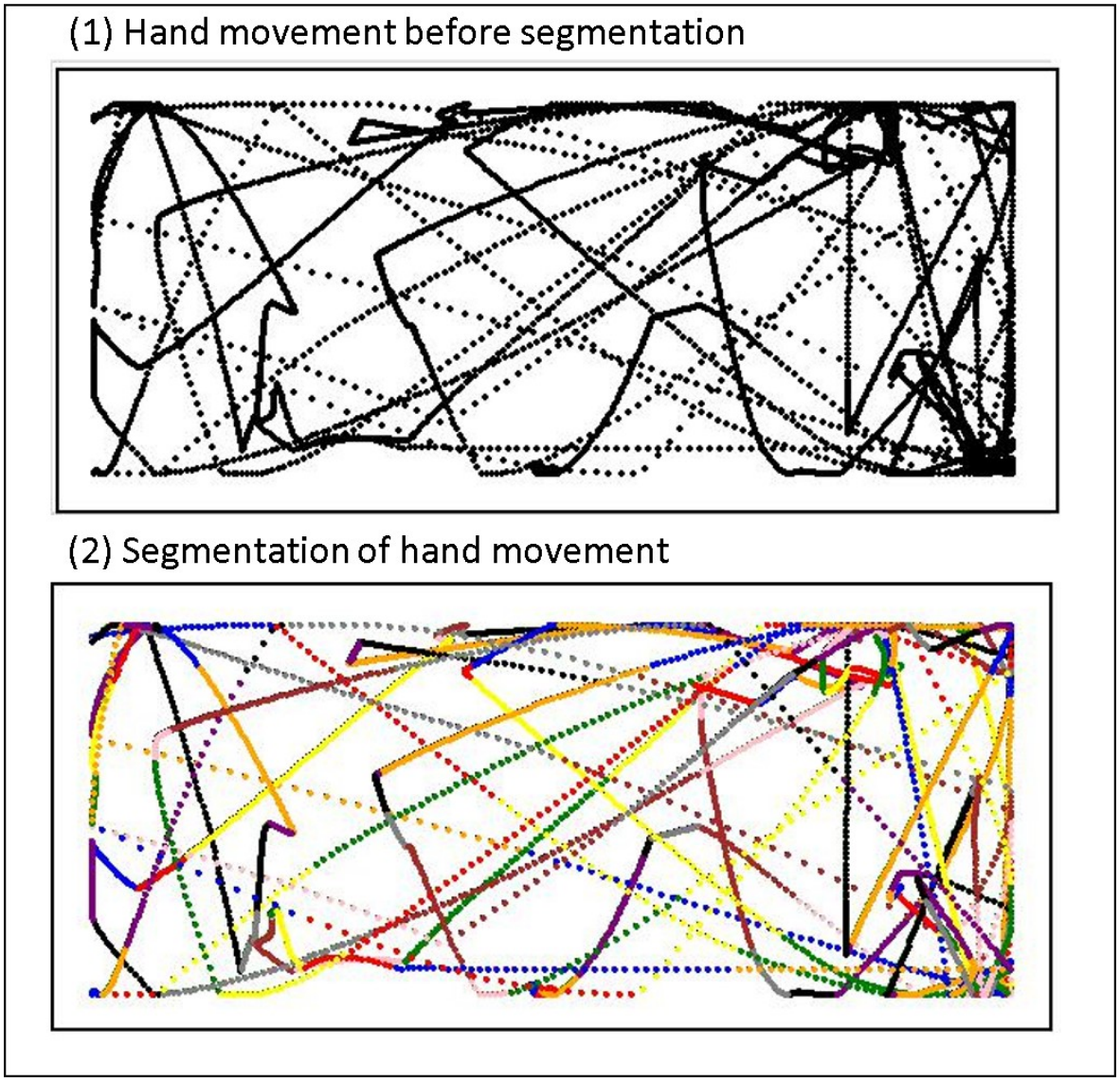


Figure 4.1: Analysing a sequence of states to identify segments consisting of hand positions on the same qualitative path

opened from halfway and releasing an object.

No attempt was made to search for segments that had coordinated hand and finger movement in them. Such coordination is necessary for actions such as throwing or a co-ordinated grasp which times closing the fingers such that they are closed at the same time the full hand movement reaches the object. Because human infants have trouble with such actions, using a more advanced mechanism such as observation and attempting to combine motor control for known actions may be better suited to learning them, but is outside the scope of the thesis.

4.2 Filtering the Segements

Because infants are interested in events and changes in the world, segments that did not contain interesting qualitative events were thrown away on the basis that an infant would

not recognise them as being significant and therefore would not learn from them. Several different qualitative events were used as heuristics for whether or not to keep a segment, for example sensor events such as touch, sound, and grasp, and contact with objects during and at the end of the action.

4.3 Conclusion

To evaluate this mechanism properly, a motor control policy learner would have to be designed to see whether or not useful actions could be learnt with the examples. Since this was not feasible the method was evaluated by determining whether it generated enough segments that matched hard coded definitions of four basic actions – Move_To, Hit, Grasp, and Ungrasp. The definitions used were:

Move_To: segments whose velocity went to or near zero towards the end, and ended with sensor events such as sound or touch and contact with an object (ball or wall)
Hit: segments where a collision with an object was detected while the velocity was high
Grasp: segments where the finger direction is positive (closing) and the grasp sensor goes true
Ungrasp: segments where the finger direction is negative (opening) and the grasp sensor goes false

By using these action definitions, the agent identified many meaningful examples of each action. This shows that using segmentation and then qualitative information to identify learning examples is a plausible approach to the problem.

4.4 Summary and Future Work

This initial exploration investigated using simple heuristics to identify meaningful actions among random behaviour generated from randomly generated sustained forces on the agent's motor variables. The exploration indicated that this approach could generate learning examples that would enable the agent to learn a motor control policy for the actions. Once motor control for the action has been learnt, the motor control policy can then be used to perform the action in a wider variety of situations as done in the Action Rule learner in Chapter 5.

As motor control learning was outside the thesis scope, the motor learning examples were not used by a motor control policy learner. Instead, hard coded motor control policies for four simple actions were used for generating data for the Action Rule learner in Chapter 5: Move_To, Hit, Grasp, and Ungrasp.

In future work, the exploration outlined in this chapter could be used as a starting point to learn a motor control policy. Unique motor actions to learn about could initially be identified by clustering the segments generated in Section 4.2 based on their velocity profiles and interactions with objects. Then motor control for each cluster could be learnt using probabilistic methods on the quantitative motor variables in each segment of the cluster. Konidaris's work outlines a possible way of doing this, [13]. Future work may find other ways of approaching the problem.This would allow the agent to identify a wider range of actions than just the ones the hard coded motor control policy has been designed for. Action Rules describing the effects of the actions generated from each action in the motor control policy could then be learnt using the Action Rule learner introduced in Chapter 5.

Human infants learn online from each learning example as it arrives. By making the motor control learning algorithm online, the agent could refine the policy as it gathers more

examples for them from both random exploration and attempting to use the incomplete policy. The qualitative information in the motor learning examples, for example the path shape, sensor inputs, and object interactions provide a source of information the agent could use to evaluate the policy and direct further refining of it. This kind of approach seems to be inline with Thelens longitudinal investigation of how infants learn to reach for objects [28].

Chapter 5

The Action Rule Learner

The main goal of the thesis is learning Action Rules from exploratory behaviour and using the Action Rules in planning problems in order to achieve simple goals. An Action Rule consists of three parts: an *intention*, which consists of a motor action type and an optional target variable, an *effect set*, which consists of a set of effects that can occur together as a result of the intention being carried out, and a set of *preconditions* that attempt to predict under what conditions the effect set is likely to occur as a result of the agent deliberately carrying out the intention. The effect set and preconditions are represented as sets of qualitative facts that contain typed variables, such as ball ("b"), normal wall("w"), or slot wall ("s").

Effects in an effect set can be either positive or negative. *Positive effects* are effects that are expected to become true as a result of the motor action. *Negative effects* are effects that are expected to become false. Positive effects are written as a qualitative fact with a "+" at the start, and negative effects with a "-".

Figure 5.1 illustrates an Action Rule the agent learns for the Move_To motor action with a target of ?b34. The effects are +Hand_Touching(?b34), +Within_Grasp_Distance(?b34), and -Hand_Touching(?w12), which specifies that moving to ?b34 results in being in within grasp distance of and touching ?b34, and no longer touching ?w12. These effects required that the ball ?b34 was on the table (precondition: On_Table(?b34)) and that the hand was initially touching wall ?w12 (precondition: Hand_Touching(?w12).

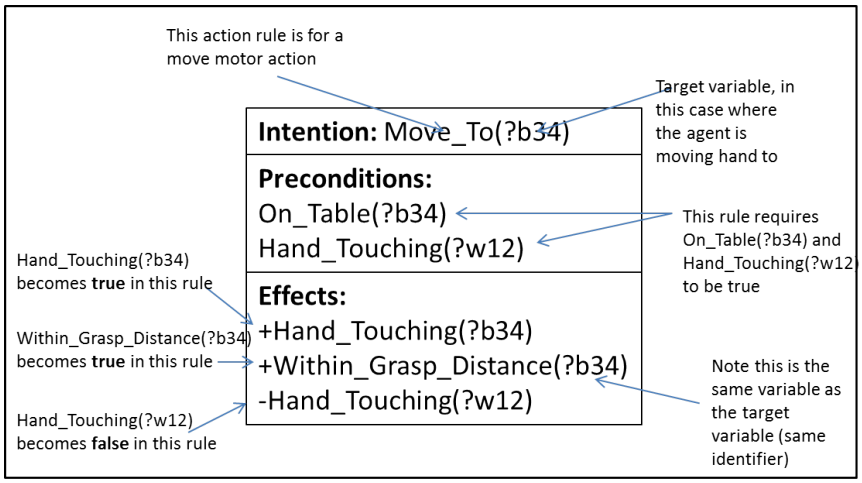


Figure 5.1: A possible Action Rule the agent can learn

The agent learns many Action Rules for each motor action. The Action Rules are made unique by their intention and effect set, because the agent is attempting to learn the different

possible effect sets for each possible intention. The precondition learning algorithm used is minimal, and thus the agent does not attempt to learn multiple Action Rules with the same intention and effect set but different preconditions. Figure 5.2 shows four Action Rules the agent learns for the Grasp motor action. While each of these four Action Rules has the same intention, they have different effect sets to make them unique.

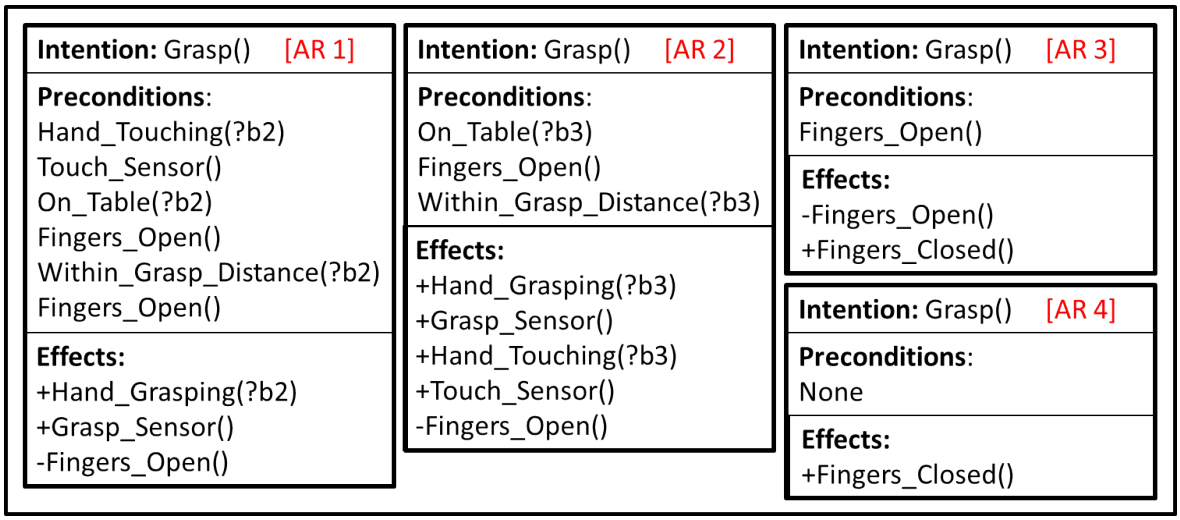


Figure 5.2: Four of the many possible Action Rules the agent can learn for the Grasp motor action

The first Action Rule (AR 1) in the figure has preconditions that describes the hand touching a ball, "?b2", and an effect set specifying that the Grasp action results in the ball being grasped. The second Action Rule (AR 2) applies in situations where the hand was within grasp distance of a ball but not quite touching it. Grasping pulls the ball in, so in addition to grasping the ball, the Grasp action also results in touching it. The third Action Rule (AR 3) applies to situations where the Grasp action just results in the fingers being closed and no longer open. For this, the fingers must have been fully open to begin with. The fourth Action Rule (AR 4) applies in situations where the only effect of the Grasp action is the fingers being closed. Unlike the previous Action Rules, this Action Rule does not require the fingers to be fully open initially.

In order to learn Action Rules, the agent needs a source of learning examples. Learning examples describe situations where the agent deliberately carried out a motor action, and qualitative changes to the world occurred as a result of that motor action being carried out. Learning examples consist of an intention, which is similar to that of Action Rules except it contains a concrete object rather than variable where a target is needed, an initial state, which is a qualitative description of the world immediately before the action was carried out, and effects, which are the qualitative difference in the world from immediately before the action was carried out to immediately after. Unlike Action Rules, all objects in a learning example are concrete rather than generalised, i.e. they are not represented with variables. Section 5.1 discusses how learning examples are generated.

The actual Action Rule learning process begins with learning effect sets. Because an effect set consists of effects that commonly occur together, this problem was approached as a frequent itemset generation problem. The Apriori Algorithm introduced in Chapter 2 is a well known algorithm for generating common itemsets in a transactional database, so was used to learn effect sets. The basic Apriori Algorithm had to be extended to work with variables in items and to work "online" rather than "offline". Section 5.2 discusses the effect set learner.

Given an effect set, the learning examples that *support* the effect set are then used to learn preconditions for the effect set. The precondition learner generalises the objects in the qualitative facts of the initial state of each learning example using a bindings dictionary that was generated when the learning example was matched against the effect set. The generalised qualitative facts are treated as candidate preconditions and are tallied to determine how many learning examples each appeared in. Those that have a tally higher than a threshold become actual preconditions that make up the preconditions component of the Action Rule. Section 5.3 discusses the precondition learner.

The implemented Action Rule learning algorithm is illustrated and evaluated in the thesis using the four motor actions defined in the previous chapter: Move_To, Hit, Grasp, and Ungrasp. However, the Action Rule learning algorithm is designed such that it could learn Action Rules for any motor action with a motor control policy, for example pushing or flicking movable objects with the fingers.

5.1 Generating Learning Examples

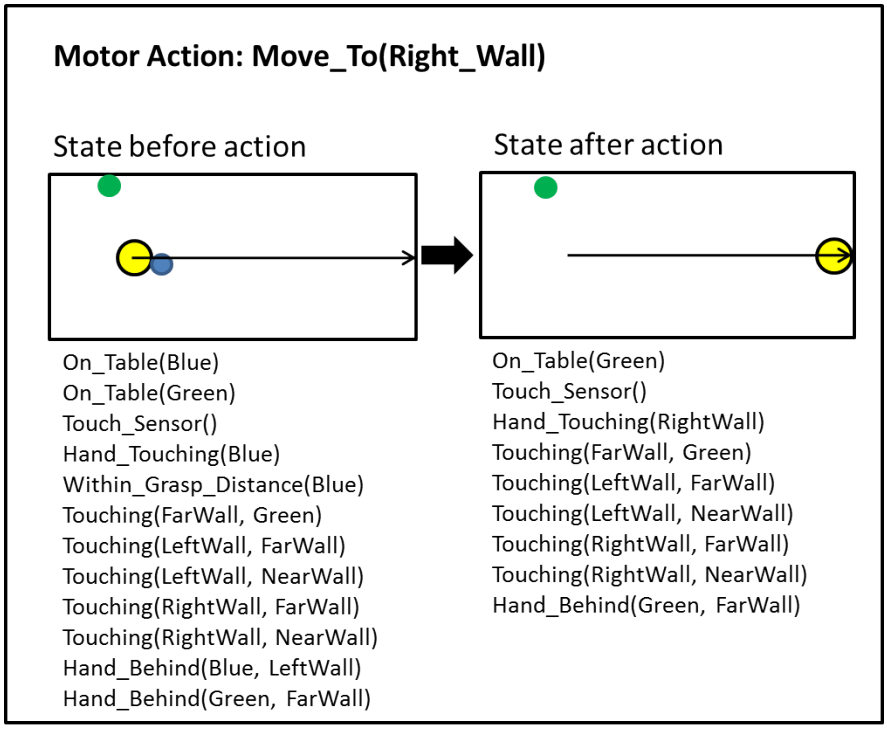


Figure 5.3: An example of a move_to motor action with the intention move_to(Right_Wall)

The Action Rule learner requires learning examples of each motor action. To generate a learning example, the agent chooses a motor action to carry out. If the motor action requires a target object, one is randomly selected. When the motor action is carried out, the agent's visual and perceptive system convert the quantitative states immediately before and after the motor action was carried out into qualitative representations. The state immediately before becomes the initial state of the learning example, and effects for the learning example are determined by comparing the initial and final qualitative states. Effects consist of qualitative facts that were present in one of these two states but not the other. Qualitative facts that were true only in the final state are positive effects and those that were true only in the

initial state are negative effects. While it is possible that a fact could have disappeared somewhere in the middle of the action, and reappeared at the end, the agent does not consider such situations.

While the agent would have been able to gather examples of the motor actions using the processes outlined in the previous chapter, these examples are not suitable for the Action Rule learner because actions that were identified as happening by coincidence have a different distribution of effects to actions that are carried out deliberately.

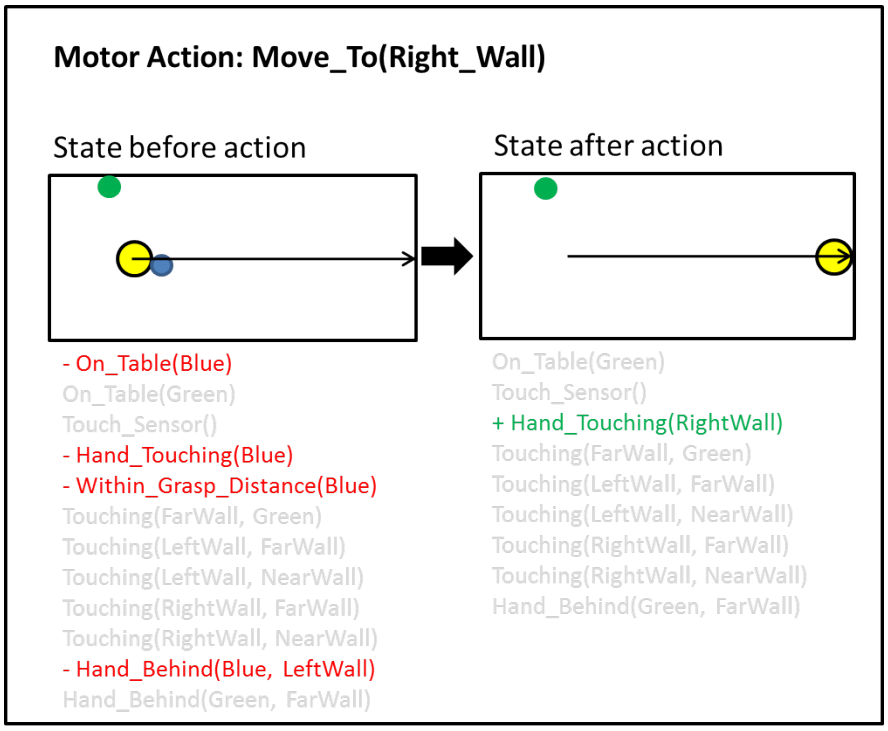


Figure 5.4: The agent determines the effects of the action carried out in Figure 5.3

Figure 5.3 shows an example of the qualitative state before and after a Move_To action with the intention Move_To(Right_Wall) is carried out. The action pushed the Blue object off the table and the hand is now touching the Right_Wall.

Figure 5.4 shows how the agent determines the effects for the action shown in Figure 5.3 . Facts that exist in both states do not become effects so are ignored (greyed out in the figure). Facts that existed in the initial state but not the final state are negative effects (coloured in red and written with a "-" in the figure). Facts that existed in the final state but not the initial state are positive effects (coloured in green and written with a "+" in the figure). This shows there are five effects: -On_Table(Blue), -Hand_Touching(Blue), -Within_Grasp_Distance(Blue), -Hand_Behind(Blue, Left_Wall), and +Hand_Touching(Right_Wall)

Figure 5.5 shows the result of putting together the initial state and intention in Figure 5.3 with the effects identified in Figure 5.4. This makes a complete learning example, suitable for the effect set learner to learn from.

Intention
Move_To(Right_Wall)

Initial State
On_Table(Blue)
On_Table(Green)
Touch_Sensor()
Hand_Touching(Blue)
Within_Grasp_Distance(Blue)
Touching(FarWall, Green)
Touching(LeftWall, FarWall)
Touching(LeftWall, NearWall)
Touching(RightWall, FarWall)
Touching(RightWall, NearWall)
Hand_Behind(Blue, LeftWall)
Hand_Behind(Green, FarWall)

Effects
- On_Table(Blue)
- Hand_Touching(Blue)
- Within_Grasp_Distance(Blue)
- Hand_Behind(Blue, LeftWall)
+ Hand_Touching(RightWall)

Figure 5.5: The learning example created by pulling together the information in Figure 5.3 and Figure 5.4.

5.2 The Effect Set Learner

When the agent carries out a motor action, the resulting effects of the action are largely dependant on the current state of the world. For example, if the agent tries to grasp when its hand is near a ball, the action is very likely to result in the ball being grasped; if the hand is not near an object, it will close with no object grasped.

The objective of the effect set learning process is to identify many different possible effects and combinations of effects that can result from each of the known motor actions. Each effect or combination of effects identified is represented as an effect set. Learning effect sets rather than single effects is somewhat unusual when compared to traditional rule learning, which tends to focus on learning about single effects. The effect set learning approach was taken because it simplifies the planner (allowing it to achieve a group of effects with a single Action Rule), allows coordinated effects to be learnt, and allows for a very simple precondition learner.

The effect set learner uses the Apriori Algorithm introduced in Chapter 2. The effects component of a learning example's effects is treated as a transaction and used to build a lattice of effect sets (referred to as itemsets in the Apriori Algorithm terminology). Singleton effect sets are identified from the individual effects in the learning examples, and candidate size n+1 effect sets are generated by combining size n effect sets, starting with the singleton effect sets. The Apriori Algorithm was chosen because it is an easy to understand algorithm that would allow the agent to begin by learning very simple rules and then extend them to more complex rules.

However, the basic Apriori Algorithm had to be modified for the thesis in two key ways. Firstly, because learning examples are generated and learnt from one at a time, a linked lattice was built up and used to connect smaller effect sets to the larger effect sets they generated. This allowed each learning example as it was generated to be propagated through

the lattice, and relevant effect sets updated and potentially combined to make new effect sets. Secondly, while learning examples contain concrete objects, effect sets should contain typed variables. This was accomplished by *generalising* the objects in the learning example into typed variables and then using a matching algorithm that identifies whether or not the intention and a subset of the generalised learning example effects can be mapped onto an effect set.

5.2.1 Definition of an effect set

An effect set is a set of generalised qualitative facts describing one possible outcome when a particular intention (motor action and optional target) is carried out. An effect set is always connected to an intention. It is possible for two distinct (by the definition in Section 5.2.2) effect sets to have the same effects, but different intentions.

Effect sets contain any number of effects. There are two kinds of effects that can be present in an effect set, both of which are represented with qualitative facts.

Effects describing what became true. This describes a fact that was false at the start of the action, but was true at the end. These effects are written with a "+" in front of their predicate. For example, +hand_touching(?b1) and +grasp_sensor().

Effects describing what became false. This describes a fact that was true at the start of the action, but false at the end. These effects are written with a "-" in front of their predicate. For example, -touch_sensor() and -hand_grasping(?b1).

Each effect set has its own copy of the intention it is an effect set for. The scope of a variable is within the effect set and copy of the intention that it appears in (and eventually Action Rule, once preconditons have been learnt for the effect set). For clarity, fresh variable names are always used in the implementation and in the examples in the thesis. If two variables within an effect set have different identifiers, they must refer to different objects.

5.2.2 Effect set equivalence

In order to determine whether or not two effect sets are equivalent, the following equivalence function was used.

Two effect sets, E1 and E2, (including their corresponding intentions), are equivalent if and only if all the following constraints are met.

1. E1 and E2 have equivalent intentions. i.e. the intentions refer to the same motor action, and the same type of target variable.
2. A *bijective* effect map exists that maps each effect in E1 to an effect in E2 that has an identical identifier, including the "+" or "-". The intention for E1 is also mapped onto the intention for E2, and these are treated the same as the effects are for the remainder of the definition.
3. A *bijective* variable map exists that maps each variable, v1, in E1 (and its intention) to a variable, v2, in E2 (and its intention) such that:
   - No variable is mapped to a variable of a different type
   - If v1 is mapped to v2, and v1 occurs at position i in an effect, e1, then v2 must occur in position i of e1's corresponding effect from the map in (2)

Figure 5.6 shows four examples of the algorithm checking whether E1 is equal to E2. In 1), the effect map and variable map are valid. Therefore, the effect sets are equivalent. In 2), the effect map is not valid because no effect with predicate Sound_Sensor exists in E2, and

no predicate with Touching exists in E1. Therefore, the effect sets are not equivalent. In 3), the effect map is valid, but the variable map is not a bijection because both ?b45 and ?b59 are mapped to ?b28. Therefore, the effect sets are not equivalent. In 4), the variable map is a bijection, but ?b98 and ?w94 which are linked have different types. Therefore, the effect set is not equivalent.

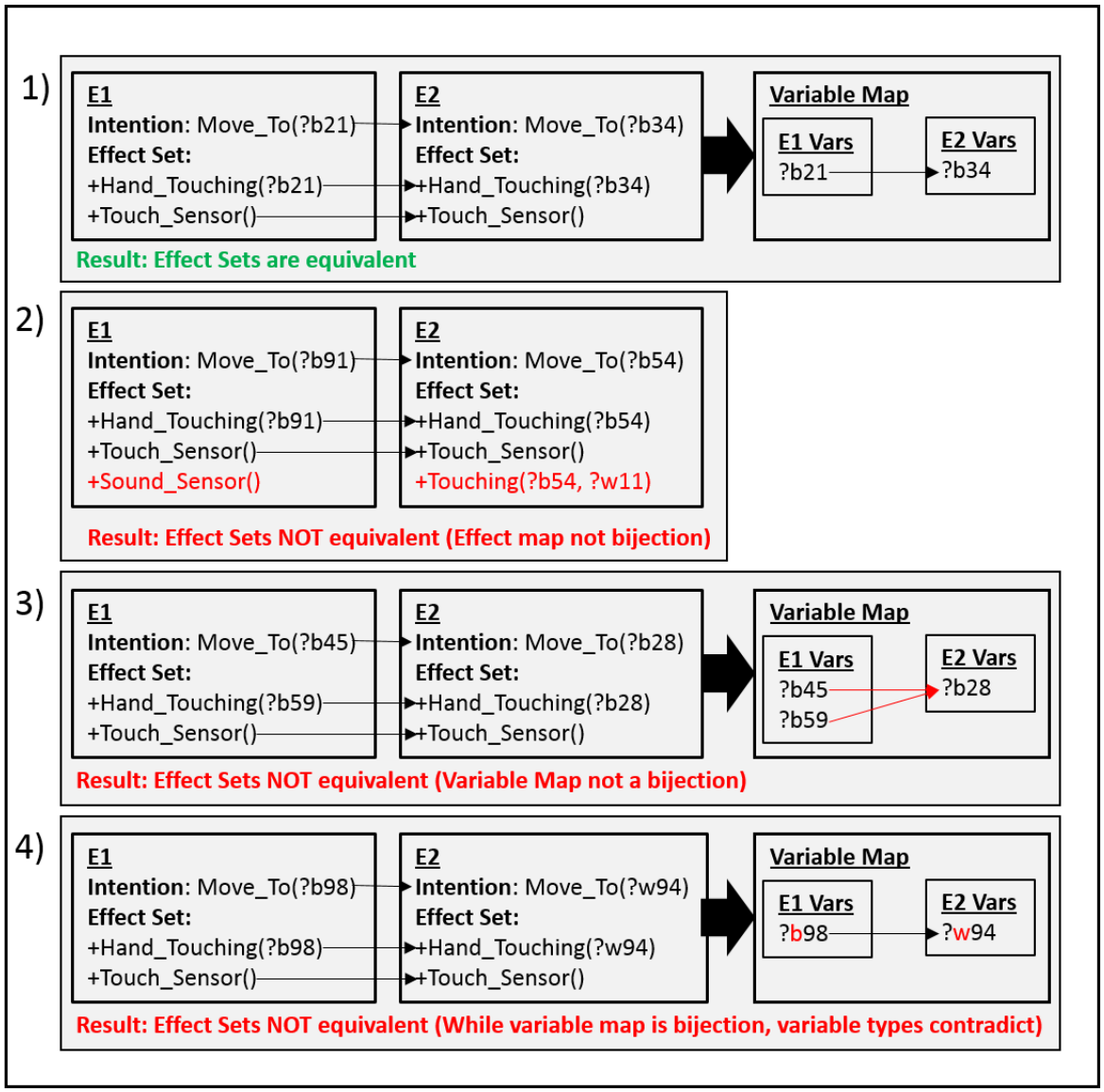


Figure 5.6: 1), 2), 3), and 4) illustrate 4 different cases for the effect set equivalence function. E1 and E2 are equivalent in 1). E1 and E2 are NOT equivalent in 2), 3), and 4) for the reasons specified

5.2.3 The effect set–learning example support function

The Apriori algorithm requires the agent to be able to determine if a given learning example *supports* a given effect set, that is, whether a subset of the Learning example's effects are a concrete instance of the effect set. Effect sets contain typed variables whereas learning examples contain concrete objects. For this definition, variables and concrete objects in the

learning example are all referred to as *parameters*, and the types of all parameters must be known. The following support function was used.

A learning example, LE, supports an effect set, ES, if and only if all the following constraints are met.

1. LE supports ES's intention. i.e. the intentions refer to the same type of motor action and have the same type of target parameter.
2. An *injective* effect map exists that maps each effect in ES to an effect in LE that has an identical identifier, including the "+" or "-". The intention for ES is also mapped onto the intention for LE, and these are treated the same as the effects are for the remainder of the definition
3. A *bijective* map between the parameters in ES (and its intention) and the parameters in the matching effects of LE that maps each parameter p1 in ES to a parameter p2 such that:
    - No parameter is mapped to a parameter of a different type
    - If p1 is mapped to p2, and p1 occurs at position i in an effect, e1, then p2 must occur in position i of e1's corresponding effect from the map in (2)

5.2.4 The lattice structure

The agent learns a lattice of effect sets for each known motor action. The lattice is organised into levels: the first level containing the effect sets with a single effect, the second level containing the effect sets with two effects, the third level containing the effect sets with three effects, etc. The maximum effect set size is determined by a constant. Each effect set has links to its immediate *subsets* and immediate *supersets*. In this thesis, the use of the terms *subsets* and *supersets* always refer to the *immediate* subsets and supersets of an effect set.

A subset-superset relationship exists between a pair of effect sets, es1 and es2 where es1 contains n effects and es2 contains n+1 effects, if and only if removing one effect from es2 makes es1 and es2 equivalent effect sets by the definition given in Section 5.2.2. This lattice is similar in structure to the implicit lattice built by the Apriori Algorithm.

Figure 5.7 shows an example of a simplified lattice the agent could learn given 6 possible singleton effect sets (in practice, each motor action has many more singleton effect sets and thus many more larger effect sets). Effect sets with a high support count are highlighted; these were used to generate larger effect sets. Each effect set links to its subsets, and to its known supersets if it was above a support count threshold. For example, the effect set Move_To(?b12) {+Hand_Touching(?b12), +Touch_Sensor()} has links to two singleton effect sets: Move_To(?b3) {+Hand_Touching(?b3)}, and Move_To(?b4){+Touch_Sensor()}, which are its subsets, and a link to one size-3 effect set, Move_To(?b18) {+Hand_Touching(?b18), -Hand_Touching(?w19), +Touch_Sensor()}, which is its only known superset. The dotted lines show subset-superset links that the agent will not have learnt yet, because the subset is not above the support threshold.

The lattice was implemented using linked nodes, where each node contained an effect set, and links to the effect set's subsets and supersets. In addition to being stored in the lattice, effect sets were also indexed. An index key was generated for each effect set by hashing an alphabetically sorted list of the predicate names in the effect set. Collisions were handled by storing effect sets with the same index key together in a set at the index. The indexes allowed the effect set learner to efficently determine whether or not an equivalent effect set for the given effect set exists in the lattice, and the planner to quickly identify suitable Action Rules for plans. A better index key able to take into account variables but ensure that two

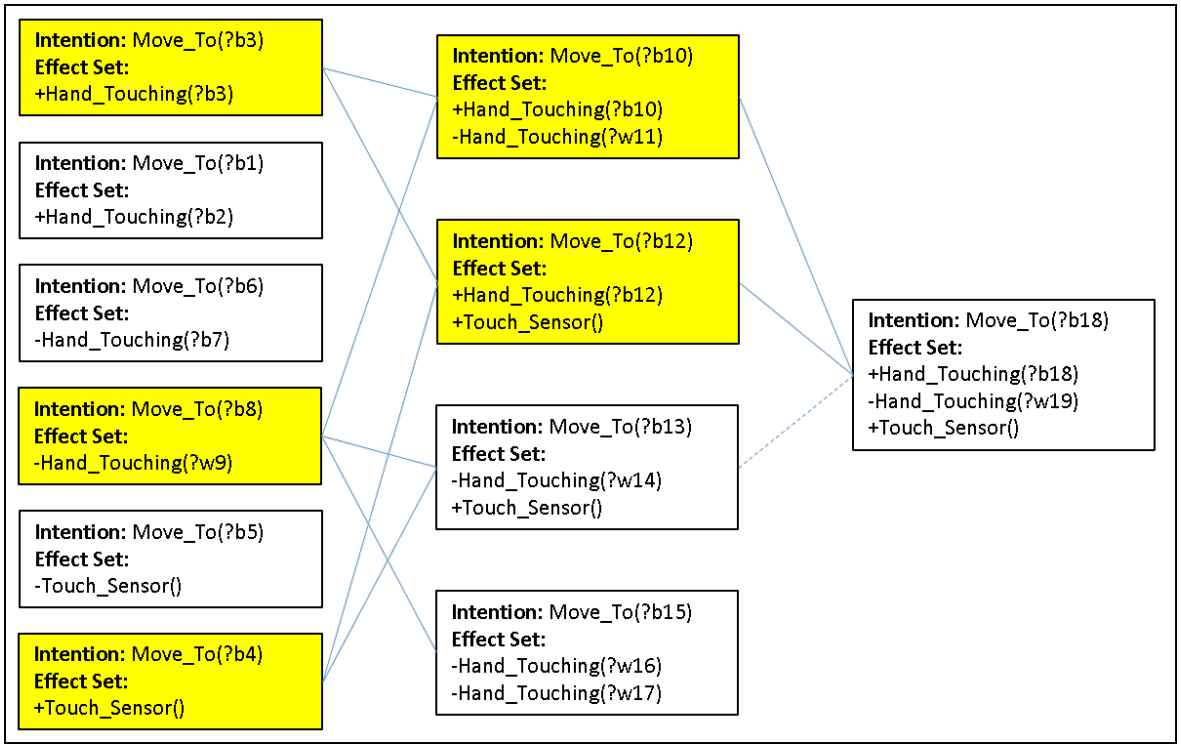


Figure 5.7: A simplified lattice of effect sets that the agent could learn.

equivalent effect sets always hash to the same value could have been designed, although was considered unnecessary and outside the scope of this thesis because collisions were not an efficiency problem.

5.2.5 Building up a lattice of effect sets

The effect set learning algorithm is outlined in Algroithm 5.1. The algorithm adds each learning example to the lattice, one at a time. It first adds the example to any singleton effect set that it supports, adding new singleton effect sets to the lattice if necessary. It then propagates the example up the lattice from those singletons, adding the example to any Effects Set that it supports. When the example pushes the support of an effect set above the combining threshold, it will attempt to create a new superset effect set to the lattice by combining the effect set with each other supported effect set of the same size with which it has a common subset. The details of the important steps are described in the following subsections.

```
define add_learning_example_to_lattice(example)
    generalised ← generalise example  5.2.6
    for each effect in generalised.effects
        singleton ← make effect set containing effect
        if singleton is equivalent to an effect set in lattice  5.2.2
            singleton ← get equivalent to singleton from lattice
        else
            add singleton to lattice
        apply_example_to_effect_set(generalised, singleton)

define apply_example_to_effect_set(generalised, effect_set)
    if generalised already marked as seen for effect_set
        return
    mark generalised as seen for effect_set
    if generalised supports effect_set  5.2.3
        increment support for effect_set by 1
        if support = COMBINING_THRESHOLD
            combine_with_neighbours(effect_set)
    for each super in known supersets of effect_set
        apply_example_to_effect_set(generalised, super)

define combine_with_neighbours(effect_set)
    for each neighbour_es in related_effect_sets(effect_set)
        for each shared subset of effect_set and neighbour_es
            for each binding_map consistent with subset, neighbour and effect_set
                new_es ← construct_effect_set(subset, effect_set, neighbour, binding_map)
                if new_es is not equivalent to any effect set in lattice  5.2.2
                    link new_es to effect_set and neighbour in lattice
                else
                    link equivalent effect_set to effect_set and neighbour

define related_effect_sets(effect_set)
    find all subsets of effect_set in lattice
    collect all supersets of each subset in lattice.
```

Algorithm 5.1: Building an effect set lattice with learning examples

5.2.6 Learning singleton effect sets

Singleton effect sets are learnt by generalising the learning example and then generating an effect set for each effect. Any effect sets that are not present in the lattice are added. In order to generalise the learning example, each unique object referred to is assigned a typed variable with a unique identifier, and each instance of the object in the learning example is replaced with the assigned variable. This allowed the agent to generate size-1 effect sets so that increasingly larger effect sets could then be generated using the size n+1 effect set learning procedure discussed in Section 5.2.7.

5.2.7 Learning size n+1 effect sets

The agent generates possible size n+1 effect sets by combining pairs of size n effect sets. In order to combine the effect sets, the agent identifies injective mappings between a subset of the variables in the first effect set and the variables in the second effect set that when applied to a copy of the effects in each effect set, will leave n+1 effects when resulting duplicates are removed. The mapping must also make the intentions of the effect sets equivalent.

In order for a suitable injective mapping to exist, the pair of size n effect sets must have at least one immediate subset (containing n-1 effects) in common. Removing any one effect from an effect set makes the effect set equivalent to one of its subsets. Intuitively, for two size n effect sets to "overlap" onto each other, n-1 effects from each must overlap, and 1 effect from each must not, giving a total of n+1 effects. Removing the non-overlapping effect from each of the two effect sets would mean they are equivalent. And because removing 1 effect from an effect set makes the effect set equivalent to one of its subsets, there must exist an immediate subset in common that is also equivalent. The agent exploits this property in order to greatly reduce the number of unsuccessful pairings processed. Multiple valid mappings are possible where there are multiple subsets in common, or where the two effects unrelated to the subset have variables that were not present elsewhere in the mapping. Multiple subsets in common can also lead to there being no valid mappings.

By having each effect set initiate the combining process just once with all other suitable effect sets above the support threshold, it is guaranteed that each valid pair of effect sets is combined exactly once. Because a size n effect set has up to n unique immediate subsets, and only two of its subsets are used to generate it, the same effect set will be generated again when another subset goes above the threshold. While duplicate effect sets are not added to the lattice, the agent exploits the additional generations to ensure that each effect set is linked to all its subsets.

Figure 5.8 illustrates an example where the agent is combining es1 with a possible es2. In (1) of the Figure, the agent identifies a common subset of es1 and es2. In (2), the agent identifies the common subset effects in each of es1 and es2. These effects are bolded and coloured blue in the figure. The agent then builds up a variable map in order to merge the common subset effects in es1 with the common subset effects in es2. In (3), the agent identifies two possible valid extensions of the map that lead to n+1 effects upon applying the mapping. Finally, the result of applying the mapping to make a new effect set is shown below each mapping. The two effect sets generated are similar, except the first effect set covers cases where the object that fell from the table is the same object as the one the agent's hand is no longer touching, whereas the second effect set covers cases where the object that fell from the table is different from the one that the agent's hand is no longer touching.

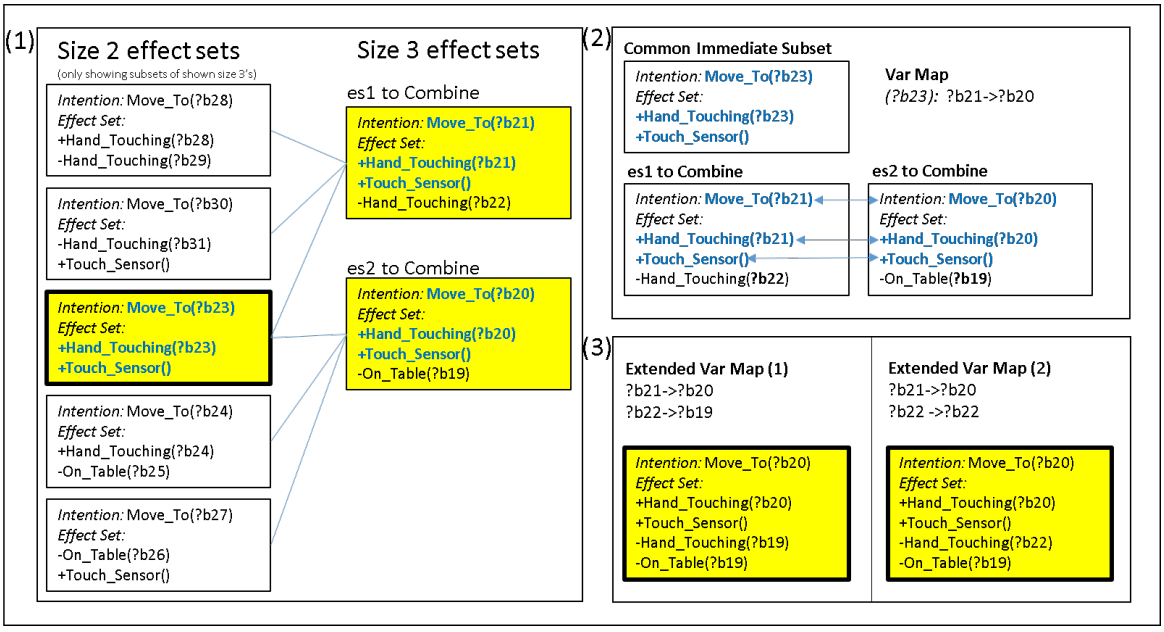


Figure 5.8: An example of the effect set combining algorithm

5.2.8 Propagating supporting learning examples

The Apriori Principle states that an item set can only be supported by a transaction if all its subsets are supported by the transaction. This also applies to the effect set lattice, as it is based on the Apriori Algorithm. This allows the agent to greatly cut down the number of effect sets a learning example is checked against, as a learning example should only be checked against an effect set if the effect set's subsets were supported by the learning example.

For simplicity of implementation, the agent checked a learning example against an effect set after it knew at least one subset was supported. In order to prevent redundant checking (because a superset could potentially be checked via all its subsets), the agent kept track of which effect sets had already been matched against a particular learning example. This approach was sufficently efficient for the purposes of this thesis. If efficiency became an issue, the algorithm could be improved by fully enforcing the Apriori Principle. When a subset sends a learning example to its supersets, the supersets could track which subsets have "sent" them the learning example. Only once all subsets have sent the learning example would the effect set check whether or not it is supported by the learning example.

5.3 Learning preconditions for Action Rules

Preconditions are learnt for each effect set in order to turn them into complete Action Rules. The preconditions strive to determine what must be true of the world in order for deliberately carrying out the motor action specified in the intention of the effect set to cause the effect set to occur. Each time the agent gets a supporting learning example for an effect set, it uses the facts in the initial state of the learning example for precondition learning. Once preconditions have been learnt for an effect set and intention, the intention, effect set, and preconditions make a complete Action Rule.

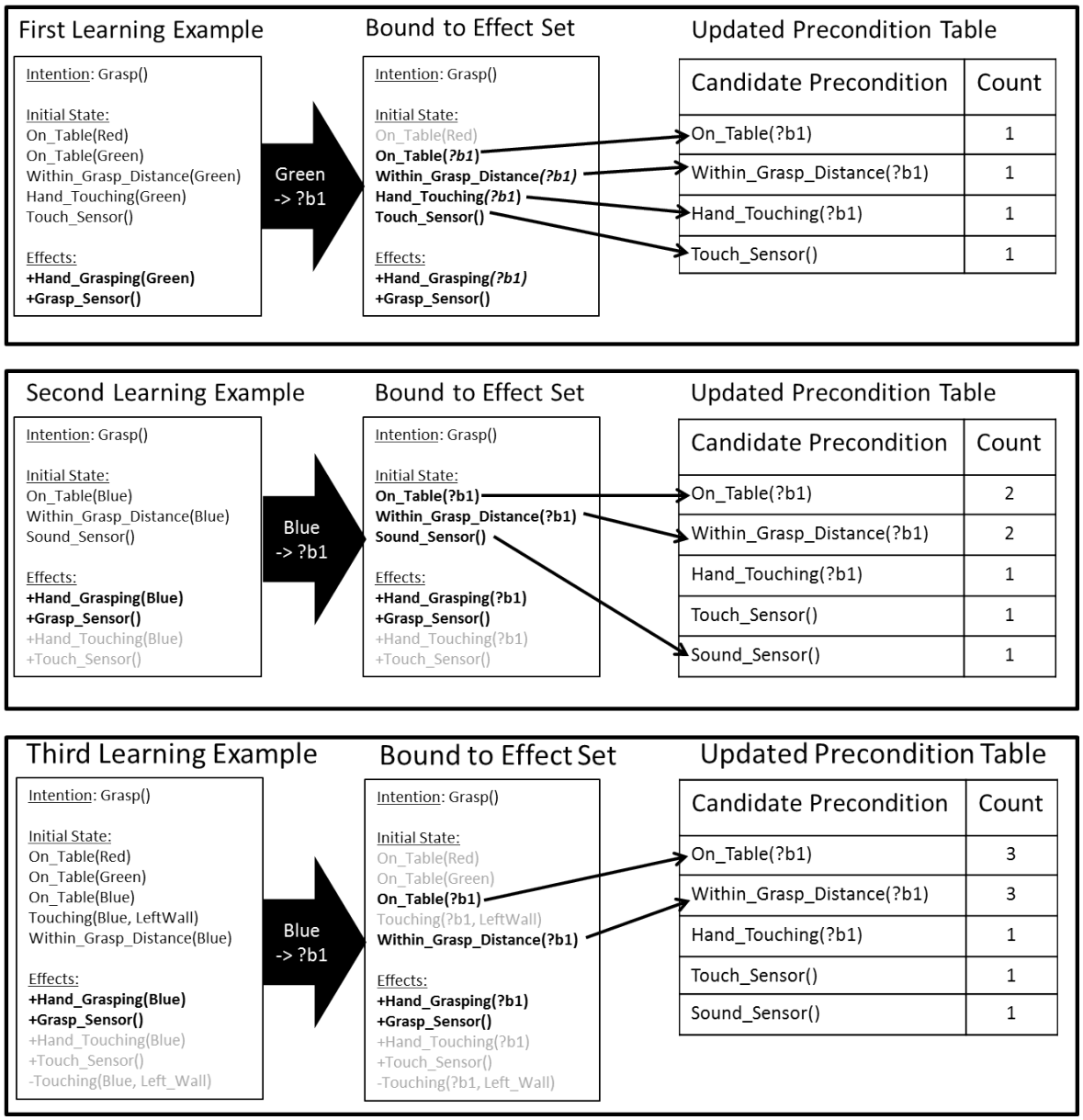


Figure 5.9: Adding three possible learning examples to an effect set with the effects +hand_grasping(?b1) and +grasp_sensor()

The precondition learning algorithm used is simplistic and assumes that if something is often true when a motor action results in an effect set, then it is a precondition. The agent is unable to learn whether or not the precondition actually has to be true or whether it just happens to be true often, and it is also unable to learn what must not be true. This simple approach was determined to be sufficient for the purpose of learning to carry out simple tasks like infants might carry out.

5.3.1 Identifying candidate preconditions

When a new effect set is identified, an empty precondition table is attached to it. The precondition table tracks candidate preconditions and how much support each candidate precondition has. When the agent needs to know the preconditions of an Action Rule, such as during the planning process, candidate preconditions that have at least a given percentage

of support are treated as actual preconditions. Learning examples that supported an effect set are used to learn candidate preconditions for the effect set.

Like effect sets, all objects in the preconditions must be represented as variables. Variables in the preconditions must be consistent with variables in the effects. In addition, preconditions must not contain variables that are not present in the intention and/or effects. In order to meet these constraints, the algorithm uses the bindings dictionary that was generated when the learning example was matched onto the effect set.

This constraint on preconditions was imposed because mapping additional precondition variables onto learning examples isn't straightforward. It also provides a simple "attention" heuristic to prevent the agent from adding lots of irrelevant preconditions that would make planning much more difficult. Ideally a more sophisticated heuristic could be used but this heuristic was simple and effective for almost all the goals in the evaluation.

Algorithm 5.2 shows how the agent learns candidate preconditions using Learning Exmaples. It assumes the effect set initially has an empty precondition table.

```
define learn_preconditions_with_example(effect_set, example)
    mapping ← generate variable bijection mapping example onto effect_set
    reverse mapping so that example objects map to effect_set variables
    for each fact in initial state of example
        mapped_copy ← apply mapping to fact
        if objects in mapped_copy were all converted to variables
            check effect_set's precondition table for precondition equivalent to mapped_copy
            if such a precondition exists
                add 1 to support count for equivalent precondition
            else
                add precondition to table with a support count of 1
```

Algorithm 5.2: Updating preconditions for an effect set with a learning example.

Figure 5.9 shows an example of the agent identifying candidate preconditons for an effect set. The first column of the figure shows raw learning examples, the second column shows the result of applying the bindings dictionary generated from the effect set matching to the learning example, and the third column shows the updated precondition table from applying the learning example to the precondition learning process. Initial state facts in the second column are greyed out if they contain non-variable objects, as they are discarded by the precondition learning process.

Adding the first learning example in Figure 5.9. The agent begins by applying the bindings dictionary that was generated by the learning example to effect set matching process to the initial state facts of the example. For this learning example, all instances of the Green ball in the initial state facts are replaced with ?b1; a variable from the effect set. The agent goes through each initial state facts that have no remaining non-variable objects and checks if the fact already exists in the precondition table. If it exists, the count for the fact is incremented by 1. Otherwise, the fact is added as a new candidate precondition with a count of 1. In this case four facts in the initial state meet the criteria of containing no objects that have not been converted to variables: On_Table(?b1), Within_Grasp_Distance(?b1), Hand_Touching(?b1), and Touch_Sensor(). Because this is the first learning example added, the table is empty and all four facts are added as new candidate preconditions.

Adding the second learning example in Figure 5.9. The bindings dictionary is applied to the initial state facts, and the agent then goes through each of the facts. The first two

facts, On_Table(?b1) and Within_Grasp_Distance(?b1), already exist in the precondition table, so their counts are incremented by 1. The third fact, Sound_Sensor(), does not exist in the table so is added as a candidate precondition with a count of 1.

Adding the third learning example in Figure 5.9. The same process is carried out again. While the effects of the learning example contain the object Left_Wall, this is not converted to a variable as it was not a part of the effect set that preconditions are being learnt for. The only two initial state facts that meet the criteria for candidate preconditions are On_Table(?b1) and Within_Grasp_Distance(?b1), so their counts are incremented by 1. While Touching(?b1, Left_Wall) contains a variable, ?b1, it also contains a non-variable object, Left_Wall. Therefore, it is not added as a candidate precondition.

5.3.2 Selecting actual preconditions from candidates

Once an effect set has a sufficient number of supporting examples, it can be used by the planner. The agent identifies which candidate preconditions should be treated as actual preconditions in the resulting Action Rule using a percentage support threshold, normally set at 80%. Any candidate precondition that was supported by at least 80% of learning examples that supported the effect set becomes an actual precondition.

Figure 5.10 shows a possible precondition table for an effect set after 50 supporting learning examples have been applied to the precondition learning process. The support threshold has been set to 80% (as it was for most experiments in the evaluation). Two preconditions in this example had sufficient support and are highlighted in yellow; On_Table(?b1) and Within_Grasp_Distance(?b1). Therefore, the agent has learnt that for a Grasp action to result in an object being grasped and the grasp sensor being activated, the Grasp actions should be carried out when the object is on the table and the hand is within grasping distance of it.

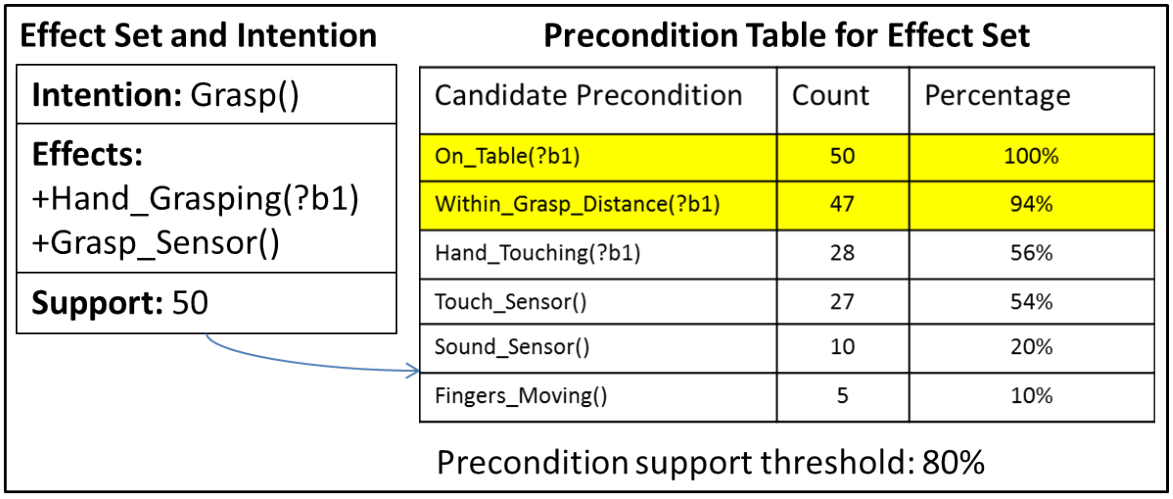


Figure 5.10: Determining which candidate preconditions to treat as actual preconditions, with a precondition support threshold of 80%

Chapter 6

Using Action Rules to Solve Simple Planning Problems

In order to achieve goals in the world, the agent requires a planner to choose suitable Action Rules for goals and to chain together Action Rules for cases where a single Action Rule cannot achieve the goal. This chapter discusses the implementation of the planner that was used.

As discussed in Chapter 2, psychology literature on the details of infant planning is limited. It is believed infants taken a Means-End Analysis approach, beginning at around 6-months old, although the exact details are unclear [29].

Therefore, a Means-End Analysis planner based on STRIPS is used in the thesis. The planner is greedy, and commits to the best known option for the current point in the plan. Therefore, there is no exhaustive searching or standard back tracking. The planner does however allow for trying again after a failure occurs by picking a new option to commit to. This results in a planner that prioritizes trying Action Rules to see whether they achieve the desired goals over reasoning about all multi-step possibilities in an attempt to find the optimal. More sophisticated planners exist for such tasks, but are unlikely to be realistic for infant planning. Just like infants, there will be tasks that the planner cannot solve.

6.1 Overview of Planning Algorithm

The planning algorithm uses a Means-End Analysis approach with an explicit stack to keep track of the current planning and reasoning. The algorithm is closely related to the STRIPS planner. Each step in the planning (a single cycle of the outer loop) involves checking the type of the stack item on the top of the stack and then either adding or removing stack items on the stack.

Stack items on the stack can either be Goal Sets or Bound Action Rules. There are three different types of Goal Set: Full Goal Set, Unsatisfied Goal Set, and Expected Goal Set.

Full Goal Set: A Goal Set that was either the original goals or all the preconditions of a Bound Action Rule immediately below it on the stack.

Unsatisfied Goal Set: A Goal Set representing the currently unsatisfied goals in the Full Goal Set immediately below it on the stack.

Expected Goal Set: A Goal Set representing the goals from the Unsatisfied Goal Set below it on the stack that the Bound Action Rule above it is expected to achieve. An Expected Goal Set will never be at the top of the stack at the end of a plan step.

Bound Action Rule: An Action Rule that should be executed as part of the plan in order to achieve Goal Set(s) below it on the stack. A Bound Action Rule is represented by an Action Rule and a dictionary of bindings to bind the variables in the Action Rule to the values in the goals of the Goal Set immediately below it on the stack. A Bound Action Rule will either have nothing above it on the stack or a Full Goal Set encompassing its preconditions as goals.

Algorithm 6.1 shows the full planning algorithm that was used. The outer loop iterates until the planning stack is empty. Each iteration of the loop requires checking the type of the top stack item in order to determine the next planning step. The key planning steps, driven by stack item types, are as follows.

If the top of the planning stack is a Full Goal Set: The agent checks if the Full Goal Set on the top of the stack is fully satisfied in the current state of the world. This is a matching process carried out by the built-in visual system (the visual system was introduced in Chapter 3), the details of which are discussed in Section 6.3. If the Full Goal Set is fully satisfied, it is popped from the stack. Otherwise, the visual system returns the unsatisfied goals which are put on the stack as an Unsatisfied Goal Set.

If the top of the planning stack is an Unsatisfied Goal Set: The agent applies greedy heuristics to select a good (not necessarily optimal) Action Rule for the goals in the Unsatisfied Goal Set currently on the top of the stack and to bind the variables in the Action Rule to the matching values in the goals. The Action Rule selection process is further discussed in Section 6.4. Three stack items are added to the stack in the following order: an Expected Goal Set containing the subset of the Unsatisfied Goal Set in which the chosen Action Rule is expected to satisfy, a Bound Action Rule containing the Action Rule and bindings, and a Full Goal Set containing the preconditions of the Bound Action Rule. The bindings are applied from the Bound Action Rule to the Full Goal Set.

If the top of the planning stack is a Bound Action Rule: The agent pops the Bound Action Rule on the top of the stack and passes it to the motor control policy (introduced in Chapter 3), which in collaboration with the visual system executes the action attached to the Bound Action Rule in the Simulation. The Expected Goal Set below is then used to evaluate the Bound Action Rule that was executed; this process is discussed in Section 6.6. The Expected Goal Set and Unsatisfied Goal Set below are popped from the stack as well. Although they might not have been satisfied, the agent needs to ensure no other goals were undone and therefore needs to recheck the Full Goal Set against the current world state.

Note that it is an impossible state for the top of the stack to contain an Unsatisfied Goal Set at the end of an outer loop cycle.

There are a few additional features of the planner in order to break infinite looping and planning, to deal with stuck plans due to no suitable Action Rules for the current Unsatisfied Goal Set, and to exploit serendipity.

```
max_depth ← 5
max_actions ← 20
define plan_for_goals(goals)
    current_depth ← 0
    current_actions ← 0
    plan_stack ← empty stack
    push goals on plan_stack as a FULL GOAL SET
    while plan_stack is not empty
        if a FULL GOAL SET is on the top of plan_stack
            current_gs ← peek at top of plan_stack
            unsatisfied ← subset of current_gs that is unsatisfied in the current world state
            if unsatisfied is an empty set
                pop the top off plan_stack to remove current_gs
            else :
                push unsatisfied on plan_stack as an UNSATISFIED GOAL SET
        else if an UNSATISFIED GOAL SET is on the top of plan_stack
            current_gs ← peek at top of plan_stack
            if a suitable action rule for satisfying some of current_gs exists
                chosen_ar ← action rule selected with heuristics
                bound_ar ← chosen_ar bound to current_gs
                subset ← subset of current_gs expected with chosen_ar
                push subset on plan_stack as an EXPECTED GOAL SET
                push bound_ar onto plan_stack
                increment current_depth by 1
                if current_depth < max_depth
                    preconditions ← preconditions from ar in bound_ar
                    push preconditions onto plan_stack as a FULL GOAL SET
            else :
                while top of plan_stack is not a BOUND ACTION RULE
                    pop the stop of plan_stack
        else if a BOUND ACTION RULE is on the top of plan_stack
            bound_action ← pop top of plan_stack
            carry out the motor control policy for bound_action in the simulation
            increment current_actions by 1
            decrment current_depth by 1
            expected ← pop top of plan_stack
            evaluate bound_action with expected
            while the top of plan_stack is not a FULL GOAL SET
                    pop top of plan_stack
            for each item from bottom to top of plan_stack
                if item is a GOAL SET and satisfied in current world state
                    remove item and all above items from plan_stack
                    break out of loop
        if current_actions >= max_actions
            while top of plan_stack is not a BOUND ACTION RULE
                    pop top of plan_stack
```

Algorithm 6.1: The planning algorithm used by the agent

Preventing infinite looping and planning: A maximum stack depth (number of action rules on the stack) and maximum plan length (number of Action Rules that have been executed as part of the plan) are used to prevent looping. When the maximum stack depth is reached, the agent pretends the preconditions of the top Bound Action Rule were satisfied even if they weren't and executes the Bound Action Rule. When the maximum plan length is reached, the agent ignores all Goal Sets (they are popped from the stack) and executes the remaining Bound Action Rules on the stack (limited by the maximum stack depth). The implications of these heuristics are further discussed in Section 6.5. In addition, once the same Action Rule has been used three times for the same underlying Full Goal Set, it will not be repeated.

Handling stuck plans: When there is no suitable Action Rule for the current Unsatisfied Goal Set, the agent pops the Unsatisfied Goal Set and underlying Full Goal Set from the stack. If there was a Bound Action Rule underneath, the next loop iteration will attempt to execute that Bound Action Rule. This allows planning to continue, and possibly succeed, even when preconditions could not be satisfied. The implications are further discussed in Section 6.5.

Exploiting serendipity: When a Bound Action Rule is executed, the agent checks if any Expected Goal Set lower down the stack was satisfied serendipitously as a side effect. If any Expected Goal Set meeting this criteria is found, all stack items above the Expected Goal Set are removed, and the Expected Goal Set itself and the Unsatisfied Goal Set below it are removed.

6.2 An Example of Planning to Grasp an Ball

Figure 6.1 shows how a possible plan for the goal of having an ball grasped is generated and carried out.

A Full Goal Set with +Hand_Grasping(?b1) is put on the stack (1). Because none of the goals in (1) are satisfied in the current state of the world, shown on the right as state (0), the agent then put an Unsatisfied Goal Set with the same goals on the stack (2). The agent then picked an Action Rule that satisfies (2). Three items were put on the stack: an Expected Goal Set for the chosen Action Rule (3); a Bound Action Rule wrapping the chosen Action Rule along with bindings to bind it to the Unsatisfied Goal Set (4), (in this case, ?b2 in the Action Rule was bound to ?b1 in the Unsatisfied Goal Set); and the preconditions for the Bound Action Rule put on as a Full Goal Set (5). The Full Goal Set (5) is then checked against the current state of the world (still (0), as no changes have been made to the world yet). In this case, one of the two goals was unsatisfied, so an Unsatisfied Goal Set containing just the unsatisfied goal is made (6). As a side effect of the matching process, ?b1 was bound to Red. The details of this are discussed in Section 6.3. An action rule is picked for the goals in (6); again an Expected Goal Set (7), the Bound Action Rule (8), and a Full Goal Set (9) are added to the stack.

The agent then checks if (9) is satisfied in the current state (0), which it is, so (9) is popped from the stack. The Bound Action Rule in (8) is then popped and executed to bring the world state into that shown in (10). The agent checks (7) to determine whether or not the Action Rule was effective, and then pops (7) and (6) from the stack. Because (5) is also satisfied in the current state (10), it is popped from the stack. (4) is then popped and executed bringing the world into the state shown in (11), and checked with (3). (3) and (2) are popped. Because the initial Full Goal Set (1) is satisfied in the current state (11), it is also popped from the

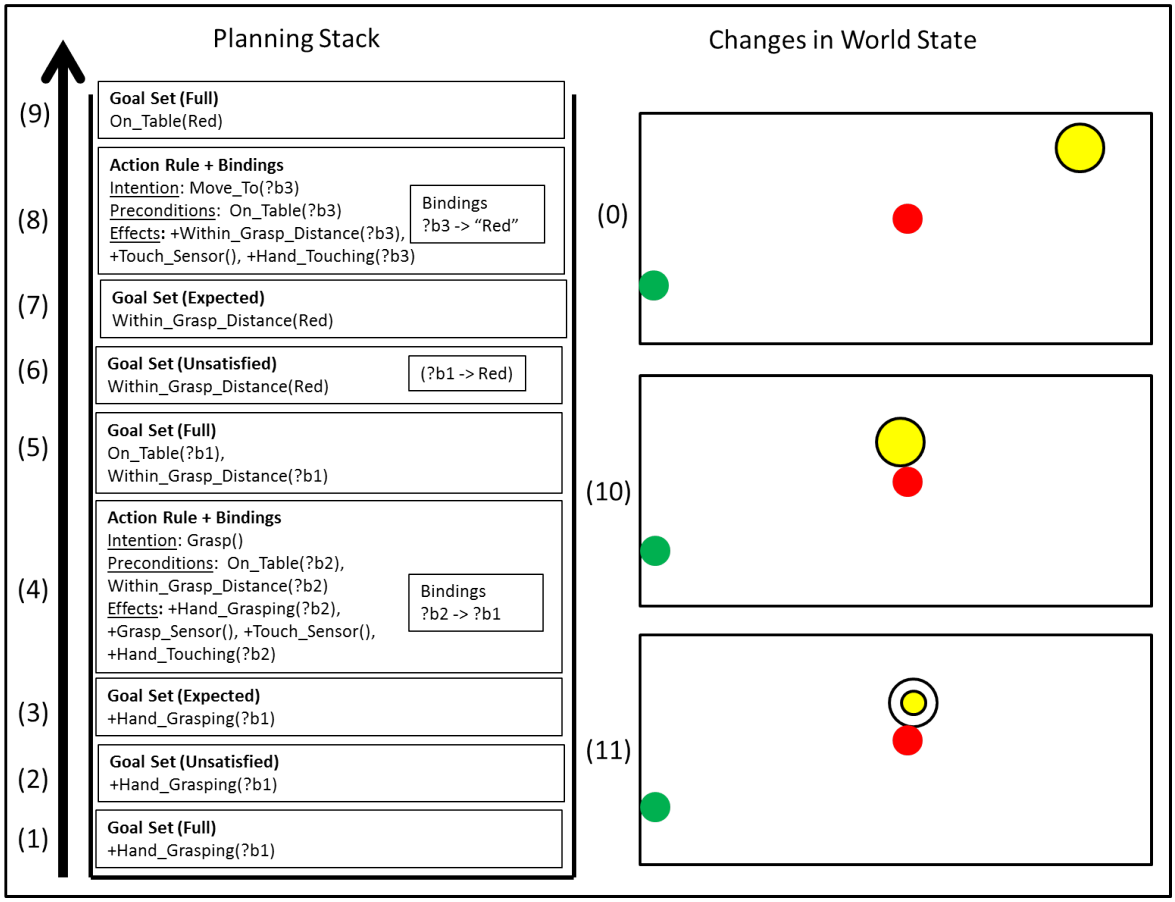


Figure 6.1: Planning to grasp a ball

stack. As the stack is now empty, the planning is complete; the plan consisted of the Move_To action followed by the Grasp.

If the agent had found that (5) was unsatisfied after carrying out (8), it would have put the unsatisfied goals back on the stack as a new Unsatisfied Goal Set and resumed planning. Because ?b1 is unbound in (5), it is possible for it to have been bound to Green instead on the second attempt. This means that even if the agent accidentally knocked Red off the table, it could still complete the plan.

6.3 Matching Goal Sets Against the Current World State

The agent matches Full Goal Sets onto the current world state in order to determine which goals of the Full Goal Set should be put into an Unsatisfied Goal Set. The same matching process is also used for determining whether or not a Bound Action Rule achieved the goals it was expected to. When a Goal Set contains no variables, the matching process simply requires checking whether or not the set of Qualitative Facts in the Goal Set exist in the qualitative representation of the current world state. When the Goal Set contains variables, the agent attempts to find objects to bind to the variables in order to satisfy as many of the goals as possible. Any bindings that were identified in this process are applied to the unsatisfied goals. Any non bound variables in the unsatisfied goals are kept as variables.

For example, in Figure 6.1, the qualitative fact hand_grasping(?b1) in (1) could not be

satisfied. Therefore, ?b1 remained in the Unsatisfied Goal Set (2). Whereas in (5), binding Red to ?b1 satisfied On_Table(?b1). This binding of ?b1 to Red was then reflected in the Unsatisfied Goal Set (6) - Within_Grasp_Distance(Red). In (5), ?b1 could alternatively been bound to Green, as this also would have satisfied one goal. Where there are multiple ways of maximising the number of satisfied goals, a random one is chosen. If the Unsatisfied Goal Set in (6) had not been satisfied, then when the agent re-planned from (5), it would have been possible for ?b1 to be bound to a different value in the new Unsatisfied Goal Set.

6.4 Selecting and Binding an Action Rule to an Unsatisfied Goal Set

This simple version of the planner used several heuristics to choose an Action Rule for an Unsatisfied Goal Set. These were roughly weighted in the order they are discussed, with the first ones holding the most weight and the later ones holding the least.

Maximise the number of goals expected to be achieved: While it is not essential for the current Unsatisfied Goal Set to be fully addressed with the chosen Bound Action Rule as the agent can re-plan for the underlying Full Goal Set again later if any goals were not satisfied, it is desirable to minimise the total number of steps carried out in the plan by picking Action Rules that satisfy as many goals as possible.

Minimise the number of unsatisfied preconditions: Action Rules with few unsatisfied preconditions in the current state of the world were preferred. This was intended to help the agent pick simple Action Rules and to minimise the depth of the stack. It was helpful in some cases, although it caused the agent to gravitate towards Action Rules with unreliable preconditions once a lot of Action Rules had been learnt.

Choose Action Rules with many effects over Action Rules with fewer effects: Action Rules with many effects were preferred. Action Rules with many effects are less likely to be a subset of multiple larger Action Rules with very different preconditions. When multiple different sets of preconditions can result in an effect set, many or all of the preconditions don't get enough support, making the overall Action Rule unreliable. This often happens with small redundant Action Rules. This heuristic was replaced in Chapter 7 with a more sophisticated detection of redundant Action Rules.

Minimise the number of variables: The number of variables in a plan is minimised by avoiding Action Rules that introduce extra variables that cannot be bound to constants in the Unsatisfied Goal Set. Such variables add unnecessary extra complexity to a plan.

Experiments showed some of these heuristics were not very effective. The revised version discussed in Chapter 7 refines them considerably.

6.5 Handling Looping and Other Failures

6.5.1 Limiting stack depth and maximum plan size

A sophisticated planner must have mechanisms for dealing with planning loops. For example if an Action Rule specifies A must be true in order for B to occur, it is possible the agent

has learnt another Action Rule specifying B must be true for A to occur. Such rules can cause infinite planning loops as the agent continually tries to find Action Rules to satisfy preconditions. Because the planning ability of human infants is very limited, it was not desirable to use sophisticated planning loop detection. However, a very simple mechanism for dealing with loops arises from infants' finite attention span and their limited working memory. The planner has a maximum plan depth and simply prevents any plan from growing beyond the limit. A side effect of this is that planning loops will automatically be broken. In addition, the total number of Action Rules executed in a plan is constrained.

Every time the stack is at the maximum depth, the agent immediately carries out the Action Rule on top, even if the preconditions are unsatisfied. In many cases, this solves the problem described at the start of this subsection. Often the looping preconditions were learnt just because they were often true in the learning examples used by the Action Rule learner, and the Action Rule learner cannot distinguish between what is merely often true and what actually needs to be true. Executing one of the Action Rules in the loop often satisfies the Goal Set below it, and thus breaking the loop. If the preconditions were necessary then the Action Rule will not accomplish the Expected Goal Set, but in this case the planner will simply try again to satisfy the lower Full Goal Set. There is a reasonable chance that the executed action will have changed the world in some way so that the planner can now achieve the goal a different way.

Once the agent has carried out the maximum number of Action Rules allowed, it ignores all preconditions and goes down the stack carrying out each of the Action Rules currently on it as a last attempt to satisfy the original Full Goal Set. The combination of limiting the maximum depth and the maximum total number of Action Rules ensures the agent eventually gives up, if it gets into a planning loop.

6.5.2 Preventing repetition

In order to ensure the same fruitless Action Rules are not repeatedly attempted in a plan, the agent keeps track of all Full Goal Sets seen in a plan and the Action Rules attempted for them. Once an Action Rule has been used too many times for an underlying Full Goal Set and repeatedly failed, it is blacklisted from that Full Goal Set for the remainder of the plan. A constant is used to say how many times the Action Rule can fail before going on the black list. For the experiments in the thesis, it was set to three. Given that the learnt Action Rules may not have perfect preconditions, it was considered important to allow more than one attempt at using the same Action Rule.

6.5.3 Handling unachievable goals

Sometimes, the agent will be unable to find a suitable Action Rule for an Unsatisfied Goal Set. This is handled by removing the Unsatisfied Goal Set and underlying Full Goal Set from the stack. This will result in the stack either being empty or having a Bound Action Rule on top. If the stack is empty, the agent gives up. Otherwise, the Bound Action Rule will now be executed. In some cases, if the impossible preconditions were unnecessary this will result in a successful plan.

One situation this mechanism helps with is when the agent has a goal involving an object such as to be grasping any object. If the object it was trying to grasp gets knocked off the table accidentally, the On_Table qualitative fact cannot be satisfied. Because it is impossible to satisfy, the agent will continually remove Goal Sets from the stack until it is down to a point at which it can bind a different object to the variable. If the original goal was to be grasping the specific object that fell from the table, the plan will fail.

6.6 Using the Expected Goal Set to Evaluate the Bound Action Rule

Some Action Rules repeatedly fail to produce the expected effects when executed deliberately in a plan, even when all preconditions were satisfied. This can occur whenthe Action Rule has deterministic rather than probabilistic effects. Often there will be multiple Action Rules with the same preconditions but different effects. Ideally the agent wants to use the Action Rules that are the most likely to be successful.

For example, hitting an object that is on the table might result in the object falling from the table, or it might result in the object moving from its original position but remaining on the table. An Action Rule exists for each possible effect set, and both have the same precondition specifying the object is on the table, and an intention specifying to hit the object. Because the two effect sets are contradictory, only one of the effect sets can occur when the intention is carried out with the preconditions satisfied. However, the agent cannot currently learn about the relationship between Action Rules with the same preconditions but different effect sets, and thus cannot identify such situations in the Action Rule learner.

As a first step towards learning about such Action Rules, the planner was used to identify Action Rules that don't achieve their expected effects when executed in a plan. Each Action Rule has a count for the number of successes and a count for the number of failures, which are used to calculate the probability of success. After a Bound Action Rule is executed by the motor control policy, the agent passes the Expected Goal Set below it to the visual system. The visual system matches the goals in the Expected Goal Set against the current state using the process outlined in Section 6.3. The Bound Action Rule is said to have succeeded if all goals were satisfied, and failed otherwise. Successes and failures are added to the relevant counts on the Action Rule that is contained within the Bound Action Rule. In cases where the preconditions were known to not be satisfied (see Section 6.5 for situations this may occur), the result from checking the Expected Goal Set is not added to the counts. The probability of success is one of the heuristics outlined in Section 6.4

The algorithm could be further extended to use unexpected results to refine Action Rules. Cases where the expected effects did not occur, despite the preconditions being satisfied, might need additional negative preconditions. In addition, cases where not all preconditions were satisfied yet all expected effects occurred suggests some of the Action Rule's preconditions are unnecessary and could be dropped. This learning is outside the scope of the thesis so was not implemented, although is discussed in Chapter 9.

6.7 Evaluation and Limitations of this Planner

The agent was able to use the planner and learnt Action Rules to plan for many one step plans such as touching objects, making sounds, causing sensor input, etc. It could also do some two step plans such as moving to and grasping an object.

The agent cannot plan for goals where the nearest point on an object is an unsuitable target. For example, to make an object touch the near wall, the agent needs to hit the object from behind relative to the near wall. But depending on the current hand position, hit might target the object from the wrong side. This same limitation prevents the agent from planning effectively for goals that make objects no longer on the table, or putting objects together. The critical issue is that the Action Rules can only specify target positions by specifying objects, but they need to be able to specify positions at a finer level of detail.

In addition, some of the heuristics used for choosing Action Rules for Goal Sets were ineffective, or had undesirable implications. Preferring Action Rules that had few unsatisfied preconditions biased the agent towards incomplete and unreliable rules, particularly

as the total number of learnt Action Rules got larger. The number of effects in the Action Rule heuristic also proved to not be as effective as expected at identifying rules that are not redundant. Efficiency was also an issue, as sometimes the agent was taking a long time to pick a rule that ended up being ineffective. A desirable feature for a planner inspired by infant development is for it to minimise "thinking" time and maximise "trying" time.

Chapter 7 discusses how the limitations outlined were addressed in an improved version of the Action Rule learner and planner. The major changes were in the Action Rule representation, and although there were some changes to the planner, the basic algorithm of the planner did not have to change.

Chapter 7

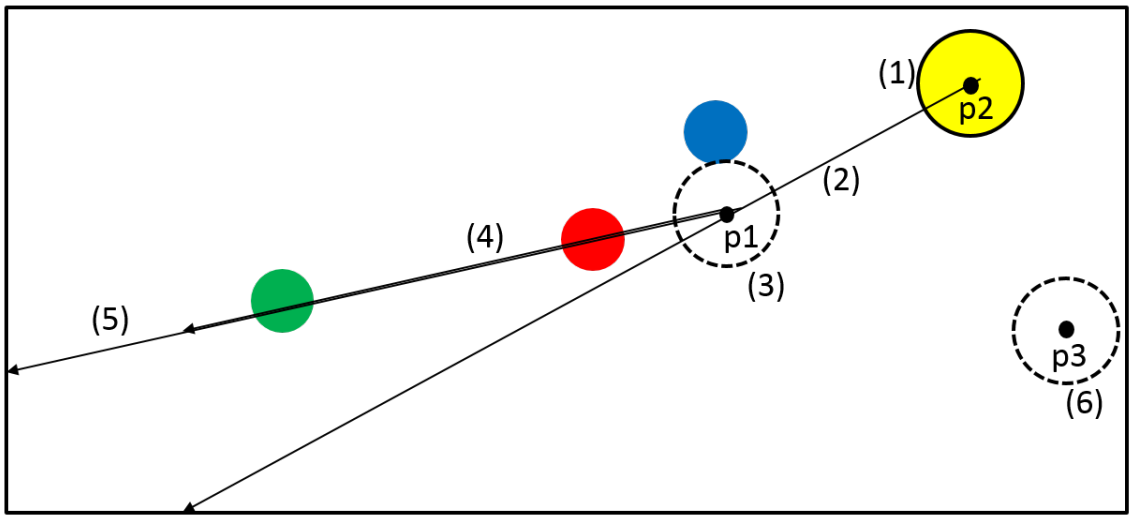
Improving the Action Rule Representation

For goals such as making an object touch a particular wall, the agent could use the Hit action to hit the object to make it move to the wall. However, it is only possible for the hit action to result in the object touching the desired wall if the point on the object that was hit is behind the object relative to the wall. This requirement is impossible to express with the representation outlined in Chapter 5, because that representation only allowed for moving to the nearest point on the target object.

This chapter discusses how an additional variable type, *places*, was added to the world representation in order to address this limitation. A place is a quantitative x, y point in the world that is made qualitatively significant by its relationship to the hand and/or objects or pairs of objects. Places allow for a richer description of the intention, preconditions, and effects of an action rule because they can be used to describe an intention target that is more than just the point nearest to the hand that touches a specified object. For the example above, places allow the agent to learn that in order to get the effect touching(?b1, ?w1), it should hit place ?p1, where ?p1 satisfies the preconditions Place_Touching(?p1, ?b1) and Place_Behind(?p1, ?b1, ?w1) (that is, ?p1 is behind ?b1, relative to ?w1).

Implementing places required enhancing the Visual System to recognise qualitative facts involving places, to include qualitative facts for interesting places (current hand position and target hand position) in the representation of a state (see Section 7.1), and to be able to identify a place that satisfies a set of given qualitative facts containing the place (see Section 7.3). Initially these additional qualitative facts were treated the same way as standard qualitative facts by the Action Rule Learner, but this did not work because effects containing qualitative facts containing places do not make sense. Also, preconditions containing places are essential for the intention to be meaningful, but always had precondition support below the threshold due to the same effects being able to be caused by slightly different sets of qualitative facts, and sometimes even by chance. Because these qualitative facts are essential for the intention to be meaningful, they were removed from the preconditions and put in a new part of the rule called the *constraints*. The constraints are treated as part of the intention, and thus multiple action rules can have the same effects but different constraints. This was implemented by modifying the storage of Action Rules, so that each effect set in the lattice can have multiple Action Rules attached to it. The details of the algorithm are discussed in Section 7.2.

In addition, some of the heuristics used for selecting the best Action Rule to acheive a Goal Set were modified and updated for the places version of the planner. These changes are discussed in Section 7.4



(1) Hand_At(p2)
(2) Place_Behind(p2, p1, Near_Wall)
(3) Place_Touching(p1, Blue)
(4) Place_Behind(p1, Red, Green)
(5) Place_Behind(p1, Red, Left_Wall)
(6) Place_Touching(p3, Right_Wall)

Figure 7.1: Some of the qualitative relationships in the current world state involving places p1, p2, and p3. The larger solid circle is the hand and the smaller solid circles are the Red, Green, and Blue balls

7.1 Adding Places to the Representation

7.1.1 Qualitative facts with places

The visual system was extended to identify contact and alignment relationships between quantiative points of interest and the hand and objects in the world. These relationships were represented as qualitative facts with the predicates Place_Touching, Place_Behind, and Hand_At.

Place_Touching(place, any) : The first parameter is a place, and the second is a place or object the hand would be touching if it was at the place specified by the first parameter.

Place_Behind(place, any, any) : The first parameter is a place, and the second and third parameters are places or objects that ensure all three parameters are aligned in the order of first parameter, second parameter, and then third parameter.

Hand_At(place) : The parameter specifies a place the center of the hand is currently on.

Given a set of quantitative points of interest, the vision system calculates all place qualitative facts and includes them in the state representation. Figure 7.1 shows an example of the visual system finding qualitative facts with places. For clarify in the diagram, some relationships and their corresponding qualitative fact were omitted.

7.1.2 Generating learning examples with places

The agent uses the same learning example generation algorithm as presented in Chapter 5. In order to include places, interesting quantitative points are defined as being the initial position and target position (for actions with a target) for the initial state of the learning example, and the initial, target (for actions with a target), and final (if different to initial and target) positions for the final state in the learning example. Qualitative facts containing the quantitative points of interest are generated as outlined in Section 7.1.1. The intention variables of the learning example are the initial position, and target position if it exists.

7.1.3 Finding suitable quantitative points for place variables

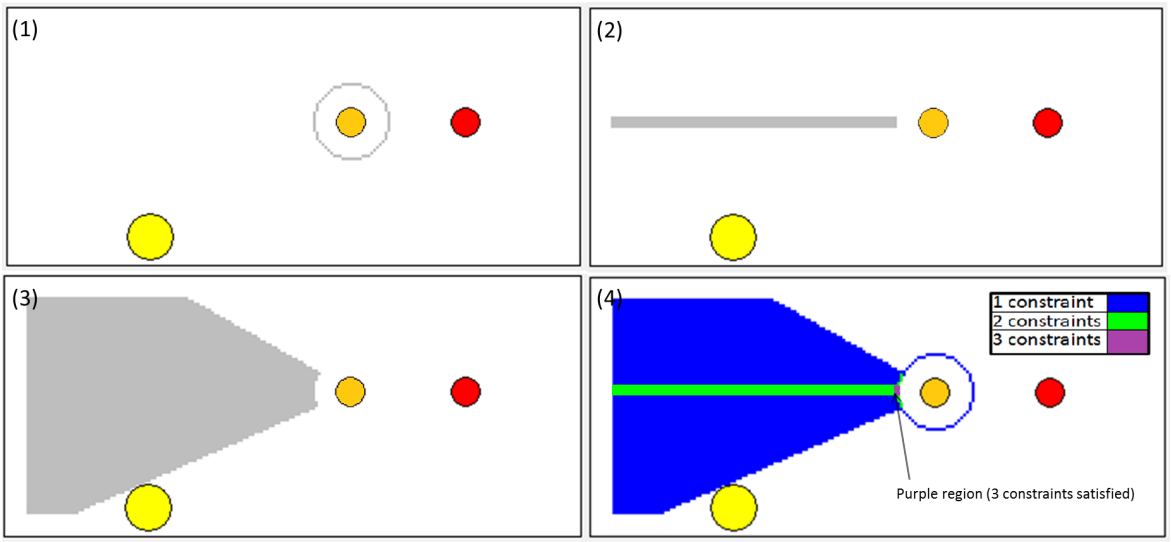


Figure 7.2: (1) Satisfying region for Place_Touching(?p1, Orange), (2) Satisfying region for Place_Behind(?p1, Orange, Red), (3) Satisfying region for Place_Behind(Orange, Right_Wall), (4) Overlapping satisfying regions for (1), (2), and (3) in order to find best points

In order to match a set of qualitative facts containing place variables onto the current state of the world, the agent needs to be able to find quantitative points for the variables that satisfy all or as many of the qualitative facts as possible. This is a visual task people are good at doing. For example with ease they can identity a point on an object such that if they hit that point the object will be hit into another object.

Finding suitable quantitative points for a place variable was implemented as part of the hardcoded visual system using Integer Programming. The space of possible quantitative points was made discrete by using a coarse grid over the world (one world unit between each grid line). For each qualitative fact containing the quantitative variable, all grid points that would make the qualitative fact true in the current world state were put into a set. It was assumed all other variables in the qualitative fact had values bound to them: the pre-processing bound randomly selected objects to variables if necessary, and where a qualitative fact contained multiple place variables, the agent ignored it until a suitable quantitative point had been found for at least one of the place variables, using the other qualitative facts containing it. The algorithm then counted the points found for each of the qualitative facts in order to identify points that satisfied the most qualitative facts. A quantitative point that satisfied the maximum number of qualitative facts was then randomly chosen. This Integer

Programming approach is not computationally optimal, although was sufficient for the table world used in the thesis. Linear Programming techniques that describe each qualitative fact as a set of constraints could be used for efficiency in complex worlds, but human visual systems may use a different approach altogether.

Figure 7.2 shows an example of the agent finding satisfying regions for variable ?p1 under three constraints; (1) Place_Touching(?p1, Orange), (2) Place_Behind(?p1, Orange, Red), and (3) Place_Behind(Orange, Right_Wall). The satisfying points within the regions were counted, and the amount of overlap for each point is shown in (4). The purple region is where the three constraints overlapped, and therefore the agent will choose a random point in that region.

7.2 Adding Constraints to Action Rules

In order to incorporate place variables and constraints into Action Rules, the intention variables are defined to be an initial place and a target place. The initial place is the place the hand is immediately before the Action Rule is executed, and the target place is the place the agent intends to go to with the action. Some actions such as Grasp and Ungrasp only have one intention variable (the initial place). Due to the simplicity of the motor control for the fingers, a target variable for these is unnecessary. The constraints of the Action Rule then specify relationships between the intention place variables and the object variables that appear in the effects and preconditions.

The Action Rule learner treats constraints as part of the intention in that Action Rules with the same effect set but different constraints are not equivalent. A reason for this is that a key role of the constraints is to give meaning to the intention. For example, Hit(?p1, ?p2) merely specifies a hit to somewhere from somewhere else. It only becomes a meaningful action when ?p1 and ?p2 are narrowed down by the constraints to be places with particular properties. Like with preconditions, an "attention" heuristic is used to prevent the agent learning irrelevant constraints related to variables not present in the effects (see Section 5.3.1 for an explanation of this heuristic). Additionally, constraint qualitative facts are not put in the pool of qualitative facts that can be learnt as effects or preconditions.

The planner treats constraints somewhat like preconditions in that they are included in Full Goal Sets and the agent should attempt to satisfy them. However they are not put into Unsatisfied Goal Sets, even if they are unsatisfied when planning because it makes no sense to try to plan to achieve a constraint on a place. For example, a place is either touching the wall or not, and cannot be made to touch the wall if it is not. Instead, given constraints the agent attempts to bind quantitative places to the place variables to make them true. When choosing places it attempts to maximise the number of satisfied constraints. One exception is the Hand_At qualitative facts which are treated as normal qualitative facts and thus can be learnt as preconditions and effects rather than constraints. This allows an unsatisfied Hand_At qualitative fact to be put in the Unsatisfied Goal Set in cases where the quantitative place bound to the initial place variable was not the current hand position. The reason for this exception is that it is feasible to act to make the hand at a place.

Figure 7.3 shows an example of an Action Rule with place variables and constraints. The intention contains two parameters, (1) and (2). This means that the hand starts at ?p21 (1) and hits ?p38 (2). The effect set specifies the Action Rule will end with ball ?b42 touching wall ?w92. The four constraints (3), (4), (5), and (6) specify how ?p21 and ?p38 should be related to these effect variables. (3) Specifies the initial hand position ?p21 must be behind ?b42 relative to ?w92. (4) Specifies that the target hand position ?p38 must also be behind

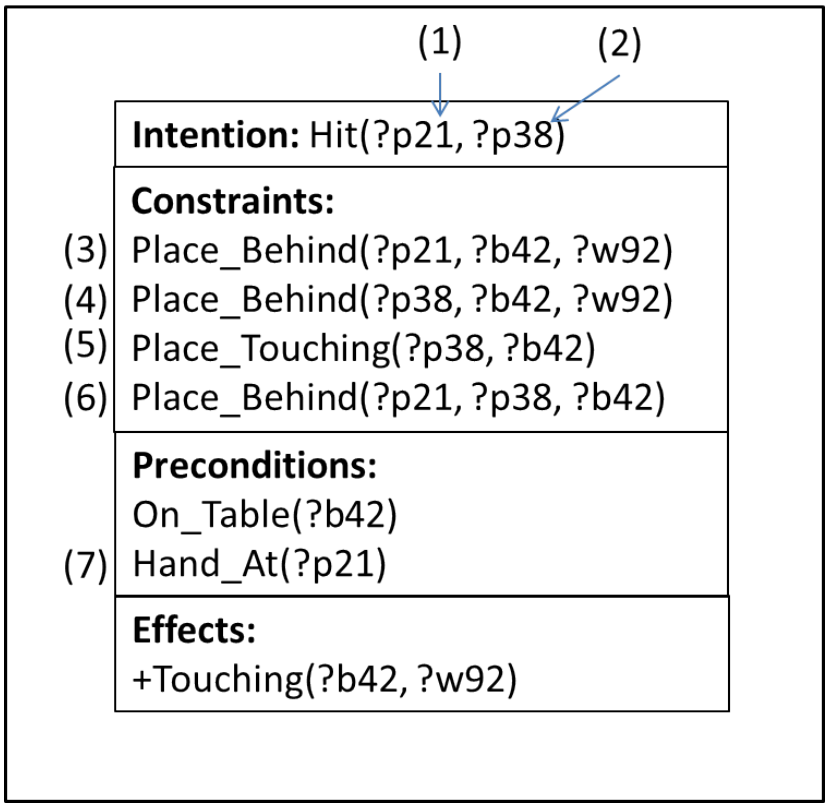


Figure 7.3: An Action Rule with place variables and constraints

?b42 relative to ?w92. (5) Specifies ?p38 must be touching ?b42. And finally (6) specifies that ?p21 must be behind ?p38 relative to ?w92.

7.2.1 Modifying the way Action Rules are stored

Because two Action Rules are distinct if their effect sets are equivalent but their constraints vary, the agent could learn many Action Rules with the same effect set. In order to avoid redundant storage and allow the agent to easily identity the best intention and set of constraints for an effect set, the storage of Action Rules was modified. Instead of creating a separate effect set in the lattice for every pair of intention and constraints, each effect set in the lattice now holds a list of all Action Rules that contain it. In order to implement this, the apply_example_to_effect_set function from Algorith 5.1 was modified.

Algorithm 7.1 shows the updated algorithm. Normal preconditions were learnt using Algorithm 5.2. Supporting examples for precondition learning were those where the effect set, intention, and constraints of the learning example all supported the Action Rule.

This code assumes the propagation process was initiated by identifying singleton effect sets that are supported by the learning example using the algorithm outlined in Chapter 5. If the learning example supports the effect set (unlike in Chapter 5, the intention is not yet taken into account), the agent checks through all Action Rules attached to the effect set to find the one that has equivalent constraints to the learning example. To do this, the learning example is bound to each Action Rule in order to check bindings. If such an Action Rule exists, the learning example is applied to it. Otherwise, a new Action Rule is created and added to the effect set's Action Rule list, and the learning example is applied to it.

```
define updated_apply_example_to_effect_set(example, effect_set)
   if example supports effect_set (intention and constraints are ignored)
      increment support for effect_set by 1
      matching_ar ← find action rule attached to effect_set supported by example
      if no matching_ar was found
         matching_ar ← new action rule with example's constraints (generalised)
         add matching_ar to the action rule list attached to effect_set
       if support for effect_set = COMBINING_THRESHOLD and all other action rules are
below thrshold
         combine_with_neighbours(effect_set) (see chapter 5)
      if support for matching_ar > COMBINING_THRESHOLD
         for each super_es in effect_set's supersets
            updated_apply_example_to_effect_set(example, super_es)
```

Algorithm 7.1: An updated version of apply_example_to_effect_set function that was introduced in Chapter 5 allowing the agent to learn multiple intentions and constraints for the same effect set. See Algorithm 5.1 for the full details.

The conditions in which the propagation procedures are carried out are also different. An effect set initiates the size n+1 effect set generation the first time one of the effect set's Action Rules goes above the combining threshold. Combining is done with same size effect sets that satisfy the heuristics for a potential successful combine that also have at least one of their attached Action Rules above the combining threshold. Propagation of a learning example to supersets of an effect set is only carried out when the Action Rule to which the learning example was matched was above the threshold. These conditions ensure new Action Rules are only generated when they have sufficient subset support.

For constraints in a learning example to match an Action Rule, the Action Rule and learning example must contain the same number of constraints, and the constraints must be able to be fully mapped between the Action Rule and learning example. This is different to matching a learning example onto an effect set, as it is okay for the learning example to have additional effects that were not mapped.

7.3 Matching Goal States with Places to the Current World State

Adding constraints required the planner to be modified as well as the Action Rule learner, though the changes were fairly small. The key difference is in determining which goals in a Full Goal Set are not yet satisfied and need to go into an Unsatisfied Goal Set.

The agent matches a Full Goal Set against the current state of the world by binding objects to variables in the goals. This process was extended to handle place variables. Given a set of goals in a Full Goal Set, the agent divides the goals into several lists:

1. Goals that contain no unbound place variables (place variables that have already been bound to quantitative points are included in this list).
2. Goals containing places but do not directly involve the hand.
3. Goals that involve the hand and places (in the current implementation this list only contains goals with the hand_at predicate)

The goals in 2) are further divided into sublists: one for each unique place variable. Goals containing multiple places are put into one sublist. The agent ensures this is done in such a way that the matching algorithm will never get into a situation where it is trying to

achieve a goal with multiple unbound place variables. Using these lists, the agent matches the goals against the current state of the world in the following order.

Firstly, the goals in 1) are matched against the current world state. This process tends to bind objects to most, if not all, of the object variables in the Goal set. If any object variables in the Goal Set are still unbound and are present in list 2), randomly chosen objects are bound to them. The agent then iterates through each sublist for 2), finding a quantitative point for the featured place variable using the method discussed in Section 7.1.3. The order in which the sublists are processed ensures any other place variables in the goals in the list were already bound in a previously processed list. Finally, the goals in 3) are matched against the current world state. Because Hand_At with the initial place is the only fact that can go in this list, if the place variable is still unbound, the current hand position is bound to it in order to satisfy it. Any unsatisfied goals from 1) and 3) are put into the Unsatisfied Goal Set. Unsatisfied goals from 2) (which originated from the constraints component of the Action Rule) are ignored even if they were unsatisfied. The planning that would be necessary to satisfy them is beyond the scope of a simple planner striving to behave like an infant. When the goal from 3) is included in the Unsatisfied Goal Set, it allows the agent to plan to move to that place using an Action Rule with a Hand_At effect.

7.4 Improving the Action Rule for Goal Set Selection Heuristics

Some of the problems identified in the initial implementation of the planner involved the heuristics for selecting a good action rule to achieve a goal set. In particular, the more Action Rule learning the agent had done, the longer it was taking to select Action Rules for plans, and the less reliable the chosen Action Rule was. Therefore, the heuristics were changed by dropping the heuristics based on the size of the preconditions and the number of unsatisfied preconditions, modifying the subsumed support heuristic, and adding three new ones. The improved set of heuristics consists of the following five heuristics:

1. The number of current goals expected to be achieved with the Action Rule,
2. The support count
3. The non-subsumed support count
4. The number of new variables introduced
5. Reliability history from previous planning with the action rule
6. A connectivity score for the variables in the action rule, that attempted to predict whether or not the rule could easily be replicated. These heuristics were combined in order to compare pairs of action rules.

This section begins by describing how each heuristic was defined and, and concludes with the algorithm used to search the lattice for an Action Rule for a given goal set.

7.4.1 The number of current goals expected to be achieved with the Action Rule

Heuristic 1 is intended to bias the agent towards simple plans to exploit the advantages of effect sets. This heuristic was helpful in most cases although sometimes in cases where it was impossible to find an effect set for all the goals the agent would choose a bad subset of them.

7.4.2 The support count of the Action Rule

Heuristic 2 is intended to bias the agent towards Action Rules that have had many supporting learning examples because they were thought to be more reliable. It turned out to not be as effective as expected because some Action Rules with very low support were reliable and useful for planning. This was because some of the more interesting effects occur rarely by chance, so few learning examples were generated for them. If the learning examples were good ones, the preconditions of the Action Rule tended to be reliable meaning the low support count was not problematic.

7.4.3 The subsumed support count of the Action Rule

Heuristic 3 is intended to bias the Action Rule selection towards an Action Rule with a large effect set over an Action Rule with a subset of that effect set on the basis that the large effect sets are more complete descriptions of the action and the subset rules are redundant, as they are explained by a larger action rule. Each time a learning example is added to the support count of an effect set and Action Rule, the learning example is then subtracted from the support of all subset effect sets and Action Rules. The non-subsumed support count of an action rule is the support count after these subtractions have been carried out.

It is unclear whether this heuristic was helpful. Experiments suggested the world and algorithm are not yet complex enough for it to be helpful. It would probably be essential if the planner was extended to prevent undoing previously achieved goals as the information needed would be held in negative effects of large effect sets.

7.4.4 The number of new variables introduced

Heuristic 4 is intended to prevent plans becoming unnecessarily complicated. While the agent tried to pick larger Action Rules where possible, it also tried to not choose ones that added extra variables beyond the ones that were in the goal. This was because it would then have to find objects to bind to the variables and could quickly get into a situation where it is trying to satisfy preconditions on objects that should have been ignored in the plan (and in the process of satisfying the preconditions, adding even more variables).

The version of the agent used in the evaluation did not allow any new variables to be introduced. Allowing a limited number of additional variables in an overall plan would be beneficial in future versions of the agent. For example, where the goal is to make the touch sensor true, the most reliable plan is to choose an object to move to. The agent cannot currently do this.

7.4.5 Reliability history of an Action Rule

Heuristic 5 biases the Action Rule selection towards Action Rules that have been shown to be reliable. To do this, the agent used previous planning history. Each action rule had two fields for this part of the algorithm: successes and failures. Each time the action rule was used in a plan, the appropriate field was incremented by one depending on whether or not the action achieved all the effects it was expected to.

A score in the range (0, 1] was generated by dividing the number of successes by the sum of successes and failures. The positive count and negative count were both initialised to 10 in order to prevent divide by 0 errors and to give unused rules an initial score of 0.5. A slightly larger increment was added for positive examples than negative examples so that the value would go up even if the Action Rule was successful only half of the time.

7.4.6 Variable connectivity score of an Action Rule

Heuristic 6 helped the agent to predict reliability of rules before it had tried them. This gave it a starting point before it was able to use heuristic 5. Even without experience using an Action Rule, the agent can identify some unreliable Action Rules by analysing how the variables in the different components of the Action Rule are connected. Action Rules where the effect set and intention variables are connected through the constraints and preconditions are more likely to be reliable.

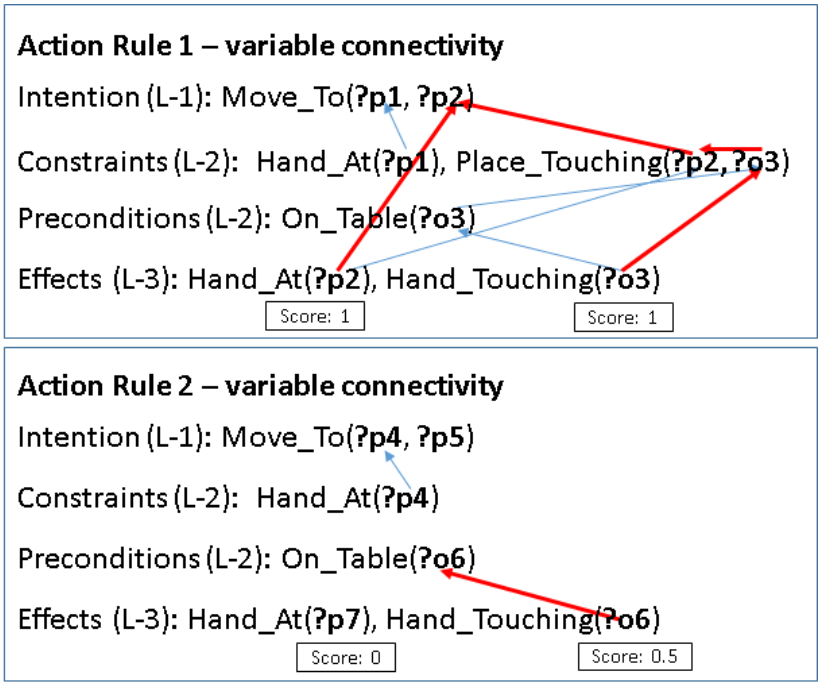


Figure 7.4: Calculating the binding scores for two action rules

For example, suppose the agent needs to plan for the goal set {+Hand_At((10, 20)), +Hand_Touching(Red)}. Two possible action rules for this plan are those shown in Figure 7.4. The first action rule specifies that for the move action to result in the hand touching an object, the place chosen to move to must be touching the object. In addition, the action rule specifies the hand will be at the target. The second action rule specifies that moving to a place with no constraints (that could be any valid position on the table) will result in touching the object, and being at a place that was not the place being moved to. Intuitively, the first action rule is more likely to make the goal set true.

The first rule in Figure 7.4 connects the ball in the effects to the intention by saying the target must be touching the ball, and the place in the effects to the intention by saying the hand must move to it. On the other hand, the second rule defines no relationship between the intention and effect variables.

The connectivity heuristic measures how well connected the variables in the effects are to the variables in the preconditions/ constraints, and intention. The stronger the connectivity, the more likely the action rule is easily replicable, and thus a good action rule to use in planning.

Algorithm 7.2 was used to calculate the connectivity of action rules and to generate a score in the range [0, 1]. A digraph is initialised and a node is added for each variable instance in the Action Rule. Edges are added between nodes that are considered to be related. Depth first search is then used from each node that originated from a variable in the effects

to see whether or not there is a path to a node that originated from the intention, and a weighted sum is calculated with the results. This heuristic proved to be very effective in determining which action rules were likely to be reliable, and which were not.

Initialise an empty digraph Add a node for each variable instance in the action rule. Each node should contain the variable identifier, var, it is for, the fact the variable instance is in, fact, and the level the fact is on (3 for effects, 2 for preconditions and constraints, and 1 for intention) Insert a directed edge from each pair of nodes a and b, if either (a) or (b) satisfied (a) a.level is greater than b.level and a.var is the same as b.var, OR (b) a.level is 2 and b.level is 2, and a.fact is the same as b.fact Assign a score to each level 3 node, specifying the lowest level it is connected to (implemented with a depth-first search) If the node is connected to a level 1 node, assign a score of 1 Else if the node is connected to a level 2 node, assign a score of 0.5 Else assign a score of 0 Insert the scores into a list sorted from lowest (worst) to highest (best) Iterate through the list, assigning decreasing weights to each score (e.g. halving the weight each time) Calculate and return the weighted sum

Algorithm 7.2: Calculating the connectivity score for an Action Rule

When the algorithm is applied to the two Action Rules in figure 7.4, the first Action Rule has effect variable node scores of 1 and 1, and the second action rule has effect variable node scores of 0 and 0.5. To combine the node scores, the agent used the formula $w = (2^{(n-k)})/((2^n) - 1)$, where n is the total number of weights needed for the Action Rule (i.e. the number of variable instances in the effects) and k is the position of the score in the sorted list that the weight is for. Using this formula with n = 2, weights of 1/3 and 2/3 are generated. The final score for the first Action Rule was 1 and the final score for the second Action Rule was 0.167. This allows the agent to predict that the first Action Rule is more reliably replicable than the second Action Rule.

7.4.7 Algorithm for choosing an Action Rule

Finally, the agent needs to put these heuristics together in order to choose a good Action Rule for the Unsatisfied Goal Set. It starts by finding an effect set in the lattice that is a subset of the goal set that firstly maximises the number of effects (goals expected to be achieved) and secondly a heuristic score for the other heuristics.

It then greedily searches up the sub lattice of the supersets of the chosen effect set by hill climbing. At each step of the search, it checks whether any of the current effect set's supersets have a higher heuristic score. If it does the superset with the highest heuristic score becomes the current effect set. Once this condition is false, the current effect set is selected for achieving the goal set. Originally, the agent would also choose a superset if it subsumed the current effect set. However, this heuristic was dropped in the evaluation because it seemed to be unhelpful.

The heuristic score for an Action Rule is calculated by using a weighted sum of the various heuristics. The heuristic score for an effect set is the maximum of the weighted scores for each Action Rule attached to it. Several calculations were tried and an optimum was not found as this thesis was focussed on outlining an overall approach to the problem rather

than optimising the parameters. Using *connectivity_score * reliability_score + non subsumed support count/1000* seemed to give reasonable results and was used in the evaluation.

7.5 Summary

These changes extended the Action Rule representation to use place variables. This allowed the agent to learn Action Rules with a richer intention. The heuristics were also updated to improve the efficiency and effectiveness of the Action Rule selection. With the place variables, the agent is now able to learn more interesting Action Rules that describe relative directions. This allows the agent to plan to make an object touch the wall without its hand getting in between the wall and the object for example. A consequence of adding constraints meant that more learning examples were needed to learn some Action Rules. The improved heuristics made this feasible as they eliminated the problem in the previous version of the agent becoming less effective as it learnt more.

+hand_touching(?b1) and +grasp_sensor()

Chapter 8

Evaluation

To evaluate the agent presented in this thesis it is not possible to use performance benchmarks, as this is the only system we know of that attempts to learn and plan with meaningful rules learnt from random exploration of a simulated world that uses realistic physics (other than QLAP). The planner is not intended to be high performing, as its behaviour is inspired by human infants who have a lot of difficulty with planning tasks. This evaluation is instead interested in whether the agent can learn meaningful Action Rules and sometimes plan successfully for a goal, as future versions of the agent will be able to use successes in order to increase planning reliability. Therefore, this evaluation focusses on examples of Action Rules the agent was able to learn and goals it was able to plan for using the learnt Action Rules.

This achieves the thesis objective of showing that meaningful Action Rules can be learnt from random exploration and then used to solve planning problems. All Action Rules presented in this section were learnt from the agent randomly carrying out motor actions in its motor control policy and then using them in the Action Rule learner. No Action Rules and as little as possible common sense knowledge was built into the agent beyond being able to describe a static state of the world with qualitative facts (the visual system) and a motor control policy.

8.1 Action Rules Learnt

This section presents a selection of Action Rules collected from a run of the Action Rule learner that learnt from 4000 learning examples. With significantly below 4000 learning examples, many of the good Action Rules do not yet have sufficient support and the system uses poor quality Action Rules in preference.

The Action Rule planner was also run for around 120 plans where the agent picked random plans out of the following list to accomplish. This is not an exhaustive list of all the plans the agent can attempt, but was rather a sample of goals to help it identify some interesting Action Rules.

- {+Hand_Touching(?b1), +Hand_Touching(?b2)}
- {+Touching(?b1, ?w1)}
- {+Touching(?b1, near_wall)}
- {+Hand_Touching(?b1), +Hand_Touching(?b2), +Hand_Grasping(?b1)}
- {+Touching(?b1, ?w1), +Touch_Sensor()}
- {+Hand_Grasping(?b1)}
- {+Touching(?b1, left_wall)}
- {+Touching(?b1, far_wall)}

- {+Touching(?b1, ?b2)}
- {+Fingers_Flat()}
- {+Touch_Sensor()}

Many of the Action Rules presented in this section were ones that the agent identified as being of a high quality. The agent identifies good Action Rules by considering those with at least two supporting examples and sorting the Action Rules by the Quality rating it has generated for each of them. The default quality rating for a rule with an ideal binding score (see Chapter 7) and at least 2 support is around 1, and this can either go up or down as the Action Rule planner attempts to use the Action Rule to achieve goals. A very bad Action Rule has a score of less than 1, and a very good Action Rule has a score up to around 2. Generally Action Rules with a score of greater than 1.2 are reliable, although an exact threshold is not used.

To construct the list below, all Action Rules with a quality score of greater than 1.1 were identified giving a list of under 100 Action Rules from which the examples were selected. Each Action Rule example shows the standard components of an Action Rule outlined in Chapter 5 and 7. The number of supporting examples is also shown, along with the quality rating.

8.1.1 Selected Action Rules for the Move_To action

The main action used by the agent is Move_To. The agent learnt that it could achieve a wide variety of goals using Move_To.

AR1

Intention	: MoveTo(?p125, ?p126)
Constraints	: Hand_At(?p125)
Preconditions	: None
Effects	: +Hand_At(?p126), -Hand_At(?p125)
Support: 887	
Quality: 1.65	

AR1 is one of the simplest Action Rules for Move_To that the agent can learn. If the hand is at one place and moves to another, it will no longer be at the old place and it will be at the new place. No preconditions are needed. This rule (and the one with only the +Hand_At fact) is useful for planning when the agent has identified a quantitative point that satisfies constraints and just needs to make the hand be at that point.

AR2

Intention:	MoveTo(?p1249, ?p1250)
Constraints:	Place_Touching(?p1250, ?b1251), Hand_At(?p1249), Place_Near(?p1250, ?b1251)
Preconditions:	On_Table(?b1251), Fingers_Flat()
Effects:	+Hand_At(?p1250), +Hand_Touching(?b1251), +Touch_Sensor()
Support: 163	
Quality: 1.6	

AR2 is the highest quality Action Rule the agent identified for making its hand touch a ball. The Action Rule contains some additional effects which commonly occur along with the hand touching the ball. In practice, the fingers don't need to be flat in order for the rule to succeed, despite the agent having learnt a precondition saying they do. This is a consequence of only learning preconditions from positive examples.

AR3

Intention:	MoveTo(?p420, ?p421)
Constraints:	Hand_At(?p420), Place_Touching(?p421, ?w423), Place_Touching(?p420, ?b422), Place_Near(?p420, ?b422), Place_Behind(?p420, ?b422, ?w423)
Preconditions:	Hand_Touching(?b422), Hand_Grasping(?b422), On_Table(?b422), hand_behind(?b422, ?w423), Touch_Sensor()
Effects:	+Touching(?b422, ?w423)
Support: 6	
Quality: 1.75	

AR3 is the most reliable Action Rule the agent identified for making a ball touch the wall. The constraints and preconditions describe having a ball grasped with the hand behind the ball relative to the wall the ball should touch, and then moving to the wall with the grasped ball. While it has low support, it has meaningful preconditions. This suggests support is of limited importance as a selection heuristic because the agent does not need many examples to meaningfully generalise and learn preconditions.

AR4

Intention:	MoveTo(?p88, ?p89)
Constraints:	Place_Touching(?p89, ?w90), Hand_At(?p88)
Preconditions:	On_Table(?w90), Fingers_Flat()
Effects:	+Hand_Touching(?w90)
Support: 407	
Quality: 1.46	

AR4 is one of the Action Rules the agent learnt for making its hand touch a wall. Again, the agent learnt that the fingers should be flat when in fact this is not necessary. The precondition On_Table(?w90) is surprising because this candidate precondition should have been ignored because it is always true (it is likely due to a bug in a special case). Nonetheless, it does not negatively impact the agent's ability to use the Action Rule, as it is always satisfied.

8.1.2 Selected Action Rules for the Grasp action

The Grasp action was a simple action but had different effects in different situations.

AR5

Intention:	Grasp(?p253)
Constraints:	Hand_At(?p253), Place_Touching(?p253, ?b254), Place_Near(?p253, ?b254)
Preconditions:	Hand_Touching(?b254), Touch_Sensor(), On_Table(?b254), Fingers_Flat()
Effects:	+Hand_Grasping(?b254)
Support: 97	
Quality: 1.84	

AR5 and its supersets are the Action Rules the agent identified for being able to make objects grasped. It was not built into the agent that grasping near a ball will make a ball be grasped, so it had to learnt this.

AR6

Intention:	Grasp(?p672)
Constraints:	Hand_At(?p672)
Preconditions:	Fingers_Flat()
Effects:	+fingers_closed(), -Fingers_Flat(), -Hand_At(?p672)
Support: 435	
Quality: 1.61	

AR6 is one of the Action Rules the agent learnt about using a Grasp action when not near a graspable ball. The -Hand_At precondition was probably learnt because of the hand moving at a very low velocity from a previous Hit or Move action. This Action Rule illustrates a limitation of not having negative preconditions: if the agent attempts to use it when near a ball, it will fail and the agent won't know why. Ideally the agent needs to be able to learn a precondition that the hand should not be touching any ball. For the purposes of this thesis, the Action Rule was good enough.

8.1.3 Selected Action Rules for the Ungrasp action

Ungrasp was the simplest action. The Action Rules for it were reliable because from the agent's perspective it is very deterministic.

AR7

Intention:	Ungrasp(?p183)
Constraints:	Hand_At(?p183)
Preconditions:	None
Effects:	+Fingers_Flat()
Support: 203	
Quality: 2	

AR7 is one of the Action Rules the agent learnt to say that ungrasping anywhere will make the fingers flat. No preconditions were learnt because this Action Rule works in any state of the world. It is the most reliable Action Rule the agent identifies because the motor control for Ungrasp is so simple that it never fails.

AR8

Intention:	Ungrasp(?p311)
Constraints:	Hand_At(?p311), Place_Touching(?p311, ?b312), Place_Near(?p311, ?b312)
Preconditions:	On_Table(?b312), Hand_Grasping(?b312), Hand_Touching(?b312), Touch_Sensor()
Effects:	-Hand_Grasping(?b312)
Support: 83	
Quality: 1.4	

AR8 is one of the Action Rules the agent learnt to say that ungrasping when grasping a ball means the ball will no longer be grasped.

8.1.4 Selected Action Rules for the Hit action

The Hit action was the most challenging action the agent was expected to learn about because Hit actions can result in wildly varying effects. Some good rules were learnt such as hitting balls can make them touch walls, although they weren't as reliable as those learnt for the other actions. The effects of a Hit action are somewhat probabilistic and would require

a finer quantitative state representation to allow the agent to reason about whether or not hitting a ball is likely to make it touch a wall based on factors such as current distance from the wall. This was not overly concerning because young children have trouble achieving a desired goal (e.g. make one ball hit another) by hitting a ball. Their hit actions tend to be very clumsy and uncoordinated.

In order to identify what the Hit action could do, experiments were run with both the bounce factor on and off (i.e. whether or not balls should bounce off the wall when they hit it)

AR9

Intention:	Hit(?p164, ?p165)
Constraints:	Hand_At(?p164), Place_Touching(?p165, ?w166)
Preconditions:	On_Table(?w166), Fingers_Flat()
Effects:	+Hand_Touching(?w166), +Touch_Sensor()
Support: 195	
Quality: 1.27	

In addition to learning that the Move_To action could be used to touch walls, the agent learnt that it could also use the Hit action.

AR10

Intention:	Hit(?p85, ?p86)
Constraints:	Place_Touching(?p86, ?b88), Hand_At(?p85), Place_Near(?p86, ?b88)
Preconditions:	+On_Table(?b88), Fingers_Flat()
Effects:	-On_Table(?b88)
Support: 4	
Quality: 1	

AR10 is one of the Action Rules the agent learnt for knocking a ball off the table. Its constraints are incomplete because the agent cannot learn the requirement to be on the side of the ball opposite the slot wall. The problem is that the "attention" heuristic used for choosing relevant preconditions and constraints prevented the agent recognising the constraint that ?p86 has to be behind ?b88 relative to the slot wall. This was because no effect directly happened on the slot wall as a result of the action, and therefore no variable for it was defined in the effects. Therefore, the constraint was ignored.

One way to fix it would be to include the slot wall in a predicate name by defining a behind drop constraint that describes when a place is behind a ball relative to a drop from the table. Another way is to try and improve the attention heuristic. The attention heuristic proved to be effective in the vast majority of Action Rules the agent learnt, so modifying it in a way that doesn't cause problems for other Action Rules might not be straightforward. Also, if given enough learning examples it would be possible for the agent to eventually learn a variant of the Action Rule that involved hitting the slot wall with a ball on the path. Such an Action Rule would have an effect saying the hand is touching the slot wall, which would allow the slot wall to be present in preconditions and constraints.

AR11

Intention:	Hit(?p3006, ?p3007)
Constraints:	+Hand_At(?p3006)
Preconditions:	fingers_moving()
Effects:	-fingers_moving(), -Hand_At(?p3006)
Support: 117	
Quality: 1.15	

While the agent learnt and identified many good Action Rules, it learnt a very large number of junk Action Rules. AR11 is one example of a meaningless Action Rule that was learnt. The fingers going from moving to not moving is not a consequence of the hit motor action but instead of the agent randomly moving its fingers when the action was started. If AR11 was used to stop the fingers moving in plans, its quality score would rapidly fall as it would be consistently unreliable.

8.1.5 Summary

The agent is able to learn Action Rules that capture the important effects that can result from its motor actions, and identify which are good using the quality heuristics. The binding score heuristic provided a good starting point for choosing Action Rules of a high quality and then the quality went up or down depending on whether or not the Action Rule did what it was expected to do. This allowed the agent to identify good Action Rules among thousands of candidates.

There are some obvious limitations, particularly in the preconditions. Because preconditions are only learnt from positive examples, the agent learnt unnecessary preconditions where a fact normally happened to be true. Also, negative preconditions saying what should not be true would have been desirable in some of the Action Rules presented in this section. An improved precondition learner is an obvious direction for future work.

Despite the limitations, Action Rules learnt and recognised as being of a high quality by the agent show that learning with random exploration is promising. The next section further evaluates the agent's Action Rules by using them in simple planning task.

8.2 Planning for Goals

The second part of the evaluation looks at the agent's ability to plan with its learnt Action Rules.

Each plan presented has a goal and a sequence of actions the agent carried out in an attempt to achieve the goal. The action sequence is shown in a figure with arrows on the initial state to show how the hand position changed. Move and Hit actions are drawn with an arrow to show approximately where the hand ended up, Grasp with a dotted circle with an arrow going towards the centre, and Ungrasp with a dotted circle and arrow going away from the center. In cases where the agent's movement was complex and not successfully working towards the goal, the path was not shown.

8.2.1 One step plans

The agent is able to plan for goals that require a single Action Rule to achieve.

Plan 1 – Goal: Fingers_Flat()

The agent achieved the goal from the initial state shown using the following Ungrasp action.

1. Ungrasp((0, 23)) with constraints: {Hand_At((0, 23))}

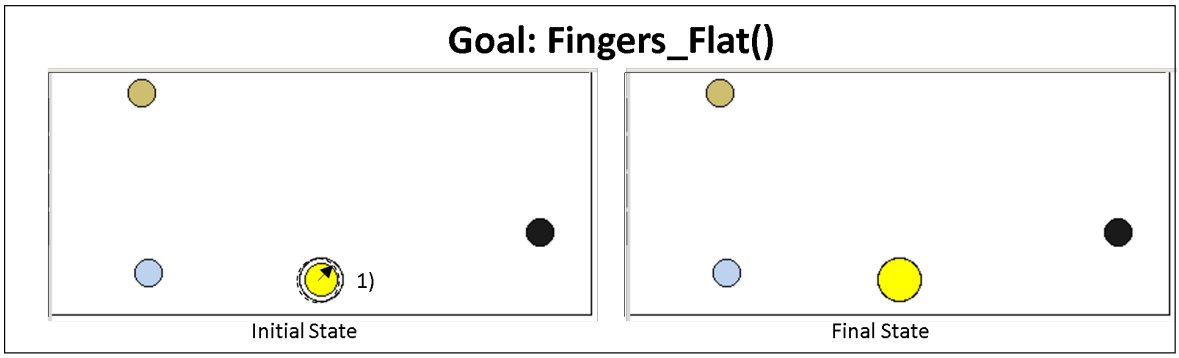


Figure 8.1: The agent planning for the goal Fingers_Flat

The agent consistently plans successfully for this goal with a one step plan.

Plan 2 – Goal: Hand_Touching('Plum3')

The agent achieved the goal from the initial state shown using the following Move_To action.

1. Move_To((0, 23), (-77, 46)) with constraints: {Place_Touching((-77, 46), 'plum3'), Hand_At((0, 23)), Place_Near((-77, 46), 'plum3')}

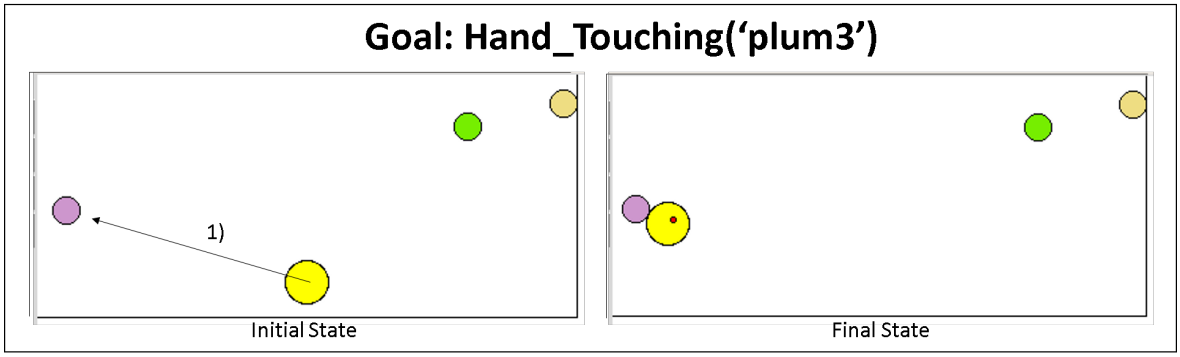


Figure 8.2: The agent planning for the goal Hand_Touching('plum3')

After a few practice attempts, the agent nearly always plans for this goal correctly. Initially it tries to use Hit actions which push the ball away. This leads to the quality scores of Action Rules with the effect Hand_Touching(?b1) quickly degrade, and the agent learns that Move_To is much more reliable for achieving the goal.

Plan 3 – Goal: Hand_Touching(?b1)

Plan 3 is a variation of Plan 2 which instead of telling the agent it must touch a specific ball, allows the agent to choose a ball to touch. Again, the agent achieved the goal from the initial state shown using a single Move_To action.

1. Move_To((0, 23), (-54, 47)) with constraints: {Hand_At((0, 23)), Place_Near((-54, 47), gray3), Place_Touching((-54, 47), gray3)}

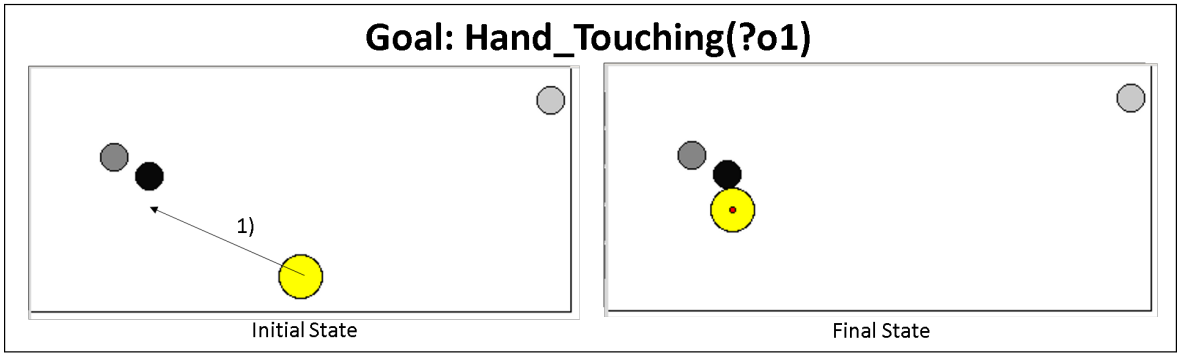


Figure 8.3: The agent planning for the goal Hand_Touching(?b1)

The agent chose gray3 to bind to ?b1.

8.2.2 Moving to a specific place

Plan 4 – Goal: Hand_At((74, 65))

The agent is able to plan to move to a specific place without any other goals.

1. Move_To((0, 23), (74, 65)) with constraints: {Hand_At((0, 23))}
2. Move_To((0, 23), (74, 65)) with constraints: {Hand_At((77, 61))}

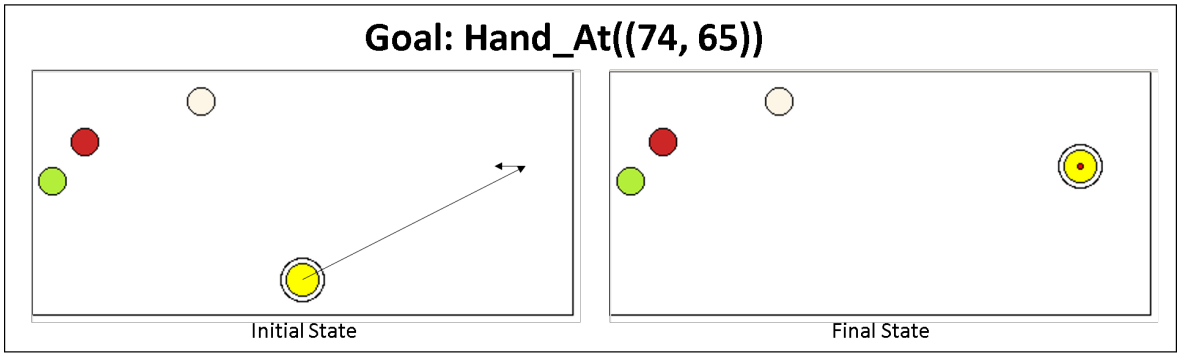


Figure 8.4: The agent planning for the goal Hand_At((74, 65))

Because the motor control policy is imperfect in some cases, the hand went slightly past the target place. When the agent checked whether or not the first Move_To achieved the goal, it recognised it had not, so planned and used another Move_To to bring the hand to the target place. This demonstrates that the agent's planner elegantly recovers from small artefacts in the motor control policy. This behaviour is not unlike humans, who when trying to move their hand to a precise point far away are likely to be near but not on the target the first attempt, but then can do a shorter move onto the exact target.

8.2.3 Grasping a ball

Plan 5 – Goal: Hand_Grasping(?b1)

The agent achieved the goal from the state shown using a Move_To action to get to the ball and then a Grasp action to grasp it.

1. Move_To((0, 23), (48, 83)) with constraints: {Place_Touching((48, 83), 'yellowgreen'), Place_Near((48, 83), 'yellowgreen'), Hand_At((0, 23))}
2. Grasp((48, 83)) with constraints: {Place_Touching((48, 83), 'yellowgreen'), Place_Near((48, 83), 'yellowgreen'), Hand_At((48, 83))}

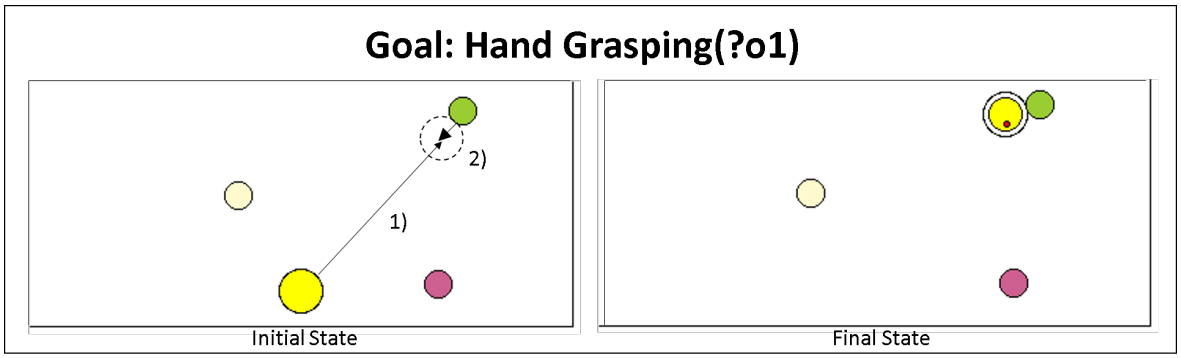


Figure 8.5: The agent planning for the goal Hand_Grasping(?b1)

Because the agent was not told which ball to grasp, it chose the 'yellowgreen' one. The agent's first attempts at grasping a ball normally involve hitting the ball in an attempt to move to it, but the agent quickly learns from experience of trying to achieve the goal that using a Move_To action is more reliable. Beyond the initial learning, the agent nearly always plans for this two-step plan efficiently and successfully.

Plan 6 – Goal: Hand_Grasping(?b1) (2)

Plan 6 is a variation of Plan 5 which starts from an initial state where the fingers are closed. The agent achieved the goal using an Ungrasp action to open its fingers and then a Move_To and Grasp actions to grasp the ball.

1. Ungrasp((0, 23)) with constraints: {Hand_At((0, 23))}
2. Move_To((0, 23), (19, 87)) with constraints: {Place_Touching((19, 87), 'darkkhaki'), Place_Near((19, 87), 'darkkhaki'), Hand_At((0, 23))}
3. Grasp((19, 87)) with constraints: {Place_Touching((19, 87), 'darkkhaki'), Place_Near((19, 87), 'darkkhaki'), Hand_At((19, 87))}

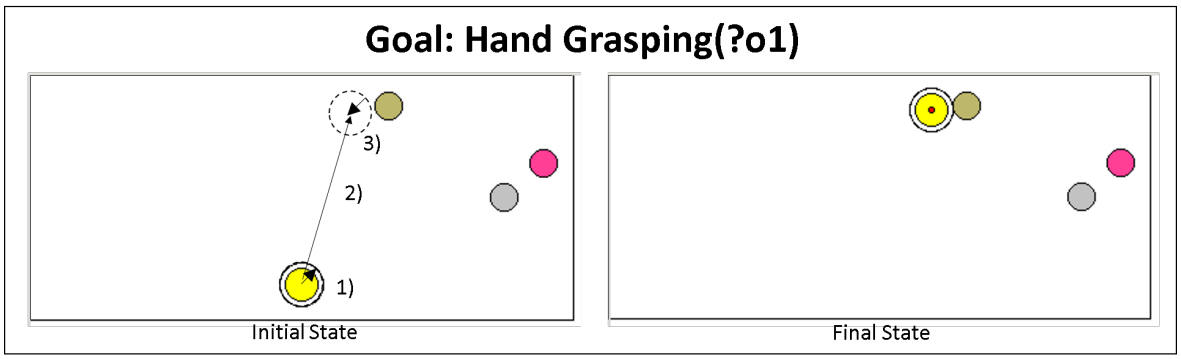


Figure 8.6: The agent planning for the goal Hand_Grasping(?b1) when its fingers are initially closed

While initially appearing that the agent deliberately open its fingers for the Grasp action, the order in which the actions were carried out suggests otherwise. The Ungrasp was actually planned to satisfy the Move_To Action Rule's preconditions (due to Fingers_Flat being an unnecessary precondition in many Action Rules). Despite this, had the Move_To action not had the unnecessary Fingers_Flat precondition, the Grasp action also had the precondition so the plan still would have suceeded, except it would have been more deliberate. Further learning about which preconditions are unnecessary would probably allow the agent to deliberately plan the Ungrasp action to allow the Grasp action to suceed.

8.2.4 Making balls touch walls

Plan 7 – Goal: Touching(?b1, ?w1)

Plan 7 is one of the more difficult goals the agent can achieve, with multiple ways of going about it. The agent approached it with a three step plan consisting of a Move_To to get to the ball it wanted to make touch the wall, a Grasp to grasp the ball, and then a Move_To to get the ball to the wall.

1. Move_To((0, 23), (-72, 21)) with constraints: {Place_Touching((-72, 21), OrangeRed4), Hand_At((0, 23)), Place_Near((-72, 21), OrangeRed4)}
2. Grasp((-72, 21)) with constraints: {Place_Near((-72, 21), OrangeRed4), Place_Touching((-72, 21), OrangeRed4), Hand_At((-72, 21))}
3. Move_To((-72, 21), (-92, 58)) with constraints: {Place_Touching((-72, 21), OrangeRed4), Hand_At((-72, 21)), Place_Touching((-92, 58), left_wall), Place_Behind((-72, 21), OrangeRed4, left_wall), Place_Near((-72, 21), OrangeRed4)}

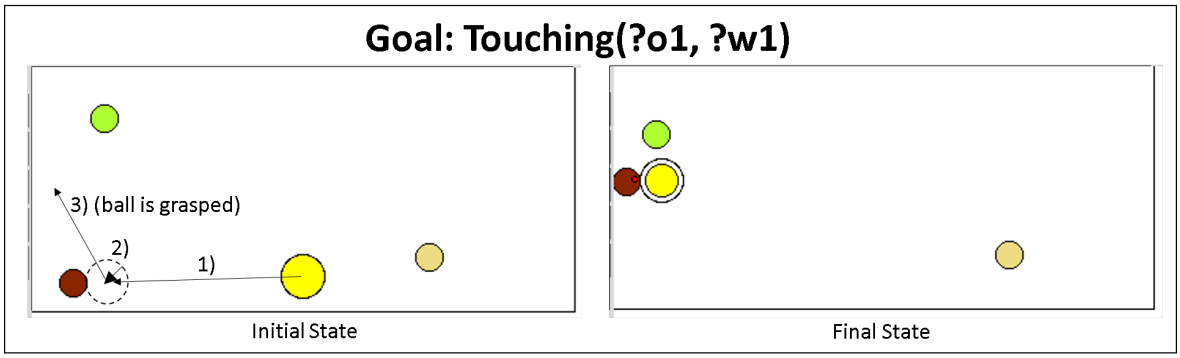


Figure 8.7: The agent planning for the goal Touching(?b1, ?w1)

Plan 7 shows the agent correctly using places and constraints in order to make a ball touch a wall. The Place_Behind constraint in step (3) ensured the ball was moved to and grasped from the side away from the left wall so that moving towards the left wall would make the ball, and not the hand, touch the left wall. The agent sometime uses Hit actions to make balls touch walls, although that is generally less reliable. The agent would not have been able to plan reliably for the goal without constraints.

Plan 8 – Goal: Touching(LightGoldenrod2, Left_Wall)

Plan 8 is a more difficult variation of Plan 7. While in Plan 7 the agent happened to have its hand on the most convienent side of the ball to begin with, in Plan 8 it does not. The agent eventually managed to get to a place on the correct side of the ball using three Move_To actions, and then a Grasp and a Move_To like in Plan 7.

1. Ungrasp((0, 23)) with constraints: {Hand_At((0, 23))}
2. Move_To((0, 23), (38, 29)) with constraints: {Place_Near((38, 29), LightGoldenrod2), Place_Behind((38, 29), LightGoldenrod2, left_wall), Place_Touching((38, 29), LightGoldenrod2), Hand_At((0, 23))}
3. Move_To((38, 29), (86, 28)) with constraints: {Place_Behind((86, 28), LightGoldenrod2, left_wall), Hand_At((38, 29))}
4. Move_To((86, 29), (61, 40)) with constraints: {Place_Near((61, 40), LightGoldenrod2), Place_Behind((61, 40), LightGoldenrod2, left_wall), Place_Touching((61, 40), LightGoldenrod2), Hand_At((86, 29))}
5. Grasp((61, 40)) with constraints: {Place_Near((61, 40), LightGoldenrod2), Place_Touching((61, 40), LightGoldenrod2), Hand_At((61, 40))}
6. Move_To((38, 29), (-92, 45)) with constraints: {Place_Touching((38, 29), LightGoldenrod2), Hand_At((38, 29)), Place_Touching((-92, 45), left_wall), Place_Behind((38, 29), LightGoldenrod2, left_wall), Place_Near((38, 29), LightGoldenrod2)}

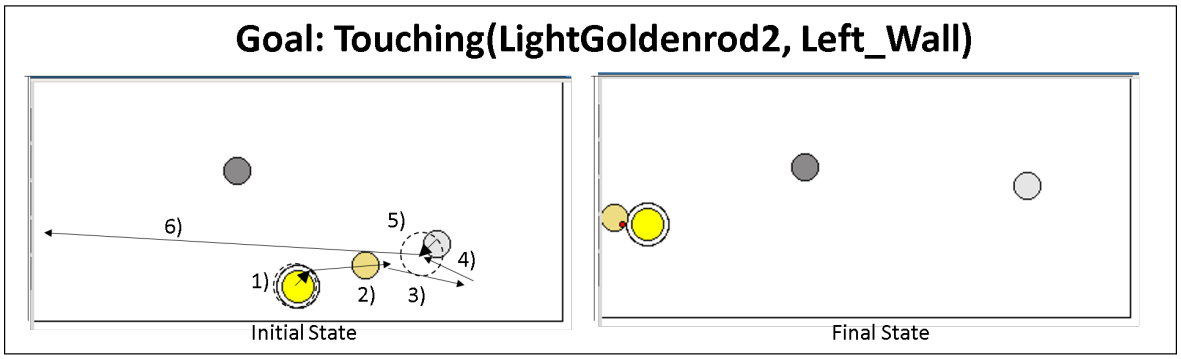


Figure 8.8: The agent planning for the goal Touching(LightGoldenrod2, Left_Wall)

Plan 8 was somewhat lucky for the agent, as sometimes when trying to get behind a ball relative ot the desired wall it accidentally knocks the ball off the table or into the opposite wall (making it impossible to get behind the ball). Nonetheless, Plan 8 further demonstrates that agent's planner is good at recovering from unexpected effects.

Plan 9 – Goal: Touching(maroon4, Near_Wall)

Plan 9 was even more difficult than Plan 8 because the target ball was so close to the far wall to begin with. This made it difficult for the agent to get its hand behind the ball, and it ended up carrying out 10 Move_To actions which initially pushed the ball against the far wall, and then demonstrated meaningless pushing of the ball back and forth along the wall and a couple of times going back to the near wall. The agent is programmed to give up after 10 actions, so did and failed the goal.

1. Move_To((0, 23), (-67, 89)) with constraints: {Place_Near((-67, 89), maroon4), Place_Behind((-67, 89), maroon4, near_wall), Place_Touching((-67, 89), maroon4), Hand_At((0, 23))}
2. Move_To((-67, 89), (-55, 82)) with constraints: {Place_Behind((-55, 82), maroon4, near_wall), Hand_At((-67, 89))}
3. Move_To((-55, 82), (-38, 90)) with constraints: {Place_Behind((-38, 90), maroon4, near_wall), Hand_At((-55, 82))}
4. Move_To((-38, 83), (39, 18)) with constraints: {Hand_At((-38, 83)), Place_Touching((39, 18), near_wall)}

5. Move_To((39, 18), (-44, 89)) with constraints: {Place_Behind((-44, 89), maroon4, near_wall), Place_Touching((39, 18), near_wall), Hand_At((39, 18))}
6. Move_To((-44, 89), (-23, 86)) with constraints: {Place_Behind((-23, 86), maroon4, near_wall), Hand_At((-44, 89))}
7. Move_To((-23, 86), (2, 89)) with constraints: {Place_Near((2, 89), maroon4), Place_Behind((2, 89), maroon4, near_wall), Place_Touching((2, 89), maroon4), Hand_At((-23, 86))}
8. Move_To((2, 89), (-7, 84)) with constraints: {Place_Near((-7, 84), maroon4), Place_Behind((-7, 84), maroon4, near_wall), Place_Touching((-7, 84), maroon4), Hand_At((2, 89))}
9. Move_To(?p25715, (-18, 82)) with constraints: {Place_Near((-18, 82), maroon4), Place_Behind((-18, 82), maroon4, near_wall), Place_Touching((-18, 82), maroon4), Hand_At(?p25715)}
10. Move_To((-67, 89), (20, 18)) with constraints: {Place_Touching((-67, 89), maroon4), Hand_At((-67, 89)), Place_Touching((20, 18), near_wall), Place_Behind((-67, 89), maroon4, near_wall), Place_Near((-67, 89), maroon4)}

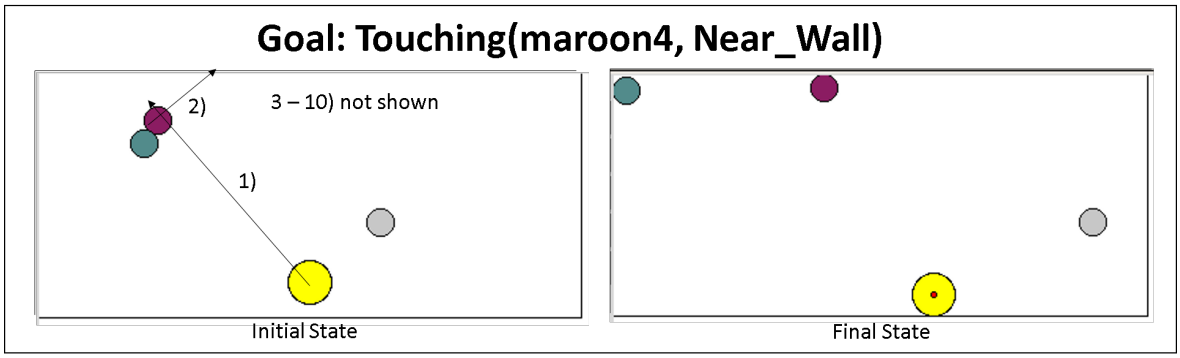


Figure 8.9: The agent planning for the goal Touching(maroon4, Near_Wall)

The agent giving up after 10 actions ensured it did not plan indefinitely. Once the ball was pushed against the far wall, the goal was nearly impossible to achieve. The agent is not able to plan to pull the ball away from the far wall and to then go behind it. In order to make such plans, the agent would need to include place variables for ball positions, and specify constraints in those positions. A special Object_At constraint much like Hand_At would also be needed so that the agent could learn Action Rules that change the place a ball is at. These constraints would be a nice extension to the agent's representation.

An alternative solution is adding constraints that make it possible for the agent to reason about not attempting to move through balls, and thus avoiding situations where it can no longer go behind the ball. For example, a clear path constraint that specifies when the path between two places need to be clear. This would allow the agent to learn that it needs to move to the place that is touching the ball from a place where there is a clear path. The agent could "chain" Move_To actions to go around balls where necessary.

8.2.5 Removing balls from the table

Plan 10 – Goal: On_Table(dodgerblue)

The agent suceeded in Plan 10, hitting the ball so that it rolled off the table.

1. Hit((0, 23), (86, 24)) with constraints: {Place_Touching((86, 24), dodgerblue), Hand_At((0, 23)), Place_Near((86, 24), drblue)}

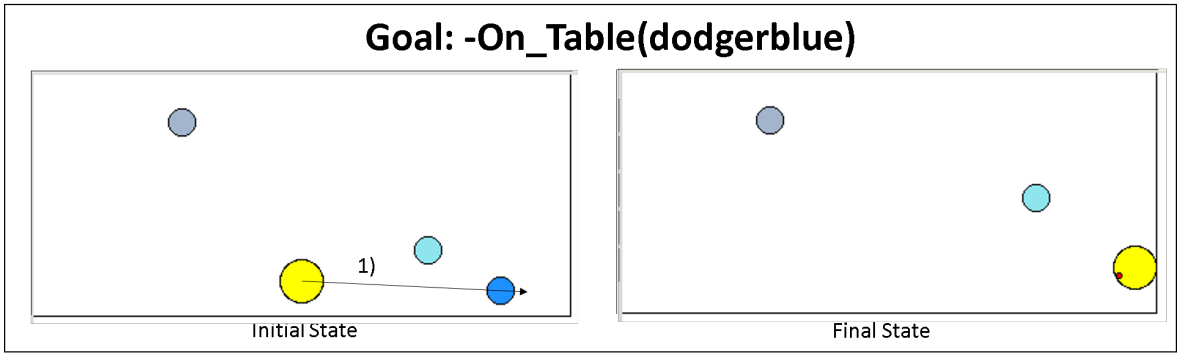


Figure 8.10: The agent planning for the goal -On_Table(dodgerblue)

The agent had learnt that hitting balls can cause them to roll from the table. The hit target selected was suitable.

Plan 11 – Goal: -On_Table(green4)

The agent failed Plan 11, carrying out 11 obviously unhelpful actions which did not achieve the goal.

1. Hit((0, 23), (0, 44)) with constraints: {Place_Touching((0, 44), green4), Hand_At((0, 23)), Place_Near((0, 44), green4)}
2. Move_To((-1, 92), (0, 83)) with constraints: {Place_Touching((0, 83), green4), Hand_At((-1, 92)), Place_Near((0, 83), green4)}
3. Grasp((0, 83)) with constraints: {Place_Near((0, 83), green4), Place_Touching((0, 83), green4), Hand_At((0, 83))}
4. Ungrasp((0, 83)) with constraints: {Place_Near((0, 83), green4), Place_Touching((0, 83), green4), Hand_At((0, 83))}
5. Move_To((0, 83), (23, 82)) with constraints: {Place_Touching((23, 82), green4), Hand_At((0, 83)), Place_Near((23, 82), green4)}
6. Hit((23, 82), (50, 50)) with constraints: {Hand_At((23, 82))}
7. Hit((78, 22), (50, 50)) with constraints: {Hand_At((78, 22))}
8. Move_To((23, 82), (50, 50)) with constraints: {Hand_At((23, 82))}
9. Hit((52, 52), (36, 87)) with constraints: {Place_Touching((36, 87), green4), Hand_At((52, 52)), Place_Near((36, 87), green4)}
10. Move_To(?p3613, (50, 50)) with constraints: {Place_Touching((50, 50), green4), Hand_At(?p3613), Place_Near((50, 50), green4)}

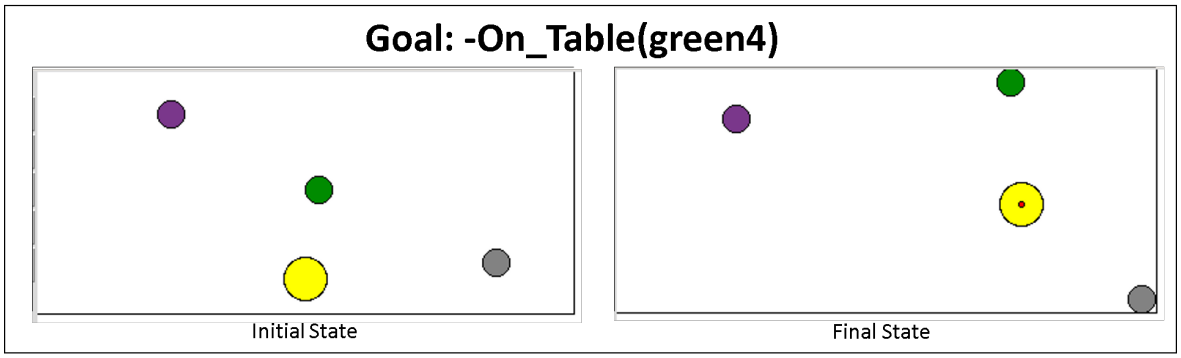


Figure 8.11: The agent planning for the goal -On_Table(green4)

The reason the agent was unsuccessful in Plan 11 is because it could not learn that it needed to go behind the ball relative to the slot wall. This limitation and possible solutions was discussed in AR10.

8.2.6 Goals involving three or more objects

Plan 12 – Goal: {**+Touching(plum4, far_wall), +Touching(IndianRed2, far_wall)**}

The agent was able to make two balls touch the far wall. It hit one towards the wall and then the other. It attempted to grasp multiple times when its fingers were already closed.

1. Move_To((0, 23), (57, 26)) with constraints: {Place_Near((57, 26), plum4), Place_Behind((57, 26), plum4, far_wall), Place_Touching((57, 26), plum4), Hand_At((0, 23))}
2. Hit((57, 21), (50, 50)) with constraints: {Hand_At((57, 21))}
3. Hit((43, 91), (50, 50)) with constraints: {Hand_At((43, 91))}
4. Grasp((57, 21)) with constraints: {Place_Near((57, 21), plum4), Hand_At((57, 21))}
5. Grasp((50, 18)) with constraints: {Place_Near((50, 18), plum4), Hand_At((50, 18))}
6. Grasp((50, 18)) with constraints: {Place_Near((50, 18), plum4), Hand_At((50, 18))}
7. Grasp((50, 18)) with constraints: {Place_Near((50, 18), plum4), Hand_At((50, 18))}
8. Grasp((50, 18)) with constraints: {Place_Near((50, 18), plum4), Hand_At((50, 18))}
9. Grasp(?p205244) with constraints: {Place_Near(?p205244, plum4), Hand_At(?p205244)}
10. Hit((57, 26), (-21, 85)) with constraints: {Place_Behind((57, 26), plum4, far_wall), Hand_At((57, 26)), Place_Near((57, 26), plum4), Place_Near((-21, 85), IndianRed2), Place_Touching((57, 26), plum4), Place_Touching((-21, 85), IndianRed2)}

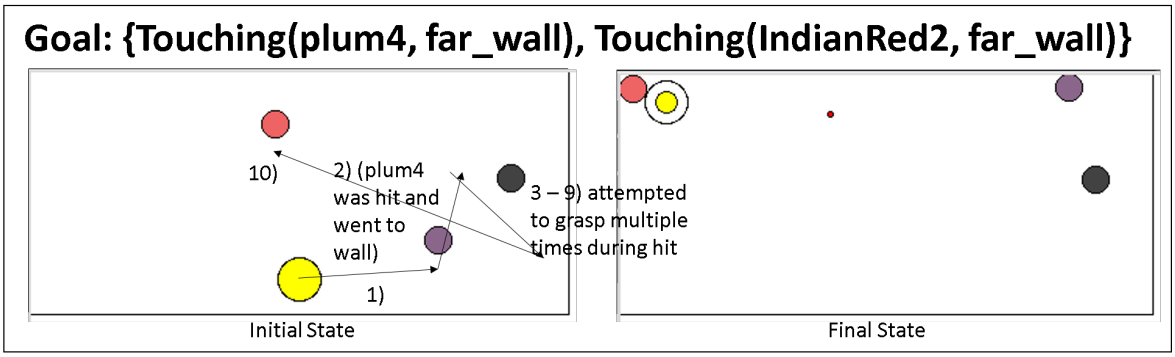


Figure 8.12: The agent planning for the goal{+Touching(plum4, far_wall), +Touching(IndianRed2, far_wall)}

The final Hit Action Rule in Plan 12 was one the agent learnt for making two balls touch the wall, and it only had 1 support and 0 quality. The figure below shows the Action Rule.

Intention:	Hit(?p11868, ?p11869)
Constraints:	Place_Behind(?p11868, ?b11872, ?w11871), Hand_At(?p11868), Place_Near(?p11868, ?b11872), Place_Near(?p11869, ?b11870), Place_Touching(?p11868, ?b11872), Place_Touching(?p11869, ?b11870)
Preconditions:	None (wasn't enough support)
Effects:	+Touching(?b11870, ?w11871), +Touching(?b11872, ?w11871)
Support: 1	
Quality: 0	

The constraints were impossible to satisfy, and it was lucky that the agent satisfied the ones concerned with IndianRed3 because plum4 was already touching the wall from an earlier action carried out. The agent did not know this when trying to satisfy the constraints because it was the first Action Rule put on the stack.The heuristics put too much weight on the number of goals satisfied and not enough on the other heuristics. An improvement would be to make the heuristics consider smaller Action Rules with a far higher quality even though they might not achieve all the goals.

The agent also did six meaningless grasp actions during the action. This was because it was trying to satisfy goals and looped. The maximum number of actions allowed in a plan ensured this loop was broken.

Overall, success for this plan depended on serendipity.

Plan 13 - {**+Touching(plum4, far_wall), +Touching(IndianRed2, far_wall)**} Unlike Plan 12, the agent did not manage to complete Plan 13 successfully. The hit actions were somewhat meaningless.

1. Hit(?p129929, (50, 50)) with constraints: {Hand_At(?p129929), Place_Near((50, 50), PaleGreen4), Place_Touching((50, 50), PaleGreen4)}
2. Hit(?p80465, (50, 50)) with constraints: {Place_Touching((50, 50), PaleGreen4), Hand_At(?p80465), Place_Behind(?p80465, PaleGreen4, far_wall), Place_Near((50, 50), PaleGreen4)}
3. Hit(?p80465, (50, 50)) with constraints: {Place_Touching((50, 50), PaleGreen4), Hand_At(?p80465), Place_Behind(?p80465, PaleGreen4, far_wall), Place_Near((50, 50), PaleGreen4), Place_Behind((50, 50), PaleGreen4, far_wall)}
4. Hit(?p80465, (50, 50)) with constraints: {Place_Touching((50, 50), PaleGreen4), Hand_At(?p80465), Place_Behind(?p80465, PaleGreen4, far_wall), Place_Near((50, 50), PaleGreen4)}
5. Hit(?p80465, (50, 50)) with constraints: {Place_Touching((50, 50), PaleGreen4), Hand_At(?p80465), Place_Behind(?p80465, PaleGreen4, far_wall), Place_Near((50, 50), PaleGreen4), Place_Behind((50, 50), PaleGreen4, far_wall)}
6. Hit(?p80465, (50, 50)) with constraints: {Place_Touching((50, 50), PaleGreen4), Hand_At(?p80465), Place_Behind(?p80465, PaleGreen4, far_wall), Place_Near((50, 50), PaleGreen4)}
7. Move_To((40, 51), (77, 22)) with constraints: {Place_Touching((40, 51), PaleGreen4), Place_Behind((77, 22), PaleGreen4, far_wall), Place_Near((40, 51), PaleGreen4), Place_Touching((40, 51), far_wall), Hand_At((40, 51))}
8. Hit((77, 22), (38, 36)) with constraints: {Place_Touching((38, 36), PaleGreen4), Hand_At((77, 22)), Place_Behind((77, 22), PaleGreen4, far_wall), Place_Near((38, 36), PaleGreen4), Place_Behind((38, 36), PaleGreen4, far_wall)}
9. Move_To((0, 23), (12, 26)) with constraints: {Place_Behind((12, 26), PaleGreen4, far_wall), Hand_At((0, 23)), Place_Near((12, 26), hotpink), Place_Touching((12, 26), hotpink)}
10. Hit((12, 26), (22, 34)) with constraints: {Place_Near((12, 26), hotpink), Place_Touching((12, 26), hotpink), Place_Behind((22, 34), PaleGreen4, far_wall), Place_Near((22, 34), PaleGreen4), Place_Touching((22, 34), PaleGreen4), Hand_At((12, 26)), Place_Behind((12, 26), PaleGreen4, far_wall)}

For difficult goals the agent tends to resort to the highly probabilistic hit actions in an attempt to achieve all the goals at the same time. While finding Action Rules that achieve all the goals is useful when the goals are currently related, it makes planning difficult when the goals are not closely related. The agent is not able to recognise when it should plan for each goal separately.

8.2.7 Coordinated effects

Plan 14 - {**Touch_Sensor(),Hand_Touching(?b1)**} The agent is able to identify Action Rules with an effect set that achieves multiple coordinated effects.

1. Move_To((0, 23), (10, 72)) with constraints: {Hand_At((0, 23)), Place_Touching((10, 72), gray70), Place_Near((10, 72), gray70)}

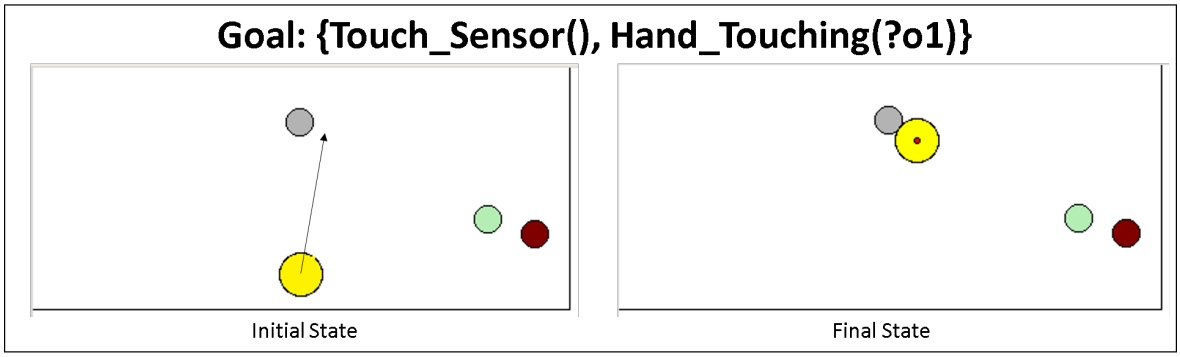


Figure 8.13: The agent planning for the goal{Touch_Sensor(),Hand_Touching(?b1)}

Where a goal has multiple parts, the agent is able to select an Action Rule to achieve all of them. This greatly simplifies the planning, particularly where there are multiple unsatisfied preconditions.

8.3 Summary

This evaluation showed that the agent is able to plan for a range of tasks, using Action Rules that were autonomously learnt from exploratory behaviour. The agent was able to plan with both one-step and multi-step plans. Constraints increased the agent's ability to achieve goals such as making an object touch the wall. This increased ability was obvious when contrasted to the agent's difficulty knocking objects off the table because it was hindered from learning effective constraints on the Action Rules for that particular task. The agent had some difficulty with the hit action because it is more probabilistic than the other actions. Overall, results were promising and areas for Future Work were highlighted.

Chapter 9

Conclusion

The thesis objective was to explore how a system could learn like a human infant in a world with realistic physics without all the limitations that were present in earlier work such as QLAP. An agent in a continuous simulated world with a simple visual system and motor control policy was implemented in Python. An Action Rule learner and Action Rule planner were then designed and implemented to enable the agent to learn meaningful rules and plan with them without requiring any teacher input. Human infants were a significant inspiration for the design of the agent. The agent was successful in that it was able to achieve a variety of goals by planning with its learnt Action Rules. For example, the agent was able to reach for and grasp objects, ungrasp objects and make objects touch walls.

The learning problem was broken into sub problems so that suitable algorithms could be selected for each part of the problem. The Action Rule learner learnt from qualitative data in order to make sense of the world.The vision system and motor control policy on the other hand would require algorithms that work with quantitative data (although learners were not constructed for the thesis). This was contrast to QLAP which attempted to learn everything qualitatively, leading to an unrealistically simple form of motor control that would not scale or work with a realistic model of an infant hand and nearly nothing in terms of a vision system which again hurt scalability.

By separating the motor control policy from the Action Rule learner and characterising actions in terms of their external behaviour (path and overall velocity profile) rather than values set to individual motor variables, the agent would be able to work with a realistic model of an infant hand. It is the motor control policys job to learn the precise details of applying forces to the motor variables in order to achieve the desired external effect. The Action Rule learner needs no knowledge of how the motor control policy actually carries out the actions as it is only concerned with what actions can be carried out and what effects they have on the world.

The Action Rule learner learnt many Action Rules, some of which were meaningful and some of which were not. Simple heuristics allowed the agent to quickly identify which Action Rules were good, and which were not. In particular, analysing how well connected the variables in the effects were to the variables in the intention via the preconditions and constraint gave the agent a good starting point. This was then further refined by using planning success history for the Action Rules.

Despite being an unusual approach, learning effect sets as the basis of Action Rules rather than single effects (as QLAP did) greatly simplified the planner. In many cases the agent was able to find a single Action Rule that would satisfy all the preconditions rather than having to look at many Action Rules and attempt to put them together.

The Action Rule planner allowed the agent to use the Action Rules learnt from exploratory behaviour to achieve deliberate goals. The planner spent little time searching

for Action Rules to use and most of its time trying them. Each time an action in the plan was carried out the agent checked the new state of the world and planned additional actions where the original action had not achieved the desired effects. In many cases, this allowed the agent to eventually achieve the goal with the plan, even if a few undesirable effects occurred along the way. This is like human infants, who experiment and retry when they fail rather than spending time thinking to try and prevent failing.

The Action Rule planner also gives opportunity for refining Action Rules and further learning. While the planner does not always succeed in planning, its success rate is sufficient because the plans that do succeed provide learning examples suitable for further learning. Furthermore, the planner can use its failures to learn more about the reliability of the Action Rules which will improve its future performance.The agent could be extended with additional learning and planning mechanisms to allow it to use the successful examples of plans in order to improve their reliability. Even examples of plans where the planning was serendipitously successful could be learnt from. This is not unlike human infants who initially have difficulty achieving their objectives but with age and experience increase their ability.

The agent learnt a special kind of precondition called constraints, particularly to handle the introduction of places. Unlike normal preconditions, constraints are considered to be part of the intention because they specify how the variables in the intention should be related to the variables in the effects. While preconditions must be planned for, constraints are handled by the agent's vision system which chooses a quantitative point to bind to each place variable in order to maximise the number of constraints satisfied. Unsatisfied constraints do not get planned for. Constraints allow the agent to specify a richer intention with place variables than it could have otherwise.

A limitation of the Action Rule learner was its precondition learning component. Because the agent only learnt from positive examples, it was unable to identify what should not be true or determine whether or not something that was often true in its learning examples actually needed to be. This issue should easily be addressable in future work by using the planner to experiment and identify unnecessary preconditions or to identify possible negative preconditions for Action Rules which repeatedly fail but otherwise did well in the quality heuristics. The connectivity heuristic in particular is probably a very strong indicator of whether or not an Action Rule is worth trying to refine and worth further investigation.

The agent also had a limited ability to reason about probabilistic effects, which greatly limited its ability to plan with Action Rules learnt for the hit action. It was possible for multiple Action Rules to have different effects but the same intention, constraints, and preconditions. Linking Action Rules with an identical intention, constraints and preconditions, much like effect sets were linked, would allow the agent to try and add additional constraints and preconditions to them that separated them out.

In some cases, preconditions containing quantitative information would be needed to separate similar Action Rules. For example, whether or not hitting an object will lead to the object hitting a wall is dependent on how much force was applied to the object and how far it was from the wall. Infants, and even some adults, struggle with reliably knowing how hard to hit an object in order for it to hit a wall. For this reason, it isnt really a limitation within the scope of this agent. If the agent was to be extended to start learning at the level of a slightly older child, the problem of using quantitative information to refine preconditions would need to be addressed.

The planner could be extended to allow it to keep a previous goal true while carrying out an additional goal. In order to achieve this, the agent would need to reason about what will be undone by an action and how to prevent it. The effect set approach combined with identifying subsumed effect sets is beneficial for this because large and complete effect sets

will contain the necessary information about side effects of the action.

In summary, the agent achieved the thesis objective because it was able to autonomously learn meaningful rules and achieve goals in a world with realistic physics. Key limitations of QLAP were avoided. The agent presented in the thesis provides a strong system to further build on and work towards the goal of building a computer program that is able to learn many of the things human infants do.

Bibliography

[1] AVRON, B., AND EDWARD, F. The handbook of artificial intelligence, 1981.

[2] BAILLARGEON, R. Representing the existence and the location of hidden objects: Object permanence in 6-and 8-month-old infants. *Cognition 23*, 1 (1986), 21–41.

[3] BAILLARGEON, R. How do infants learn about the physical world? *Current Directions in Psychological Science* (1994), 133–140.

[4] BAILLARGEON, R. The acquisition of physical knowledge in infancy: A summary in eight lessons. *Blackwell handbook of childhood cognitive development 1* (2002), 46–83.

[5] BAILLARGEON, R., NEEDHAM, A., AND DEVOS, J. The development of young infants' intuitions about support. *Early development and parenting 1*, 2 (1992), 69–78.

[6] ELSNER, B., AND ASCHERSLEBEN, G. Do i get what you get? learning about the effects of self-performed and observed actions in infancy. *Consciousness and cognition 12*, 4 (2003), 732–751.

[7] FABRICIUS, W. V. The development of forward search planning in preschoolers. *Child development* (1988), 1473–1488.

[8] FIELD, T. *Infancy*. Harvard University Press, 1990.

[9] GIBSON, E. J. Exploratory behavior in the development of perceiving, acting, and the acquiring of knowledge. *Annual review of psychology 39*, 1 (1988), 1–42.

[10] GOSWAMI, U. *Cognitive development: The learning brain*. Psychology Press Hove, UK, 2008.

[11] GUERIN, F. Learning like a baby: a survey of artificial intelligence approaches. *Knowledge Engineering Review 26*, 2 (2011), 209–236.

[12] HESPOS, S. J., AND BAILLARGEON, R. Reasoning about containment events in very young infants. *Cognition 78*, 3 (2001), 207–245.

[13] KONIDARIS, G., KUINDERSMA, S., GRUPEN, R., AND BARTO, A. Robot learning from demonstration by constructing skill trees. *The International Journal of Robotics Research* (2011), 0278364911428653.

[14] LUNGARELLA, M., METTA, G., PFEIFER, R., AND SANDINI, G. Developmental robotics: a survey. *Connection Science 15*, 4 (2003), 151–190.

[15] MUGAN, J. Autonomous qualitative learning of distinctions and actions in a developing agent. *PhD Thesis* (2011).

[16] MUGAN, J., AND KUIPERS, B. Learning distinctions and rules in a continuous world through active exploration. In *Proceedings of the Seventh International Conference on Epigenetic Robotics (EpiRob-07)* (2007), pp. 101–108.

[17] MUGAN, J., AND KUIPERS, B. Learning to predict the effects of actions: Synergy between rules and landmarks. In *Development and Learning, 2007. ICDL 2007. IEEE 6th International Conference on* (2007), IEEE, pp. 253–258.

[18] MUGAN, J., AND KUIPERS, B. Continuous-domain reinforcement learning using a learned qualitative state representation. In *22nd International Workshop on Qualitative Reasoning* (2008).

[19] MUGAN, J., AND KUIPERS, B. Towards the application of reinforcement learning to undirected developmental learning. In *Proceedings of the Eighth International Conference on Epigenetic Robotics (EpiRob-08)* (2008), pp. 85–92.

[20] MUGAN, J., AND KUIPERS, B. Autonomously learning an action hierarchy using a learned qualitative state representation. In *Proceedings of IJCAI* (2009), pp. 1175–1180.

[21] MUGAN, J., AND KUIPERS, B. A comparison of strategies for developmental action acquisition in QLAP. In *Proceedings of the Ninth International Conference on Epigenetic Robotics (EpiRob-09)* (2009), pp. 121–128.

[22] MUGAN, J., AND KUIPERS, B. Autonomous learning of high-level states and actions in continuous environments. *Autonomous Mental Development, IEEE Transactions on 4*, 1 (2012), 70–86.

[23] NEEDHAM, A., BARRETT, T., AND PETERMAN, K. A pick-me-up for infants exploratory skills: Early simulated experiences reaching for objects using sticky mittens enhances young infants object exploration skills. *Infant Behavior and Development 25*, 3 (2002), 279–295.

[24] QUINN, P. C. The categorization of above and below spatial relations by young infants. *Child Development 65*, 1 (1994), 58–69.

[25] ROSE, S. A., FELDMAN, J. F., AND JANKOWSKI, J. J. Visual short-term memory in the first year of life: Capacity and recency effects. *Developmental psychology 37*, 4 (2001), 539.

[26] SLATER, A., MORISON, V., AND ROSE, D. Perception of shape by the new-born baby. *British Journal of Developmental Psychology 1*, 2 (1983), 135–142.

[27] TAN, P.-N., STEINBACH, M., AND KUMAR, V. *Introduction to Data Mining, (First Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2005.

[28] THELEN, E., CORBETTA, D., KAMM, K., SPENCER, J. P., SCHNEIDER, K., AND ZERNICKE, R. F. The transition to reaching: Mapping intention and intrinsic dynamics. *Child development 64*, 4 (1993), 1058–1098.

[29] WILLATTS, P. Beyond the couch potato infant: How infants use their knowledge to regulate action, solve problems, and achieve goals. *Infant development: Recent advances* (1997), 109–135.

[30] WILLATTS, P. Development of means–end behavior in young infants: Pulling a support to retrieve a distant object. *Developmental Psychology 35*, 3 (1999), 651.