Planning in an Imperfect World Using Previous Experiences

Jen–Lung Chiu

A Dissertation Submitted in Partial Fulfillment

of the Requirements for the Degree of

Doctor of Philosophy

Department of Computer Science

Graduate School of Arts and Sciences

New York University

May 1995

Approved: ________________________________

Ernest S. Davis, Research Advisor

To My Parents

Acknowledgments

I would like to express my deepest gratitude to my research advisor, Professor Ernie Davis, for his guidance, advice, supports, and encouragements over these years.

My special thanks are to Professor Ralph Grishman, Professor Richard Wallace, Professor Jim Hendler, and Professor Robert Hummel for being in my Thesis Proposal Committee or Dissertation Defense Committee, as well as Professor Martin Davis for being in my Oral Exam Committee. They have helped me not only at the committee, but also throughout my graduate school study years.

My thanks should also go to all of the friends I met in New York University. They have shared my ups and downs during the past years. In particular, my special thanks are due to Dr. Pei Ouyang, Dr. Frank Chee–Da Tsai, Jyh–Jong Liu, Yaw–Tai Lee, and Shih–Chen Huang for their helpful discussions and encouragement.

I also appreciate the departmental staff in the Courant Institute. Their kind assistance helped me adjust to the new environment with ease and made the study here a smooth and pleasant experience. Special thanks are due to Anina Karman–Meade, Rosemary Amico, and Tamar Arnon. The assistantship from the Courant Institute of Mathematical Sciences is also highly appreciated.

Last but not least, All my family members deserve the greatest merits and special thanks. They have been always very supportive during my graduate study at New York University. Abundant credits should go to my wife, for her love, support, understanding and sacrifice. Together we went through our enjoyable times and tough roads. I feel happy to share everything with her and our lovely son in my entire life.

Planning in an Imperfect World Using Previous Experiences

Jen–Lung Chiu

Advisor : Professor Ernest S. Davis

Abstract

This thesis studies the problem of planning and problem solving in an unpredictable environment by adapting previous experiences. We construct a single agent planning system CADDY and operate it in a simple golf world testbed. The study of CADDY combines the studies of probabilistic, spatial, and temporal reasoning, adapting and reusing plans, and the tradeoff between gains and costs based on various considerations.

The CADDY planning system operates in an uncertain and unpredictable environment. Despite limited perception, incomplete knowledge, and imperfect motion control, CADDY achieves its goal efficiently by finding a plan that is already known to work well in a similar situation and applying repair heuristics to improve it. The capability of adapting experiences makes CADDY a planning system with learning capability.

In this thesis, we discuss the structure of the CADDY planning system and the results of experimental tests of CADDY when we applied to a simulated golf world. We compare CADDY with several other research projects on probabilistic planners and planners which utilizes experiences. We also discuss how CADDY can be characterized in terms of theoretical work on plan feasibility. Finally, we point out possible directions of system extension and generalizations of the idea learned from CADDY to other problem domains. Currently CADDY is not directly applied to real–world problems, but it shows an interesting and promising direction of study. By combining the techniques of probabilistic reasoning, planning, and learning, the performance of planning on real–world domains can be improved dramatically.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis studies the problem of planning and problem solving in an unpredictable environment by adapting previous experiences. We construct a single agent planning system CADDY and operate it in a simple golf world testbed. The structure of CADDY suggests a new direction for the development of automated planners. It combines the study of probabilistic, spatial, and temporal reasoning, of adapting and reusing previous experiences, and of the tradeoff between costs of sensing and costs of acting. The result of the project is to demonstrate that adaptive planning can be used effectively by agents with imperfect perception and control.

1.1 Motivation

Planning has always been a central topic in artificial intelligence. A large number of techniques have been introduced in progressively more ambitious systems over a long period of time. [THD90, McD91] have provided an excellent overview of work in the development of programs for generating plans. [AHT90, Dav90, Hen92, Ham94] provide a wide ranging discussion about issues related to planning.

A plan can be viewed as a program that controls the actions of an autonomous agent.

Given a problem, the planning system constructs a set of actions (or a plan) that can reach a desired goal. This plan can then be passed to an effector, such as a mobile robot, a robot arm, or a manufacturing system, to produce a desired result. Some planners form a complete plan before any action takes place; others interleave planning and execution. Interleaving planning and execution enables the planners to repair or enhance the plan if the planner detects errors caused in plan execution, if changes in the environment force corrective replanning or permit opportunistic replanning, or if the user requests changes to the plan.

The problem of planning can be divided into these major parts:

1. Representation. How to represent the plan and its component actions and how to represent the background knowledge (spatial, temporal, and so on).

2. Physical, epistemic, or other feasibility problem during the execution of plans. The includes problems about physical preconditions as well as knowledge preconditions; limitations on perceptions; and exact control of motions (can the plan agent executes primitive actions perfectly?).

3. How to construct a desired plan for the given starting point; how to search a good plan from the problem space. Sometimes we can find an adequate plan from the beginning; sometimes we can only find a partial plan due to the uncertainty of the planning environment.

   One way to solve the uncertainty problem is to interleave plan building and plan execution. The planner can generate a sequence of steps at a time, execute these steps just generated and collect information, then re–generate another sequence of steps for execution. This allows the planner to guide the planning process and make the planner less susceptible to errors caused by the changes of the environment, since the planner can generate only those steps the planner has complete knowledge at a time, instead of a partial plan with uncertainty in it. Work on interleaving planning and execution can be found in NASL [McD78], SIPE [Wil83], and PRS [PL87].

The other way to solve the uncertainty problem is plan repair or plan correction. The planner constructs and executes the plan. When an error is detected, the planner repairs the plan to overcome the error. Works on re–planning can be found in Hayes [Hay75], Daniel [Dan84], the SIPE planning system [Wil83], and the PRIAR reuse framework [Kam89, KH89].

Our study mainly focuses on the third problem, how to construct a desired plan for a given starting point; but we also discuss the other two problems briefly in our study.

The aim of CADDY is to address the following gaps in classical planning systems:

1. Classical planning systems do not consider uncertainty and unpredictability. They assume that the planning agents have complete and accurate knowledge of the environment and perform actions error–free. They thus avoid the consideration of problems such as knowledge acquisition, error adjustment, failure recovery, and so on.

2. They do not consider the reuse of experience. Classical planning systems always emphasize the construction of the desired plans from scratch, ignoring the experiences the systems may have encountered in previous planning.

3. They do not consider the quality of plans. Some plans for a given problem are more efficient than others. Classical planning systems emphasize the correctness of the constructed plans, rather than their efficiency.

Several previous studies have addressed one or another of these problems (see chapter 2 for detail), but few have considered all three.

1.2 Our Work

The CADDY planning system operates in an uncertain and unpredictable environment, and constructs plans by learning from its experiences to build better and better plans. CADDY maintains a library of plans which records the quality of each plan under various circumstances. Given a new situation, CADDY selects from the library the plan that appears most suitable, modifies it using domain–specific heuristics based on properties of the current situation, tests the resulting plan, and extends the library accordingly. As a testbed, we use a simulated world in which CADDY plays golf on a foggy day. In this world both perception (how far CADDY can see the ball and how accurately it can track its motions) and control (how accurately CADDY can hit the ball) are limited.

CADDY deals with imperfect perception and control, which are modeled probabilistically. It knows its limitation of perception, and knows that it cannot predict the exact outcome after executing an action. Given a problem, CADDY will first search, in the plan library where it stores its experiences, for a plan expected to fit the current situation best. It then uses several heuristics to adjust that plan in order to get a better one for later execution. The quality of the plan is determined using repeated simulations. If the new plan is determined to be efficient, CADDY puts this plan, labelled with its average cost in this setting into its plan library for future use; otherwise, CADDY discards the plan. Thus, CADDY is both a planning system and a learning system.

The CADDY planning system is not applied directly to real–world problems, but it shows an interesting and promising direction of study. With the ability to learn from experiences, the performance of planning systems should be improved dramatically.

1.3 Organization of this Thesis

The thesis will focus primarily on the description and discussion of the CADDY planning system.

Chapter 2 surveys the previous related literature. We point out the problems of classical planning systems, then survey research work about planning in uncertainty, planning by using previous experiences, and plan feasibility problems.

Chapter 3 gives an overview of CADDY. Here we just present the basic block architecture, without mentioning the underlying testbed. We keep the discussion abstract within this chapter so that these basic ideas can be applied to other problem domains.

Chapter 4 describes in detail the simple golf world which we use as the system testbed. We will present the spatial and temporal models we use within the testbed, the primitive actions the planning agent can execute, the representation of plans, the measurement of cost of each primitive action as well as the whole plan, and the motivation of this testbed.

Chapter 5 extends chapter 3 with detailed discussion of each module of CADDY as applied to the golf world testbed. We will present implementation details and suggestions for further improvement.

Chapter 6 discusses several issues of the CADDY planning system. We compare CADDY to other planning systems mentioned in section 2.2 and section 2.3. We also discuss the plan feasibility problem. We informally present the physical preconditions and knowledge preconditions of primitive actions as well as control–structure operators we use to construct the desired plan, then describe how the CADDY planning system conquers the plan feasibility problem.

Chapter 7 presents simulated results of the CADDY planning system. We present several simulation results regarding different separate modules as well as the whole planning system. We also have a brief discussion of these simulation results.

Finally, chapter 8 discusses some possible directions for improvement and extension of the CADDY planning system, and how to generalize CADDY to other richer problem domains.

Chapter 2

Literature Review

2.1 Problems of Classical Planning Systems

Classical planning systems (like STRIPS [FN71], NOAH [Sac75a, Sac75b, Sac77], and TWEAK [Cha87]) assume a completely controlled environment. The planning agent is assumed to have complete and error–free knowledge of its environment and perfect control of its actions. The agent knows the initial situation of the environment and the effects of each action. It can therefore predict the state of the environment after executing a planned course of actions. In particular, the agent can know ahead of time whether a specific plan will or will not achieve the goal. With these assumptions, they do not need to deal with many problems like imperfect or uncertain knowledge, limited perception, or imperfect mobile control.

Unfortunately, these assumptions are unrealistic in many real world problems. People may not know some detail of an action until they get more information (A thief cannot open a combination lock until he/she knows the combination); people have limited perception capability (We cannot see what happens in Washington D.C. if we are currently in New York City); and people cannot perform any action perfectly without any error (Michael Jordan cannot shoot the basket 1000 times without a miss). Therefore these planning systems can only be applied to simplified and limited domains.

Another limitation of most classical planning systems is that they emphasize constructing plans from scratch, rather than on reusing, learning, or adapting known experiences. That is, these planning systems have no knowledge about plans they have constructed before. They will always perform the same search task to find the same plan again and again, in case they are asked to solve the same problem. Many resources and much time are wasted in this way.

2.2 Previous Work on Uncertainty in Planning

Some work related to uncertainty in planning has been done in recent years. Some testbeds (like the Tileworld, the Truckworld, and Phoenix discussed in [HPC93]) also use probabilistic models to govern the occurrence of events.

2.2.1 Probabilistic Temporal Reasoning

Dean and Kanazawa [DK87, DK88] first introduced probabilistic temporal reasoning, which uses easily obtainable statistical data to provide expectations concerning how long propositions are likely to persist in the absence of specific knowledge to the contrary, and described an approach to compute probabilistic assessments.

The commonsense law of inertia [McC80, McC86] states that in order to infer that a proposition ceases to be true, an event with contravening effects must be predicted to occur. Such predictions are often difficult to make due to the lack of perfect knowledge. Probabilistic temporal reasoning uses past experiences to provide an estimate of how long certain propositions are likely to persist. The persistence rule states that the probability of " proposition P holds at time t" is influenced by the following conditional probabilities and the probabilities of the conditions:

Natural Attrition represents the conditional probability that, given no direct evidence of events to affect P and that P is true in a prior situation, P tends to remain true. The law of inertia claims that this conditional probability ought generally to be 1, but in practice, this conditional probability decreases as time passing by, and the decreasing rate depends on the persistence nature of P.

Causal Accretion represents the conditional probability that, given the occurrence of events that make P true, P becomes true. This conditional probability is generally 1.

Negative Causal Accretion represents the conditional probability that, given the occurrence of events that make P false, P becomes true. This conditional probability is generally 0.

Spontaneous Causation represents the conditional probability that, given no direct evidence of events known to affect P and P is false in a prior situation, P suddenly becomes true. In view of the law of inertia, this conditional probability is generally 0.

By giving the probabilistic distribution of the initial situation, the probability of events known to affect proposition P, and above conditional probabilities, it is easy to determine the probability of P being true by simple probability calculation.

By expressing knowledge of causal rules in terms of conditional probabilities, probabilistic temporal reasoning can make appropriate judgments concerning the persistence of propositions and the probabilistic projection problem [DK87], which is a probabilistic version of the projection problem [DM87], which involves computing the consequences of a set of conditions (observations) given a set of cause–and–effect relations (causal rules). Further work can be found in Kanazawa & Dean [KD89] and Dean *et. al* [DFM89].

2.2.2 BURIDAN: Probabilistic Planning

BURIDAN [KHW93, KHW94] is an implemented partial–order planer that uses probabilities to represent uncertainty in the world state and in the effects of its operators. Given a probabilistic distribution over possible initial world states, a boolean combination of propositions representing the goal, conditional probability distributions over operator outcomes, and a probability threshold τ,BURIDAN produces a plan that achieves the goal with probability at least τ.

BURIDAN extends the STRIPS representation to allow conditional and probabilistic effects. The initial state is represented as a probability distribution over possible world states. The goal state is represented as a boolean combination of propositions which must holds at the end. The action representation is adapted from Hanks' work on probabilistic projection [Han90a, Han90b]. An action's effects are modeled by partitioning world states into mutually exclusive and exhaustive equivalence classes and mapping each class into a particular outcome. This action model assumes that every action can be executed in any world state, but the effect of executing the action depends on the state at the execution time as well as on random chance.

Refining a plan with conditional and probabilistic operators differs from refining a plan in classical planning systems in two important ways.

- First, classical plan refinement establishes a single link between a producing step (a step which sets propositions' truth values) and a consuming step (a step which can be feasible only if the propositions hold), and the link ensures that the supported propositions will be true. Causal support in a probabilistic plan is a cumulative concept: multiple links may be necessary to support a proposition since no single link makes the proposition sufficiently likely to be true, but the disjunction of these links does. The more links supporting a proposition, the more likely it is that the proposition will be true.

- Second, classical plan refinement resolves threats (a step which voids the effect of producing step so that the consuming step will not be feasible if the threat occurs between them) by ordering the threatening step either before the producing step (demotion) or after the consuming step (promotion). Probabilistic plan refinement can also resolve the threats without changing the structure of the proposed plan, as long as the threat is sufficiently unlikely to occur (confrontation).

BURIDAN searches through a space of partial plans. Each plan consists of a set of action instances $\{a_i\}$, a partial temporal ordering relation $<$ over $\{a_i\}$, and a set of causal links. A causal link has form $i/\alpha\frac{Q}{S}j$, meaning that proposition Q is a trigger condition for action a_j's outcome, outcome α of action a_i contains Q and that BURIDAN is counting on a_i/α to make Q true and remain true until a_j is executed. S stores information about which threats are resolved by confrontation. A step a_k is said to threaten link $i/\alpha\frac{Q}{S}j$ if some outcome of a_k asserts $\bar{Q}$ and a_k can possibly occur between a_i and a_j. The probability that link $i/\alpha\frac{Q}{S}j$ succeeds in producing Q for a_j is the probability that executing a_i results in outcome α and that each step representing a confronted threat results in an outcome that does not make Q false. BURIDAN confronts a threat by modifying the threatening step so that each outcome which does not change Q to false produce a safety condition S_i and includes S_i into set S. The safety condition is then adopted as an additional goal. Since the safety condition is only produced by the threatening step's outcome which does not change Q to false, planning for the safety condition amounts to planning to force the threatening step to have the outcome which does not change Q to false.

BURIDAN begins from the initial plan which contains

- the initial step a_0, which has several outcomes. Each outcome describes a possible initial world. The corresponding trigger records that world's probability;

- the goal step a_G, which has a single outcome that produce the success of the plan triggered by the goal conditions; and
- the ordering constraint $a_0 < a_G$.

BURIDAN sequentially executes the following two steps to construct the plan:

Plan Assessment Determine if the probability of the current plan achieving the goal exceeds τ. Terminate execution and return the resulting plan if so.

Plan Generation Otherwise, try to increase the probability of goal satisfaction by refining the current plan. A plan can be refined by either resolving a thread to a causal link (by demotion, promotion, or confrontation), or adding a link to increase the probability that a desired trigger condition will be true. Each refinement generates a new partial plan. If there are no possible refinements, signal failure; otherwise choose a new partial plan nondeterministically and loop again.

C–BURIDAN [DHW94] is an extension of BURIDAN which both include actions of imperfect sensing and combine it with a framework for contingent actions. Further works are also presented in Bagchi *et. al* [BBK94], Barret & Weld [BW94b, BW94a], and Williamson & Hanks [WH94].

2.2.3 Markov Decision Processes

Domains in which actions have probabilistic results and the planning agent has direct access to the state of the environment can be formalized as Markov Decision Processes (**MDP**s) [How60]. An MDP is defined as a tuple (S, A, T, R) where S is a set of possible world states, A is a set of actions, T is a set of state transition functions which map elements of $S \times A$ to discrete probability distributions over S, and R is a set of reward functions which map $S \times A$ to real numbers that specify the instantaneously reward of taking the action at the state. A policy π is a mapping from S to A specifying an action to be taken at each state. MDPs models have played an important role in planning research (Dean *et. al* [DKKN93]), but the assumption of complete observability limits their applicability to real–world problems.

An extension of MDPs models to Partially Observable Markov Decision Processes (**POMDP**s) provides an elegant solution to the problem of acting in partially observable domains. POMDPs extend MDPs by including a set B of possible observations and a set O of observation functions which map $A \times S$ to discrete probability distributions over B. A belief state is a discrete probability distribution over the set of world states S. A controller of POMDPs contains two components: a state estimator and a policy. The policy maps belief states into actions; the state estimator takes last belief state, the most recent action, and the most recent observation as inputs and produces an updated belief state (a vector of probabilities over world states). The policy can be constructed from π and the state estimator can be constructed from T and O by straightforward application of Bayes' rule. Monahan [Mon82] and Lovejoy [Lov91] give extensive surveys of the operations research literature on POMDPs. Chrisman [Chr92] and McCallum [McC93] describe algorithms for including a POMDPs from interactions with the environment and use relatively simple approximations to the resulting optimal value function. Cassandra *et. al* [CKL94a, CKL94b] present an algorithm for finding arbitrarily good approximations to optimal policies and a method for the compact representation of many such policies.

MDPs model provides a framework of gain/cost tradeoff, while POMDPs model adds the consideration of incomplete knowledge.

2.3 Previous Work on Plan Adaptation

Employing heuristic rules to generate initial plans and then debugging if the generated plans is incorrect has proven to be a useful problem solving and planning strategy. The effectiveness of this strategy depends on the assumptions that the heuristics can be used to generate efficiently plans that are correct or near–correct so that debugging plans, while not necessary efficient, is robust enough to solve the problems which are incorrectly handled by the heuristics of generating plans. Memory management, plan indexing, plan modification, and learning are central issues in planning from experiences.

Due to the gains being made in machine learning and the new work on case–based reasoning, designing planning systems which learn from experience has emerged as an important new research direction. Some planning systems have used past experiences in developing plans for new problems. Here we discuss several famous ones.

2.3.1 HACKER

HACKER [Sus75] is a blocks–world planning system with the capability of failure–driven learning. It solves problems by looking up plan schemas in its Plan Library, and fitting them together. Given a problem statement, HACKER constructs a plan by either looking up in Plan Library whether the input problem instance is an example of a generalized problem and there exists a generalized plan schema to solve it, or building up a new plan by knowledge from its Programming Techniques Library and Blocks World Knowledge Library. If the execution of the constructed plan is halted by bugs, HACKER "criticizes" and corrects the plan for re–execution. Critics can also be learned by summarizing the experiences of bug analyzers so that HACKER will avoid the same bug the next time. HACKER has never been fully implemented yet.

2.3.2 Case–Based Planning and Memory–Based Planning

Case–based planning is an extension of Sussman's ideas of retrieval and repair and an application of case–based reasoning to planning. This approach is also characterized as memory–based because the organization of the memory of previous plans is changed during the planning process.

CHEF [Ham86a, Ham86b, Ham89] (and WOK [Ham83]) is a case–based planner that builds new plans out of its memory of old ones. CHEF's domain is Chinese Szechwan cooking and its task is to build new recipes on the basis of users' requests. CHEF's input is a set of goals for different tastes, textures, ingredients, and type of dishes and its output is a single plan (recipe) that satisfies all of the goals. CHEF consists of six processes:

Problem Anticipation

The problem anticipator notices features in the current input which have previously participated in past planning problems. The problem anticipator tries to predict any problems that might occur as a result of planning of those input features. If the features that are related to past problems are presented, the memory of the failure is activated and the planner is informed of the possibility of the failure occurring. A goal to avoid these problems is also added to the set of goals to be planned for.

Plan Retrieval

The plan retriever searches through the plan library for a plan that satisfies as many of its current goals as possible while avoiding the problems that have been predicted by the problem anticipator. The plan retriever uses three kinds of knowledge in finding the best match for a set of goals: a plan memory that is indexed by the goals the plans satisfy and the problems they avoid; a similarity metric that allows the plan retriever to notice partial matches between goals; and a value hierarchy that allows it to judge the relative importance between goals.

Plan Modification

The plan modifier alters the plan found by the plan retriever so as to satisfy goals that are not yet achieved. For each goal that is not yet satisfied, the plan modifier looks for the modification rules associated with the goal and the general plan type. If no rule exists for a particular goal, the plan modifier steps up in an abstraction hierarchy and finds rules for a more general version of the goal. Once rules are found, actions described in the rules are merged with existing steps into the plan.

If the goal is partially satisfied by the plan, the plan modifier does not have to go to the modification rules. It can replace the old items by the new ones, remove steps that were added by old items' critics and add new steps required by the new items' critics.

Plan Repair

If the plan fails after simulation, the plan repairer fixes the faulty plan by building up a causal explanation of why the failure occurred and using the explanation to find different strategies for repairing it. The causal explanation is built by back chaining from the failure to the initial steps that causes the failure using a set of causal rules. The explanation is then used to find a structure that organizes a set of strategies for solving the problem described by the causal explanation. These structures are indexed by the description of a particular type of planning problem.

Credit Assignment

While repairing a faulty plan, CHEF also attempts to repair the characterization of the world that allowed CHEF to create the faulty plan in the first place. CHEF uses the causal explanation of why the failure occurred to identify the features in the input that led to the problem so that the failure can be predicted by plan anticipator later.

Plan Storage

Plans are placed in the plan library indexed by the goals satisfied by the plans and the problems avoided by them.

Plan retrieval, modification, and storage are essential to the basic planning loop that allows old plans to be modified in service of new goals. Plan repair, credit assignment are used when constructed plans fail to satisfy the given goals. Plan anticipator uses knowledge collected by credit assignment to predict possible problems during plan construction.

Hammond [Ham86a] demonstrated how CHEF works. In planning for the goals to include SNOW PEAS and CHICKEN in a STIR FRY dish,

1. The plan anticipator is reminded of a past failure CHEF encountered in building the BEEF AND BROCCOLI plan. CHEF had tried to stir fry BEEF and BROCCOLI together, which allowing the liquid from the BEEF to make the BROCCOLI soggy. CHEF adds a new goal to avoid this problem.

2. The plan retriever uses the descriptions of the food (CHICKEN, SNOW PEAS), tastes (hot, spicy, savory, fresh, etc), textures (crunchy, chewy, firm, etc), and types of dishes (STIR–FRY, pasta, etc) as indices to retrieve the plan from plan library. It chooses BEEF–BROCCOLI, STIR–FRY recipe as the base plan.

3. The plan modifier alters the plan by substituting CHICKEN for BEEF and substi-

tuting SNOW PEAS for BROCCOLI, then adding a step of boning the CHICKEN before chopping by the critic under the concept CHICKEN.

4. CHEF does not run into any failure for this STIR–FRY dish with CHICKEN and SNOW PEAS, so there is no need to run the plan repairer and credit assignment.

5. The newly generated recipe is stored into plan library.

Other Case–Based Planners

Kambhampati's work on the PRIAR reuse framework [Kam89, KH89, KH92] applies a case–based approach in a classical planning framework. PRIAR is an extension of NONLIN [Tat77a, Tat77b] which allows the planning systems to annotate plans being created with information about the dependency structure between operators in the completed plan. This information can also be used to guide retrieval, reuse, and re–planning.

R. Oehlmann, D. Sleeman, and P. Edwards [OSE92, OSE93] suggest that in addition to the standard features of plan modification, execution, and repair, a planning system should not only be able to transform plans based on knowledge about actions, but also be able to learn the transformation strategies in order to apply to other plans in future use. Their approach is based on the principle that learning a more appropriate plan can be supported by the generation of appropriate self–questions and answers. If an answer cannot be found, a reasoning component plans and executes experiments in an attempt to acquire the missing knowledge. They implement their approach of plan transformation in an exploratory discovery system, IULIAN (Interrogative Understanding and Learning In AberdeeN), and test the system in various problem domains.

2.3.3 Adaptive Planning

Adaptive planning [Alt86a, Alt86b, Alt88] is an approach to the flexible utilization of the old plans. It takes advantage of the details associated with specific plans while still maintaining the flexibility of a planner that works from general plans to specific ones. The adaptive planning techniques are thus sufficiently robust to handle a wide range of relationships between an old specific plan and the current circumstances.

Adaptive planners use a process of *situation matching* to retrieve a specific plan from *background knowledge*, which includes general plans, categorization knowledge, and causal knowledge. It uses the position of the old plan in a planning network as a starting point for finding a match to the planning system's current circumstances. The interaction of the planning knowledge and current situation determine a plan which fits to the current context and realizes the goal. If current circumstances indicate goals not accounted for by the old plan, one of several different types of *situation differences* is encountered: failing preconditions, step–out–of–order, differing goals, or failing outcome. Associated with each type of situation differences are strategies which can be used to fix the problems. Adaptive planners consider a failing step of a plan as a *representative of a category* of action to be accomplished. They use category knowledge and abstraction/specialization technique based on an *is–a* or *step/sub–step* hierarchy to find an alternate version of a failed step to repair it. Alterman built an adaptive planner PLEXUS and demonstrated how PLEXUS successfully adjusts an old plan of riding BART (Bay Area Rapid Transit, a subway system used at San Francisco, California) to a new plan of riding the New York City subway.

Consider the subway riding example. There are several differences between the plan of riding BART and the new plan of riding the NYC subway. In the following, differences are pointed out and solutions are also presented.

Ticket Purchase

In BART, the ticket can only be bought from the ticket machine; while in the NYC subway (as of 1986), the subway token can only be bought from a teller. This is a case of *failing preconditions*: no ticket machine is encountered. PLEXUS abstracts out the failing condition "there exists a ticket machine" to the general plan of "buying a ticket". It then adapts the similar sub–category "buying a theater ticket from the theater teller" by changing "ticket" to "token" to produce the correct step "buying a token from the token teller".

Entrance

A BART rider inserts the ticket into the entrance machine, gets the returned ticket, then pushes through the turnstile; while a NYC subway rider inserts the token then pushes through the turnstile, but the token is not returned. This is a case of *step–out–of–order*, the second step of entering BART (get back the ticket) is unnecessary for NYC subway riding.

Exit

In BART, the ticket is needed to exit and is returned after exiting; while in the NYC subway, the token is not needed to exit. This is a case of *failing outcome*; the outcome of the step " ticket returned" in entering BART is that the rider "has the ticket", and the reason for "ticket returned" is that it can be used when exiting the station. PLEXUS applies one of the strategies associated with failing outcome and finds an alternate interpretation of the situation where the outcome of "ticket returned" is no longer necessary. PLEXUS re–interprets exiting BART in such a manner that it can exit without the ticket, and decide to delete the "get returned ticket" step in the entering sub–plan. Now PLEXUS encounters *failing precondition*, no ticket to insert when exiting, and abstracts to "exit institution" then specializes to similar sub–category “exit turnstile door”.

2.3.4 GTD Paradigm

The GTD paradigm (Generate, Test, and Debug problem solving paradigm) [Sim88a, Sim88b] uses a generator to construct an initial hypothesis by matching a library of heuristic rules against the initial and goal circumstances and by composing the partial sets of actions suggested by each rule. The tester tests if the generated hypothesis satisfies the goals. If the hypothesis is tested successfully, the hypothesis is accepted as a solution plan. Otherwise, the debugger uses causal dependency structures and several domain–independent debugging strategies to locate and repair the faulty assumptions. The repaired hypothesis is then tested again. This debug/test loop continues until the test succeeds or the generator is invoked to produce a new hypothesis because the debugger appears to be moving far from the solution.

Simmons [Sim88b] proposed a theory of debugging by using four general reasoning techniques: 1) bugs are located by tracing through causal dependency structures; 2) assumption changes are indicated by regressing values back through the dependency; 3) bugs are repaired by domain–independent repair strategies that repair faulty assumptions; and 4) the goodness of proposed repairs is estimated by determining the effect on the overall problem. Simmons claimed that a small set of assumptions and associated repair strategies are needed to handle a wide range of bugs over a large class of problem domains. This assumption–oriented debugging approach stands in contrast to other approaches in which repair heuristics are associated either with bug manifestations (adaptive planning) or with certain stereotypical patterns of causal explanations (HACKER and case–based planning).

Simmons also built GORDIUS, an implementation of GTD paradigm, and used to solve problems in several domains, including geologic interpretation, block worlds planning, and the Tower of Hanoi problem.

2.4 Previous Work on Plan Feasibility

As mentioned in Chapter 1, one central concept in the theory of planning is the feasibility of a plan being executed. The conditions under which a plan is feasible to be carried out are the *preconditions* of the plan. Preconditions can be divided into two categories:

- *Physical preconditions* deal with what should be true in the physical world for the planning agent to execute the desired plan. If the physical preconditions of a plan is not satisfied at a situation, the plan cannot be executed.

- *Knowledge preconditions* deal with what the agent has to know in order to execute the plan. Unlike physical preconditions, a plan may still occur even if its knowledge preconditions is not satisfied. Consider that an agent is about to execute the plan "visit the biggest city of the United States". The agent can successfully execute the plan by going to New York City even if the agent does not know that New York City is the biggest city of the United States. In this case, this plan is not executed deliberately.

The branching features of the situation calculus [MH69] or McDermott's temporal logic [McD82, McD85] make them suitable to express the notion of the physical preconditions. In McDermott's formalism, for example, a sequence of events "can" occur in situation S if it occurs in some branch of the time structure starting in S.

Most AI planning systems consider only physical preconditions, not knowledge preconditions. In STRIPS [FN71] and TWEAK [Cha87], each primitive action expresses explicitly the physical preconditions for the action to be executed. They are thus effectively limited to problems in which the planner has all the relevant knowledge from the start.

Reasoning about the satisfiability of the knowledge preconditions of plans requires a theory that integrates temporal reasoning and reasoning about knowledge. In order to determine whether an agent knows enough to execute a plan, we must be able to characterize what he knows at the beginning of the plan execution, and how the state of the world and knowledge of the agent changes as the result of the plan execution.

McCarthy and Hayes [MH69] were the first to address the Knowledge Preconditions Problem. They suggests writing down explicit knowledge precondition axioms for each action so that the planning system can reason that the planning agent knows how to perform an action if the relevant knowledge precondition axioms are true. This approach has a major drawback that it leads to an explosion of knowledge precondition axioms for every action encountered.

The first extensive discussion of knowledge preconditions for actions was that of Moore [Moo80, Moo85]. Moore presented a theory integrating knowledge and actions. He used two languages:

- A model language contains the following two operators:
  - An epistemic operator $KNOW(A, P)$, "Agent A knows the proposition P".
  - A dynamic operator $RES(ACT, P)$, "Proposition P will be true as the result of performing action ACT".

  The operator $KNOW$ is referentially opaque; that is, the substitution of arguments by their equal terms may change the truth value.

- The language of possible worlds uses situations, fluents, and actions precisely as in the situation calculus, but adds two new relations between situations:
  - $K(A, W_1, W_2)$ means "Possible world W_2 is knowledge accessible from W_1 relative to agent A".

  - $R(ACT, W_1, W_2)$ means "Possible world W_2 will be the result of performing action ACT in W_1".

The two languages are related by axioms for translating one to the other.

Using this representation, Moore was able to define the notion of "can". The operator $CAN(A, ACT, P)$, meaning that an agent A could achieve property P by performing action ACT, is defined formally as

$$(\forall a)((\exists x)KNOW(a, ((x = ACT) \wedge RES(DO(a, ACT), P))) \supset CAN(a, ACT, P))$$

The axiom states that an agent A can achieve P by performing action ACT if he knows what ACT is and that P would be true as the result of performing ACT. An agent knows what action ACT is if he knows the general procedures of the action ACT and what objects the procedures are to be applied to. For example, an agent can perform the action $UNLOCK(SAFE_1, COMBINATION(SAFE_1))$ if he knows

1. The general procedure of the action $UNLOCK$. That is, to dial the combination of the safe to unlock it.

2. Which particular safe $SAFE_1$ represents.

3. The combination of $SAFE_1$.

In this case, the term representing the general procedure of ACT must be a rigid designator; that is, it must have the same denotation in every possible world.

A modification and extension of Moore's theory was proposed by Morgenstern [Mor87, Mor88]. Morgenstern began by identifying two problems that a theory of flexible planning would encounter:

- The Knowledge Preconditions Problem for Actions. The theory must explain under what circumstances an agent knows how to perform an action.

- The Knowledge Preconditions Problem for Plans. The theory must explain how an agent can carry a plan that includes steps he does not currently know how to perform. Actually, Moore also addressed this problem within his formalism.

Morgenstern built her theory on a first order logic in which knowledge was represented as a relation on strings. Strings are taken by predicates like $HOLD$ and $KNOW$ as arguments to represent a propositional sentence. For example,

$$KNOW(John, \text{``}At(Joe, NYC, S_3)\text{''}, S_5)$$

means that at situation S_5, John knew that Joe was in NYC at situation S_3. The use of strings as parameters of predicates makes these predicates *referentially opaque.* For example, consider the predicate

$$know\_how\_to\_perform(agent, action\_string, situation)$$

Assuming that $favorite\_block(Mary) = BLOCK_1$, the proposition

$$know\_how\_to\_perform(John, \text{``}put\_on(favorite\_block(Mary), BLOCK_2)\text{''}, S_1)$$

is not equivalent to the proposition

$$know\_how\_to\_perform(John, \text{``}put\_on(BLOCK_1, BLOCK_2)\text{''}, S_1)$$

unless we explicitly express that John knows $favorite\_block(Mary) = BLOCK_1$ at situation S_1.

Morgenstern's theory solved the knowledge preconditions problem for actions by positing that an agent knows enough to do an action if he knows a standard identifier for the action. For example, an agent A knows how to perform the following action

$$put\_on(favorite\_block(Mary), favorite\_block(Jane))$$

if he knows the basic procedures for the action *put_on* as well as the standard identifiers for these two blocks.

The theory solved the knowledge preconditions problem for plans by positing that an agent knows enough to perform a plan if he knows initially that at each step he will know how to proceed with the plan. The general rule was axiomatized by giving specific recursive conditions for knowing enough to carry out various control structure operators such as sequences, conditionals, loops, and concurrency.

Davis [Davng] proposes a simple formalism to solve the knowledge preconditions problem for plans. The formalism uses the situation calculus with branching time feature as the temporal model. The basic idea behind the formalism is that a plan P is executable for a planning system at situation T if and only if

- P will eventually terminate when starts execution at T.
- After P begins execution starting at T,

  (1) The planning agent will know whether P has successfully finished;

  (2) The planning agent will know of every action whether or not it is a next step of P; and

  (3) All the next step actions of P are feasible.

In [Dav94], Davis extends the above formalism to a continuous model of time.

Chapter 3

An Overview of the CADDY Planning System

In this chapter, we describe the abstract structure of the CADDY planning system in domain–independent terms. The testbed and detailed implementation will be discussed in the following chapters. Our purpose in doing this is that the same abstract structure can be applied to other problem domains later. This will be important in the discussion of generalization in section 8.2.

As discussed in chapter 1, the central issues addressed in CADDY are limited perception and control, plan reuse, and plan quality. The architecture of the CADDY planning system reflects this focus. In order to reuse stored plans in a new situation, CADDY must have the following capabilities:

1. A measurement method to determine the quality (or cost) of a plan at one specific situation. With this cost value, CADDY can determine whether a plan is better than another one for the current situation

2. A library of plans, together with associated information used to estimate the cost of a

plan in different situations. This library serves as the previous experiences CADDY knows.

3. Knowledge of several heuristic rules, either domain independent or domain dependent, which CADDY can use to adjust the selected plan to get a new plan with expected better quality.

Figure 3.1 shows the basic block architecture of the CADDY planning system. The function of each block is described as follows:

Plan Library (PL)

The Plan Library includes a set of plans, each of which can be chosen as a base plan for future adjustment, and a plan table which records the cost of each plan at each template starting situation.

Estimation Module (EM)

Given a plan and a starting situation, the Estimation Module estimates the cost of the selected plan under the input starting situation using the recorded costs of that plan in related circumstances as recorded in the Plan Library.

Plan Selection Module (PSM)

Given a starting situation, the Plan Selection Module will choose from the Plan Library a plan which expected to be best for the starting situation. the Plan Selection Module will use the Estimation Module to estimate the cost of each plan stored in the Plan Library and select the one with the lowest cost.

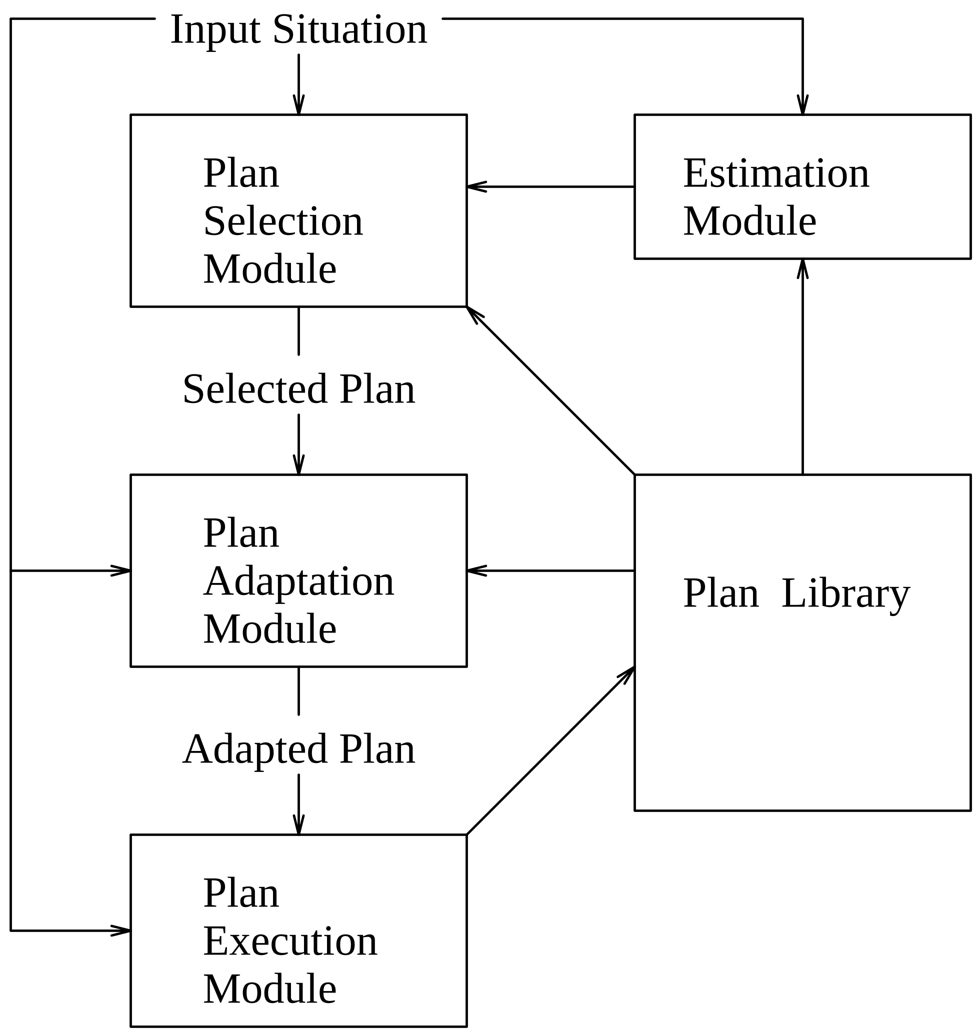

Figure 3.1: Basic Architecture of the CADDY Planning System

Plan Adaptation Module (PAM)

Given the plan selected by the Plan Selection Module and a starting situation, the Plan Adaptation Module uses heuristics to adjust the input plan with small changes in order to form a new plan with expected better quality.

Plan Execution Module (PEM)

Given the plan adjusted by the Plan Adaptation Module and a starting situation, the Plan Execution Module will simulate and monitor the execution of that selected plan until it reaches the goal. The Plan Execution Module executes numbers of simulations to get the average cost of the adapted plan. If the simulation cost of the adapted plan is better than the estimated cost of the original selected plan, the Plan Execution Module outputs the adapted plan as well as the simulation result to the Plan Library.

Chapter 4

The Testbed: A Simple Golf World

This chapter describes the underlying testbed we are using. A detailed description of the testbed (the spatial model, the temporal model, the primitive actions CADDY can execute, and so on), the representation of plans, and definition of quality of plans are given. The motivation of the choice of the golf world as the testbed is also discussed.

4.1 Description of the Testbed

Our testbed is a simple golf world: CADDY plays golf on a foggy day. Beginning a distance D from the hole, CADDY's goal is to hit the ball into the hole, which is defined as bringing the ball to within distance ϵ to the hole.

We can see that CADDY works on a fixed goal: getting the golf ball into the hole. Therefore, the goal of CADDY is not an input to the system. In this respect, the CADDY planning system has elements in common with Schoppers' Universal Plans [Sch87].

Spatial Model

We use a two dimensional X–Y coordinate system as our spatial model to represent the golf field, with the hole always at the origin $(0.00, 0.00)$. CADDY and the golf ball are represented by their position. CADDY always know its position as well as hole's position, but CADDY may have imperfect knowledge of the position of the golf ball if the golf ball is outside CADDY's perception distance.

Temporal Model

We use the situation calculus as our temporal model. Each situation is represented by CADDY's position (R_x, R_y), the actual position of the golf ball (B_x, B_y), the CADDY's estimate of golf ball's position (K_x, K_y), and the position from which CADDY has most recently hit the ball (H_x, H_y). (R_x, R_y) and (B_x, B_y) represent the physical state of one situation while (K_x, K_y) and (H_x, H_y) represent CADDY's knowledge state of one situation. The significance of (K_x, K_y) and (H_x, H_y) will be described in detail later in this section.

Initially, CADDY and the golf ball are at the same position. The goal situation is described as the distance between (B_x, B_y) and the origin being less than ϵ.

Primitive Actions

There are five primitive actions CADDY can execute. Each primitive action, along with the transition functions, is described below. We use the STRIPS rule to avoid the frame problem; that is, any state of a situation that is not mentioned in the transition functions remains unchanged during action execution.

(hit X θ Z ψ)

where X and θ are input parameters while Z and ψ are output parameters.

CADDY tries to hit the golf ball to distance X in direction θ. Due to the imperfect control of hitting, golf ball actually goes to distance Y in direction ϕ, where Y follows a normal distribution centered at X with standard deviation $\beta_1 \cdot X$ and ϕ follows a normal distribution centered at θ with standard deviation α_1.

After hitting the ball, CADDY looks at the flying ball and makes an estimate of its final position. Due to imperfect perception, CADDY's estimate differs from the true position; it perceives that the golf ball will land at a position with distance Z and direction ψ from CADDY's current position, where Z follows a normal distribution centered at Y with standard deviation $\beta_2 \cdot Y$ and ψ follows a normal distribution centered at ϕ with standard deviation α_2. All these distributions are independent.

After (hit X θ Z ψ), (R_x, R_y) remains the same, (B_x, B_y) changes to $(R_x + Y \cdot \cos(\phi), R_y + Y \cdot \sin(\phi))$, (K_x, K_y) changes to $(R_x + Z \cdot \cos(\psi), R_y + Z \cdot \sin(\psi))$, and (H_x, H_y) changes to (R_x, R_y).

The cost of (hit X θ Z ψ) is *c_hit*, independent of the hitting distance.

(move X θ**)**

where X and θ are input parameters.

CADDY moves distance X in direction θ. We assume that CADDY has perfect control in moving.

After (move X θ), (R_x, R_y) will change to $(R_x + X \cdot \cos(\theta), R_y + X \cdot \sin(\theta))$; (B_x, B_y), (K_x, K_y), and (H_x, H_y) remain the same.

The cost of (move X θ) is *c_move* times X.

(look Z ψ**)**

where Z and ψ are output parameters.

CADDY looks around and searches for the golf ball. Since CADDY has limited perception, CADDY may not always locate the position of the ball after executing *look*. If the ball is within distance λ (the perception range of CADDY's sensors), Z and ψ will be distance and direction of the ball from CADDY; otherwise Z will be -1.

After (look Z ψ), (K_x, K_y) changes to (B_x, B_y) if CADDY finds the ball; otherwise (K_x, K_y) will be undefined. (R_x, R_y), (B_x, B_y), and (H_x, H_y) remain the same.

The cost of (look Z ψ) is a constant *c_look*.

(scan $PATH$ Z ψ**)**

where $PATH$ is input parameter while Z and ψ are output parameters.

$PATH$ is a list of (*distance*, *direction*) pairs. CADDY moves along $PATH$ scanning for the golf ball. If it finds the ball (that is, the ball is within distance λ), then it stops, and returns Z and ψ to be the distance and direction of the ball from CADDY's current position. Otherwise, it completes scanning the whole $PATH$ and returns $Z = -1$. We can view scan as a sequence of *look*ing and *move*ing a short distance along the pre–defined route $PATH$.

After (scan $PATH$ Z ψ), (R_x, R_y) changes to the position where CADDY stops while (K_x, K_y) changes to (B_x, B_y) if CADDY finds the ball; otherwise (K_x, K_y) will be undefined. (B_x, B_y) and (H_x, H_y) remain the same.

The cost of (scan $PATH$ Z ψ) is *c_look* plus *c_scan* times the distance CADDY actually moves through the $PATH$. *c_scan* should be greater than *c_move* but less than *c_move* + *c_look*/λ. The first inequality ensures that *scan* is more expensive than the sequence of *move*s, which form the $PATH$; while the second inequality ensures that *scan* is cheaper than repeatedly *move* a short distance λ and *look*.

(replace)

CADDY gives up looking the golf ball, returns to the position of the last hit, and starts again with a new ball.

After (replace), values of (R_x, R_y), (B_x, B_y), and (K_x, K_y) change to (H_x, H_y) while (H_x, H_y) remains the same.

The cost of (replace) is *c_replace* plus *c_move* times the distance CADDY travels to return back to the position where CADDY hits the ball last time.

We can see that *look* is a pure sensing action (actions that provide information of the physical environment); *move* and *replace* are effect actions (actions that change the state of the physical environment); while *scan* and *hit* involve both sensing and physical changes.

The planner can perfectly predict the effects of actions *move* and *replace*. It cannot perfectly predict the effects of *look* and *scan*, due to the limits of perception, nor the effect of *hit*, due to the limited perception and imperfect physical control.

CADDY's knowledge about the ball's position varies over time. If the golf ball is within CADDY's perception distance and CADDY has executed a search operation to look for the ball, (K_x, K_y) and (B_x, B_y) should have the same value. Otherwise CADDY only knows that the position of the golf ball is within an area, which is controlled by two independent normal probabilistic distributions described in *hit* operation. The only information CADDY knows regarding to the position of golf ball are (K_x, K_y) returned by previous *hit* and those four physical parameters controlling the normal distributions. CADDY knows that it is unlikely that (K_x, K_y) is the exact position of the ball; instead, CADDY treats (K_x, K_y) as a reference position to guide its search.

The characteristic of the golf world is determined by six physical parameters (α_1, β_1, α_2, β_2, λ, and ϵ) and five cost parameters (*c_hit*, *c_move*, *c_look*, *c_scan*, and *c_replace*).

4.2 Plans and Quality of Plans

CADDY can have several different strategies for hitting the ball, for searching the ball in the field, and for deciding to replace the ball. Different sequence of strategies lead to

```
(sequence (cond (ge (distance agent hole) 120)
                (hit 120 (direction agent hole) Dist Dir)
                (hit (distance agent hole) (direction agent hole) Dist Dir))
          (while (not ball_in) (sequence
                (move Dist Dir)
                (look Dist Dir)
                (cond (ge Dist 0.0) (move Dist Dir) (replace))
                (cond (ge (distance agent hole) 120)
                      (hit 120 (direction agent hole) Dist Dir)
                      (hit (distance agent hole) (direction agent hole) Dist Dir)))))
```

Figure 4.1: An Example of a Selected Plan

different plans. Since CADDY adapts and uses pre–stored plans for problem solving, the Plan Library must have methods to store known plans.

Table 4.1 gives the syntax which is used by CADDY to represent a plan in the Plan Library. It has a LISP–like syntax with three control–structure operators: sequence, condition, and while loop. A *task* is defined as one specific execution of a plan under given parameter assignments and a starting situation.

Figure 4.1 shows an example of a simple plan. It repeatly hits a fixed short distance towards the hole, moves to the reference point returned by *hit*, searches the ball, then either moves to the ball (if it finds the ball) or gives up and replaces a new ball (if it does not find the ball) for next hit.

A plan may perform very well at one starting situation but very poorly at another starting situation. The quality of a plan is the measurement of how efficiency the plan may be under current parameter assignments. Since several uncertainty factors are included in the testbed, different executions of the same plan under the same parameter assignments may lead to different outcome. The quality of a plan under one parameter assignments cannot be determined by the cost of one execution of the plan; instead, it should be determined

```
PLAN ≡      (hit EXP EXP Variable Variable) |
            (move EXP EXP) |
            (look Variable Variable) |
            (scan (PATH) Variable Variable) |
            (replace) |
            (sequence PLAN PLAN) |
            (cond COND PLAN PLAN) |
            (while COND PLAN)

COND ≡      ball_in |
            (eq EXP EXP) | (ne EXP EXP) |
            (gr EXP EXP) | (ge EXP EXP) |
            (le EXP EXP) | (ls EXP EXP) |
            (and COND COND |
            (or COND COND) |
            (not COND)

PATH ≡ (EXP EXP) | (EXP EXP) PATH

EXP ≡       Constant | Variable |
            (+ EXP EXP) | (- EXP EXP) |
            (* EXP EXP) | (/ EXP EXP) |
            (exp EXP EXP) | (sqrt EXP) |
            (sin EXP) | (cos EXP) |
            (abs EXP) |
            (distance POINT POINT) |
            (direction POINT POINT)

POINT ≡ robot | hole | ball
```

Table 4.1: Syntax of a Simple Planning Language Used by the Golf World Testbed

by the average cost of executions of the plan. The cost of an execution is the sum of the costs of the primitive actions executed.

The goal of the CADDY planning system is to construct a plan which achieves the goal of hitting the golf ball into the hole with the least cost.

4.3 The Golf World Testbed

The golf world testbed is designed as a setting which is simple but still raises interesting problems of imperfect control and knowledge. It was specifically intended to be unlike classical planning domains such as the blocks world; the difficult issues of the blocks world do not arise in the golf world and vice versa.

1. This is a single agent system, so we do not need to consider issues about multiple agents, like problems about how to communicate and coordinate behaviors between agents, how to specify the effect of simultaneous actions.

2. There is a clear interface between *CADDY* and the environment and a simple yet well–defined model of time (the situation calculus) and space (the 2–D coordinate system).

3. There is a simple definition of cost of each primitive actions as well as the measures of plan quality, which will ease the design of the Plan Library and the Estimation Module.

4. The uncertainty of behavior is controlled by a set of numerical parameters. This provides a convenient way for us to control different experiments by varying these parameters systematically and analyze the performance over a set of sample problems and conditions, which will further help us the design of the Plan Library.

5. Unlike other planning systems in which the planning agent usually have different top–level goals for different executions, *CADDY* has a constant top–level goal (hitting the ball into the hole) for each execution. The choice of a plan to be executed depends on the initial situation and physical parameters.

6. It has a simple set of primitive actions, which are also clearly defined, and a simple causal structure: *CADDY* sequentially hits the golf ball, looks for the ball, then moves to a specified point for another hit. This will ease the design of the Plan Execution Module.

If we compare CADDY with the blocks world planners in these respects, we note the following similarities and contrasts: Point (1) applies equally to both. Point (2) applies even more strongly to the blocks world, which, in effect, uses a discrete model of space. Point (3) is irrelevant for most blocks world planners since they generally ignore costs. Point (4) does not apply in the blocks world since it is deterministic. Point (5) is false in the blocks world. As regards point (6), the blocks world also has a simple model of actions. The physical causal structure of the golf world is simpler than that of the blocks world: a sequence of actions in the golf world is physically feasible if and only if each *hit* is preceded either by *replace* or by *move* to the position of the ball. On the other hand, the epistemic structure is more complex. The blocks world, which assumes perfect knowledge, has no epistemic structure.

Despite these simplicities, the golf world is nonetheless rich enough to support interesting planning.

- Planning in the world involves probabilistic, epistemic, and spatial reasoning, and the tradeoff of costs based on those various considerations.

For example, in deciding whether to hit the ball far, CADDY must balance the gain of making physical progress towards the hole against the loss of losing sight of the ball. Which is better depends in a complex way on the control of hit (α_1, β_1), the power of perception (α_2, β_2, λ), and the relative costs of searching and hitting.

Another example: in choosing a path to scan, CADDY must find a path that probably takes it within λ of the ball.

- *CADDY* has limited perception and incomplete knowledge of the test environment and it cannot predict the exact outcome of primitive actions it intends to execute. The support of uncertainty in the testbed increases the complexity in planning.

- Though there is only an unique top–level goal, the choice of optimal plan may vary under different circumstances. Depending on the quantities of physical parameters and the current situation, different strategies may be appropriate.

  For example, *CADDY* may prefer to replace the golf ball every time it hits the golf ball to a wrong direction if cost of replacement is far cheaper than cost of searching the ball flying to wrong direction; but if cost of replacement is expensive, *CADDY* may rather search for the golf ball even if it flies to the wrong direction.

  Another example: *CADDY* may wish to hit a long distance if its perception distance is large; but if its perception distance is short due to the foggy environment, *CADDY* may prefer to hit a short distance to avoid more cost on ball searching.

Chapter 5

Implementation

This chapter gives a detailed discussion about the implementation of each module in CADDY's architecture as applied to the golf world testbed. It also discusses briefly possible improvements to each module.

5.1 Plan Library

The Plan Library contains the following:

- A collection of plans. Plans are represented as a combination of control–structure operators and primitive actions. Currently we use a LISP–like notation with three simple control–structure operators (sequence P_1 P_2), (cond Q P_1 P_2), and (while Q P_1). Figure 4.1 is an example of a plan.
- A Best Table (BT) recording the history of CADDY's execution. It stores a collection of actual executions CADDY has performed (the system parameter setting, the starting situation, the plan executed, and the simulated cost). Every time CADDY executes under one specific circumstance, CADDY updates information in the Best Table: if similar circumstance has been met before, update the plan entry and sim-

ulated cost entry whenever current run results in a less simulated cost; otherwise it adds the new result in the Best Table.

- A Failure Table (FT) recording the failed experiences CADDY has encountered. Every time CADDY knows that the adapted plan is outperformed by the original selected plan, CADDY records the selected plan, the system parameter settings, and the heuristic rules applied to the Failure Table so that next time CADDY encounters similar situations, CADDY will not apply the same set of heuristic rules again.

- A Simulation Table (ST) recording the simulated costs of each plan under different selected circumstances. Every time a new plan has been included in the Plan Library, the Simulation Table needs to be updated.

- A collection of maintenance modules. These include routines of table retrieval, table update, plan retrieval, plan addition, etc.

The Best Table and the Failure Table are implemented as K–D trees [FBF77]. A K–D tree is a K dimensional binary search tree such that at each level a different attribute value is tested to determine the direction in which a branch is to be made. For example, in a 2–D tree, the first attribute is compared at the root and at even levels (assuming root is level 0), while the second attribute is compared at odd levels. Search and insert/delete operations are the same as in an ordinary binary search tree, except that the appropriate attribute value is used at each level.

The Best Table is a 5–D tree for the four system parameters (λ, α_1, β_1, $c\_replace$) and the starting situation (the distance between the hole and CADDY at the beginning). New entries are inserted into the Best Table, or old entries in the Best Table may be modified during CADDY's execution. The Best Table is not a balanced tree.

The Failure Table is also a 6–D tree for the plan id, the four parameters (λ, α_1, β_1, *c_replace*) and the starting situation (the distance between the hole and CADDY at the beginning). New entries are inserted into the Failure Table but old entries are not modified or deleted. The Failure Table is not a balanced tree.

The Simulation Table is a linear table, with each entry represents a plan stored in the Plan Library. Each element in the Simulation Table contains the name of the plan and a five dimensional linked list which stores several simulated costs of selected system parameter settings and the starting situations. Each node in the top dimension linked list contains a specific value of parameter *c_replace* and a four dimensional linked list; each node in the four dimensional list contains a specific value of parameter α_1 and a three dimensional linked list; and so on.

The reason of using a linked list, instead of a balanced search tree, in each level of the Simulation Table is that CADDY only needs to store a few value settings of each parameter. Therefore using binary search does not improve run–time performance much. Once a entry has been inserted into the Simulation Table, the entry will not be modified.

5.2 Estimation Module

The Estimation Module is an assistance tool used by the Plan Selection Module. It uses information stored in the Simulation Table and calculates the estimated cost of a plan under one circumstance.

Using information in the Simulation Table, the Estimation Module uses linear interpolation on current parameter settings and starting situation. There are 5 stages of interpolation, with 1 for cost parameter *c_replace*; 3 for physical parameters λ, α_1, and β_1; and 1 for the distance between the hole and the ball at the starting distance. For each stage, the Estimation Module linearly searches through the linked structure of the Simulation Table, chooses two nearest values of the current parameter, and applies linear interpolation to get the estimated cost under current interpolation stage. The order of choices of parameters to interpolate follows the order how the linked list structure of each plan in the Simulation Table stores the simulated result. After passing down then passing up five stages, the Estimation Module will get the estimated cost of a specific plan under current circumstance.

For example, with the starting circumstance which ($c\_replace$, λ, α_1, β_1, $dist$) equals to (130.00, 35.00, 20.00, 0.20 ,200.00), the Estimation Module first interpolates the $c\_replace$ by 100.00 and 200.00; for each of these two $c\_replace$ value, the Estimation Module interpolates the α_1 by 10.00 and 24.00; for each α_1 value, the Estimation Module interpolates the β_1 by 0.10 and 0.24; for each β_1 value, the Estimation Module interpolates λ by 20.00 and 60.00; and finally for each λ value, the Estimation Module interpolates $dist$ by 100.00 and 300.00. After the Estimation Module gets the interpolated cost at the $dist$ level, these values are passed up to the λ level and the Estimation Module gets the interpolated cost with λ and $dist$ equal to current setting. After five levels of passing up, the Estimation Module gets the interpolated cost under current circumstance.

These interpolation calculations are computation intensive. Currently the result of the Estimation Module is acceptable but still needs improving. We should make the difference between the estimated cost and the simulated cost as small as possible.

5.3 Plan Selection Module

The objective of the Plan Selection Module is to select the expected best plan from the Plan Library, given the current system parameter settings and the starting situation.

The Plan Selection Module first looks through the Best Table to find if CADDY has met similar circumstances before. If a match is found, the Plan Selection Module retrieves the plan from the Plan Library and passes it to the Plan Adaptation Module; otherwise the

Plan Selection Module uses the Estimation Module to get the estimated cost of each plan in the Plan Library and passes the plan with the least estimated cost to the Plan Adaptation Module.

The current CADDY planning system needs to interpolate every plan in the Plan Library to get the plan with the least estimation cost. As more new plans are added in the Plan Library, it will be time consuming for the Plan Selection Module to get the expected best plan. The most obvious way of improving the performance of the Plan Selection Module is to reduce those unnecessary interpolations or improve the performance of the interpolations.

5.4 Plan Adaptation Module

Since one main objective of the CADDY planning system is to adapt previous experiences, the Plan Adaptation Module plays the key role in CADDY. The execution of the Plan Adaptation Module contains three stages:

1. The Parsing Stage.

2. The Analysis Stage.

3. The Adaptation Stage.

The Parsing stage reads in the selected plan and translates into the internal representation. The internal representation of a plan is a step/sub–step relation tree. Each internal node represents a control–structure operator, containing an ID which identifies the type of the internal node (*sequence*,*cond*, or *while*), and pointers to the condition structure (*cond* and *while*) and sub–plans (*while* has one sub–plan while *sequence* and *cond* have two sub–plans). Each external node represents a primitive action, containing information about the type of the primitive actions and the parameters. Figure 5.1 is the result of the Parsing Stage of a plan in figure 4.1.

The Analysis Stage reads in the internal plan representation and attaches to each node description information about knowledge or causal relation, etc. This information helps CADDY to gain global knowledge of the selected plan and then adequately apply heuristic repair rules during the Adaptation Stage. Consider the plan segment and its internal representation in figure 5.2. This plan segment *hit*s the ball towards the hole with distance X, if possible. Figure 5.2 also shows information attached to each plan node by the Analysis Stage.

After the Analysis Stage, CADDY knows that the purpose of the *cond* node is to hit the ball towards the hole with appropriate distance, and the first *hit* is to hit the ball with constant distance while the second *hit* is to hit the ball with distance between CADDY and the hole. If the Plan Adaptation Module decides to adjust the hitting distance, it will know only to change the X parameter of the first *hit*, but not of the second *hit*.

The information attached to plan nodes should also prevent the Adaptation Stage from applying the same heuristic rules again and again. Figure 5.3 gives the result of the Analysis Stage of the plan in figure 4.1.

The Analysis Stage passes the internal plan representation with information attachment to the Adaptation Stage to perform plan modification by using a set of domain–independent or domain–dependent heuristic rules. Currently we only implement a small set of domain–dependent heuristic rules within CADDY, but we will explore more in the near future. Rules currently used include:

- Rules for adapting *hit*.

  H1: If CADDY's hit is accurate under current parameter settings (α_1 and β_1 are both very small), CADDY will substitute *hit* operations by another *hit* operations

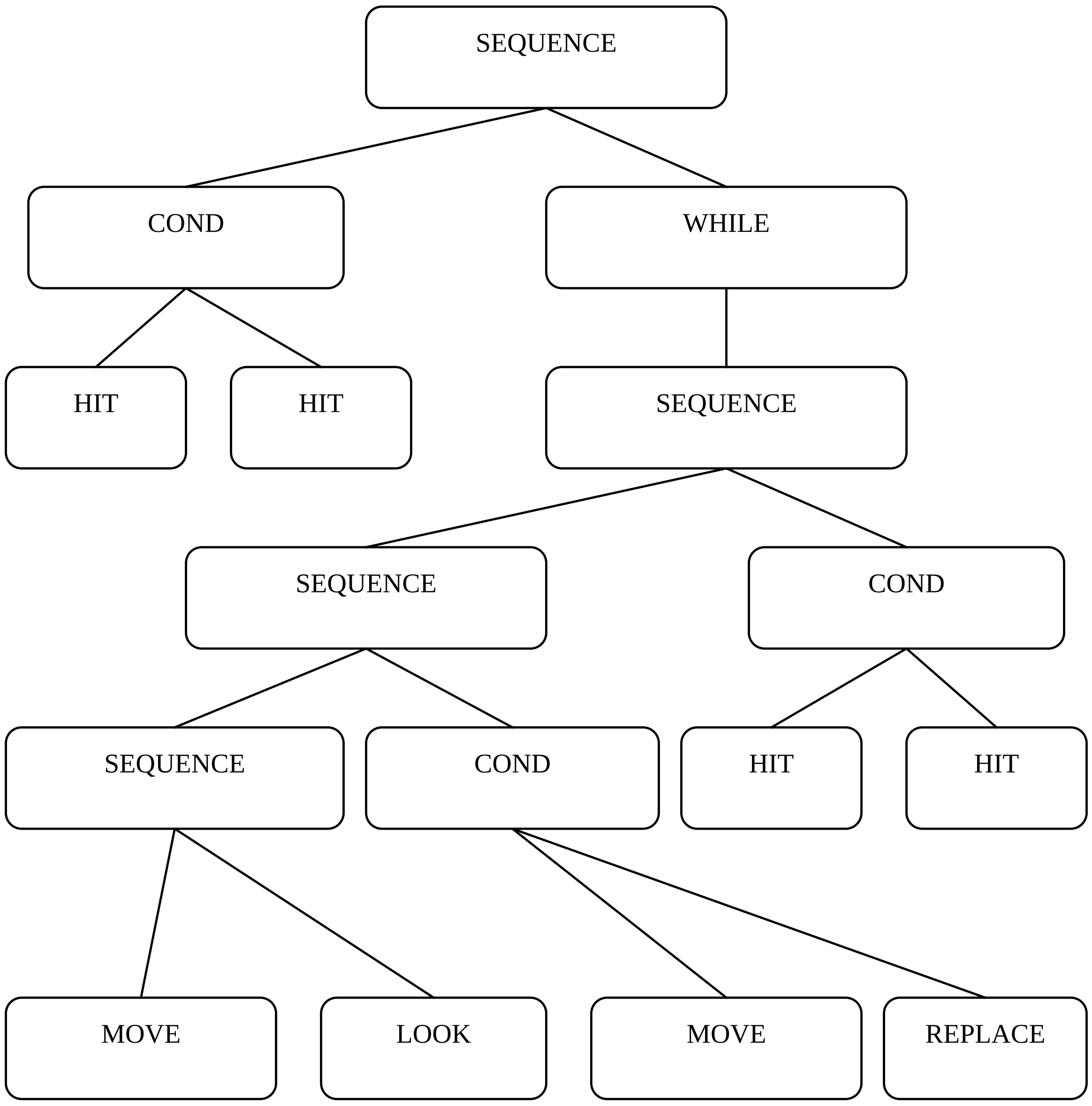

Figure 5.1: Result of the Parsing Stage of the Selected Plan

```
(cond (ge (distance Agent Hole) X)
      (hit X theta Z psi)
      (hit (distance Agent Hole) theta Z psi))
```

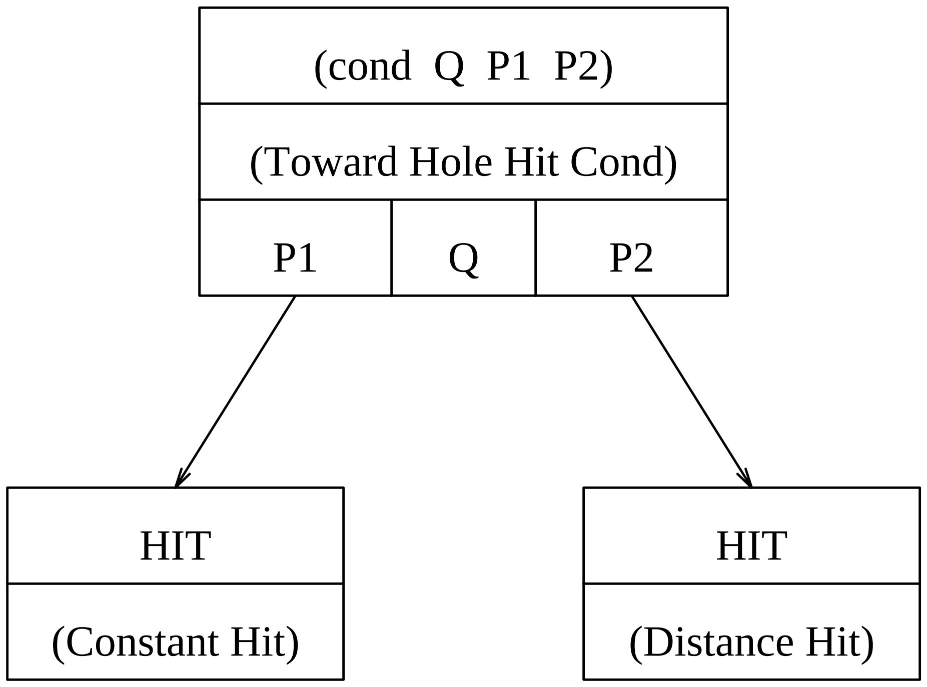

Figure 5.2: A Plan Segment and Its Internal Representation with Information Attached by the Analysis Stage

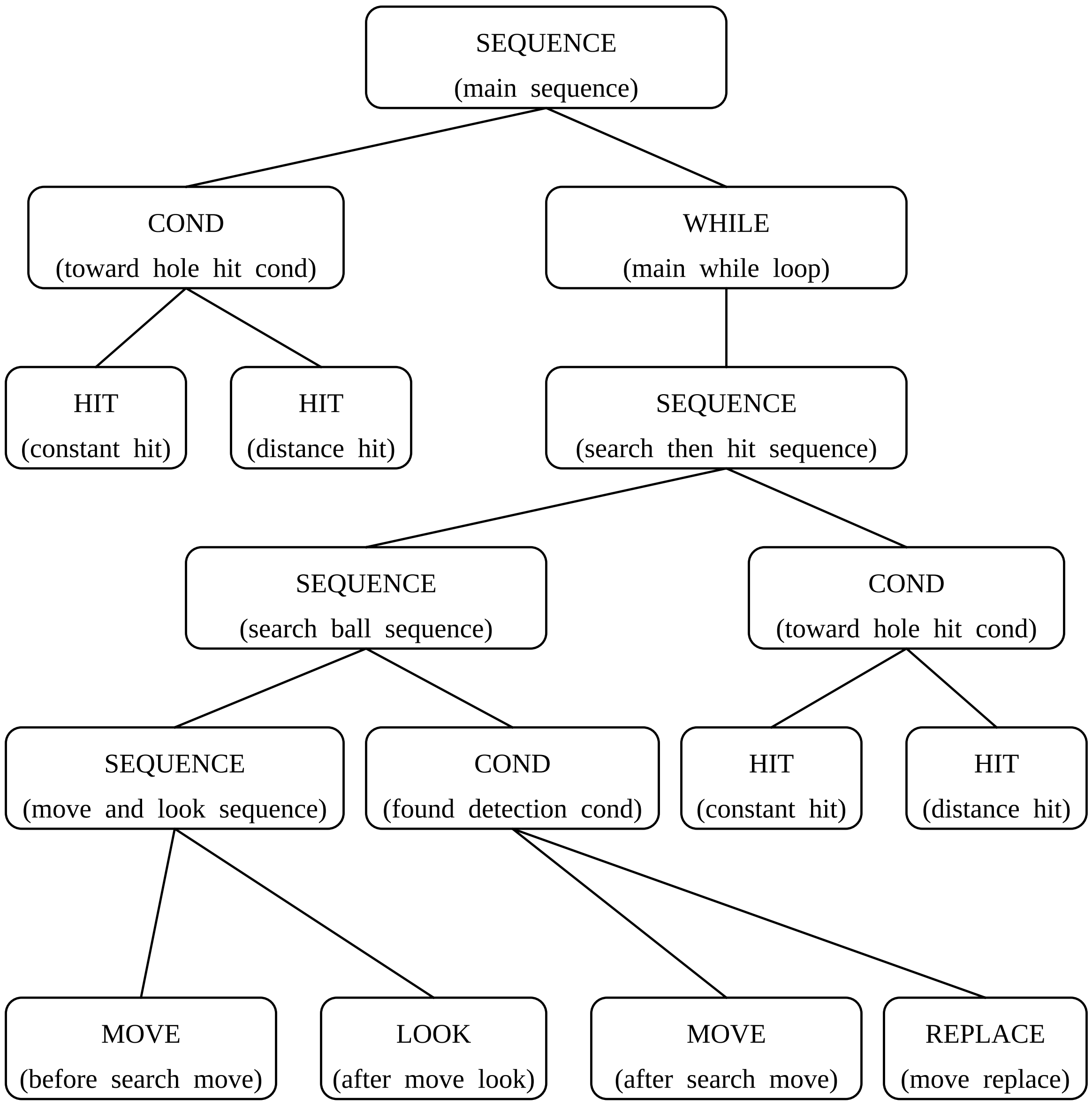

Figure 5.3: Result of the Analysis Stage of the Selected Plan

with the maximum distance CADDY can hit.

H2: In order to have a good chance that the ball is within CADDY's perception distance λ while CADDY can still gain more forward progress, the Plan Adaptation Module adjusts the hitting distance of any constant *hit* operation by the following rule: For ($hit\ X\ \theta\ Z\ \psi$) while X is a constant, if X is larger than $\beta_1 * \lambda$, the Plan Adaptation Module decreases X by d, a small adjustment constant; otherwise the Plan Adaptation Module increase X by d.

- Rules for *replacing* the golf ball.

  R1: If replacing the ball costs less than moving distance d_1 (a pre–defined constant) then looking ($c\_replace$ is less than $c\_look + c\_move * d_1$) and CADDY can hit the ball with direction accuracy (α_1 is small), the Plan Adaptation Module will insert the plan segment before the search operations that CADDY can *replace* the ball, instead of starting the search, if CADDY detects the ball flies out of direction (angle between θ and ψ is greater than $\pi/3$).

  R2: If replacing the ball costs less than moving distance d_1 then looking ($c\_replace$ is less than $c\_look + c\_move * d_1$), the Plan Adaptation Module will insert the plan segment before the search operations that CADDY can *replace* the ball, instead of starting the search, if CADDY detects the forward progress is too small ($Z \ll X$ after (hit $X\ \theta\ Z\ \psi$)).

  R3: If replacing the ball costs less than moving backwards for distance d_2 (a pre–defined constant), hitting the ball back, then moving back to current position ($c\_replace$ is less than $c\_hit + c\_move * 2 * d_2$), the Plan Adaptation Module will insert the plan segment before the search operations that CADDY can *replace* the ball, instead of starting the search, if CADDY detects that the ball flies backwards (angle between θ and ψ is greater than $\pi/2$).

- Rules for avoiding *replace*.

  S1: If replacing ball is too costly under current parameter settings (*c_replace* is very large), CADDY will either extend the *scan* path by distance d, where $d \leq$ *c_replace*/*c_scan*, if the search operation before *replace* is *scan*; or insert a *scan* operation with path of length d if the search operation before *replace* is *look*. The format of path depends on the values of α_1 and β_1. If no search operations before *replace* (the *replace* operation is inserted by heuristic rules **R1** to **R3**), CADDY will do nothing.

  Currently the inserted/extended *scan* path is a spiral–like path centered at the position where CADDY currently stands. In the future, the structure of the path should depend on the physical parameters α_1, β_1, α_2, and β_2, which affect the outcome of primitive action *hit*. For example, if α_1 is large and β_1 is small, which means that CADDY can can *hit* the ball with more distance accuracy but poor directional accuracy, the inserted/extended path should emphasize on a wide direction range but a narrow distance range. Similarly, the inserted/extended path should emphasize on a wide distance range but a narrow direction range if α_1 is small while β_1 is large.

After collecting heuristic rules which CADDY intends to apply to the selected plan, the Plan Adaptation Module checks the Failure Table whether applying these rules matches previous failed experiences. If not, the Plan Adaptation Module is safe to apply those collected heuristic rules to the selected plan; otherwise the Plan Adaptation Module randomly chooses a subset of the collected heuristic rules and checks if the newly formed collection of heuristic rules still matches previous failed experiences. The Plan Adaptation Module continues choosing subset of the collected heuristic rules until the subset does not match

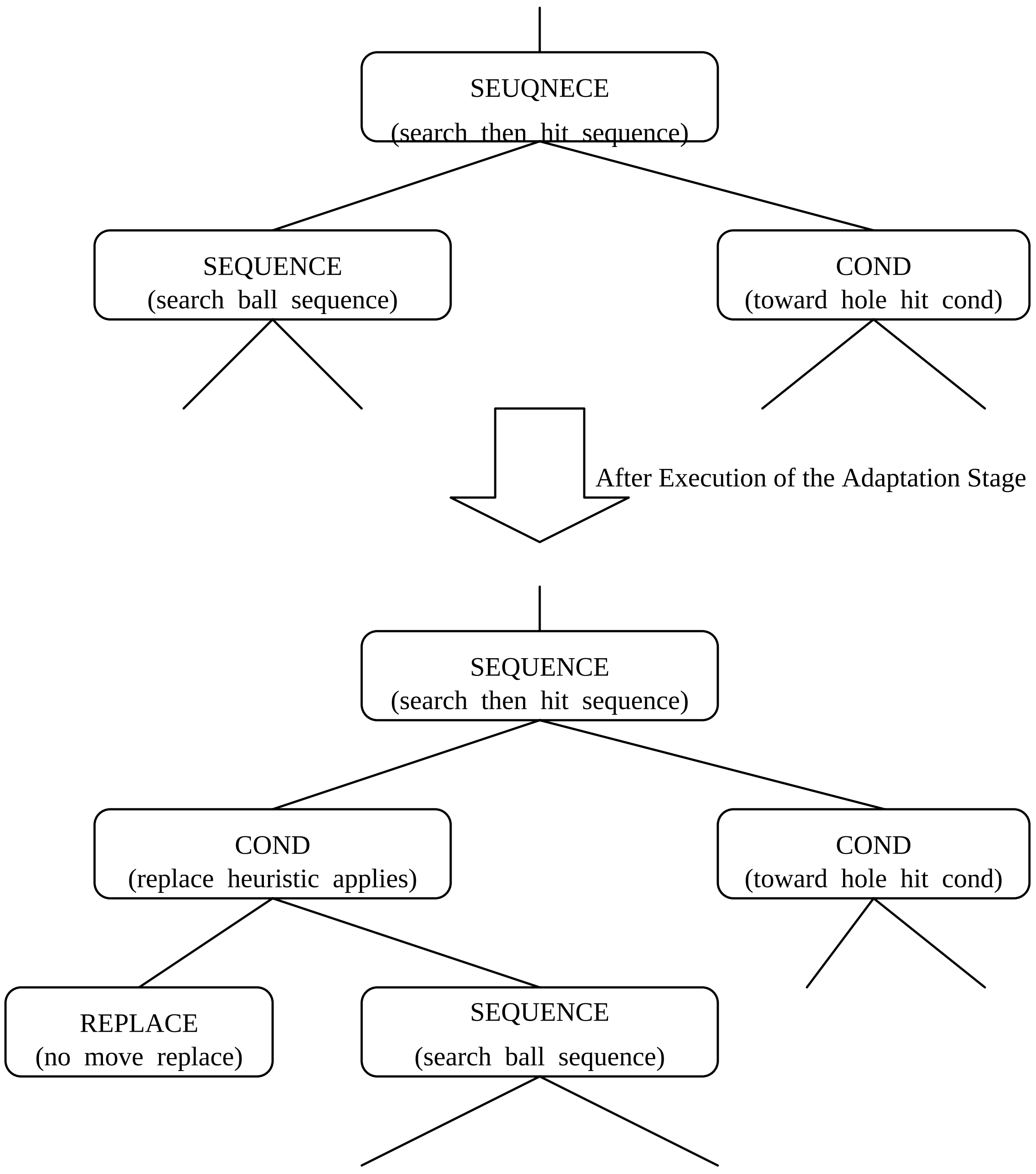

Figure 5.4: A Replace Heuristic Rule Applied to an Internal Representation of a Plan

any failed experience in the Failure Table.

Figure 5.4 shows how heuristic rule for *replacing* the golf ball (either **R1**, **R2**, or **R3**) applies to an internal plan segment.

For example, with the following parameter assignments where ($c\_replace$, λ, α_1, β_1) = (120.00, 35.00, 3.00, 0.03) and the starting situation where CADDY the golf ball are at position (200.0, 200.0), the Plan Adaptation Module applies heuristic rules **H1** (since $\alpha_1 = 3.00$ and $beta_1 = 0.03$ are both small) and **R3** (since $c\_replace = 120.0$ is not less than $c\_move * 120.0 = 120.0$ but is less than $c\_move * 2 * 120.0 + c\_hit = 300.0$) to the plan in figure 4.1 and gets the adapted plan in figure 5.5.

We can view the Plan Selection Module as selecting a plan which is expected to suit current circumstance best; while adapting the plan by the Plan Adaptation Module as small adjustments of the parameters within the plan.

Currently LISP–like plans are stored in the Plan Library and are parsed and analyzed by the Plan Adaptation Module. We did this to simplify the representation of plans and hence to simplify the construction of the initial Plan Library. One way to improve the run–time performance of the Plan Adaptation Module is to store the whole internal structure of plans (the step/sub–step tree and the information attached by the Analysis Stage) in the Plan Library and a new retrieval routine in the Plan Adaptation Module to read in the plan. This will save the processing time spent by the Parsing Stage and the Analysis Stage. But we need to transform each plan currently in the Plan Library to the new form.

5.5 Plan Execution Module

The Plan Execution Module simulates the execution of the plan generated by the Plan Adaptation Module until it reaches the goal: the golf ball is in the hole. CADDY has a graphic interface which can demonstrate the execution of a plan on a graphic board and

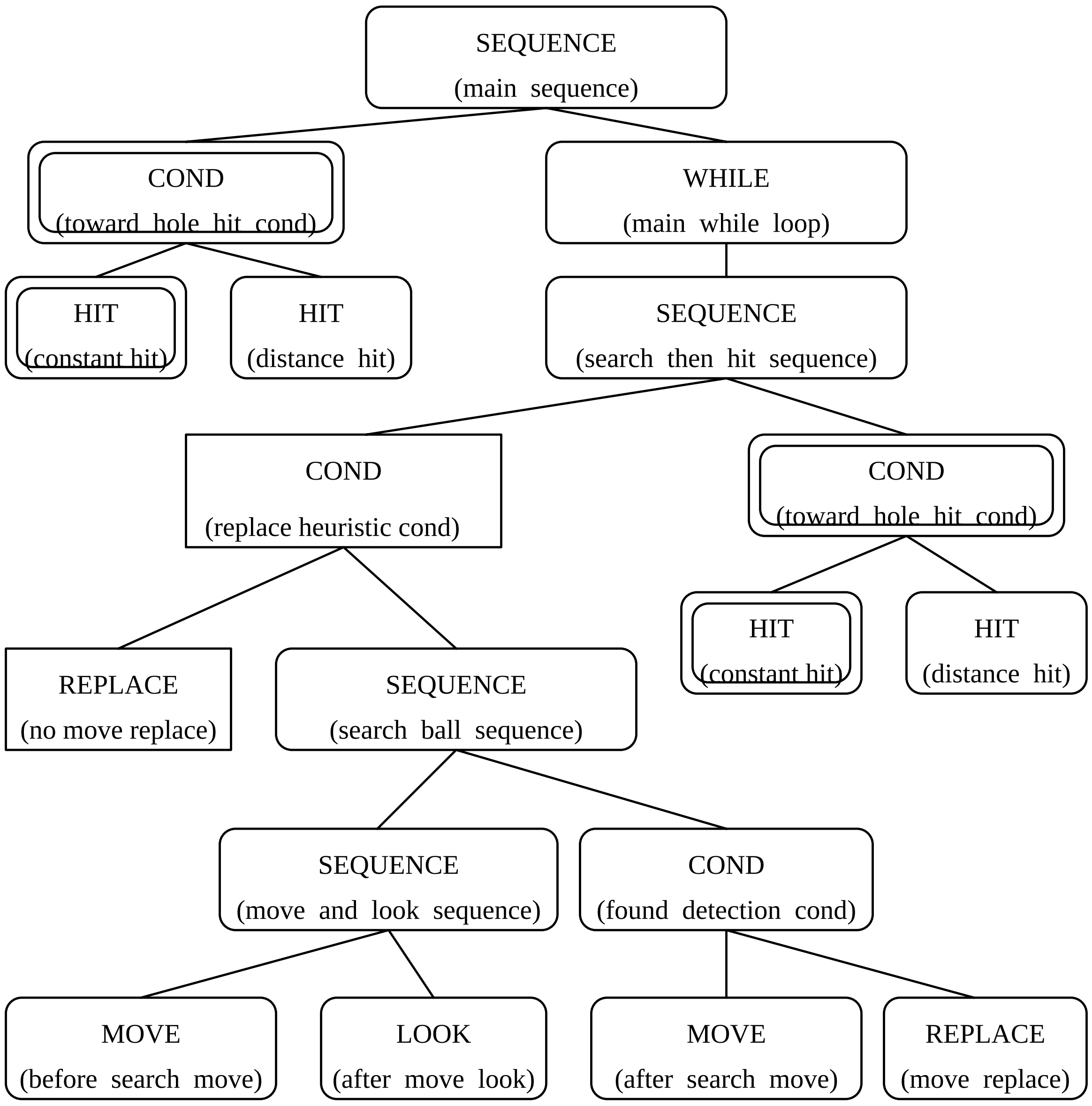

Figure 5.5: Result of the Adaptation Stage of the Selected Plan

```
(sequence (cond (ge (distance agent hole) 150)
                (hit 150 (direction agent hole) Dist Dir)
                (hit (distance agent hole) (direction agent hole) Dist Dir))
          (while (not ball_in) (sequence
                (cond (ge (abs (- Dir (direction agent hole))) 90) (replace) (sequence
                      (move Dist Dir)
                      (look Dist Dir)
                      (cond (ge Dist 0.0) (move Dist Dir) (replace))))
                (cond (ge (distance agent hole) 150)
                      (hit 150 (direction agent hole) Dist Dir)
                      (hit (distance agent hole) (direction agent hole) Dist Dir)))))
```

Figure 5.6: External Representation of the Adapted Plan

give helpful information like the primitive action currently executed, the current situation and system parameter settings, and the total cost so far. Another text window also shows the execution log for trace purpose.

If the plan generated by the Plan Adaptation Module shows a lower average cost than the estimation cost of the original plan selected by the Plan Selection Module, the Plan Execution Module will

1. store the newly generated plan in the Plan Library.

2. record the simulated cost of the current system parameter settings in the Best Table so that the same plan can be selected again by the Plan Selection Module if similar circumstance is encountered again. The information in the Best Table thus saves the future computation of a series of interpolations and plan adaptation.

3. invoke several simulation runs under selected circumstances and record the simulated costs in the Simulation Table so that it can be interpolated and selected by the next run.

For example, if the plan in figure 5.5 (adapted plan) outperforms the plan in figure 5.3 (original plan), the internal plan representation will be transformed to the plan in figure 5.6 and be recorded in the Plan Library, the simulation results are also recorded in the Best Table and the Simulation Table. The differences between figure 5.6 and figure 4.1 are underlined in figure 5.6.

If the plan generated by the Plan Adaptation Module shows a greater average cost than the estimated cost of the original plan selected by the Plan Selection Module, the Plan Execution Module discards the newly generated plan and updates the Best Table with the original plan selected by the Plan Selection Module and its estimated cost. The failed experience is also recorded in the Failure Table.

If the plan chosen by the Plan Selection Module is not changed by the Plan Adaptation Module, the Plan Execution Module updates the Best Table with the plan and its simulated cost.

The current system treats physical parameters as constants during plan execution. We would like to extend this by allowing physical parameters varying values during plan execution. This variation can either follow pre–defined functions or be obtained from probabilistic models.

Chapter 6

Comparison of CADDY with Other Planning Systems

This chapter compares the CADDY planning system with some recent planners and considers how theoretical work on plan semantics applies to CADDY.

6.1 CADDY and Uncertainty in Planning

Several probabilistic planning systems or probabilistic reasoning techniques are discussed in section 2.2. We will compare CADDY with BURIDAN [KHW93, KHW94] and Partially Observable Markov Decision Processes [CKL94a, CKL94b] in a number of respects.

Model of Actions

CADDY, BURIDAN, and POMDPs all use similar representations of primitive actions with probabilistic outcomes. BURIDAN's action model extends the standard STRIPS representation to allow conditional and probabilistic effects. The possible effects of an action are partitioned into mutually exclusive and exhaustive classes, and each class is mapped a particular probabilistic value. POMDPs defines state transition functions as mappings from environmental states and actions into discrete probabilistic distributions over a set of states.

The action model of CADDY differs from those of BURIDAN and POMDPs in that POMDPs and BURIDAN assume a finite (in practice, a small) state space as the effects of an action; while CADDY uses a continuous (uncountably infinite) state space as the effects of an action.

In the golf world testbed, only one action has a probabilistic outcome: the *hit* operation. All other actions have deterministic outcomes. We can use either of the above probabilistic models to represent the *hit* operations. Consider the POMDPs model of action (hit X θ Z ψ) with X and θ as given constants. Each situation listed below are represented by three points: CADDY's current position (R_x, R_y), the golf ball's current position (B_x, B_y), and CADDY's belief of the ball's position (K_x, K_y). Before executing *hit*, CADDY and the ball are at the same position.

$$S_0 = (R_x,\, R_y),\, (R_x,\, R_y),\, (R_x,\, R_y)$$

After *hit*, the ball flies distance Y in the direction ϕ, where Y follows a normal distribution centered at X with standard deviation $\beta_1 \cdot X$, and ϕ follows a normal distribution centered at θ with standard deviation α_1. The original situation transfers to the following situation

$$S_1 = (R_x,\, R_y),\, (R_x + Y\cos(\phi),\, R_y + Y\sin(\phi)),\, (R_x,\, R_y)$$

with probability

$$\begin{aligned} P_1 &= \frac{1}{\sqrt{2\pi}X\beta_1}\exp(-\tfrac{1}{2}(\tfrac{Y-X}{X\beta_1})^2)\cdot\frac{1}{\sqrt{2\pi}\alpha_1}\exp(-\tfrac{1}{2}(\tfrac{\phi-\theta}{\alpha_1})^2) \\ &= \frac{1}{2\pi X\alpha_1\beta_1}\exp(-\tfrac{1}{2}[(\tfrac{Y-X}{X\beta_1})^2 + (\tfrac{\phi-\theta}{\alpha_1})^2]) \end{aligned}$$

for all possible Y and ϕ combinations, since the two normal distributions are independent.

After *hit*, CADDY believes that the ball flies distance Z in the direction ψ, where Z follows a normal distribution centered at Y with standard deviation $\beta_2 \cdot Y$, and ψ follows a normal distribution centered at ϕ with standard deviation α_2. The original situation transfers to the following situation

$$S_2 = (R_x,\, R_y),\, (R_x + Y\cos(\phi),\, R_y + Y\sin(\phi)),\, (R_x + Z\cos(\psi),\, R_y + Z\sin(\psi))$$

with probability

$$\begin{aligned} P_2 &= \tfrac{1}{\sqrt{2\pi}X\beta_1}\exp(-\tfrac{1}{2}(\tfrac{Y-X}{X\beta_1})^2)\cdot\tfrac{1}{\sqrt{2\pi}\alpha_1}\exp(-\tfrac{1}{2}(\tfrac{\phi-\theta}{\alpha_1})^2) \\ &\cdot\tfrac{1}{\sqrt{2\pi}Y\beta_2}\exp(-\tfrac{1}{2}(\tfrac{Z-Y}{Y\beta_2})^2)\cdot\tfrac{1}{\sqrt{2\pi}\alpha_2}\exp(-\tfrac{1}{2}(\tfrac{\psi-\phi}{\alpha_2})^2) \\ &= \tfrac{1}{4\pi^2 XY\alpha_1\alpha_2\beta_1\beta_2}\exp(-\tfrac{1}{2}[(\tfrac{Y-X}{X\beta_1})^2 + (\tfrac{\phi-\theta}{\alpha_1})^2 + (\tfrac{Z-Y}{Y\beta_2})^2 + (\tfrac{\psi-\phi}{\alpha_2})^2]) \end{aligned}$$

for all possible Y, ϕ, Z, and ψ combinations. Again, these four normal distributions are independent. P_2 now can be served as the transition probability of action *hit* (from situation S_0 to S_2). Thus, the probability distribution of (hit X θ Z ψ) is $\sum_Y \sum_\phi P_2$ for all possible Z and ψ combinations.

The representation of *hit* operation using BURIDAN's model can be constructed similarly.

BURIDAN does not contain perception actions, it assumes that the perceptions are of all–or–nothing style. POMDPs uses incomplete perceptions. a discrete probabilistic distribution over a set of possible physical states. As for CADDY (within the golf world testbed), outcomes of sensing actions (*look* and *scan*) are of all–or–nothing style; CADDY either knows the exact position of the ball or knows nothing after sensing actions. On the other hand, action *hit* implicitly contains a probabilistic perception action (the output parameter Z and ψ).

Plan Construction

BURIDAN and POMDPs generate plans from scratch while CADDY reuses or adapts pre–stored plans.

CADDY and BURIDAN follow the traditional planning models, completing plan construction before executing the plan. On the other hand, POMDPs interleaves planning and execution [1]; they only generate one action to be executed next at each step.

In order to re–use and adapt experiences, CADDY allows external plan representation containing control–structure constructs like *sequence*s, *conditional*s, and *while loop*s. The plan generated by POMDPs can be viewed as a set of test/action rules; POMDPs generates actions to be executed next according to the current circumstance at each step. The plan generated by BURIDAN can be viewed as a sequence of primitive actions. C–BURIDAN also uses conditionals. Loops can be achieved implicitly during the plan–assessment/plan–generation loop, they are unnecessary to be represented in the final plan.

Definition of Successful Planning

The conditions to end plan construction are different. BURIDAN continues plan assessment and plan generation loop until the constructed plan can achieve the goal with probability at least τ, a user–specified probability threshold. POMDPs continues generating next steps until it reaches a state which satisfies the goal conditions. CADDY adapts previous experiences and constructs a plan which can reach its goal efficiently. The issue of efficiency is critical in the golf world testbed, since there are many very costly plans that can still reach the goal with high probability.

BURIDAN and POMDPs are guaranteed to produce a plan which satisfy the goal requirement. CADDY is heuristic; the generated plan may or may not achieve the goal efficiently.

[1] POMDPs pre–computes a whole table of what actions to apply under what circumstances, which can be considered as a pre–computed complete plan.

Efficiency Consideration

BURIDAN does not consider the issue of plan efficiency while CADDY and POMDPs does consider the efficiency of the generated plans.

POMDPs uses reward functions which maps a environmental state and an action to a real number that specify the instantaneous reward the planning agent derives from taking the action in the given state. At each state, POMDPs enumerates all possible actions and all possible outcomes and decides to execute a specific action which narrows the gap between the current state and the goal with possible maximum reward.

CADDY uses cost as a measurement of plan efficiency, which can be viewed as a negative reward. CADDY tries to find a plan with low estimated costs.

6.2 CADDY and Plan Adaptation

We next compare CADDY with several plan adaptation systems. HACKER [Sus75], WOK [Ham83], CHEF [Ham86a, Ham86b, Ham89], PLEXUS [Alt86a, Alt86b, Alt88], GTD paradigm [Sim88a, Sim88b], and CADDY are similar in a number of respects. They are all single agent planning systems; they attempt to use past experiences to overcome current problems encountered; they use a collection of plans to serve as past experiences; and they use heuristics (or critics) to correct or adjust plans if they do not fit current circumstance. Still, several characteristics distinguishes CADDY from the others:

- CADDY deals with an uncertain environment, in which the agent has limited perception and imperfect control when executing primitive actions. HACKER deals with a blocks world with complete knowledge about the environment and perfect control of action execution; while WOK, CHEF, PLEXUS, and GTD paradigm generate a final plan with rather "abstract" actions. None of them deal with problems of knowledge, perception, and control.

- CADDY considers not only the correctness of the generated plans (the plan will indeed lead to the final goal), but also the quality of plans (the plan will lead to the final goal efficiently); while other systems (HACKER, WOK, CHEF, PLEXUS, and GTD paradigm) only deal with the correctness of the generated plan.
- In considering the golf world testbed, CADDY has an unique top–level goal: hitting the golf ball into the hole. Other systems usually encounter different goals during different executions.

Comparing the plan adaptation mechanism, CADDY is more like the case–based planning system. HACKER and GTD paradigm use pattern match techniques to retrieve pre–stored programs and subroutines; adaptive planning employs abstraction/specification of plans as transformation strategies. CADDY and case–based planning can be characterized as memory–based because the organization of the plans can be changed during the planning processes.

Let us consider how CADDY achieves the tasks of processes in CHEF:

Problem Anticipation

The purpose of the Problem Anticipator is to notify features in the current input that have participated in past planning problems. The Problem Anticipator avoids CHEF from making the same fault it has made before. In CADDY, the Plan Adaptation Module uses information in the Failure Table to prevent CADDY from constructing failed plans. The difference is that the Problem Anticipator is executed at the beginning and these features are added as a goal to avoid these problems; while in CADDY, the Failure Table serves as to inform the Plan Adaptation Module that the adapted plan is not a good one.

Plan Retrieval

The Plan Selection Module chooses plan either by consulting information in the Best Table or by choosing the plan with the least cost with the assistance of the Estimation Module.

Plan Modification

The Plan Adaptation Module uses domain–dependent heuristic rules to adapt the plan selected by the Plan Selection Module.

Plan Repair and Credit Assignment

CHEF fixes the faulty plan when a plan fails to achieve its goal, and CHEF also records the causal explanation of why the failure has occurred so that the Problem Anticipator can identify the features in the input that lead to the same problem in future planning executions. CADDY does not try to repair the faulty plan but CADDY records the failure experience in the Failure Table to prevent from making the same plan given the same circumstance.

Since CHEF does not deal with uncertainty and efficiency issues, it can correctly predict whether the constructed plan will succeed, and it only considers the correctness of the constructed plan. While in CADDY, the uncertainty can make a plan be an efficient one in one try but an awful one in the next try. CADDY uses simulation to estimate the effectiveness of the generated plan, but it cannot predict the outcome of an actual execution. This is why we do not repair the generated plan, a plan can be poorly executed although it is projected as a good one.

Plan Storage

The Plan Execution Module stores needed information in the Best Table, in the Failure Table, or in the Simulation Table. The generated plan is also stored in the Plan Library if it outperforms the original selected plan.

6.3 Plan Feasibility Problems

As discussed in section 2.4, plan feasibility has been playing a central concept in the theory of planning. In this section, we first present the preconditions of each primitive action in the golf world testbed and the knowledge preconditions of the control–structure operators in informal terms, then discuss how CADDY solves the plan feasibility problems.

Preconditions Axioms

For all primitive actions, we assumes that CADDY knows the values of the input parameters before executing the action, and knows the values of the output parameters after executing the action.

For (hit X θ Z ψ), the one physical precondition required is that CADDY and the golf ball be at the same position ($(R_x, R_y) = (B_x, B_y)$). No knowledge preconditions are required. The result of executing *hit* is unpredictable by CADDY and is controlled by the underlying testbed (the values of α_1, α_2, β_1, and β_2).

For (look Z ψ) and (scan $PATH$ Z ψ), no physical and knowledge preconditions are needed. The result of executing *look* or *scan* depends on the CADDY's perception capability (the value of λ).

For (move X θ), no physical and knowledge preconditions are required and CADDY executes *move* error–free.

For (replace), no physical preconditions are required. The one knowledge precondition required is that CADDY knows the position that it most recently hits the ball (the value of (H_x, H_y)). CADDY executes *replace* error–free.

The knowledge preconditions of the control–structure operators are adopted from Morgenstern's model [Mor87]. For (sequence P_1 P_2), the knowledge preconditions require that CADDY knows how to execute sub–plan P_1 at the beginning, and knows how to execute sub–plan P_2 after P_1 is complete. For (cond Q P_1 P_2), the knowledge preconditions require that CADDY knows CADDY knows the truth value of Q at the beginning, and knows either how to execute sub–plan P_1 if Q is $TRUE$ or how to execute sub–plan P_2 if Q is $FALSE$. For (while Q P_1), the knowledge preconditions are defined recursively in terms of sequences and conditionals since (while Q P_1) is equivalent to (cond Q (sequence P_1 (while Q P_1)) $\emptyset$).

It seems that the epistemic feasibility problem is rather simple in CADDY, since only *replace* requires knowledge preconditions. The problem with this "local view" is that CADDY does not consider the causal structure (hit, search, then move or replace loops) of the plans. We must extend the "local view" to a "global view" of the whole plan structure. Several conditions must be preserved during plan generation so that the generated plans are feasible for later execution.

S_1 : In order to *hit* the ball, CADDY must *move* to the the same position where the ball resides or *replace* the ball. This requires that every *hit* must be preceded by either a *move* to the position of the ball or a *replace*.

S_2 : In order to *move* to the position of the ball, CADDY must know the position. This requires that a successful *look* or *scan* must have taken place.

S_3 : In order to *replace* the ball, CADDY must know the position where it most recently *hit*s the ball. Condition S_3 can be easily achieved since CADDY can store the required position into its internal knowledge each time it *hit*s the ball.

In this "global view" of CADDY, a plan is epistemically feasible if the following two conditions are satisfied:

Q_1 : When CADDY is about to execute a primitive action, the preconditions of the primitive action is satisfied.

Q_2 : Conditions S_1, S_2, S_3 are preserved during plan execution. Since S_3 can be easily preserved with extra internal storage, we focus on S_1 and S_2.

We now discuss how CADDY preserves the feasibility conditions during plan generation (the execution of the Plan Selection Module followed by the execution of the Plan Adaptation Module).

Solutions to Plan Feasibility Problem

For planning systems which assume complete knowledge, the plan feasibility problem reduces to the physical preconditions problems. The agents always have complete and error–free knowledge of the underlying environment, all knowledge preconditions axioms will be satisfied. Classical planning systems like STRIPS and TWEAK are within this category.

For planning systems which interleave planning and acting, the plan feasibility problem is implicitly solved by the planning stage. At each planning stage, the planning systems only consider those actions which are feasible for the agent. That is, the planning system only considers those actions whose physical preconditions and knowledge preconditions are satisfied at the time the planning systems are about to generate next steps. POMDPs are within this category.

CADDY operates in an unknown environment and has limited perceptions, and CADDY will not take any execution until the plan has been generated. CADDY does not fall into either of the above categories, CADDY must have ways to solve the feasibility problem. Instead of constructing plans from scratch, CADDY adapts experiences. To solve the feasibility problem, CADDY must take care of the following two issues:

- The plans which are pre–store in the Plan Library satisfy the requirements of preconditions.
- The heuristic rules which are used to adapt the selected plans must maintain the requirements of preconditions.

In the simplest execution, CADDY just selects a plan from the Plan Library and executes it. In order to solve the plan feasibility problem, the plans which reside in the Plan Library must satisfy the requirement of physical preconditions as well as knowledge preconditions. The plans initially in the Plan Library are constructed with care so that CADDY is feasible to execute either of these plans if it is selected. That is, condition Q_1 and Q_2 are preserved during the execution of every plan stored in the Plan Library.

The plans stored by the Plan Execution Module are plans modified by the Plan Adaptation Module. After the Plan Selection Module chooses a plan, the Plan Adaptation Module modifies the plan by applying the heuristic rules. These modifications change the internal structure of the plans and have potential to destroy the feasibility of the plans, CADDY should apply these rules with care so that conditions Q_1 and Q_2 are still preserved even the plan structure is altered.

Applying the heuristic rules for adapting *hit* (**H1** and **H2**) only changes the X parameter of the primitive action *hit*, but the internal structure of the plan remains the same. *Hit* does not require knowledge preconditions and the physical preconditions are still satisfied (the adaptation rules do not alter the ball's position or CADDY's position before the *hit*), Q_1 is preserved. Q_2 is also preserved since changing the hitting distance of *hit* does not

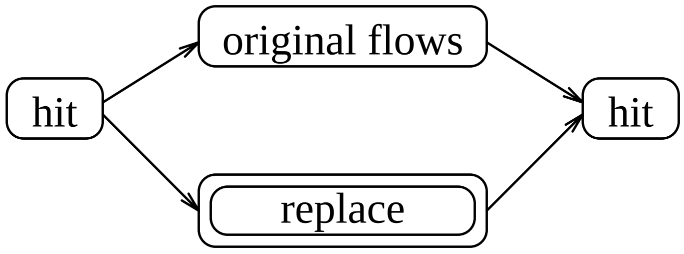

Figure 6.1: Causal Structure Changed after R1, R2, or R3 is applied

alter the causal structure of the plan, S_1 and S_2 are unaffected.

Applying the heuristic rules for adapting *replace* (**R1**, **R2**, and **R3**) changes the internal structure of the plan by adding a *cond* plan segment before searching the ball. The plan segment tests some condition and executes *replace* if the condition is $TRUE$, otherwise resumes the search execution. Action *replace* does not require physical preconditions and the knowledge required can be extracted from the instantaneous situation (the (H_x, H_y) point in each situation); the condition testing does not change anything if the condition is $FALSE$, so the preconditions needed for the first action of the search sub–plan are still satisfied. Q_1 is preserved. The insertion of the plan segment changes the causal structure of the plan to the one shown in figure 6.1. The new flow (*hit–replace–hit*) preserves condition S_1 and S_2 (*hit* is preceded by a *replace*, no *move* is involved in the causal flow), the original flows are known to preserve conditions S_1 and S_2. Q_2 is also preserved.

Applying the heuristic rules for preventing *replace* (**S1**) changes the internal structure by adding a *scan* action before *replace* action to extend the search. Since *scan* does not require physical preconditions and knowledge preconditions, *replace* does not require physical preconditions and the knowledge required can be extracted from the instantaneous situation (the (H_x, H_y) point), Q_1 is preserved. The extra search step added into the plan changes the causal structure of the plan to the one shown in figure 6.2. The added *scan* action can be viewed as s search flows extension, and the global causal structure remains

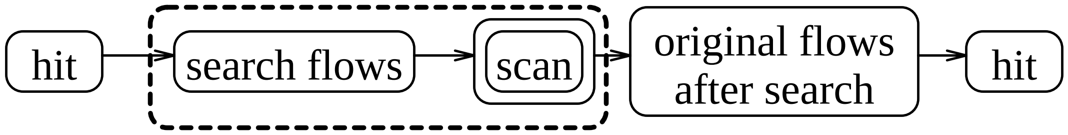

Figure 6.2: Causal Structure Changed after S1 is applied

the same. Q_2 is also preserved.

The plans initially in the Plan Library are feasible to be executed, and the heuristic adaptation rules do not destroy the feasibility of the plans. This concludes that the plan which CADDY is about to execute will not cause any feasibility problem.

Chapter 7

Simulation Results

Currently the CADDY planning system along with the golf world testbed are implemented on a SUN workstation, with a simple graphic interface. All program modules are written in C. The Plan Library initially contains 18 different plans, the Best Table is empty, and the Simulation Table contains the simulated cost results of the 18 plans under different parameter assignments.

The graphic interface enables users to set the value of physical parameters α_1, β_1, λ, and cost parameter *c_replace*; while $\alpha_2 = 0.75 * \alpha_1$ and $\beta_2 = 0.75 * \beta_1$. The golf world testbed currently treats physical parameter ϵ and cost parameters *c_hit*, *c_move*, *c_look*, *c_scan* as constants. We decided against relaxing these to variable parameters in order to keep a reasonable bound on the dimensionality of the search space and hence on the computational demands.

The graphic interface enables users to change parameter settings as well as to initialize the position of CADDY and the golf ball at the starting situation. While the graphic interface will simulate and display the effect of each primitive action performed and the execution of the selected plan, another text window will also display the log for tracing purpose.

Since the performance of several key modules will affect the performance of the whole system, we will first discuss several simulation result of individual modules within the CADDY planning system, then present some simulation result of the whole system in the following sections.

7.1 Simulation Result of the Estimation Module

We show the relation between the simulation cost and those system parameters (λ, α_1, β_1, *c_replace*, and the distance between the golf ball and the hole at the starting situation *dist*), then compare the result of the Estimation Module from the averaged simulation cost. Figure 7.1 through figure 7.5 show the relation between the simulated cost of a plan and the system parameters. For each selected parameter settings, four values are displayed.

- **x** indicates the estimated cost EC of the plan, as generated by the Estimation Module.
- **o** indicates the simulated cost SC of the plan, as generated by averaging the cost of hundreds of simulations of the plan under the selected parameter settings.
- Two $+$'s indicates the variance of the simulation cost derivation. These two $+$'s mark the cost value of $(SC - \sigma)$ and $(SC + \sigma)$ where σ is the standard derivation.

The four plans used (**a3**, **b2**, **c3**, and **e1**) are enumerated in figures 7.6 and 7.7.

Figure 7.1 shows the relation between cost parameter *c_replace* and the cost of the chosen plan with (λ, α_1, β_1, *dist*) bound to (20.00, 24.00, 0.24, 300.00). EC is close to SC. The cost increases while *c_replace* increases except for simulation result of **e1**, in which it seems that the cost and *c_replace* are independent. The reason may be that **e1** does not immediately replace the golf ball until a long $PATH$ of *scan*, while other plans give up search either after a *look* or after a short $PATH$ of *scan*.

Figure 7.1: Relation between Plan Cost and Cost Parameter c_replace.

Figure 7.2: Relation between Plan Cost and Physical Parameter alpha.

Figure 7.3: Relation between Plan Cost and Physical Parameter beta.

Figure 7.4: Relation between Plan Cost and Physical Parameter lambda.

Figure 7.5: Relation between Plan Cost and Starting Situation distance.

Figure 7.2 shows the relation between physical parameter α_1 and the cost of the chosen plan with ($c\_replace$, λ, β_1, $dist$) bound to (100.00, 30.00, 0.08, 400.00). The difference between EC and SC is small. When α_1 increases, both the cost and variation increase.

Figure 7.3 shows the relation between physical parameter β_1 and the cost of the chosen plan with ($c\_replace$, λ, α_1, $dist$) bound to (100.00, 30.00, 8.00, 400.00). EC is usually larger than SC but within $SC+\sigma$. When β_1 increases, both the cost and variation increase.

α_1 and α_2 determine the directional accuracy of CADDY's *hit* action, and β_1 and β_2 determine the distance accuracy of CADDY's *hit* action. Larger αs or βs means a larger area of points the ball may land (or points CADDY assumes the ball are landing) after CADDY *hit*s the ball. Increasing either value not only potentially increases the *hit* error, which increases the total cost either by increasing the cost of searching the golf ball, by increasing the times of replacing the ball, or by both; but also increases the variation of the total cost due to the increase of variation of cost of *hit* action. The large variation in the result of $c\_replace$ simulation (figure 7.1) is explained in the same way.

Figure 7.4 shows the relation between physical parameter λ and the cost of the chosen plan with ($c\_replace$, α_1, β_1, $dist$) bound to (100.00, 8.00, 0.08, 400.00). EC stays close with SC, and with larger λ, EC is almost equal to SC. Cost decreases when λ increases under current parameter settings.

Figure 7.5 shows the relation between the initial situation $dist$ and the cost of the chosen plan with ($c\_replace$, λ, α_1, β_1) bound to (100.00, 30.00, 8.00, 0.08). EC is usually larger than SC but stays close to SC. Cost increases when $dist$ increases under current parameter settings.

Table 7.1 gives the comparison of EC and SC for 3750 trials of randomly chosen parameter assignments. The difference percentage $\frac{|SC-EC|}{SC}$ may sometimes be rather large (for example, over 30%), but EC is almost always within the range ($SC-\sigma$, $SC+\sigma$). The

$d = \frac{\lvert SC-EC \rvert}{SC}$					
$d \le 1\%$	$1\% < d \le 5\%$	$5\% < d \le 10\%$	$10\% < d \le 20\%$	$20\% < d \le 30\%$	$d > 30\%$
236 6.29%	931 24.83%	998 26.61%	1256 33.49%	257 6.58%	72 1.92%

$SC - \sigma \le EC \le SC + \sigma$	Otherwise
3747 99.92%	3 0.08%

Table 7.1: Simulation Result of the Estimation Module.

λ	10.00	50.00	100.00	10.00	50.00	100.00	10.00	50.00	100.00
α_1	0.00	0.00	0.00	20.00	20.00	20.00	40.00	40.00	40.00
β_1	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Plan Selected	a3	a3	a3	e3	a3	d3	e1	e2	d3
λ	10.00	50.00	100.00	10.00	50.00	100.00	10.00	50.00	100.00
α_1	0.00	0.00	0.00	20.00	20.00	20.00	40.00	40.00	40.00
β_1	0.20	0.20	0.20	0.20	0.20	0.20	0.20	0.20	0.20
Plan Selected	d3	d3	d3	e2	e3	d3	e1	e2	b2
λ	10.00	50.00	100.00	10.00	50.00	100.00	10.00	50.00	100.00
α_1	0.00	0.00	0.00	20.00	20.00	20.00	40.00	40.00	40.00
β_1	0.40	0.40	0.40	0.40	0.40	0.40	0.40	0.40	0.40
Plan Selected	d3	d3	d3	e1	e3	d3	e1	e2	c2

Table 7.2: Plans Selected under Different Parameter Assignments

performance of the Estimation Module is thus applicable judging from above simulation result.

7.2 Simulation Result of the Plan Selection Module

Two simulation results are demonstrated here:

- The Plan Selection Module selects different plans for different parameter assignments.
- For several input starting situations, the Plan Selection Module usually selects the actual optimal plans or near–optimal plans within the Plan Library.

Plan **a3**

```
(sequence (cond (ge (distance agent hole) 120)
                    (hit 120 (direction agent hole) Dist1 Dir1)
                    (hit (distance agent hole) (direction agent hole) Dist1 Dir1))
          (while (not ball_in) (sequence
                    (move Dist1 Dir1)
                    (look Dist2 Dir2)
                    (cond (ge Dist2 0.0) (move Dist2 Dir2) (replace))
                    (cond (ge (distance agent hole) 120)
                          (hit 120 (direction agent hole) Dist1 Dir1)
                          (hit (distance agent hole) (direction agent hole) Dist1 Dir1)))))
```

Plan **b2**

```
(sequence (cond (ge (distance agent hole) 60)
                    (hit 60 (direction agent hole) Dist1 Dir1)
                    (hit (distance agent hole) (direction agent hole) Dist1 Dir1))
          (while (not ball_in) (sequence
                    (look Dist2 Dir2)
                    (cond (ge Dist2 0.0) (move Dist2 Dir2) (sequence
                          (move Dist1 Dir1)
                          (look Dist2 Dir2)
                          (cond (ge Dist2 0.0) (move Dist2 Dir2) (replace))))
                    (cond (ge (distance agent hole) 60)
                          (hit 60 (direction agent hole) Dist1 Dir1)
                          (hit (distance agent hole) (direction agent hole) Dist1 Dir1)))))
```

Plan **c3**

```
(sequence (cond (ge (distance agent hole) 120)
                    (hit 120 (direction agent hole) Dist1 Dir1)
                    (hit (distance agent hole) (direction agent hole) Dist1 Dir1))
          (while (not ball_in) (sequence
                    (move (/ Dist1 2) Dir1)
                    (scan ((Dist1 Dir1)) Dist2 Dir2)
                    (cond (ge Dist2 0.0) (move Dist2 Dir2) (replace))))
                    (cond (ge (distance agent hole) 120)
                          (hit 120 (direction agent hole) Dist1 Dir1)
                          (hit (distance agent hole) (direction agent hole) Dist1 Dir1)))))
```

Figure 7.6: Some Example Plans Selected by Simulation of Plan Selection

Plan **e1** (**e2** and **e3** are similar, with larger hitting distance)

```
    (sequence (cond (ge (distance agent hole) 30)
                    (hit 30 (direction agent hole) Dist1 Dir1)
                    (hit (distance agent hole) (direction agent hole) Dist1 Dir1))
              (while (not ball_in) (sequence
                    (move (- Dist1 lambda) Dir1)
                    (scan (((* (sqrt 2) lambda) (+ Dir1 45))
                           ((* (sqrt 2) lambda) (- Dir1 45))
                           ((* 2 (* (sqrt 2) lambda)) (+ Dir1 225))
                           ((* 2 (* (sqrt 2) lambda)) (+ Dir1 135))
                           ((* 3 (* (sqrt 2) lambda)) (+ Dir1 45))
                           ((* 3 (* (sqrt 2) lambda)) (- Dir1 45))
                           ((* 4 (* (sqrt 2) lambda)) (+ Dir1 225))
                           ((* 4 (* (sqrt 2) lambda)) (+ Dir1 135))
                           ((* 5 (* (sqrt 2) lambda)) (+ Dir1 45))
                           ((* 5 (* (sqrt 2) lambda)) (- Dir1 45)))
                           Dist2 Dir2)
                    (cond (ge Dist2 0.0) (move Dist2 Dir2) (replace))))
                    (cond (ge (distance agent hole) 30)
                           (hit 30 (direction agent hole) Dist1 Dir1)
                           (hit (distance agent hole) (direction agent hole) Dist1 Dir1)))))
```

Figure 7.7: Some Example Plans Selected by Simulation of Plan Selection (continued)

Table 7.2 shows that, with the same starting situation which CADDY and the golf ball are at $(200.00, 200.00)$ and fixed cost parameter *c_replace*, different plans are selected by the Plan Selection Module for different settings of physical parameters λ, α_1, and β_1. Some plans selected by this simulation are enumerated in figure 7.6 and figure 7.7. The major difference between these plans is the search strategy. The intended hitting distance of these plans are also different.

- **a3** moves to the position returned by *hit* action then looks around to locate the golf ball. If CADDY finds the ball, moves towards the ball and re–hits again; otherwise gives up search immediately and replaces the ball for next hit.

- **b2** differs from **a3** in one respect. **b2** looks around to locate the ball right after CADDY *hit*s the ball. **b2** can locate the ball earlier if either the hitting distance is short or λ (the perception distance) is fairly large.

- **c3** *scan*s a short straight line through the direction returned by *hit*. If α_1 is rather small while β_1 is large, **c3** has a great chance locating the ball through the scan path.

- **e1** looks for the golf ball through a spiral–like path centered at the position returned by a *hit* action. **e1** should execute the replacement rarely due to the long search path and the large search area. Figure 7.8 shows the *scan* path after the *move*.

The Plan Selection Module selects plans either (1) by remembering a near–perfect match of experiences or (2) by selecting a plan with the least estimated cost (refer to section 5.3 for more detail). Case (1) is simple: The plan for near–matched experience has been constructed by previous execution of the Plan Selection Module and the Plan Adaptation Module, and has been stored in the database marked as optimal (or near optimal, depends on the performance of the Plan Selection Module at that time) for that starting situation.

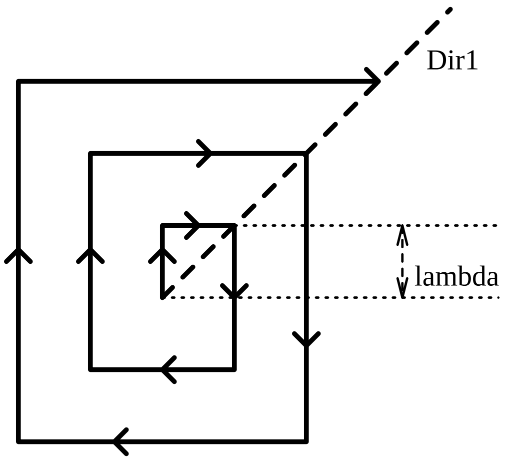

Figure 7.8: The scan PATH of Plan ‘e1’.

$P_1 = P_2$	$P_1 \neq P_2$			
653 (63.77 %)	371 (36.23 %)			
	$\delta \leq 1\%$	$1\% < \delta \leq 5\%$	$5\% < \delta \leq 10\%$	$\delta > 10\%$
	102 (9.96 %)	218 (21.29 %)	44 (4.30 %)	7 (0.68 %)

Table 7.3: Simulation Result of the Plan Selection Module.

We next consider the performance of the Plan Selection Module for case (2). In order to demonstrate that the Plan Selection Module is effective in selecting the expected best plan in the Plan Library for the current starting situation, we compare the result of the Plan Selection Module and the actual optimal plan by the following simulation procedure. Given different randomly selected parameter settings, the simulation procedure first evokes the Plan Selection Module to select a plan P_1 with estimated cost C_1, then simulates each plan within Plan Library under the given parameter settings and chooses P_2 with the smallest average cost C_2. If P_1 is different from P_2, the simulation procedure also compares the cost difference by calculating the difference percentage value $\delta = \frac{|C_1 - C_2|}{C_2}$. Table 7.3 shows the simulation result after 1024 random trials.

The simulation result shows that the percentage of the Plan Selection Module selecting the right plan (P_1 is equal to P_2) is only 63.77%. While more than one third times the Plan Selection Module does not select the optimal plan, most of the time the difference (δ) is rather small. We can see that only 0.68% times the difference is rather large ($\delta > 10\%$) while 73.73% times the Plan Selection Module either selects the right plan or the difference is very small ($\delta \leq 1\%$); if we increase the error percentage to 5%, 95.02% times the Plan Selection Module's performance satisfies the request. This shows that the Plan Selection Module satisfies, to some extent, the requirement of selecting the optimal or near–optimal plan from the Plan Library.

7.3 Simulation Result of the Plan Adaptation Module

We have demonstrated in section 5.4 how the Plan Adaptation Module can apply heuristic rules to adapt the plan selected by the Plan Selection Module. Since the resulting plan adapted by the Plan Adaptation Module cannot be guaranteed to be better than the original plan selected by the Plan Selection Module (though we hope this always is the case), we just present the following simulation result.

Of 256 random trials, the Plan Adaptation Module applies heuristic rules to adapt selected plans 192 times while remains selected plans unchanged 64 times. Of those 192 trials, P_2, the adapted plan generated by the Plan Adaptation Module, outperforms P_1, the original plan selected by the Plan Selection Module, 126 times while P_1 outperforms P_2 66 times. This also demonstrates that these heuristic repair rules do not always work.

7.4 Simulation Result of the Whole System

Demonstrated below are several typical simulation results we got from the current system.

With ($c\_replace$, λ, α_1, β_1) = (120.00, 35.00, 3.00, 0.03) and initial position (200.0, 200.0), the Plan Selection Module chooses plan **a3** with estimated cost 714.16, and the Plan Adaptation Module applies heuristic rules **H1** and **R3** to plan **a3**. After 256 simulations, the average costs of both plans are 752.66, 670.84; with 3.96, 3.28 *hit* operations and 0.00, 0.00 *replace* operations. Actual simulation results show that after adapting the selected plan, CADDY gets a better plan with less average cost and less *hit* operations. The newly generated plan **p0** is stored in the Plan Library; the record (**p0**, $c\_replace$ = 120.00, λ = 35.00, α_1 = 3.00, β_1 = 0.03, *dist* = 282.00, *cost* = 670.84) is recorded in the Best Table; and the Plan Execution Module evokes simulations of the plan **p0** under selected circumstances and records the result in the Simulation Table.

With ($c\_replace$, λ, α_1, β_1) = (54.00, 26.00, 28.00, 0.24) and initial position (200.0, 200.0), the Plan Selection Module chooses plan **e3** with estimated cost 1526.65, and the Plan Adaptation Module applies heuristic rules **H2** and **R1** to plan **e3**. After 256 simulations, the average costs of both plans are 2074.67, 1934.58; with 8.26, 9.02 *hit* operations and 0.04, 1.48 *replace* operations. Though with higher number of *hit* and *replace* operations, the adapted plan shows a less average cost because it now spends less on *scan*ing those bad *hit*s. Again, simulation results show that applying heuristics makes CADDY get a better plan with lower average cost. Again, the newly generated plan **p1** is stored in the Plan Library; the record (**p1**, $c\_replace$ = 54.00, λ = 26.00, α_1 = 28.00, β_1 = 0.24, *dist* = 282.00, *cost* = 1934.58) is recorded in the Best Table; and the Plan Execution Module evokes simulations of the plan **p1** under selected circumstances and records the result in the Simulation Table.

With ($c\_replace$, λ, α_1, β_1) = (100.00, 56.00, 20.00, 0.20) and initial position (200.0, 200.0), the Plan Selection Module chooses plan **e3** with estimated cost 1259.75, and the Plan Adaptation Module applies heuristic rules **H2** and **R1** to plan **e3**. After 256 simulations, the average costs of both plans are 1359.77, 1420.30; with 6.69, 7.07 *hit* operations and 0.00, 0.45 *replace* operations. This time, the adapted plan is worse than the original plan with higher cost. The newly generated plan is then discarded by the Plan Execution Module, the adaptation experience (**e3**, $\lambda = 56.00$, $\alpha_1 = 20.00$, $\beta_1 = 0.20$, $c\_replace = 100.00$, $dist = 282.00$, {**H2**, **R1**}) is recorded in the Failure Table, and the record (**e3**, $c\_replace = 100.00$, $\lambda = 56.00$, $\alpha_1 = 20.00$, $\beta_1 = 0.20$, $dist = 282.00$, $cost = 1359.77$) is recorded in the Best Table.

With ($c\_replace$, λ, α_1, β_1) = (54.00, 26.00, 28.00, 0.24) and initial position (200.0, 200.0), the Plan Selection Module chooses plan **p1** with estimated cost 1934.58 since CADDY remembers the similar situation has been inputed before (**p1** was generated from **e3** by the Plan Adaptation Module with the same input starting situation). the Plan Adaptation Module does not apply any heuristic repair rules to the selected plan, and the Plan Execution Module shows similar simulation result.

With ($c\_replace$, λ, α_1, β_1) = (100.00, 56.00, 20.00, 0.20) and initial position (200.0, 200.0), the Plan Selection Module chooses plan **e3** with estimated cost 1259.75, and the Plan Adaptation Module intends to apply heuristic rules **H2** and **R1** to plan **e3** but finds the failed experience in the Failure Table, so the Plan Adaptation Module decides to apply only rule **H2** only. After 256 simulations, the average costs of both plans are 1359.77, 1398.16; with 6.69, 6.78 *hit* operations and 0.00, 0.00 *replace* operations. This time, the adapted plan is still worse than the original plan with higher cost. The newly generated plan is then discarded by the Plan Execution Module, the adaptation experience (**e3**, $\lambda = 56.00$, $\alpha_1 = 20.00$, $\beta_1 = 0.20$, $c\_replace = 100.00$, $dist = 282.00$, {**H2**}) is recorded in the Failure Table, and the record (**e3**, $c\_replace = 100.00$, $\lambda = 56.00$, $\alpha_1 = 20.00$, $\beta_1 = 0.20$, $dist = 282.00$, $cost = 1359.77$) is updated in the Best Table. Though the adapted plan is still outperformed by the original selected plan, the simulated cost this time (1398.16) is slightly better than the simulated cost last time (1420.30).

7.5 Discussion of Simulation Result

We have shown the following simulation results to demonstrate the performance of the CADDY planning system under the golf world testbed.

- We have studied the relation between plan cost and each of the system parameters. Though we only demonstrate the relation with specific fixed settings of parameters and a small set of plans, we believe the relation should be similar under other parameter assignments and other plans.

- The performance of the Estimation Module and the Plan Selection Module is adequate to our purposes.

- Heuristic adaptation rules do not always improve the quality of plans. The Plan Adaptation Module may or may not adapt the input plans, and the adapted plans may not always outperform the original ones.

- The performance of the integrated system is satisfactory since the Estimation Module can get satisfactory estimate cost of plans; the Plan Selection Module can choose a near–optimal plan from the Plan Library for current parameter settings and the starting situation; the Plan Adaptation Module can apply heuristic repair rules to adjust the selected plan; and the Plan Execution Module can learn experiences by storing the better quality plans and simulation result back to the Plan Library.

Though the current system is satisfactory, it is far from perfect. We will discuss some directions for improvement in section 8.1.

Chapter 8

Extension and Generalization

In this chapter, we will first point out directions where the performance of the CADDY planning system can be improved, then discuss how to generalize the ideas we get from CADDY to other problem domains.

8.1 System Extension

Although the current system gives us some useful and promising results, CADDY is far from perfect. A couple of directions for system improvement are addressed here:

- Improve the run–time performance.

  The current prototype is implemented using several naive and straightforward methods, which simplify the implementation at the cost of efficiency. Chapter 5 has pointed out several directions of implementation improvement, which will be addressed in detail within this section.

- Extend the testbed complexity.

  The current golf world testbed is, to some extent, too simplistic to be considered as a real–world problem. Several extension concerns are also addressed later to complicate the testbed so that the gap between real–world problems and the testbed is narrowed.

- Explore more heuristic repair rules.

  The current system only includes some simple heuristic adaptation rules. More heuristics should be explored, especially when the underlying testbed is extended.

8.1.1 Run–Time Performance Improvement

As we have pointed out in chapter 5 and chapter 7, while CADDY is tested in the golf world testbed, work can be done to improve the run–time performance, both the accuracy (the interpolations) and the efficiency. We will present some directions of improvement here.

Plan Library

The performance of the K–D tree (the Best Table and the Failure Table) and the 5–dimension linked list structure (the Simulation Table) are fair. The biggest problem that can affect the performance of CADDY is the increasing size of the Simulation Table. As more plans being stored in the Plan Table, the size of the Simulation Table increases dramatically. One way to control the size of the Simulation Table is to periodically check all plans in the Plan Library and discard plans which are outperformed by others under selected circumstances. Another way is to replace these discrete entries in the Simulation Table by functions of the plans and system parameters. This will needs a comprehensive numerical analysis of the simulation result to get the cost estimation function of each plan in the Plan Library.

Estimation Module and Plan Selection Module

We can see, from the simulation result in section 7.1, that the estimated cost and the simulated cost are not always compatible. For example, the Estimation Module usually over–estimates the cost when β_1 is large (see figure 7.3). One way to improve the accuracy of estimation is to increase the reference points in the Simulation Table so that the linear interpolation can choose points closer to the current setting, but this will increase the size of Simulation Table and then delay the search of two nearest points. Another possible improvement is to use other interpolation methods or other simulation techniques.

Controlling the size of the Plan Library will also improve the performance of the Plan Selection Module. Remember that the Plan Selection Module needs to perform interpolations of each plan stored in the Plan Library, removing plans which are outperformed by others can reduce the number of unnecessary interpolations.

Plan Adaptation Module

We should explore more heuristic rules. Especially when the underlying testbed is extended, more domain–dependent heuristics should be easily included.

Parallel Execution

Two major performance bottlenecks are encountered during the execution of the CADDY planning system:

1. Searching for a new plan from the Plan Library. The Plan Selection Module uses the Estimation Module to interpolate every plan within the Plan Library and chooses the one with least cost. As we have pointed out, the interpolation calculation in the Estimation Module is a computation intensive process, so the sequence of interpolations performed by the Plan Selection Module can be expected to be time consuming.

2. Storing generated plans and simulation results in the Plan Library. The Plan Execution Module needs simulations of the adapted plan under selected circumstances and records the simulation result in the Simulation Table. We have pointed out that the

Plan Execution Module needs hundreds of simulations in order to get the average cost of a plan under one specific circumstance. A large number of simulated executions of a plan is also expected to be time consuming.

Distributing these computations to multiple processors will speed up these bottlenecks. The Plan Selection Module can be modified to use parallel execution to speed up its performance, with several machines running the Estimation Module independently to calculate interpolations of different plans. The Plan Execution Module can be modified to use parallel execution to speed up its performance, with several machines independently simulating the plan under different circumstances then outputting these result to the Simulation Table. Putting these modules in parallel execution will help improving the run–time performance of CADDY.

8.1.2 Testbed Extension

We have pointed out in section 4.3 that, while the golf world testbed, like the blocks world, has the advantages of simplicity, it also raises interesting problems of imperfect perception, control, and knowledge. However, the golf world testbed ignores several issues which would be encountered in a real world golf course. Extending the golf world testbed for flexibility will narrow the gap. The extension of the golf world may be of no practical interest, but the discussion of the extension can help us to understand more about testbed designing, which can then help us to design other testbeds when we generalize CADDY to other domains.

One obvious way of extending the testbed is to include more system parameters. For example, considering the wind factor and the golf field factor will certain complicate the testbed. The wind factor will affect the hitting distance and direction, which then affects CADDY to decide what distance/direction to hit. The field factor (the obstacles will block CADDY's perception, CADDY should avoid hitting the ball into the pond or the sand ground, etc) also will complicate the underlying environment, which then affects CADDY's decision of plan actions.

To include more primitive actions which CADDY can execute is also a possible way of extending the testbed. For example, we can extend the *hit* operation with one more parameter *club*, which represents which golf club CADDY uses to hit the ball. Different clubs can have different hitting abilities (different hitting distance, different accuracy, one club may fit at one kind of field while another may not, etc), which surely affects CADDY's decision on which club to use under different circumstances. Another example is that we can extend *look* operation so that CADDY has different accuracy of perception under different distances. Allowing CADDY have partial perception accuracy increases the information CADDY can gather (the current *look* model is a kind of all–or–nothing, either CADDY knows the exact location of the ball or CADDY gains nothing), but also complicates the degree of uncertainty.

The use of uncertainty model gives another potential extension of the testbed. The uncertainty model currently used is a normal probabilistic distribution with fixed parameters, the values of the physical parameters and the cost parameters are unchanged during plan execution. Allowing parameters changes values during plan execution and the use of different probabilistic distribution models increase the complexity of the underlying testbed. This variation can either follow pre–defined functions or be obtained from meta–level probabilistic models.

8.2 Generalization

Generalization of CADDY to other domains will be our main concern. We have already pointed out that the golf world is a rather simplified testbed with a build–in set of parameters which characterize the behavior of the tested world and simple uncertainty model. We will consider how the insights gained from the study of CADDY can be extended to other richer domains.

Each new domain requires a specification of the testbed, used for simulation; the planning language; the cost function on plan execution; the estimation techniques to be used by the Estimation Module; and the heuristic adaptation rules used by the Plan Adaptation Module.

Bibliography

[AHT90] James F. Allen, James Hendler, and Austin Tate. *Readings in Planning.* Morgan Kaufmann Publishers, Inc., San Mateo, California, U.S.A., 1990.

[Alt86a] Richard Alterman. Adaptive Planning : a Case of Flexible Knowing. Technical report, University of California at Berkeley, 1986.

[Alt86b] Richard Alterman. An Adaptive Planner. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 65–69, Philadelphia, Pennsylvania, U.S.A., August 11 – 15 1986. American Association for Artificial Intelligence.

[Alt88] Richard Alterman. Adaptive Planning. *Cognitive Science*, 12:393–421, 1988.

[BBK94] Sugato Bagchi, Gautam Biswas, and Kazuhiko Kawamura. Generating Plans to Succeed in Uncertain Environments. In Kristian J. Hammond, editor, *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems*, pages 1–6, Chicago, Illinois, U.S.A., June 13 – 15 1994.

[BW94a] Anthony Barret and Daniel S. Weld. Partial–Order Planning : Evaluating Possible Efficiency Gains. *Artificial Intelligence*, 67(1):71–112, 1994.

[BW94b] Anthony Barret and Daniel S. Weld. Schema Parsing : Hierarchical Planning of Expressive Languages. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, Seattle, Washington, U.S.A., July 31 – August 4 1994. American Association for Artificial Intelligence.

[Cha87] David Chapman. Planning for Conjunctive Goals. *Artificial Intelligence*, 32(3):333–377, 1987.

[Chr92] L. Chrisman. Reinforcement Learning with Perceptual Aliasing : The Perceptual Distinctions Approach. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 183–188, San Jose, California, U.S.A., July 12 – 16 1992. American Association for Artificial Intelligence.

[CKL94a] Anthony R. Cassandra, Leslie Pack Kaelbling, and Michael L. Littman. Acting Optimally in Partially Observable Stochastic Domains. In *Proceedings of the*

Twelfth National Conference on Artificial Intelligence, Seattle, Washington, U.S.A., July 31 – August 4 1994. American Association for Artificial Intelligence.

[CKL94b] Anthony R. Cassandra, Leslie Pack Kaelbling, and Michael L. Littman. Algorithms for Partially Observable Markov Decision Processes. Technical Report CS–94–14, Department of Computer Science, Brown University, 1994.

[Dan84] L. Daniel. Planning and Operations Research. In Tim O'Shea and Marc Eisenstadt, editors, *Artificial Intelligence : Tools, Techniques, and Applications*. Harper and Row, New York, New York, U.S.A., 1984.

[Dav90] Ernest S. Davis. *Representations of Commonsense Knowledge*. Morgan Kaufmann Publishers, Inc., San Mateo, California, U.S.A., 1990.

[Dav94] Ernest S. Davis. Branching Continuous Time and the Semantics of Continuous Action. In Kristian J. Hammond, editor, *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems*, pages 231–236, Chicago, Illinois, U.S.A., June 13 – 15 1994. Poster.

[Davng] Ernest S. Davis. Knowledge Preconditions for Plans. *Journal of Logic and Computation*, Forthcoming.

[DFM89] Thomas L. Dean, R. James Firby, and David Miller. Hierarchical Planning Involving Deadline, Travel Times, and Resources. *Computational Intelligence*, 4(4):381–398, 1989.

[DH93] Marek J. Druzdzel and Max Henrion. Efficient Reasoning in Qualitative Probabilistic Networks. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 548–553, Washington D.C., U.S.A., July 11 – 15 1993. American Association for Artificial Intelligence.

[DHW94] Denise Draper, Steve Hanks, and Daniel S. Weld. Probabilistic Planning with Information Gathering and Contingent Execution. In Kristian J. Hammond, editor, *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems*, pages 31–36, Chicago, Illinois, U.S.A., June 13 – 15 1994.

[DK87] Thomas L. Dean and Keiji Kanazawa. Persistence and Probabilistic Inference. Technical Report CS–87–23, Department of Computer Science, Brown University, 1987.

[DK88] Thomas L. Dean and Keiji Kanazawa. Probabilistic Temporal Reasoning. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 524–528, St. Paul, Minnesota, U.S.A., August 21 – 26 1988. American Association for Artificial Intelligence.

[DKKN93] Thomas L. Dean, Leslie Pack Kaelbling, Jack Kirman, and Ann Nicholson. Planning with Deadlines in Stochastic Domains. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 574–579, Washington D.C., U.S.A., July 11 – 15 1993. American Association for Artificial Intelligence.

[DM87] Thomas L. Dean and Drew V. McDermott. Temporal Data Base Management. *Artificial Intelligence*, 32:1–55, 1987.

[Dru93] Marek J. Druzdzel. *Probabilistic Reasoning in Decision Support Systems : from Computation to Common Sense*. PhD thesis, Department of Engineering and Public Policy, Carnegie Mellon University, 1993.

[FBF77] J. H. Friedman, J. L. Bentley, and R. A. Finkel. An Algorithm for Finding Best Matches in Logarithmic Expected Time. *ACM Transactions on Mathematical Software*, 3:209–226, 1977.

[FN71] Richard E. Fikes and Nils J. Nilsson. STRIPS : A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence*, 2(3):189–208, 1971.

[Ham83] Kristian J. Hammond. Planning and Goal Interaction: The Use of Past Solutions in Present Situations. In *Proceedings of the Third National Conference on Artificial Intelligence*, pages 148–151, Washington D.C., U.S.A., August 22 – 26 1983. American Association for Artificial Intelligence.

[Ham86a] Kristian J. Hammond. CHEF : A Model of Case–based Planning. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 267–271, Philadelphia, Pennsylvania, U.S.A., August 11 – 15 1986. American Association for Artificial Intelligence.

[Ham86b] Kristian J. Hammond. Learning to Anticipate and Avoid Planning Problems through the Explanation of Failures. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 556–560, Philadelphia, Pennsylvania, U.S.A., August 11 – 15 1986. American Association for Artificial Intelligence.

[Ham89] Kristian J. Hammond. *Case–Based Planning : Viewing Planning as a Memory Task*. Academic Press, Inc., San Diego, California, U.S.A., 1989.

[Ham94] Kristian J. Hammond, editor. *Proceedings The Second International Conference on Artificial Intelligence Planning Systems*, Chicago, Illinois, U.S.A., June 13 – 15 1994. American Association for Artificial Intelligence.

[Han90a] Steve Hanks. Practical Temporal Projection. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 158–163, Boston, Massachusetts, U.S.A., July 29 – August 3 1990. American Association for Artificial Intelligence.

[Han90b] Steve Hanks. *Projecting Plans about Uncertain Worlds.* PhD thesis, Computer Science Department, Yale University, January 1990.

[Hay75] Philip J. Hayes. A Representation for Robot Plans. In *Advance Papers of the Fourth Annual International Joint Conference on Artificial Intelligence*, pages 181–188, Tbilisi, Georgia, USSR, September 3 – 8 1975.

[HD91] Max Henrion and Marek J. Druzdzel. Qualitative Propagation and Scenario–Based Approaches to Explanation of Probabilistic Reasoning. In Piero P. Bonissone, Max Henrion, L. N. Kanal, and J. F. Lemmer, editors, *Uncertainty in Artificial Intelligence 6*, pages 17–32. North–Holland:Elsevier, 1991.

[Hen92] James Hendler, editor. *Proceedings The First International Conference on Artificial Intelligence Planning Systems*, College Park, Maryland, U.S.A., June 15 – 17 1992. American Association for Artificial Intelligence.

[How60] Ronald A. Howard. *Dynamic Programming and Markov Processes.* The MIT Publisher, Cambridge, Massachusetts, U.S.A., 1960.

[HPC93] Steve Hanks, Martha Pollack, and Paul Cohen. Benchmarks, Testbeds, Controlled Experimentation, and the Design of Agent Architectures. Technical Report 93–06–05, Department of Computer Science and Engineering, University of Washington, June 1993.

[Kam89] Subbarao Kambhampati. *Flexible Reuse and Modification in Hierarchical Planning : a Validation Structure Based Approach.* PhD thesis, Department of Computer Science, University of Maryland at College Park, 1989.

[KD89] Keiji Kanazawa and Thomas L. Dean. A Model for Projection and Action. In *Proceedings of the Eleventh Annual International Joint Conference on Artificial Intelligence*, pages 985–990, Detroit, Michigan, U.S.A., August 20 – 25 1989.

[KH89] Subbarao Kambhampati and James Hendler. Flexible Reuse of Plans via Annotation and Verification. In *Proceedings of the Fifth IEEE Conference on Applications of Artificial Intelligence*, Miami, Florida, U.S.A., 1989.

[KH92] Subbarao Kambhampati and James Hendler. A Validation–Structure–Based Theory of Plan Modification and Reuse. *Artificial Intelligence*, 55:193–258, 1992.

[KHW93] Nicholas Kushmerick, Steve Hanks, and Daniel S. Weld. An Algorithm for Probabilistic Planning. Technical report, Department of Computer Science, University of Washington, January 1993.

[KHW94] Nicholas Kushmerick, Steve Hanks, and Daniel S. Weld. An Algorithm for Probabilistic Least–Commitment Planning. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, Seattle, Washington, U.S.A., July 31 – August 4 1994. American Association for Artificial Intelligence.

[Lov91] W. S. Lovejoy. A Survey of Algorithmic Methods for Partially Observed Markov Decision Processes. *Annals of Operations Research*, 28(1):47–65, 1991.

[McC80] John McCarthy. Circumscription – A Form of Nonmonotonic Reasoning. *Artificial Intelligence*, 13(1,2):27–39, 1980.

[McC86] John McCarthy. Applications of Circumscription to Formalizing Common–Sense Knowledge. *Artificial Intelligence*, 28(1):89–116, 1986.

[McC93] R. A. McCallum. Overcoming incomplete Perception with Utile Distinction Memory. In *Proceedings of the Tenth International Conference on Machine Learning*, Amherst, Massachusetts, U.S.A., 1993. Morgan Kaufmann.

[McD78] Drew V. McDermott. Planning and Acting. *Cognitive Science*, 2(2):71–109, 1978.

[McD82] Drew V. McDermott. A Temporal Logic for Reasoning about Processes and Plans. *Cognitive Science*, 6(2):101–155, 1982.

[McD85] Drew V. McDermott. Reasoning about Plans. In Jerry R. Hobbs and Robert C. Moore, editors, *Formal Theories of the Commonsense World*, pages 269–317. ABLEX Publishing Corp., Norwood, New Jersey, U.S.A., 1985.

[McD91] Drew V. McDermott. A Reactive Plan Language. Technical Report Yale Research Report 864, Computer Science Department, Yale University., August 1991.

[MH69] John McCarthy and Patrick J. Hayes. Some Philosophical Problems from the Standpoint of Artificial Intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence 4*, pages 463–502. Edinburgh University Press., Edinburgh, United Kingdom, 1969.

[Mon82] G. E. Monahan. A Survey of Partially Observable Markov Decision Processes : Theory, Models, and Algorithms. *Management Science*, 28(1):1–16, 1982.

[Moo80] Robert C. Moore. Reasoning about Knowledge and Action. Technical Report 191, SRI International, Menlo Park, California, U.S.A., 1980.

[Moo85] Robert C. Moore. A Formal Theory of Knowledge and Action. In Jerry Hobbs and Robert C. Moore, editors, *Formal Theories of the Commonsense World*, pages 319–358. ABLEX Publishing, Norwood, New Jersey, U.S.A., 1985.

[Mor87] Leora Morgenstern. Knowledge Preconditions for Actions and Plans. In *Proceedings of the Tenth Annual International Joint Conference on Artificial Intelligence*, pages 867–874, Milan, Italy, August 23 – 28 1987.

[Mor88] Leora Morgenstern. *Foundations of a Logic of Knowledge, Action, and Communication.* PhD thesis, Courant Institute of Mathematical Sciences, New York University, 1988.

[OSE92] R. Oehlmann, D. Sleeman, and P. Edwards. Self–Questioning and Experimentation in an Exploratory Discovery System. In *Proceedings of the ML–92 Workshop on Machine Discovery*, pages 41–50, 1992.

[OSE93] R. Oehlmann, D. Sleeman, and P. Edwards. Learning Plan Transformations from Self–Questions : A Memory–Based Approach. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 520–525, Washington D.C., U.S.A., July 11 – 15 1993. American Association for Artificial Intelligence.

[PL87] Michael P.Georgeff and Amy L. Lansky. Reactive Reasoning and Planning. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, pages 677–682, Seattle, Washington, U.S.A., July 13 – 17 1987. American Association for Artificial Intelligence.

[Sac75a] Earl D. Sacerdoti. A Structure for Plans and Behavior. Technical Report 109, SRI Artificial Intelligence Center, Menlo Park, California, U.S.A., 1975.

[Sac75b] Earl D. Sacerdoti. The Nonlinear Nature of Plans. In *Proceedings of the Forth Annual International Joint Conference on Artificial Intelligence*, pages 206–214, Tbilisi, Georgia, USSR, September 3 – 8 1975.

[Sac77] Earl D. Sacerdoti. *A Structure for Plans and Behaviour.* Elsevier–North, Holland, 1977.

[Sch87] M. J. Schoppers. Universal Plans for Reactive Robots in Unpredictable Domains. In *Proceedings of the Tenth Annual International Joint Conference on Artificial Intelligence*, pages 1039–1046, Milan, Italy, August 23 – 28 1987.

[Sim88a] Reid G. Simmons. A Theory of Debugging Plans and Interpretations. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 94–99, St. Paul, Minnesota, U.S.A., August 21 – 26 1988. American Association for Artificial Intelligence.

[Sim88b] Reid G. Simmons. *Combining Associational and Causal Reasoning to Solve Interpretation and Planning Problems.* PhD thesis, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, 1988.

[Sus75] Gerald Jay Sussman. *A Computer Model of Skill Acquisition.* American Elseview Publishing Company, Inc., New York, New York, U.S.A., 1975.

[Tat77a] Austin Tate. Generating Project Networks. In *Proceedings of the Fifth Annual International Joint Conference on Artificial Intelligence*, Cambridge, Massachusetts, U.S.A., August 22 – 25 1977.

[Tat77b] Austin Tate. Project Planning Using a Hierarchical Non–linear Planner. Technical Report 25, Department of Artificial Intelligence, Edinburgh University, 1977.

[THD90] Austin Tate, James Hendler, and Mark Drummond. A Review of AI Planning Techniques. In James F. Allen, James Hendler, and Austin Tate, editors, *Readings in Planning*, pages 26–49. Morgan Kaufmann Publishers, Inc., San Mateo, California, U.S.A., 1990.

[Wel87] Michael P. Wellman. Probabilistic Semantics for Qualitative Influences. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, pages 660–664, Seattle, Washington, U.S.A., July 13 – 17 1987. American Association for Artificial Intelligence.

[Wel90a] Michael P. Wellman. Fundamental Concepts of Qualitative Probabilistic Networks. *Artificial Intelligence*, 44(3):257–303, 1990.

[Wel90b] Michael P. Wellman. Qualitative Probabilistic Networks for Planning under Uncertainty (revised). In G. Shafer and J. Pearl, editors, *Readings in Uncertain Reasoning*. Morgan Kaufmann Publishers, Inc., San Mateo, California, U.S.A., 1990.

[WH94] Mike Williamson and Steve Hanks. Optimal Planning with a Goal–directed Utility Model. In Kristian J. Hammond, editor, *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems*, pages 176–181, Chicago, Illinois, U.S.A., June 13 – 15 1994.

[Wil83] David E. Wilkins. Representation in a Domain–Independent Planner. In *Proceedings of the Eighth Annual International Joint Conference on Artificial Intelligence*, pages 733–740, Karlsruhe, West Germany, August 8 – 12 1983.