A SURVEY OF PLANNING IN INTELLIGENT AGENTS: FROM EXTERNALLY MOTIVATED TO INTERNALLY MOTIVATED SYSTEMS

by

Daphne Hao Liu

Technical Report TR-2008-936
in the Department
of
Computer Science

University of Rochester
June 2008

Abstract

The ability to plan is a essential for any agent, artificial or not, wishing to claim intelligence in both thought and behavior. Not only should a planning agent persist in pursuing a goal as long as the situation justifies the agent's perseverance, but an *intelligent* planning agent must additionally be proficient with responding to failures, opportunities, and threats in the environment. This distinction leads naturally to a discussion of externally motivated and internally motivated planning systems.

We first survey externally motivated planners, which exist and work only to accomplish user-given goals. These planners can be further classified as either non-hierarchical or hierarchical, depending on whether a high level plan is first developed and then successively elaborated. We then review internally motivated planners, which are endowed with self-awareness and such mental attitudes as beliefs, desires, and intentions. Finally, we present a preliminary proposal of a self-aware, opportunistic planning agent that maximizes its own cumulative utility while achieving user-specified goals.

Keywords: non-hierarchical planners, hierarchical planners, intention, belief-desire-intention (BDI), self-awareness, (internally) motivated systems.

Acknowledgments

My sincere gratitude goes to my advisor, Prof. Len Schubert, for his invaluable guidance and feedback on this work, and for his insights imparted in his courses. I am indebted to the University of Rochester for the Sproull Fellowship, and also very thankful for the support I have received under my advisor's NSF grant IIS-0535105.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In Artificial Intelligence (AI), *planning* is conventionally defined as the design or generation of a sequence of actions which, when executed, will result in the achievement of some desired goal. The ability to plan is a requisite for any agent, artificial or not, wishing to claim intelligence in both thought and behavior. Given a goal, a planning agent should persist in pursuing the goal to attainment as long as the situation justifies the agent's perseverance; that is, the agent should be sensitive to changes in the environment. Additionally, an *intelligent* planning agent must be equally proficient with responding to failures, opportunities, and threats in the environment.

Traditionally, planning systems have been regarded as mindless and unconscious slaves to human users, existing and working only to accomplish user-given goals. We consider such systems *externally motivated*. Deprived of self-awareness and such mental attitudes as beliefs, desires, and intentions, these systems act with total disregard for what they themselves stand to gain in cumulative utility from the fulfillment of user-given goals.

On the other hand, in alignment with AI's objective to instill or attain human level intelligence in machines, we are ultimately interested in designing and developing *internally motivated* planning agents. Not only do such agents possess self-awareness, but they can also act opportunistically and reason in both static and dynamic environments. By opportunistically, we mean that the agents can recover from unexpected action failures, seize unexpected favorable opportunities, and avoid unexpected threats.

Furthermore, we envision an opportunistic and self-aware planning agent, driven primarily by how well it can do for itself, maximizing its own cumulative utility while achieving user-specified goals. Given this directive, it would be interesting to see how, and assess to what extent, a self-serving agent can also exhibit cooperative behavior in practice.

The dichotomy between the externally and internally motivated systems leads to a natural division of our survey paper into Part I (Chapters 2 - 5) and Part II (Chapters 6 - 8), respectively. Parts I and II each begin with a preface/introduction, and end with a conclusion, of their respective contents. In focusing on this dichotomy, we set aside some important aspects of planning discussed in the broader planning literature. These include planning under uncertainty, multiagent planning, and path planning for robot motion and navigation, and isolated topics such as Markov decision processes, reinforcement learning, and conformant planning. Of the omitted topics, planning under uncertainty is probably the most important, and will eventually have to be taken into account in formulating planning strategies for an internally motivated agent that strives to optimize its cumulative rewards in an uncertain, incompletely known world.

In the remainder of this chapter, we outline the remaining chapters of our survey paper. We devote Chapter 2 mainly to investigating both human cognition and theorem proving as early influences in the design and development of planning systems. Specifically, GPS [63, 65, 9] looks to human cognition as a basis for planning, while the logical formulations of Green [41], Kowalski [49], and Bibel [9] lend the power of theorem proving to planning.

As planning systems can be further divided into non-hierarchical systems and hierarchical systems, we decompose our discussion into Chapter 3 and Chapter 4, accordingly. Although both non-hierarchical planning and hierarchical planning find a sequence of actions to achieve a given set of goals, the latter further distinguishes goals critical to the success of the plan from those that are merely details, first developing a higher level plan and then elaborating on it at successively finer levels later. Therefore, in contrast to the former, the latter can avoid getting bogged down in the minutiae of unnecessary plans.

Chapter 3 surveys several non-hierarchical planners: STRIPS [28], Warplan-C [99], UCPOP [69] along with with its ancestors TWEAK [16] and SNLP [53], GRAPHPLAN [12], and SATPLAN [47]. On the other hand, Chapter 4 reviews several hierarchical planners: STRIPS augmented with triangle tables [29], ABSTRIPS [79], NOAH [80, 81] along with its successors NONLIN [91] and SIPE [101], and DEVISER [95]. As well, we describe *hierarchical task network planning* extensively. Our discussion of the aforementioned planners is complete with algorithmic descriptions and numerous examples.

Part I concludes with Chapter 5. In addition to summarizing Chapters 2 - 4, we compare and contrast linear (total order) and nonlinear (partial order) planning, as well as progressive planning (forward chaining) and regressive planning (backward chaining).

In Chapter 6, we study the integral role such mental attitudes as beliefs, desires, and

intentions play in the design of intelligent agents. Our study is informed by Bratman's philosophical perspective of intention [14, 15], as well as Cohen and Levesque's theory of intention [17]. Additionally, we review Rao and Georgeff's related belief-desire-intention (BDI) framework [74, 76] founded on beliefs, desires, and intentions, before looking at the BDI architecture as realized in Georgeff and Lanky's PRS system [36].

Chapter 7 explores various views of self-awareness by Schubert [84], Minsky [60], and McCarthy [55]. Notably, Schubert [84] defined and differentiated *explicit self-awareness* from other weaker notions of self-awareness including self-monitoring. Anderson and Perlis's metacognitive loop [5], a self-monitoring agent, is reviewed. Moreover, through a review of literature by McCarthy [57] and Cox [20, 21], we consider introspection in planning agents. We then discuss Vere and Bickmore's Homer [96], an exemplar of a conscious agent exhibiting both perceptual awareness and self-awareness.

Finally, Chapter 8, the last chapter of our survey paper, first summarizes Chapters 6 - 7. Then in Section 8.2, we present our preliminary proposal of a self-aware, opportunistic planning agent that maximizes its own cumulative utility while achieving user-specified goals.

Part I

Externally Motivated Systems

Part I, comprised of Chapters 2 - 5, of our paper is focused on the *externally motivated* planning systems. These systems, in stark contrast to *internally motivated* systems, are devoid of self-awareness and mental attitudes such as beliefs, desires, and intentions. Consequently, these altruistic systems exist and work *only* to accomplish user-given goals without concern for what they themselves stand to gain in cumulative utility from their fulfillment of user-given goals. Alternatively, one may presume that these systems optimize their cumulative utility vicariously through users' contentment with the realization that the user-given goals have been achieved.

In Chapter 2, we explore two particular areas of scientific studies, namely human cognition and theorem proving, as early influences in the design and development of planning systems. Specifically, GPS [63, 65, 9] is an exemplar of how human cognition serves as a basis for planning, while logical formulations independently devised by Green [41], Kowalski [49], and Bibel [9] lend the robustness of theorem proving to planning.

As planning systems can be further divided into non-hierarchical systems and hierarchical systems, we decompose our discussion into Chapter 3 and Chapter 4, accordingly. First, *non-hierarchical planning* consists of finding a sequence of actions to achieve a given set of goals, while not distinguishing critical goals from less important ones. Chapter 3 surveys several non-hierarchical planners: STRIPS [28], Warplan-C [99], UCPOP [69] along with with its ancestors TWEAK [16] and SNLP [53], GRAPHPLAN [12], and SATPLAN [47].

In contrast to non-hierarchical planning, *hierarchical planning* distinguishes important goals from less critical ones, dividing a domain into various abstraction levels with successively lower levels realizing increasingly refined subplans. Chapter 4 reviews several hierarchical planners: STRIPS augmented with triangle tables [29], ABSTRIPS [79], NOAH [80, 81] along with its successors NONLIN [91] and SIPE [101], and DEVISER [95]. As well, an elaborate description of *hierarchical task network planning* is presented.

Lastly, Part I concludes with Chapter 5, which summarizes Chapters 2 - 4 and offers an analysis of various planning techniques. We compare and contrast linear (total order) and nonlinear (partial order) planning, as well as progressive planning (forward chaining) and regressive planning (backward chaining).

Chapter 2

Early Influences

Allen, Hendler, and Tate [2] acknowledge three areas of scientific studies as being influential in the development of planning systems; specifically, these are operations research, human cognition, and theorem proving. In sections 2.1 and 2.2, we discuss GPS [63, 65, 9] and three logical planners [41, 49, 9], respectively, as paragons of how the psychology of human cognition and the robustness of logic provide a foundation for planning systems.

Paraphrased briefly, the Merriam-Webster dictionary broadly characterizes *operations research* (OR) as the application of mathematical methods (e.g., statistics and algorithms) to the analysis of problems concerned with the optimization of the minima or maxima of some objective function. We refrain from delving into OR in this survey on planning because of its disparate focus. Whereas planning is a creative process of finding a sequence of actions to transform an initial configuration to a final configuration, OR strives to schedule tasks optimally by performing given actions so as to meet constraints and maximize utility.

2.1 GPS

Drawing on the study of human psychology, Newell and Simon [63, 64, 65] devised GPS – the general problem solver – in 1959, as a theory of human problem solving. GPS was gestated in a context reflecting a balance between the poles of *behaviorism* and *Gestalt psychology*. *Behaviorism* tasks psychology with providing objective explanations and measurements for an organism's responses as a function of stimuli; therefore, it espouses the mechanistic assumption that stimuli and responses are maintained as deterministic functions of the environment. In contrast, *Gestalt psychology* rejects this mechanistic assumption in favor of more holistic principles of organization; it postulates that thinking is not the mere

association of sensations and ideas without modification, but rather is assigned direction by the task or the context. To strike a balance between the two poles, Newell and Simon [65] conceded that no simple schemes of modern behavioristic psychology could fully account for a human's intricate system, but at the same time, all notions must be made operational via behavioral measures but treated with a dose of skepticism.

In order to evaluate GPS as a theory of human problem solving, Newell and Simon [65] conducted an experiment to obtain a verbatim protocol of a human subject verbalizing while attempting to transform a given logic expression. In the experiment, given twelve logical expression transformation rules each maintaining the semantic equivalence of the left operand and right operand in the pair, a college level engineering student with no prior knowledge of symbolic logic was first taught how to transform a logic expression using these rules, and then asked to transform a new logic expression into a syntactically different but semantically equivalent expression using the same rules. The student was encouraged to reason aloud and was recorded while solving the given task.

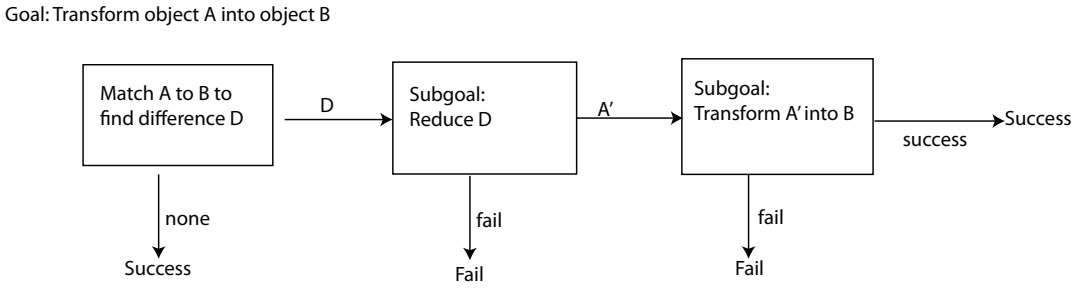

Figure 2.1: GPS's *Transform* Goal

While GPS is an attempt to simulate the processes which humans use to solve problems [64], it is also an endeavor to accomplish with machines the tasks which humans perform [63]. GPS operates in a task environment consisting of *objects* that can be transformed by *operators*. GPS gleans and organizes information regarding the task environment into *goals*. A *goal* is a conglomeration of information regarding what constitutes goal achievement and how this goal is related to other goals. There are three types of goals as follows:

1. *Transform* object A into object B;

2. *Reduce* difference D between object A and object B;

3. *Apply* operator Q to object A (to reduce a detected difference).

In the context of the aforedescribed experiment, the objects are logic expressions; the

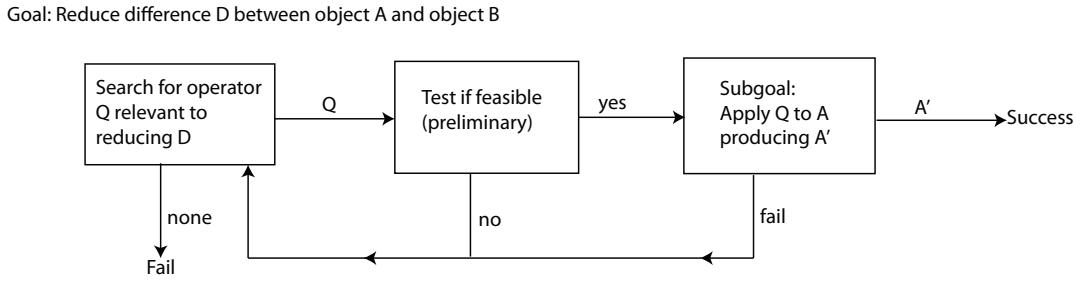

Figure 2.2: GPS's *Reduce* Goal

operators are the twelve rules; the differences include expressions like "change logical connective" or "add a term," and, as such, encapsulate ways to relate operators to their respective effects on objects. The objects and operators are specified by the task, whereas the differences are an additional perspective that GPS injects into the task.

GPS achieves a goal by establishing subgoals whose achievement leads to the fulfillment of the initial goal. To this end, GPS employs three crucial methods, one for each goal type. Figures 2.1, 2.2, and 2.3 present the high level algorithmic flowcharts for the *Transform*, *Reduce*, and *Apply* goal types, respectively. Readers interested in further details are referred to [63].

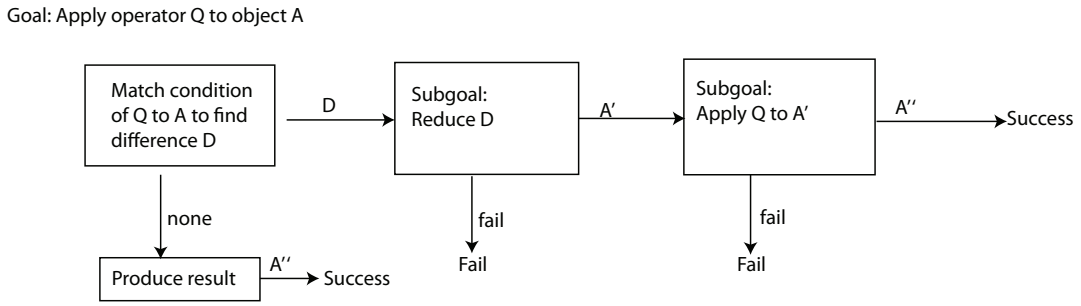

Figure 2.3: GPS's *Apply* Goal

To assess the competence of GPS as a model of the symbolic manipulations executed by the human subject in the aforedescribed experiment, Newell and Simon analyzed the trace of GPS performing the same logical expression transformation task as was given to the subject. In particular, they examined correspondence from the protocol to the trace; it was expected that every feature of the protocol concerning the task be mirrored in the trace, but not conversely as many task-related details certainly occurred without the subject's commenting on or perhaps being aware of them [65].

The GPS trace accorded well with the subject's protocol except for some exceptions.

Specifically, GPS could not differentiate between the internal and external worlds, nor could it process sets of items in parallel by using compound expressions. In essence, the key criticisms leveled against GPS are twofold. First, GPS has no continuous hindsight about its past actions and, thus, has difficulty mirroring the conscious thought in humans along the lines of "should have done something else instead" or "should have done that this other way instead." Second, GPS is a misnomer in that there are no general problem solving methods, but only specific methods suitable for specific domains and problem types. Nevertheless, the lesson distilled here underscores the significance of domain specific knowledge; domain specific knowledge plays an integral role in the general problem solving paradigm of identifying the differences between the current state and a desired state, acting accordingly to eliminate the differences, and then advancing to the desired state.

2.2 Logic Planners

The robustness of logic and the power of theorem proving were exploited in early planning systems. In sections 2.2.1, 2.2.2 and 2.2.3, we describe Green's [41] and Kowalski's [48] formulations of planning, as well as Bibel's [9] deductive solution of generating plans for solving robot tasks.

2.2.1 Green's Formulation

In 1969, Green [41] presented a logical characterization of planning in the formalism of the situation calculus [58]; additionally, Green introduced QA3, an implementation based on an earlier resolution theorem prover [42] and aimed at solving problems in several robotic applications at SRI. Green's seminal contribution is his illustration of the direct correspondence between theoretical specification and practical implementation. The axioms used in the formal description of the planning task constitute the basis of the representation in the implemented planner; furthermore, each step taken by the planner corresponds to a step in the construction of a proof of the existence of a suitable plan [86].

Green's formulation of a robot planning problem encompasses three sets of assertions with the closed world assumption, one set specifying the initial state of the world, another describing the effects of all possible actions that a robot can take on states, and a third set of *frame assertions*, which stipulate what remains unchanged during an action. All variables in assertions are assumed to be universally quantified, unless bound by an existential quantifier. Unless its truth value is independent of state, a predicate includes a state or situation

variable. The goal is denoted by a formula with an existentially quantified state variable. In the system, all assertions and the negation of the goal formula are first converted to clausal form and passed to the resolution theorem prover QA3; then, the prover attempts to find a contradiction, using resolution refutation with an answer extraction literal, and finding a goal state if one exists. A goal state is expressed as a composition of *do* functions, showing the sequence of actions that lead to the goal state. For instance, the function $do(\alpha, \delta)$ denotes the state that results from executing action α in state δ.

To illustrate Green's formulation, consider an example [66] in which objects are denoted by uppercase letters, variables by lowercase letters, and predicates by uppercase words. Suppose in the initial world S_0, there are a clear position C and two clear-topped blocks A and B directly on positions D and E, respectively, expressed by the set of formulas $\{BLOCK(A),\ BLOCK(B),\ POSN(D),\ POSN(E),\ POSN(C),\ ON(A, D, S_0),\ ON(B, E, S_0),\ CLEAR(A, S_0),\ CLEAR(B, S_0),\ CLEAR(C, S_0)\}$. Further suppose that there is an action $move(x, y, z)$, which moves x currently directly on y and places x directly on z iff both x and z are different and clear-topped; we can formulate the positive literal effects of this action as the implication $[CLEAR(x, s) \wedge CLEAR(z, s) \wedge DIFF(x, z) \wedge ON(x, y, s)] \rightarrow [CLEAR(x,\ do(move(x, y, z),\ s)) \wedge CLEAR(y,\ do(move(x, y, z),\ s)) \wedge ON(x,\ z,\ do(move(x, y, z),\ s))]$. Assuming the simple goal of having block A directly on top of block B, we express the goal formula as $(\exists s)ON(A, B, s)$. With resolution refutation using answer extraction, we obtain $s_1 = do(move(A, B), S_0)$ as the single action required in achieving the goal.

It is imperative that we have frame assertions for *each* predicate relation not changed by an action. In the aforedescribed trivial example, we would need assertions to express that blocks not moved stay in the same position, and that a block w remains clear if it is clear when a different block u is placed directly on top yet another different block v. Had our goal been the compound goal $(\exists s)[ON(A, B, s) \wedge ON(B, C, s)]$, we would have had to use a frame assertion to prove that B stays on C when A is placed on B.

Shanahan [86] claimed that it is widely believed that planning through theorem proving is impractical and inefficient. Despite Green's acknowledged contribution bridging theory and practice, Russell and Norvig [77] also lamented the smaller-than-expected impact Green's work has had on later planning systems in the following excerpt:

> Unfortunately a good theoretical solution does not guarantee a good practical solution .... To make planning practical we need to do two things: (1) Restrict the language with which we define problems .... (2) Use a special purpose algorithm

> ... rather than a general purpose theorem prover to search for a solution. The two go hand in hand: every time we define a new problem description language, we need a new planning algorithm to process the language .... The idea is that the algorithm can be designed to process the restricted language more efficiently than are solution theorem prover. [Russell & Norvig, 1995, page 342 of *Artificial Intelligence: A Modern Approach*]

2.2.2 Kowalski's Formulation

A successor of Green's work [41], Kowalski's planning formulation [48] reifies what would normally be predicates in Green's formulation as an attempt to simplify frame assertions. To this end, Kowalski introduced the $HOLDS$ predicate, as well as the $PACT$ and $POSS$ predicates. For instance, $POSS(s)$ means that state s is a possible state and one that can be reached; literal $HOLDS(l, s)$ can be used instead of the single literal l to denote the *concept* of the term l being true in state s; $PACT(\alpha, s)$ states that it is possible to execute action α in state s. The primary advantage of Kowalski's predicate-reifying formulation is that only one frame assertion is required for each action, as is apparent in the following example.

In accordance with the example in Section 2.2.1, we express the initial state as the set of formulas $\{POSS(S_0)$, $BLOCK(A)$, $BLOCK(B)$, $POSN(C)$, $POSN(D)$, $POSN(E)$, $HOLDS[CLEAR(A), S_0]$, $HOLDS[CLEAR(B), S_0]$, $HOLDS[CLEAR(C), S_0]$, $HOLDS[ON(A,\ D),\ S_0]$, $HOLDS[ON(B,\ E),\ S_0]\}$. We cast the each relation made true by (i.e., each positive literal effect of) an action using a separate $HOLDS$ literal; thus, for our action $move(x, y, z)$, we obtain the formulas $HOLDS[CLEAR(x),\ do(move(x, y),\ s)]$, $HOLDS[CLEAR(y),\ do(move(x, y),\ s)]$ and $HOLDS[ON(x,\ z),\ do(move(x,\ y),\ s]$. The preconditions of action $move(x, y, z)$ are given by the formula $[HOLDS[CLEAR(x),\ s] \wedge HOLDS[CLEAR(z),\ s] \wedge DIFF(x, z) \wedge HOLDS[ON(x,\ y),\ s]] \rightarrow PACT[move(x,\ y,\ z),\ s]$. We also need the axiom $[POSS(s) \wedge PACT(u,\ s)] \rightarrow POSS[do(u,\ s)]$ to assert that the state resulting directly from performing an action whose preconditions are all satisfied in a possible state is also a possible state. The single frame assertion required here is $[HOLDS(v,\ s) \wedge DIFF[v,\ CLEAR(z)] \wedge DIFF[v,\ ON(x,\ y)]] \rightarrow HOLDS[v,\ do(move(x,\ y,\ z),\ s)]$, which specifies that all terms different from $CLEAR(z)$ and $ON(x,\ y)$ still hold in all states generated by executing $move(x,\ y,\ z)$. Our goal would be $(\exists s)[POSS(s) \wedge HOLDS[ON(A,\ B),\ s]]$, with conjunct $POSS(s)$ stipulating that the goal state s be reachable.

Despite the elegant treatment of the frame problem, two main deficiencies of the formulation need to be addressed. First, the formulation fails to account for the ramification problem. The ramification problem is concerned with representing what happens indirectly and implicitly as a result of an action. For example, if we were to move an object H currently in a house K and place H on top of a box J located in another house M, Kowalski's formulation would not be able to express that as a result of this move, H would be in house M. See [87, 45, 52, 54] for some solutions to both the frame and ramification problems. Second, the formulation cannot model concurrent actions and their interactions, namely the *precondition interaction problem* and the *effects interaction problem* [71]. The former arises when actions can or cannot be be executed, depending on whether other actions are executed concurrently; e.g., some actions might require a shared resource that is unavailable if they are performed in parallel. The latter occurs either when two concurrent actions cancel each other's effects, or when two concurrent actions synergistically produce effects that neither action performed in isolation could. Refer to [71] for a situation calculus extension that accommodates the interactions of concurrent actions.

2.2.3 Bibel's Formulation

In designing a deductive solution to generating plans for solving robot tasks, Bibel first looked to program synthesis as a source of inspiration. Supposing one can specify input-output relation $R(x, z)$ for any input x and any output z, then under certain circumstances one may extract a function f such that $f(x) = z$ from a proof establishing z's existence. A powerful theorem prover would make possible this elegant and straightforward way of programming. Bibel then conjectured that generating a plan to a given robot task seems similar in nature, but for the exception that actions to be executed by the plan were actions in the environment as opposed to in a computer. Noting the analogy between program synthesis and plan generation, Bibel devised a plan generation solution [9], one which he claimed to be as elegant and straightforward as the one for program synthesis.

In this solution [9], the problem description of a task consists of an initial configuration and a final configuration both given by logical formulas of arbitrary structure, as well as a set of available actions formalized as $antecedent \rightarrow consequent$ rules whose antecedent and consequent are both expressed as arbitrary logical formulas. A solution to the task is then generated by a proof of the problem description with the exception that proofs here are strictly linear (to be explained shortly). This linearity renders Bibel's solution amenable to any theorem prover based on the connection method [8], in addition to accelerating the

proof search.

Specifically, in Bibel's approach [9], situations are given as sets of objects satisfying certain properties expressed as arbitrary logical formulas. Primitive actions to change situations and at the disposal of the robot take the *antecedent* $\rightarrow$ *consequent* form, with the antecedent and consequent of each action specified by two separate arbitrary logical formulas. Notably, no properties impertinent to the action under consideration need be mentioned in the rules; thus, this approach also solves the frame problem. The robot task is then represented by a formula that there exists a goal set satisfying the goal properties under the circumstances given by the situation descriptions and rules. If a strictly linear connection proof can be found, then the final value of the output variable corresponds to a plan. A strictly linear connection proof [8] is one in which any instance of a literal is contained in at most one connection required for establishing the proof.

Drawing on PROLOG's pairing of goals and heads, a connection proof [8] is comprised of the connections along with the substitutions obtained by unifying connected literals. A formula is then visualized in a two dimensional matrix in which disjunctions are presented horizontally and conjunctions vertically. Consider the following example from [9], where a and b are constants, s and z are, respectively, the initial and final state, predicates are denoted by single capital letters, and literals are without parentheses. Predicate S, short for $STATE$, acknowledges that actions change the state of the world. The initial configuration F_1 is $Ss \wedge Cb \wedge Oba \wedge Ta \wedge E$; the goal configuration G_1 is $Ta \wedge Tb \wedge E \wedge Sz$, with z as the output variable; the first action rule R_1 is $\forall w \forall x \forall y \exists u(Sw \wedge Cx \wedge Oxy \wedge E \rightarrow Su \wedge Hx \wedge Cy)$; the second action rule R_2 is $\forall w' \forall v \exists p(Sw' \wedge Hv \rightarrow Sp \wedge Tv \wedge E)$. Given these formulas, we wish to prove $P_1 : F_1 \wedge R_1 \wedge R_2 \rightarrow G_1$.

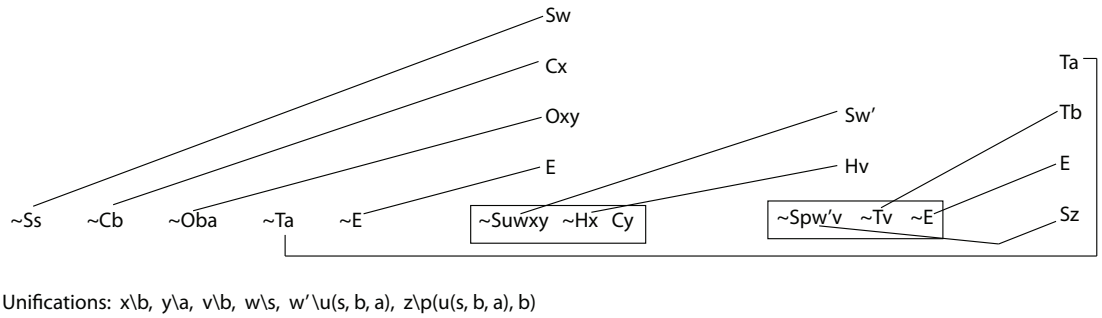

Figure 2.4: An Example Connection Proof of P_1

Figure 2.4 presents the connection proof of P_1 after its transformation into the semantically equivalent formula $\neg Ss \vee \neg Cb \vee \neg Oba \vee \neg Ta \vee \neg E \vee (Sw \wedge Cx \wedge Oxy \wedge E \wedge (\neg Suwxy \vee$

$\neg Hx \vee Cy)) \vee (Sw' \wedge Hv \wedge (\neg Spw'v \vee \neg Tv \vee E)) \vee (Ta \wedge Tb \wedge E \wedge Sz)$ as depicted in the matrix form. In Figure 2.4, a tilde denotes the logical *not* operator, P_1's universal quantifiers have been deleted, and P_1's existential quantifiers for u and p replaced by Skolem functions $u(w, x, y)$ and $p(w', v)$, respectively. Paraphrased briefly, the initial configuration F_1 is one in which a clear-topped block b is directly on top of block a, which is, in turn, directly on top of a table with an empty hand readily available to pick up a block; the goal configuration G_1 finds blocks a and b both clear-topped and directly on top of the table with the hand being empty; the resultant value $p(u(s, b, a))$ of z stipulates an appropriate plan of first unstacking b from a and then putting down b.

Bibel proclaimed the generality and notational ease of his solution as its two primary advantages. First, once a problem description stated in terms of logical formulas as described has been formulated, any theorem prover may take the description with the proviso that the prover must obey the linearity restriction. In terms of the connection method [8], the prover must mark instances of literals used in the proof and must not use a marked literal again except upon backtracking. Second, this solution seems less notationally cumbersome than Green's [41] and Kowalski's [48] formulations. In particular, this solution avoids the former's additions of a state variable in each literal and of frame assertions for each relation unchanged by an action, without the inclusion of predicates such as *POSS* and *HOLDS* of the latter. Other advantages are that the linearity restriction accelerates the proof search, and that previously generated macro steps may easily be added to the solution of more difficult problems by incorporating the respective preprocessed proof fragments.

Bibel later introduced *transition logic* (TL) [11], based on the linear connection method [8]. [31] then developed LBC – the linear backward chainer – for converting a given planning problem in TL into the language of the SETHEO theorem prover [1], and any proof found would represent a plan of actions for the given problem. SETHEO was selected because it had won the first place in the Automated Theorem Proving competition at the 1996 Conference on Automated Deduction. It was reported in [11] that the combined use of LBC and SETHEO outperformed UCPOP [69] by several orders of magnitude. However, as noted in [10], LBC does not provide a reduction, but rather an encoding, of planning to deduction. The reason is that LBC does not directly translate TL into first-order logic; instead, LBC marks available propositions to prevent two connections from sharing a single proposition. Furthermore, no tests have been conducted to compare LBC/SETHEO with other algorithms that also have outstripped UCPOP.

Chapter 3

Non-Hierarchical Planners

Non-hierarchical planning consists of finding a sequence of actions to achieve a given set of goals. In non-hierarchical planning, one does not distinguish critical goals from less important ones, so one could waste a significant amount of time searching for solutions to non-critical parts of a plan, only to find a more critical part of the plan unsolvable. For instance, if the problem were to get from home to your office by public transit, then taking a bus with a suitable schedule would be more critical than deciding whether to take the stairs or the elevator up to the office floor after your arrival at the office building.

Heeding the deficiencies of non-hierarchical planning, those designing non-hierarchical planners should strive for a balance between creating overly detailed plans and overly vague plans. Being bogged down in solving uncritical parts in an overly detailed plan is a waste of time, while an overly vague plan cannot sufficiently specify which problem solving methods to use at a given point. Since striking such a balance is not an exact science, hierarchical planning has gained popularity in recent decades. We will review hierarchical planners in Chapter 4.

In this chapter, we survey several non-hierarchical planners: Fikes and Nilsson's STRIPS [28] in Section 3.1; Warren's conditional linear planner Warplan-C [99] in Section 3.2; Penberthy and Weld's partial order planner UCPOP [69], as well as its two direct ancestors, namely Chapman's TWEAK [16] and McAllester and Rosenblitt's SNLP [53], in Section 3.3; Blum and Furst's Graphplan [12] in Section 3.4; and Kautz and Selman's SATPLAN [47] in Section 3.5. Our discussion is supplemented with algorithmic pseudocode and examples.

3.1 STRIPS

STRIPS (STanford Research Institute Problem Solver) [28] is a problem solver devised by Fikes and Nilsson in 1971. Given an initial world model and a goal formula, STRIPS strives to find a sequence of operators to tranform the initial model into one in which the goal formula holds. A world model is represented as a collection of first-order predicate calculus formulas, and a resolution theorem prover, QA3.5 [32], is consulted in answering questions of particular models.

The problem space in STRIPS is defined by three entities:

1. An initial world model, DB_0, which is a set of closed arbitrary well-formed formulas describing the initial state of the world; e.g.,

$$DB_0 = \left\{ \begin{array}{l} InRoom(box_1, supplies), Connected(doorA, supplies, kitchen), \\ Connected(doorA, kitchen, supplies), InRoom(box_2, closet), \\ Box(box_2), Connected(doorB, closet, supplies), Box(box_1), \\ InRoom(robot, kitchen), Connected(doorB, supplies, kitchen) \end{array} \right\}$$

2. A set of operators, including their preconditions and effects (formulas to be added to or deleted from the world) stated as arbitrary well-formed formulas; e.g.,

$pushThru(x, d, r_1, r_2)$

Precondition:	$InRoom(robot, r_1), InRoom(x, r_1), Connected(d, r_1, r_2)$
Delete list:	$InRoom(robot, r_1), InRoom(x, r_1)$
Add list:	$InRoom(robot, r_2), InRoom(x, r_2)$

$goThru(d, r_1, r_2)$

Precondition:	$InRoom(robot, r_1), Connected(d, r_1, r_2)$
Delete list:	$InRoom(robot, r_1)$
Add list:	$InRoom(robot, r_2)$

3. A goal condition specified as an arbitrary well-formed formula whose free variables are understood existentially; e.g., $Box(x) \wedge InRoom(x, kitchen)$.

In STRIPS, we assume the world we are dealing with satisfies the following criteria:

(a) Only one action can occur at a time;

(b) Actions are effectively instantaneous in the model (though not always so in the actual world as in Shakey the robot [67]);

(c) Nothing changes except as the result of planned actions (as a solution to the frame problem).

A prudent search strategy is employed in STRIPS to avoid generating an undesirably large and thus impractical tree of world models from all the operators applicable to any given world model. To this end, STRIPS adopts the GPS [63, 64, 65] two-pronged strategy of (i) extracting *differences* between the current world model and the goal and then (ii) identifying operators pertinent to reducing these differences. A *difference* is a literal which cannot be resolved and so hampers derivation of the empty clause from the set union of the current world model and the negated goal. A pertinent operator is one with an effect on its add list that resolves against a difference literal. Once STRIPS has determined a pertinent operator, it attempts to solve the subproblem of producing a world model to which the operator is applicable, wherein the preconditions of the selected operator constitute a subgoal to be achieved. If such a model is attainable (i.e., the subgoal has been achieved), then STRIPS applies the operator and reconsiders the original goal in this new model.

In the original STRIPS, the hierarchy of goals, subgoals, and models generated in the search process is represented by a search tree, in which each node has the form ($\langle$*world-model*$\rangle$, $\langle$*goal-list*$\rangle$) and represents the problem of achieving the subgoals on the ordered goal list from the world model. Typically, more than one pertinent operator will be found, and the search tree allows for the exploration of different pertinent operators. In essence, STRIPS operates as follows:

(1) Select a subgoal and attempt to establish that it is true in the appropriate model. If so, go to step (4). Otherwise, go to step (2).

(2) Choose as a pertinent operator one whose add list specifies clauses that allow the incomplete proof of step (1) above to be continued. Go to step (3).

(3) The appropriately instantiated precondition well-formed formulas of the chosen operator constitute a new subgoal. Go to step (1).

(4) If the subgoal is the main goal, terminate. Otherwise, create a new model by applying the operator whose precondition is the subgoal just established, and go to step (1).

STRIPS successfully solves a given problem when it generates a world model satisfying the goal well-formed formula. A solution is a sequence $\langle Act_1\theta_1, ..., Act_n\theta_n\rangle$, where Act_i is

the name of an operator with Pre_i, Add_i, and Del_i as the corresponding components, and θ_i is a substitution of constants for the variables in that operator, and where the sequence satisfies the following:

(a) For all i such that $1 \leq i \leq n$, $DB_i = (DB_{i-1} \setminus Del_i\theta_i) \cup Add_i\theta_i$;

(b) For all i such that $1 \leq i \leq n$, $DB_{i-1} \vdash Pre_i\theta_i$;

(c) For some θ, $DB_n \vdash Goal\theta$.

3.1.1 The Reasoning Component in the Original STRIPS

The original STRIPS uses the resolution theorem prover QA3.5 [32] when attempting to prove goal and subgoal well-formed formulas (WFFs). The general situation is that given some goal $G(p)$ and a set M of clauses, we want to prove $G(p)$ from M, where p is a set of parameters appearing in the goal WFF. Using resolution refutation, we attempt to prove the inconsistency of the set $\{M \cup \neg G(p)\}$; that is, we try to find an instance p' for p for which $\{M \cup \neg G(p')\}$ is inconsistent.

The standard unification algorithm of resolution refutation is used along with the following modifications for parameters (non-constants, non-quantified variables):

- Terms that can be substituted for a variable are variables, constants, parameters, and functional terms not containing the variable;
- Terms that can be substituted for a parameter are constants, parameters, and functional terms not containing Skolem functions, variables, or the parameter;
- Suppose clauses C_1 and C_2 resolve to form clause C and that some term t is substituted for some parameter p in the process. Then we must make sure that p is replaced by t in all clauses that descend from C.

It must be noted that in later versions of STRIPS, the reasoning component has been completely removed to increase efficiency. A clear negative implication is that we cannot derive logical consequences to address the ramifications of actions. In fact, in the two progressive planning and regressive planning examples given in Section 3.1.2, whenever there is a test of the form $LHS \models RHS$ by resolution refutation, resolution is not carried out; instead, variable bindings are applied to check whether there exist instances of the RHS in the LHS, that is, whether $RHS \subseteq LHS$. Therefore, the test or check amounts to

pattern matching following variable bindings, akin to what occurs in a production system except there the matched production rules would fire automatically.

According to Pollock [72], the replacement of the deductive approach by the algorithmic approach to planning in STRIPS was dictated largely by the inability to solve the frame problem. Pollock [72] defined the frame problem as that of building a reasoner able to reason correctly about all the consequences of actions. Since later versions of STRIPS do not reason about the consequences of action, they allow only for literals in the add- and delete-lists of actions. This means that the design of a STRIPS planning system stripped of its reasoning component cannot encode knowledge as a set of first-order formulas.

3.1.2 Algorithms of Propositional STRIPS

In this section, we review two planning algorithms of STRIPS in the propositional case first presented in [13]. In the propositional case, DB contains no variables, whereas Pre, Add and Del may do so.

Algorithm 1 Progressive Planning Algorithm for Propositional STRIPS

1: **procedure** PROGPLAN($DB, Goal$)
2: **if** $DB \models Goal$ by resolution refutation **then**
3: **return** the empty plan
4: **end if**
5: **for** each operator $\langle Act, Pre, Add, Del \rangle$ such that $DB \models Pre\theta$ by resolution refutation, where θ is an arbitrary substitution of constants for variables **do**
6: $DB' \leftarrow (DB \setminus Del\theta) \cup Add\theta$
7: $Plan \leftarrow ProgPlan(DB', Goal)$
8: **if** $Plan \neq$ failure **then**
9: **return** $Act\theta.Plan$
10: **end if**
11: **end for**
12: **return** failure
13: **end procedure**

Consider the depth first, propositional version of the progressive planner in Algorithm 1, which takes as inputs a world model DB and a goal formula $Goal$, and returns as output a plan or failure. We apply this algorithm to the example introduced in Section 3.1; enumerated below are the steps.

(1) Given DB_0 and $Goal$ from Section 3.1, we cannot derive $\{\}$ from $DB_0 \cup \neg Goal$.

(2) Since we can derive $\{\}$ from $DB_0 \cup (\neg InRoom(robot, r_1) \vee \neg Connected(d, r_1, r_2))$

using substitution $\theta_1 = (d/doorA,\ r_1/kitchen,\ r_2/supplies)$, we apply $goThru(doorA,\ kitchen,\ supplies)$ and generate a new progressed world model:

$$DB_1 = \left\{ \begin{array}{l} Connected(doorA, kitchen, supplies), InRoom(robot, supplies), \\ Connected(doorA, supplies, kitchen), InRoom(box_2, closet), \\ Connected(doorB, closet, supplies), Box(box_1), Box(box_2), \\ Connected(doorA, supplies, closet), InRoom(box_1, supplies) \end{array} \right\}$$

(3) We cannot derive $\{\}$ from $DB_1 \cup \neg Goal$.

(4) Since we can derive $\{\}$ from $DB_1 \cup (\neg InRoom(robot,\ r_1)) \vee (\neg InRoom(x,\ r_1)) \vee (\neg Connected(d,\ r_1,\ r_2))$ using substitution $\theta_2 = (x/box_1,\ d/doorA,\ r_1/supplies,\ r_2/kitchen)$, we apply $pushThru(box_1,\ doorA,\ supplies,\ kitchen)$ and generate a new progressed world model:

$$DB_2 = \left\{ \begin{array}{l} Connected(doorA, kitchen, supplies), InRoom(box_1, kitchen), \\ Connected(doorB, closet, supplies), InRoom(box_2, closet), \\ Connected(doorA, supplies, closet), Box(box_1), Box(box_2), \\ Connected(doorA, supplies, kitchen), InRoom(robot, kitchen) \end{array} \right\}$$

(5) Since we can derive $\{\}$ from $DB_2 \cup \neg Goal$, we are done, and the plan is $\langle goThru(doorA,\ kitchen,\ supplies),\ pushThru(box_1,\ doorA,\ supplies,\ kitchen)\rangle$.

Algorithm 2 Regressive Planning Algorithm for Propositional STRIPS

```
1: procedure RegPlan(DB, Goal)
2:     if DB ⊨ Goal by resolution refutation then
3:         return the empty plan
4:     end if
5:     for each operator ⟨Act, Pre, Add, Del⟩ such that (Delθ ∩ DB) = ∅, where θ is an
   arbitrary substitution of constants for variables do
6:         Goal′ ← (Goal \ Addθ) ∪ Preθ
7:         Plan ← RegPlan(DB, Goal′)
8:         if Plan ≠ failure then
9:             return Plan.Actθ
10:        end if
11:    end for
12:    return failure
13: end procedure
```

Algorithm 2 is a depth first, propositional version of the regressive planner, which takes as inputs a world model DB and a goal formula $Goal$, and returns as output a plan or failure. Continuing with the same example, we show step by step how a plan is found regressively.

(1) Given DB_0 and $Goal$ from Section 3.1, we cannot derive $\{\}$ from $DB_0 \cup \neg Goal$.

(2) Since $Goal \cap \{InRoom(robot,\ r_1),\ InRoom(x,\ r_1)\} = \emptyset$ using substitution $\theta_1 = (x/box_1, d/doorA, r_1/supplies, r_2/kitchen)$, we apply $pushThru(box_1, doorA, supplies, kitchen)$ and generate a new regressed goal:

$$Goal_1 = \left\{ \begin{array}{l} Connected(doorA, supplies, kitchen), Box(box1), \\ InRoom(box_1, supplies), InRoom(robot, supplies) \end{array} \right\}$$

(3) We cannot derive $\{\}$ from $DB_0 \cup \neg Goal_1$.

(4) Since $Goal_1 \cap \{InRoom(robot, r_1)\} = \emptyset$ using substitution $\theta_2 = (d/doorA, r_1/kitchen, r_2/supplies)$, we apply $goThru(doorA,\ kitchen,\ supplies)$ and generate a new regressed goal:

$$Goal_2 = \left\{ \begin{array}{l} Connected(doorA, supplies, kitchen), InRoom(robot, kitchen), Box(box1), \\ Connected(doorA, kitchen, supplies), InRoom(box_1, supplies) \end{array} \right\}$$

(5) Since we can derive $\{\}$ from $DB_0 \cup \neg Goal_2$, we are done and the plan is $\langle goThru(doorA,\ kitchen,\ supplies),\ pushThru(box_1,\ doorA,\ supplies,\ kitchen)\rangle$.

We postpone the discussion of regressive (backward chaining) versus progressive (forward chaining) planning until Chapter 5, the concluding chapter of Part I.

3.1.3 Example Problems STRIPS Cannot Solve

We describe two example problems that STRIPS cannot solve, namely the Sussman anomaly [90] and the register exchange problem. Both problems illustrate a flaw of non-interleaved planners, including STRIPS. A non-interleaved planner is one which, when given two goals G_1 and G_2, attempts to generate either a plan for G_1 concatenated with a plan for G_2, or a plan for G_2 concatenated with a plan for G_1.

In the Sussman anomaly problem, initially a clear-topped block B rests directly on a table, with a clear-topped block C directly atop a block A set directly on the table.

Permitted to move only clear-topped blocks and only one such block at a time, the agent wishes to stack the three blocks such that clear-topped A is directly atop B, which is, in turn, directly atop C resting on the table.

Non-interleaved planners, including STRIPS, would normally divide the goal of placing A atop B atop C into two subgoals of (i) getting A atop B and of (ii) getting B atop C. Suppose the agent starts with subgoal (i). The agent would try to first unstack C off A, and then move A atop B, in order to achieve (i); however, the agent cannot now pursue subgoal (ii) without undoing (i), since both A and B must be moved atop C and only clear-topped blocks can be moved. Suppose, instead, that the planner begins with subgoal (ii). To accomplish (ii), the agent would directly move B atop C as the most efficient solution; however, the agent cannot now pursue subgoal (i) without undoing (ii). STRIPS preserves already achieved goals and protects them from subsequent actions; as a result of its inherent goal protection mechanism, STRIPS fails the Sussman anomaly.

Yet another problem STRIPS cannot solve is the register exchange problem. In this problem, initially there are three registers: a register R_1 containing a value N_1, a register R_2 containing a value N_2, and an empty register R_3. The initial state is, thus, given by $\{Contains(R_1, N_1), Contains(R_2, N_2)\}$. The only action at the agent's disposal is $Copy(x, y, z)$, which copies contents x of register y to register z, with precondition $Contains(y, x)$, delete list $Contains(z, \$)$, and add list $Contains(z, x)$. The agent wishes to exchange the contents of R_1 and R_2; the goal state is, thus, captured by $\{Contains(R_1, N_2), Contains(R_2, N_1)\}$.

As a linear (total order) planner, STRIPS encodes a plan as a totally ordered sequence of actions under the linearity assumption that subplans can be merged only by concatenation and not by interleaving. On the other hand, nonlinear (partial order) planners, which impose ordering constraints only as required at execution time, can solve both the Sussman anomaly and the register exchange problem. We discuss UCPOP [69], an exemplar of nonlinear planners, in Section 3.3.

3.2 Warplan-C

Warren's Warplan-C [99] is a system for synthesizing conditional linear plans and programs from problem descriptions precisely specified in predicate logic. It holds the distinguishing honor of being the very first conditional planner. A strictly forward planner, Warplan-C uses extended STRIPS rules to express uncertain outcomes, designed to support automatic

programming.

Warplan-C [99] is a direct successor of Warren's linear planner Warplan [98], extended to handle conditionals. A conditional plan is required when uncertainties either in an environment or in the results of actions invalidate the selection of a single course of action to achieve a goal [70]. A conditional planner generates a set of plans for all projected contingencies in advance, instead of replanning at execution time. As such, Warplan-C differs from a reactive planner, which improvises solutions to address both foreseen and unforeseen uncertainties that arise at execution time.

Warplan-C is a strictly forward planner, constructing a plan by progressively modifying a partial solution until there exist no unsatisfied goals. Figure 3.1 depicts an example partial plan in which the goal v has to be true at time T. There are two cases to consider. Supposing v is already true T; then the system need not do anything. Suppose otherwise. Then a new action achieving v must be inserted in the plan at some time prior to T, say at time T'; furthermore, the system ensures both that the new action does not affect any already attained goals, and that no intervening actions between time T' and time T destroy v. Generally, the new action inserted at time T' would require that further goals be formed at T'.

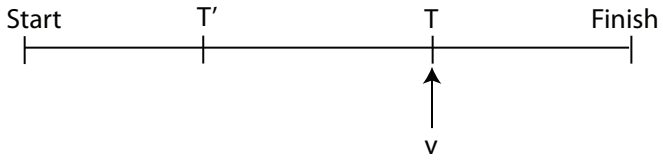

Figure 3.1: An Example Partial Plan

In Warplan-C, certain actions comprising a plan are labeled conditional. A conditional action has two possible outcomes p and $\neg p$, where p denotes an arbitrary proposition. The planning passes proceed iteratively until all different branches of each conditional action have been taken. In each pass, the planner develops a plan by exploring a different branch for each conditional action. Consider an example illustrated in Figure 3.2 adapted from that of Peot and Smith [70], in which a tilde denotes the logical *not* operator, the S and G square nodes represent, respectively, the start and goal states, the numbered circular nodes indicate the actions in the plan, and there exists exactly one conditional action in the plan with possible outcomes p and $\neg p$.

Paraphrased from the explanation in [70], in Figure 3.2A, the planner generates an

unconditional plan under the assumption that the conditional action 1 has outcome p. After developing this plan, in Figure 3.2B, the planner plans for the other conditional branch, using the same initial segment (the start state and action 1), but inserts another action 3 immediately after action 1 to preserve the preconditions of action 2 and the goal. Like action 1 (or perhaps as a result of immediately following action 1), action 3 also yields outcome p or $\neg p$. Finally in Figure 3.2C, the planner obtains the final plan by combining these two plans. Notably, the planner never fuses different branches of the conditional plan.

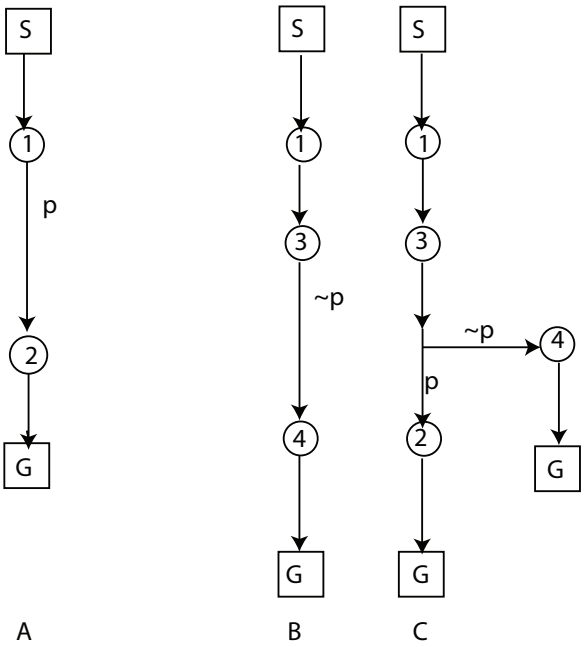

Figure 3.2: Example Operations of Warplan-C

The explanation above paraphrased from [70] is puzzling, however. Our interpretation of the explanation of [70] is that a part of the effect of action 1 can be p or $\neg p$, and that actions 1-4 are each an arbitrarily complex subplan. Suppose that action 1 has either outcome $p \wedge q$ or $\neg p \wedge r$, where p, q, and r can be arbitrarily complex formulas. In the former case, the remaining plan works, with action (subplan) 2 completing it. But for the latter outcome, $\neg p$ may falsify the preconditions of action 2 or a goal condition, and to repair this, an action 3 is inserted that repairs those conditions. Nevertheless, if action 3 repairs what is necessary for the preconditions of action 2 and G, it is unclear why one would still need to branch on $\neg p$. Although p may still be false as only those aspects that affect action 2 and G are repaired, it should not matter if some other part of p is still false – by definition this would not impact action 2 or G.

Warplan-C adheres to certain criteria that help it not only restrict its search, but also guide the search more quickly to a solution [99]. The first criterion is consistency, used by the planner to reject inconsistent goals when the planner has information that certain statements cannot be simultaneously true. The second, minimality criterion ensures that the planner never tests a condition more than once on the same path through a problem. Loop checks are the optional, third criterion. Generally, loop checks streamline the search; while *weak* loop checks reject goals that are subgoals of another goal at the same point in the plan, *strong* loop checks dismiss subgoals that are unifiable with the supergoal. However, loop checks break the planner's completeness as they may prevent genuine solutions to certain problems from being found [99].

Distinguished as the very first conditional planner, Warplan-C without the aforementioned loop checks is also conjectured to be complete; that is, if a precise problem presented to the system has a solution, the system can, in principle, find a/the solution. Warren [99] cites the planner's major deficiency as the inherent redundancy in the existence of plans that are identical except for the precise ordering of the actions involved. Sacerdoti's ABSTRIPS [82], which we will review in Section 4.2, resolves this problem by creating plans that are only partially ordered with the objective of avoiding unnecessary orderings.

3.3 UCPOP

Directly motivated by Chapman's TWEAK [16] and McAllester and Rosenblitt's SNLP [53], Penberthy and Weld's UCPOP [69] is a partial order planner working with actions with conditional effects, universally quantified preconditions and effects, and universally quantified goals. Given its lineage, UCPOP is also *well-founded* like its two immediate ancestors. A *well-founded* planner is one formulated with sufficient transparency and rigor to allow for theoretical analysis; in fact, UCPOP was proved both sound and complete in [69].

We first give a brief review of UCPOP's inspirations, namely TWEAK [16] and SNLP [53], in Section 3.3.1. Following in Section 3.3.2 is a description of UCPOP's simplified algorithm. Lastly in Section 3.3.3, we conclude with an example of how it solves the Sussman anomaly.

3.3.1 TWEAK and SNLP

The first well-founded partial order planner, Chapman's TWEAK [16] is a rigorous mathematical construction of earlier partial order planners such as Sacerdoti's NOAH [80], to be described in Section 4.3. Specially, a provably correct modal truth criterion was used in establishing both the correctness and the completeness of the planner. It was Chapman's endeavor to "[define] an algebra of plan transformations ... a body of formal theory about the ways in which interacting subgoals can be dealt with," the lack of which in NOAH had been rued by Sacerdoti in [80].

While TWEAK works on a given problem, it has a continually evolving, incomplete plan which it defines incrementally by specifying partial descriptions or constraints that must be satisfied. In fact, the incomplete plan represents a class of exponentially many plans, each of which is one way the incomplete plan could be completed, depending on the constraints added. Planning is finished when all the completions of the incomplete plan solve the given problem. Adding a constraint to a plan can often eliminate all the completions, resulting in an inconsistent set of constraints and the invalidity of the incomplete plan; at this point, backtracking occurs. Chapman devised a polynomial time algorithm to compute the possible and necessary properties of an incomplete plan.

Central to Chapman's formulation is the *modal truth criterion*. The notions of *necessity* and *possibility* employed in this criterion refers to what is entailed by, and what is allowed by (consistent with), the current plan constraints, respectively. A proposition q is *necessarily true* in a state s iff two conditions[1] hold, namely (i) there exists a state t either identical or necessarily prior to s in which q is necessarily asserted, and (ii) for every step A possibly before s and for every proposition p possibly codesignating with q which A denies, there exists an intervening step B necessarily between A and s which asserts a proposition r, such that r and q codesignate whenever q and p codesignate. The *possible truth* criterion reads analogously, with "necessary" and "possible" interchanged. Codesignation is an equivalence relation on variables and constants, specifically the binding of variables appearing in preconditions and in post-conditions to particular constants.

TWEAK then achieves a goal by interpreting the modal truth criterion as a nondeterministic procedure. As the criterion specifies all the ways a proposition could be necessarily true, the procedure nondeterministically picks one of them and modifies the incomplete plan

[1] In plainer English, q needs to be established by some step prior to s, and if q is clobbered, then it needs to be rescued again, prior to s. In fact, later authors observed that this "white knight" condition (for rescuing q) was a quite unnecessary complication, as one just needs to ensure that there is a prior state s' in which q is established and that no step that potentially lies between s' and s clobbers q.

accordingly. The procedure can add temporal constraints and codesignation constraints; the former to stipulate that some step occur before another, and the latter to enforce the codesignation or non-codesignation of two propositions. Should newly added constraints conflict with existing ones, backtracking occurs in which certain constraints are retracted and new ones incorporated. Although shown pictorially as partially ordered plans (then known as *task networks*), TWEAK plans were composed of non-hierarchical STRIPS-like actions, and consequently encompassed no logical reasoning or state constraints.

McAllester and Rosenblitt's SNLP [53] (Systematic Non-Linear Planner) is a simple, sound, and complete partial order planner. Since SNLP handles only propositional goals, its STRIPS-like operators have no parameters but only atomic preconditions and post-conditions; therefore, SNLP is expressively weaker than TWEAK. SNLP operates by generating a tree of alternative plans, deciding which plan to refine next, until it obtains a complete and flawless plan. Each plan to be further refined has an associated *agenda* of flaws yet to be fixed. Once SNLP selects a specific flaw for repair, it entertains all possible ways of fixing the flaw and, thus, generates alternative successor plans accordingly.

Flaws are either threats, or unsatisfied goals or preconditions. Threats are encapsulated in the formal notion of a *causal link*; a *causal link*, going from one action to another to indicate their dependency, requires that a condition produced by the first action be preserved until after the execution of the second action and thus essentially guards the condition. For instance, a causal link $A_b \rightarrow^Q A_c$ indicates that action A_b has an effect Q which achieves precondition Q for action A_c. Accordingly, a threat is realized in a situation where an action yields a condition contradicting some goal guarded by a causal link. A threat can then be repaired and avoided by placing a threatening action either before or after the threatened causal link, corresponding respectively to *demoting* or *promoting* the action with respect to the causal link. On the other hand, an unsatisfied goal or precondition can be resolved by inserting a new action, or using an existing action or initial conditions, to satisfy it.

3.3.2 Algorithm

POP and UCPOP [69] are both generalizations of SNLP to accommodate operators with open parameters, but UCPOP additionally accepts universally quantified preconditions, effects, and goals (hence the first letter "U") as well as conditional effects (hence the second letter "C"). As in SNLP, both POP and UCPOP work by generating a tree of alternative plans, deciding which plan to refine next, until a complete and flawless plan is attained. Each plan in need of further refinement also has an agenda of yet to be repaired flaws which

are either threats to causal links, or unsatisfied goals or preconditions.

Allowing for open parameters necessitates keeping a set of *binding constraints* for each alternative plan in POP and UCPOP. *Binding constraints* are either *EQ*-constraints or *NEQ*-constraints; the former are added by unifications when goals or preconditions are created as a result of performing an action, while the latter typically arise when an operator's preconditions require that two parameters be distinct, or parameters be different from certain constants. Typical *NEQ*-constraints, though considered a part of an operator's preconditions, are not effected by actions but rather constrain the constants that can be used to instantiate the operator.

Allowing for open parameters also complicates threat elimination. Consider an example in which the condition $NextTo(A_2, v)$ is protected by a causal link, and an action whose effects include $\neg NextTo(u, w)$ is introduced. Then this action can potentially destroy the guarded condition, exactly if both $u = A_2$ and $w = v$ hold. The threat can be removed in two ways. In the first, *separation* approach, the planner explicitly adds either the *NEQ*-constraint $NEQ(u, A_2)$ or $NEQ(w, v)$, yielding two alternative, refined plans. In the second approach, addressing the threat is postponed until the threat is realized as definite, precisely when the plan includes *EQ*-constraints that establish both $u = A_2$ and $w = v$.

Algorithm 3, reproduced from [62], is Weld's POP algorithm [100]. It is a simplified version of UCPOP which does not handle operators with open parameters, universal quantification, conditional effects, or disjunctive preconditions. Readers interested in the full UCPOP algorithm are referred to [69].

A plan is a triple $\langle A, O, L\rangle$, where A is a set of actions $\{A_1, A_2, ..., A_n\}$ in the plan, O a set of ordering constraints on actions $\{A_i < A_j, ..., A_m < A_n\}$, and L a set of causal links indicating how actions support each other. Action A_0, identified with **start**, has no preconditions but all facts in the initial state as its effects, whereas action A_{inf}, known also as **end**, has no effects but the goal conditions as its preconditions. A causal link has the form $A_i \rightarrow^Q A_j$, indicating that action A_i has an effect Q which achieves action A_j's precondition Q. An agenda is comprised of either threats to causal links, or unsatisfied goals or preconditions; each element of an agenda is a pair $\langle Q, A_i\rangle$, where Q is a precondition of A_i that needs substantiation.

In the context of Algorithm 3, the initial, null plan has the components $A = \{A_0, A_{\text{inf}}\}$, $O = \{A_0 < A_{\text{inf}}\}$, and $L = \{\}$, and the initial agenda is a conjunction of the goals (preconditions of **end**). Algorithm 3 is a regressive planner that begins with the null plan and continually updates it by inserting new actions and removing threats; it terminates when all

Algorithm 3 UCPOP Algorithm for the Propositional Case

procedure POP($\langle A, O, L\rangle, agenda$)
  if agenda is empty **then**
    return $\langle A, O, L\rangle$ ▷ **Termination**
  end if
  $\langle Q, A_{need}\rangle \leftarrow$ a pair on agenda ▷ **Goal selection**
  ▷ **Action selection**
  $A_{add} \leftarrow$ an action which adds Q, and which is either a new action or nondeterministically chosen from A
  if A_{add} is null **then**
    return failure
  end if
  $L' \leftarrow L \cup \{A_{add} \rightarrow^{Q} A_{need}\}$
  $O' \leftarrow O \cup \{A_{add} < A_{need}\}$
  if A_{add} is a new action **then**
    $A' \leftarrow A \cup \{A_{add}\}$
    $O' \leftarrow O' \cup \{A_0 < A_{add} < A_{\inf}\}$
  else
    $A' \leftarrow A$
  end if
  ▷ **Update goal set**
  $agenda' \leftarrow agenda \setminus \{\langle Q, A_{need}\rangle\}$
  if A_{add} is a new action **then**
    for each conjunct Q_i of A_{add}'s precondition **do**
      $agenda' \leftarrow agenda' \cup \{\langle Q_i, A_{add}\rangle\}$
    end for
  end if
  ▷ **Causal link protection**
  for each action $A_t \in A$ which threatens a causal link $A_p \rightarrow^{Q} A_c$ **do**
    $oc \leftarrow$ a nondeterministically chosen ordering constraint either (a) **demotion** $A_t < A_p$, or (b) **promotion** $A_c < A_t$
    $O' \leftarrow O' \cup \{oc\}$
    if neither ordering constraint added results in a consistent O' **then**
      return failure
    end if
  end for

  call POP($\langle A', O', L'\rangle, agenda'$) ▷ **Recursive invocation**
end procedure

preconditions of each action in the plan are supported by some unthreatened causal links.

3.3.3 An Example

Consider the Sussman anomaly example [90]. Below we show pictorially and explain how UCPOP successfully solves the Sussman anomaly, which is unsolvable by linear planners like STRIPS.

The planning problem in the Sussman anomaly is to transform the initial state in which a clear-topped block C is atop a block A resting on a table with a clear-topped block B resting on the table, into a goal state in which a clear-topped A is atop B which is, in turn, atop C resting on the table. Furthermore, it is assumed that there is only one action schema in the example: $Move(x, y, z)$ with preconditions $Clear(x)$, $Clear(y)$, and $On(x, y)$; delete-effects $On(x, y)$ and $Clear(z)$; add-effects $On(x, z)$, $Clear(y)$, and $Clear(Table)$.

Figure 3.3 depicts the initial state at the outset of Algorithm 3. The set of ground literals after a box denoting an action instance corresponds to the action's effects, while the set of ground literals before a box denoting an action instance represents the action's preconditions.

In step 1 as illustrated in Figure 3.4, $\langle On(A, B), *end* \rangle$ is picked and removed from the agenda, and accordingly, an action instance $Move(A, Table, B)$ achieving $On(A, B)$ is added along with its corresponding ordering constraint. As well, the preconditions of $Move(A, Table, B)$ form new pairs which are inserted into the agenda. A causal link is created from $Move(A, Table, B)$ to $*end*$ contingent on the ground literal $On(A, B)$, and the link remains unthreatened by any other action instances.

Step 2 as shown in Figure 3.5 picks and removes $\langle Clear(A), Move(A, Table, B) \rangle$ from the agenda, and then adds an action instance $Move(C, A, Table)$ which achieves $Clear(A)$. The preconditions of this action instance form new agenda elements. A causal link is added from $Move(C, A, Table)$ to $Move(A, Table, B)$ to indicate the dependency of the ground literal $Clear(A)$; additionally, an ordering constraint imposing $Move(C, A, Table)$ before $Move(A, Table, B)$ is added. At this point, the two causal links remain unthreatened by any other action instances.

We describe what happens in the next three steps. In step 3, $\langle Clear(C), Move(C, A, Table) \rangle$ is picked off the agenda, action $*start*$ is selected as $Clear(C)$ already holds in the initial state, and a causal link is added from $*start*$ to $Move(C, A, Table)$ contingent on $Clear(C)$. In step 4, $\langle On(C, A), Move(C, A, Table) \rangle$ is picked off the agenda, action $*start*$ is selected as $On(C, A)$ already holds in the initial state, and a causal link is added from $*start*$

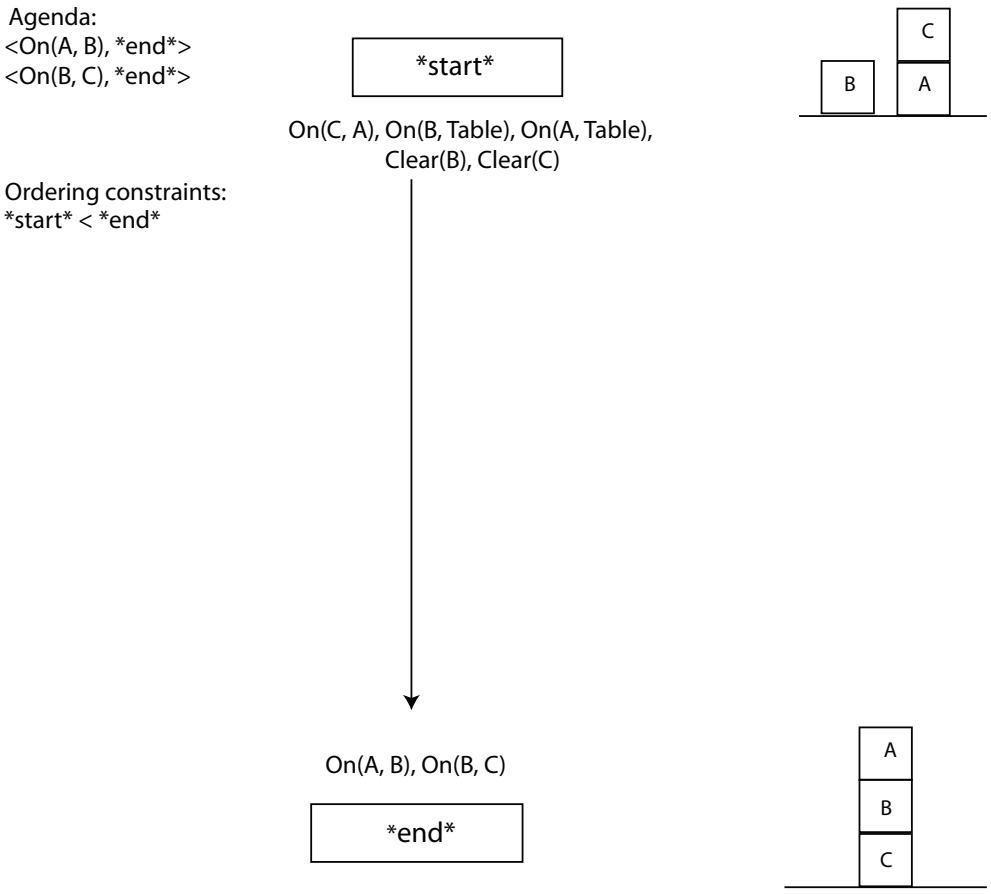

Figure 3.3: Initial State in UCPOP's Sussman Anomaly Example

to $Move(C, A, Table)$ contingent on $On(C, A)$. In step 5, $\langle Clear(B), Move(A, Table, B)\rangle$ is picked off the agenda, action $*start*$ is selected as $Clear(B)$ already holds in the initial state, and a causal link is added from $*start*$ to $Move(A, Table, B)$ contingent on $Clear(B)$. In each of these three steps, all the causal links remain unthreatened, and the ordering constraints stay the same.

In step 6 as pictured in Figure 3.6, $\langle On(A, Table), Move(A, Table, B)\rangle$ is picked off the agenda, action $*start*$ is selected as $On(A, Table)$ already holds in the initial state, and a causal link is added from $*start*$ to $Move(A, Table, B)$ contingent on $On(A, Table)$. All the causal links remain unthreatened, and the ordering constraints stay the same.

Step 7 illustrated in Figure 3.7 first picks $\langle On(B, C), *end*\rangle$ off the agenda, then chooses an action instance $Move(B, Table, C)$ to achieve $On(B, C)$, and adds the corresponding ordering constraint. A causal link is created from $Move(B, Table, C)$ to $*end*$ dependent on $On(B, C)$. At this point, $Move(B, Table, C)$ threatens the causal link from $*start*$ to

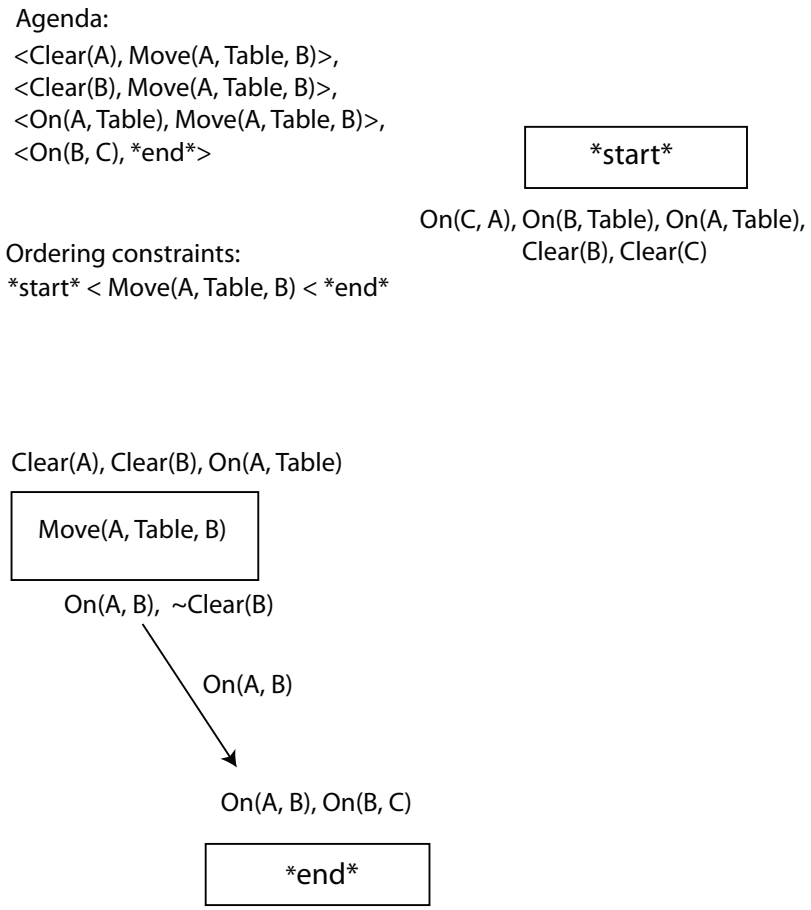

Figure 3.4: Step 1 in UCPOP's Sussman Anomaly Example

$Move(C, A, Table)$ contingent on $Clear(C)$. To resolve this threat, let's choose to promote $Move(B, Table, C)$ by adding the ordering constraint $Move(C, A, Table) < Move(B, Table, C)$.

In the next step, $\langle Clear(B), Move(B, Table, C)\rangle$ is picked off the agenda, action $*start*$ is selected as $Clear(B)$ already holds in the initial state, and a causal link is added from $*start*$ to $Move(B, Table, C)$ dependent on $Clear(B)$. At this point, $Move(A, Table, B)$ threatens the causal link from $*start*$ to $Move(B, Table, C)$ contingent on $Clear(B)$. To resolve this threat, let's choose to promote $Move(A, Table, B)$ by adding the ordering constraint $Move(B, Table, C) < Move(A, Table, B)$. The plan solution now is the sequence of action instances $Move(C, A, Table)$, followed by $Move(B, Table, C)$, followed by $Move(A, Table, B)$. The remaining open conditions on the agenda can be directly achieved by the $*start*$ action as they already hold in the initial state; furthermore, no new threats will be introduced.

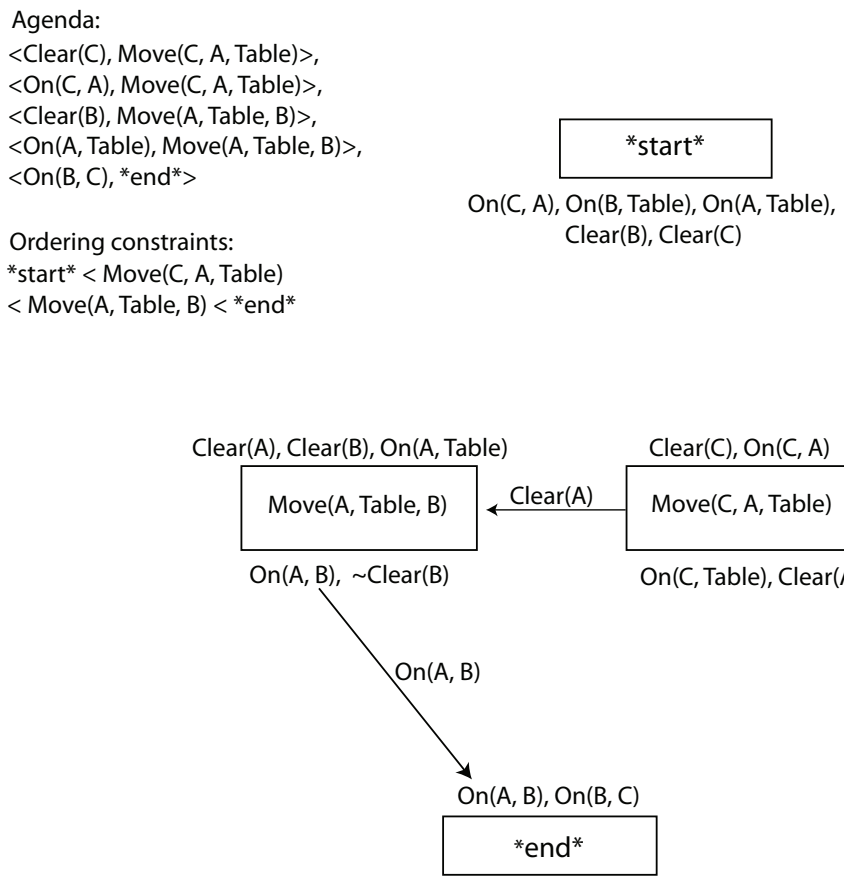

Figure 3.5: Step 2 in UCPOP's Sussman Anomaly Example

3.4 Graphplan

Graphplan, due to Blum and Furst [12], is a sound and complete partial order planner that plans in STRIPS-like domains by constructing and analyzing a *planning graph* structure. Graphplan always returns a shortest possible partial order plan if one exists, given a *planning problem*. A *planning problem* consists of a set of operators with preconditions, delete-effects, and add-effects, all of which are conjunctions of literals; a set of objects; a set of literals as the initial conditions; and a set of literals as the problem goals required to be true at the end of a plan. Not only can a planning graph be constructed in polynomial time and is thus of polynomial size, but it can also help organize and maintain search information so as to guide the search for a plan [12].

Pivotal in Graphplan is the *planning graph* structure. A *planning graph* is a leveled graph comprised of three kinds of edges and two kinds of nodes. The levels alternate between *proposition levels* and *action levels*. The former are composed of *propositional nodes* each

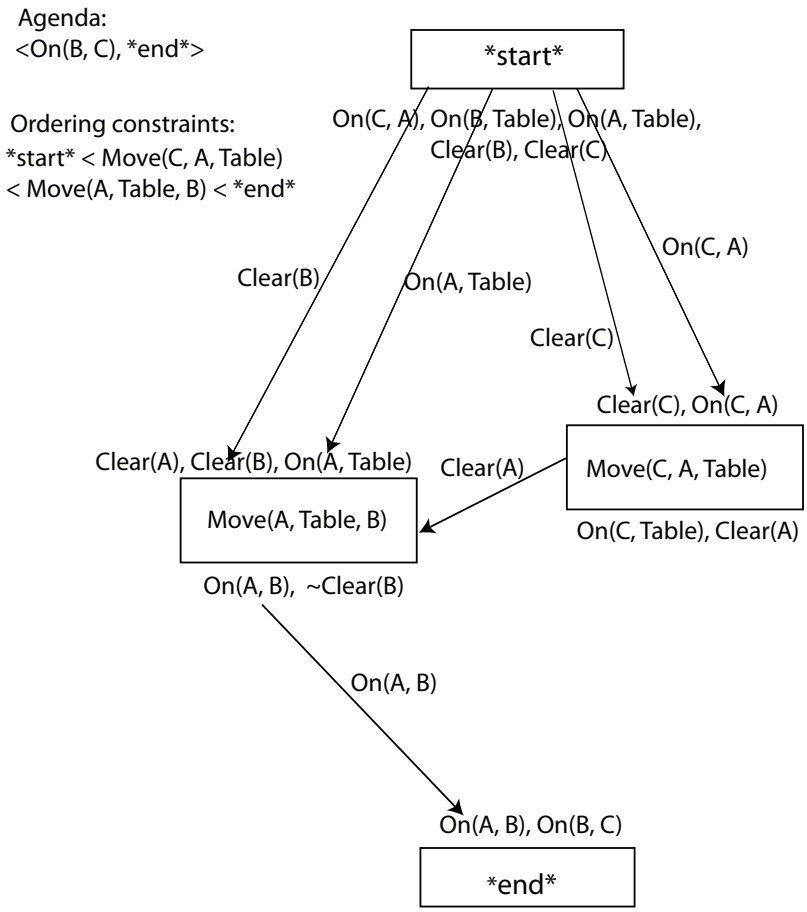

Figure 3.6: Step 6 in UCPOP's Sussman Anomaly Example

labeled with a ground literal, and the latter of *action nodes* each labeled with an action. An action is either a fully instantiated operator, or a null action, whose sole purpose is to model the persistence of state properties by propagating exactly one ground literal from one propositional level to the next proposition level. Edges represent relations between actions and propositions. The action nodes in action level k are connected by the *add-edges* and *delete-edges* to their corresponding add-effects and delete-effects in proposition level $k+1$, respectively, and by the *precondition edges* to their respective preconditions in proposition level k.

The levels of the graph are, in chronological order from left to right, propositional level 1 (level 0), action level 1 (level 1), propositional level 2 (level 2), action level 2 (level 3), and so on. Level 0 consists solely of all the ground literals true initially; level 1, only of all the action instances possible initially, each connected by edges to the level 0 ground literals

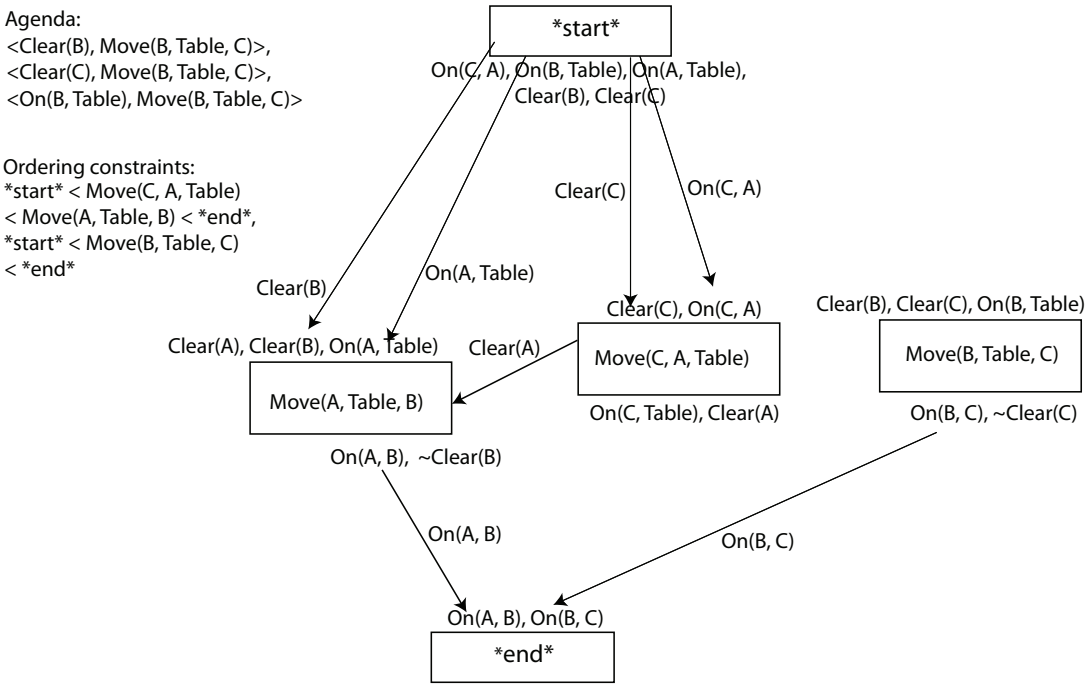

Figure 3.7: Step 7 in UCPOP's Sussman Anomaly Example

constituting its preconditions. At level 2 are all the ground literals that either are the effects of the level 1 action instances, or are inherited by persistence from level 0. Action instances that follow at level 3 are all the possible action instances given the level 2 ground literals.

Pairwise *mutual exclusion relations* among nodes are an integral part of the analysis a planning graph, as they help prune the search for a subgraph that might correspond to a valid plan. Two propositions at a proposition level are *mutually exclusive* and connected by a mutex link if no valid plan that could make both true exists; this applies analogously to two actions at an action level.

Graphplan identifies and records mutual exclusion relations by propagating them through a planning graph using the following criteria [69]:

- Two actions a and b at an action level are marked as exclusive of each other, if either action deletes a precondition or add-effect of the other, or if a precondition of a and a precondition of b are both marked as exclusive of each other at the previous proposition level;

- Two ground literals p and q at a proposition level are marked as exclusive of each other, if all ways of creating p are exclusive of all ways of creating q, i.e., if each action with an add edge to p is marked as exclusive of each action with an add edge to q.

Graphplan works in a loop interleaving two main tasks, namely the construction of a planning graph, and the extraction of a plan solution from the constructed graph [78]. Graphplan first checks if all the original goal literals are present at the current level in the graph without any pairwise mutex links. If so, then a solution might exist in the graph so Graphplan tries to extract a solution. Otherwise, Graphplan expands the graph by adding an additional action level followed by a propositional level. The loop continues until either a solution is found, or it has been found that no solution exists.

Extracting a plan solution from a planning graph makes use of backward chaining. Assuming the original goal conditions are true and non-exclusive of each other at the highest level t of the graph, Graphplan finds a set of level $t-1$ non-exclusive actions that generate these goals. Then the preconditions of these level $t-1$ actions become a set of subgoals at level $t-1$, reducing the problem to a smaller one of showing that these subgoals can be accomplished in $t-1$ steps. The rationale is that the existence of a solution to the smaller problem implies that the original goal conditions can indeed be achieved in t steps. If the set of subgoals turns out to be unsolvable, Graphplan backtracks and selects alternative actions in the regression. A plan solution exists iff Graphplan successfully regresses to the initial state. [78] outlines a proof establishing that Graphplan always terminates on any given planning problem.

Solutions to large planning problems are usually found by orders of magnitude faster in Graphplan than in such regression planners as STRIPS and UCPOP. This is primarily attributed to the substantially tighter constraints on the regression search imposed by the propagated ground literals, actions, and mutual exclusion relations. Admittedly, regression in general remains exponentially complex.

There are three main deficiencies of Graphplan. First, since Graphplan works only in STRIPS-like domains, it cannot handle certain planning situations, particularly those in which actions can create new objects or the effect of an action cannot be determined statically. For instance, given the action "give every student in this class a grade of A," one cannot determine the effect of this action statically, as the set of students getting a grade of A depends on who happens to be in the class at the time of performing the action. Second, storing the mutex links can consume space quandratic in the number of nodes at a given level, and this large storage complexity could hamper the planning attempt. Lastly, Graphplan guarantees to find a/the shortest possible plan, if one exists, to a given problem; as a result, it may expend extra effort even in situations not requiring a/the shortest possible plan. It would be desirable to incorporate into the planning graph structure tradeoffs between the

speed of planning and plan quality [69].

3.5 SATPLAN

SATPLAN, due to Kautz and Selman [47], models planning as Boolean satisfiability, exploiting advances made in fast satisfiability (SAT) solvers. A SAT solver is a program which, given a set of propositional formulas, finds a satisfying truth assignment (if one exists) to the sentential variables appearing in the formulas. In this approach, a planning problem is translated into a set of axioms in propositional logic such that any model of the axioms amounts to a valid plan. A model of the axioms assigns true to actions that form the corresponding plan, and false to other actions. Conversely, if no plan exists, no models of the axioms exist.

A given planning problem is encoded by the following axioms:

- axioms describing the initial state at time step 0;
- axioms describing the goal at some time step T_{max};
- action exclusion axioms, asserting that only one action occurs at a time, precluding simultaneous actions;
- precondition axioms for each possible action instance at each possible time step, asserting that the action at a certain time step implies its preconditions at the same time step;
- successor state axioms for each possible action instance at each possible time step, enumerating not only the action's effect axioms, but also all the ways in which the truth value of the fluent predicates representing the effects of the actions can be changed;
- optional state constraints.

Some additional clarifications are warranted for the axioms. First, the action exclusion axioms force every plan to be totally ordered, and thus can potentially increase the computation time by increasing the number of time steps in the plan. This potential overhead can be alleviated by requiring only partial exclusion, i.e., disallowing only mutually interfering, simultaneous actions, or by using state constraints in lieu of action exclusion axioms [78]. Third, an extra time argument is added to fluent predicates to indicate what is true at

various times; e.g., the atom $ON(C, D, 3)$ can be taken to state that block C is atop block D at time 3. By an action instance is meant an instantiated, ground action literal. Ground literals are treated as Boolean sentential variables.

Algorithm 4 SATPLAN Algorithm

```
1: procedure SATPlan(problem, T_max)
2:     for T = 0 to T_max do
3:         cnf, mapping ← Translate-to-SAT(problem, T)
4:         assignment ← SAT-Solver(cnf)
5:         if assignment is not null then
6:             return Extract-Solution(assignment, mapping)
7:         end if
8:     end for
9:     return failure
10: end procedure
```

Algorithm 4 from [78] takes as input a planning problem *problem* and an upper limit for plan length T_{max}. T_{max} is arbitrarily chosen and provided only to ensure termination; one can gradually increase T_{max} as deemed necessary. First, *problem* is translated into a conjunctive normal form sentence comprised of axioms for each time step up to and including time step T, as well as assertion of the goal at T. Then, if the employed SAT solver finds a model of the sentence, a plan is extracted and returned consisting of actions whose corresponding sentential variables are assigned true in the model. Otherwise, the algorithm repeats with the goal advanced one time step further.

According to [78], the primary drawback of SATPLAN is the size of the propositional knowledge base generated from the original planning problem. Notably, the number of action instances generated from each action schema is exponential in the arity of the schema; e.g., the action schema $Fly(p, a_1, a_2, t)$ denoting that plane p flies from a_1 to a_2 at time step t yields $|Planes| * |Airports|^2 * |T|$ different ground action literals, where $|T|$ is the size of the discrete time steps considered. Generally, the total number of ground action literals in the problem is upper bounded by $|T| * |Act| * |D|^A$, where $|Act|$ is the total number of action schemata, $|D|$ is the total number of objects in the domain, and A is the maximum arity of any action schema.

Since the arity of the action schema is the dominating exponential factor in the size of the propositional knowledge base, [78] suggests applying *symbol splitting* to an action symbol. *Symbol splitting*, borrowed from semantic networks, reduces predicates with more than two arguments to a set of binary predicates describing each argument separately. As a

result of symbol splitting, the total number of ground action literals in the problem is now upper bounded by $|T| * |Act| * |D| * A$. Furthermore, [78] claims that it also reduces the size of the knowledge base as some split predicates will be impertinent to, and thus omitted from, certain axioms. One caveat is that the inclusion of action exclusion axioms barring any parallel actions is now absolutely required, in order to understand the truncated, split predicates. See [78] for an example.

In the "rocket" domain [12] and the "logistics" domain [94][46], [46] reports that SATPLAN is orders of magnitude faster than Graphplan, while finding optimal plans. The reasons are twofold. SATPLAN benefits from the speed of relatively newer SAT solvers (particularly WalkSAT), and the direct propositional encodings abstract away subgoal ordering and the action's interactions. On the other hand, SATPLAN is memory-intensive and not complete when using stochastic SAT procedures. If the optimal plan length cannot be determined in advance, SATPLAN must work iteratively gradually increasing the T_{max} bound. Furthermore, SATPLAN seems incapable of conditional planning and of interleaving planning and executing; [46] attributes this limitation to SATPLAN's undirected search for a plan solution in contrast to the guided search in progression or regression planners.

Chapter 4

Hierarchical Planners

Hierarchical planning arose from the discontent with the inefficiency of non-hierarchical planning; in particular, the latter could waste a significant amount of time searching for solutions to non-critical parts of a plan, only to find a more critical part of the plan unsolvable. A hierarchical planner, in contrast, divides a domain into various abstraction levels, providing a hierarchical representation of a plan. In such a representation, a top level abstract plan leaving details unspecified is first constructed, successively lower levels realize increasingly refined subplans, and the lowest level is sufficiently detailed to solve a given planning problem if a solution exists.

Yang [105] suggested two approaches in which details can be inserted into a plan. The first approach is formalized in hierarchical task network planning, in which a high level task is successively decomposed into a partially ordered set of lower level tasks, until only primitive tasks remain in a plan. Hierarchical task network planning is reviewed extensively in Section 4.3.1. The second approach, providing an abstraction through precondition elimination, emulates how a human would explore and solve subgoals in an order designated by their relative priorities. This approach is embodied in ABSTRIPS [79], examined in Section 4.2.

In this chapter, we survey several hierarchical planners: Fikes and Nilsson's STRIPS augmented with triangle tables [29] in Section 4.1; Sacerdoti's ABSTRIPS with a hierarchy of abstraction spaces [79] in Section 4.2; Sacerdoti's NOAH [80, 81], the very first partial order and hierarchical task network planner, in Section 4.3; NOAH's related, successor systems NONLIN [91], O-PLAN [92], and SIPE [101] in Section 4.3.4; and Vere's DEVISER [95], the first planner with temporal considerations, lastly in Section 4.4. Our discussion is supplemented with algorithmic descriptions and examples.

4.1 STRIPS Triangle Tables

In 1972, Fikes, Hart, and Nilsson [29] proposed the addition of *triangle tables* to STRIPS for the dual purposes of both self-learning and plan-monitoring in STRIPS. *Triangle tables* [67] are used to store generalized plan solutions to a family of tasks; consequently, used as MACROPS, they enable STRIPS to define new operators, understood synonymously with action schemas, on the basis of previously discovered problem solutions, and to apply the new operators to more challenging problems. Furthermore, generalized plans assist the STRIPS planner in responding to unexpected consequences of actions in the world.

Our detailed discussion is decomposed into several subsections. In Section 4.1.1, we present the syntax and semantics of triangle tables. Their structure and meaning naturally place STRIPS with triangle tables in the paradigm of hierarchical planning. Section 4.1.2 describes the generalization procedure, namely that of generalizing a plan by replacing problem-specific constants in the plan by problem-independent parameters. The integral role played by the triangles tables in guiding and monitoring a plan's execution is discussed in Section 4.1.3.

4.1.1 Syntax and Semantics

A triangle table of rank N is an N x N lower triangular array of cells, each possibly containing a collection of formulas possibly involving schema variables. The rows are numbered from the top, starting with row 1 or time-step 1; the columns, numbered from the left starting with column 0. Each column but column 0 is headed by an action schema. The schema variables in a formula are a subset of those action schema variables in the action, if any, heading the column containing that formula. The schema variables in an action schema are a subset of those formula schema variables appearing in the cells in the row to the left of that action schema.

Figure 4.1 is the general schema of a rank-5 triangle table for the plan consisting of a sequence of 4 action schemas OP_1, OP_2, OP_3, and OP_4. For each column $i \in \{1, 2, 3, 4\}$, A_i, which is the add-effect(s) of OP_i, is placed in the top cell, with the portion of A_i surviving the application of subsequent action schemas placed in consecutive cells below. For example, A_1 is the formulas added by OP_1; $A_{1/2}$ is A_1 minus formulas deleted by OP_2; $A_{1/2,3}$ is A_1 minus formulas deleted by OP_2 or by OP_3; and so forth for $A_{1/2,3,4}$. The conjunction of the formulas in the last row of a triangle table is the effect of the plan represented by the table.

A *marked* formula in a triangle table is one prefixed with an asterisk. The conjunction of

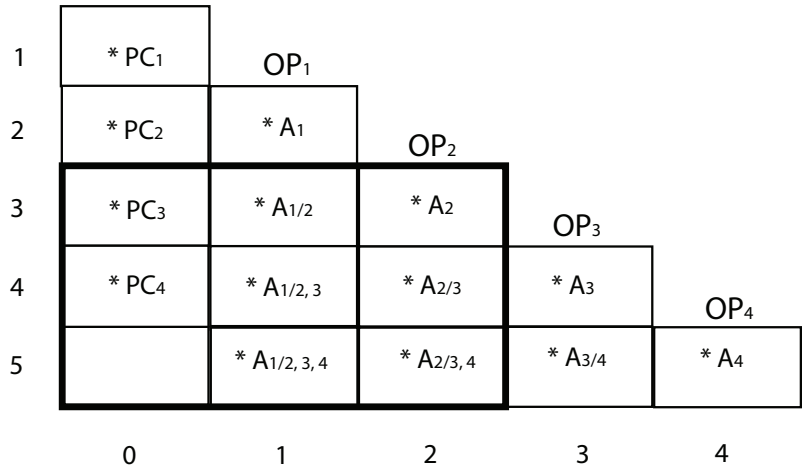

Figure 4.1: The General Schema of a Rank 5 Triangle Table

all the marked formulas in the row to the left of an action schema constitutes the precondition(s) for performing that action schema. All formulas in column 0 are marked, by default, and while all other formulas are also marked in Figure 4.1, this certainly need not be the case for all triangle tables. For each $i \in \{1, 2, 3, 4\}$, PC_i is precisely the precondition(s) of OP_i true in the initial world model and not deleted by any of the first $i - 1$ action schemas. Since column 0 of a table gives a set of sufficient conditions for the applicability of the entire plan represented by the table, the conjunction of all column 0 formulas is considered the precondition of the entire plan.

An example triangle table from [68] is illustrated in Figure 4.2, in which the tilda denotes the logical *not* operator. The expressions $a_1(x)$, $a_2(y)$, and $a_3(y)$ are action schemas, while $A \wedge B(x)$, $D(y)$, $\neg C(x)$, $E \wedge F$, $G(y)$, H, and I are formula schemas.

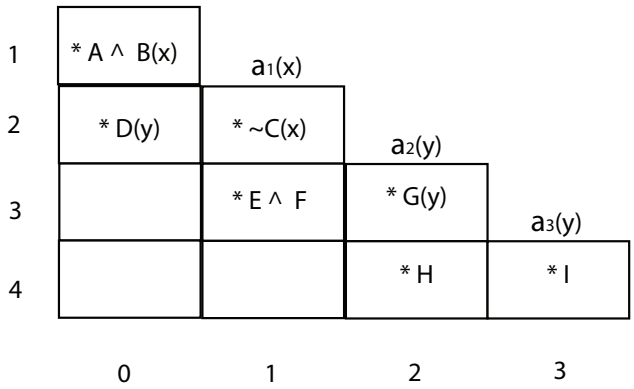

Figure 4.2: An Example Triangle Table

In Figure 4.2, for instance, $D(y) \wedge \neg C(x)$ forms the preconditions of $a_2(y)$, meaning that if any instance of the preconditions is true, then the corresponding instance of a_2 can be performed. Instances of the formula $G(y)$ are effects of corresponding instances of the action schema $a_2(y)$, with H being an additional effect of all instances of $a_2(y)$.

By definition, the *jth kernel* of a rank-N triangle table is the conjunction of all the marked formulas in the unique rectangular subtable consisting of the bottom $N - (j - 1)$ rows of the leftmost j columns. Instances of the jth kernel are sufficient conditions for corresponding instances of the action schema sequence $\langle OP_j, OP_{j+1}, ..., OP_N \rangle$ to be executable and to achieve the effects appearing in the last row of the table. For instance, the heavily outlined box in Figure 4.1 is the third kernel, providing sufficient conditions for performing the action schema sequence $\langle OP_3, OP_4 \rangle$.

4.1.2 Plan Generalization Procedure

The motivation for plan generalization in a learning system is readily apparent. Triangle tables enable STRIPS to solve problems more quickly, if a problem of the same type was previously solved and abstracted as a triangle table. In fact, speed-ups by a factor of 5 were reported for Shakey the robot using STRIPS with triangle tables.

Consider the specific plan found for the example from Section 3.1.2: $\langle goThru(doorA,$ $kitchen, supplies), pushThru(box_1, doorA, supplies, kitchen) \rangle$. While this sequence solves the original example, one probably cannot justify saving it unless one expects the robot would frequently need to go from *kitchen* through *doorA* to *supplies* to push back the specific box box_1 back into *kitchen* through *doorA*. Rather, it would be more useful to generalize the plan so that it could be used in situations involving arbitrary rooms, doors, and boxes.

The plan generalization procedure [29] constructs generalized triangle tables by abstraction from successful plans. Given a triangle table, the procedure builds a corresponding generalized plan, also known as a *MACROP*, as follows:

(1) Replace each occurrence of a constant in the column 0 formulas of a given triangle table by a new parameter, while substituting distinct parameters for multiple occurrences of the same constant. Go to step (2).

(2) Fill in the remainder of the table with appropriate add-effect formulas assuming completely uninstantiated action schemas (i.e., as these add-effect formulas appear in the action schema descriptions), and assuming the same deletions as in the original

table. Go to step (3).

(3) Redo each action schema's precondition proof using both the marked formulas in the *lifted* table as axioms, and the precondition formulas from the action schema descriptions as the theorems to be proved. Go to step (4).

(4) Make uniformly throughout both the *lifted* table and the general plan, substitutions of parameters for constants or for other parameters in the new proofs.

A few clarifications are in order. The *lifted* table obtained after steps (1) and (2) is the most general form of the given triangle table; therefore, step (3) determines constraints that could be imposed on the lifted table to restrict its generality. Most importantly, any constraints should ensure that the marked formulas in each row remain the preconditions of the action schema of that row, and that the original table stays an instance of the lifted table. Each new proof in step (3) is isomorphic to the corresponding original proof of preconditions in that at each step, resolutions on the same clauses and unifications on the same literals are performed as in the original proof. The substitutions made in step (4) function as constraints on the generality of the plan. Before a generalized plan is stored, two refinements are made to remove possible inconsistencies and to increase efficiency; refer to [29] for additional details.

4.1.3 Execution and Plan-Monitoring

To understand how triangle tables are used in guiding and monitoring plan execution, let's turn our attention to Shakey the robot [67], an SRI project involving planning with STRIPS in conjunction with MACROPs. In particular, our discussion is centered on the PLANEX system, which is used as both a plan executor and a plan monitor in Shakey. We use MACROPS and triangle tables interchangeably in the ensuing discussion.

Before PLANEX can use a MACROP, the MACROP's parameters must be partially instantiated using the specific constants of the goal formula, or of the conjunction of all the given goal formulas if given more than one goal formula. The partial instantiation is effected by placing in the lower leftmost cell of the MACROP those formulas from the initial world model which were used by STRIPS to prove the goal formula. Then the goal formula is proved using only a subset of all formulas in the last row of the MACROP, with substitutions made in the proof applied to the whole MACROP. Furthermore, those formulas in the last row used in the proof of the goal formula are marked. PLANEX uses this version of the MACROP below to control execution.

PLANEX attempts to execute a MACROP when it wishes to achieve an instance of the MACROP's effect. In order to reflect the heuristic that it is best to execute the legal action least removed from the goal, PLANEX finds the highest indexed kernel i whose marked formulas are all true in the current world model and executes the corresponding action OP_i. Such a kernel i is called the *active kernel*; and OP_i, the *active operator* or the *active action*. Action sequences associated with active kernel instances are called *active sequences*. Whenever there is an active kernel, executing a corresponding active sequence will achieve the corresponding instance of the effects of the MACROP. As a result, to execute a MACROP is to compute and then execute its active operator.

If the goal kernel (the last row) is true in the current world model, execution halts; otherwise, PLANEX determines whether the second-to-last kernel is true by checking whether *some* instance of the conjunction of the marked clauses in the kernel can be proved from the current model, and so on, until it finds a true kernel i along with its corresponding tail of the plan $\langle OP_i, ..., OP_N \rangle$, where N is the rank of the MACROP. Then PLANEX executes the corresponding action instance of OP_i. Next, PLANEX checks the outcome in the new world model by searching for its highest indexed true kernel. On the other hand, if no kernels are true in the current world model, PLANEX passes control back to STRIPS so STRIPS can replan by reinstantiating parameters of action schemas [29].

To find the highest indexed kernel with all its marked formulas true, PLANEX scans the MACROP cell by cell, evaluating each cell as either true if all its marked formulas are true in the current model, or false otherwise. A kernel is *potentially true* at some point in the scan if all evaluated cells of the kernel are true. Notably, PLANEX scans the cells in a left-to-right, bottom-to-up order, succinctly described by the statement:

> Among all unevaluated cells in the highest indexed, potentially true kernel, evaluate the leftmost cell. Break "leftmost ties" arbitrarily. [Fikes, Hart, & Nilsson, 1972, page 271 of "Learning and Executing Generalized Robot Plans"]

[29] claims this scanning algorithm is optimal because on average, it evaluates fewer cells than does any other scan guaranteed to find the highest indexed true kernel, although the claim has not been proved.

In addition to assisting the STRIPS planning system in learning, triangle tables are also an efficient representation that readily supports plan monitoring. Particularly, with the aid of triangle tables, PLANEX can identify the role each action plays in an overall plan, specifically what its important effects are and why these effects are required in the plan. At every step, triangle tables also endow PLANEX as the plan monitor with the ability to

answer the following questions [29]:

- Has the portion of the plan executed thus far produced the expected results?
- What portion of the plan need be executed next so that the intended task will be completed after its execution?
- Can this portion to be executed next be executed in the current state of the world?

In addition to their instrumental role in learning and monitoring a plan, triangle tables boast several other strengths. First, finding and constructing plans from previously stored generalized MACROPS can significantly reduce the time taken to solve a planning problem. For instance, speed-ups by a factor of 5 were realized for Shakey the robot using STRIPS with triangle tables.

Their tabular representation readily makes triangle tables amenable to hierarchical and reactive planning [68]. Although so far the action schemas in a MACROP are treated as primitive, one could make an action schema non-primitive by programming it as a triangle table. When a non-primitive action schema is invoked, control could be transferred to its triangle table, the *called* triangle table, to execute only the active action in the subordinate table. This might result in a new active action in the *calling* triangle table. This scheme goes down a hierarchy of tables until primitive actions are reached and executed. In this hierarchy the topmost table has the ultimate control, passing action selection downward through the hierarchy until a primitive action is selected and executed, and only then does the topmost table start the process over again. The advantage of this hierarchical scheme is that since active actions are determined for all the calling tables in the hierarchy after a primitive action is executed, the feedback loops are short, and the actions always are pertinent to the current state of memory. As a result, as long as changes are perceived and recorded in memory, the planning system, not committed to "open-loop" execution of action sequences, can react to unexpected changes in the world appropriately and quickly [29].

As a result of the order in which PLANEX scans the cells of a MACROP to find the highest indexed true kernel, PLANEX can sometimes fix certain mistakes made by STRIPS [29]. Perhaps due to the use of an inadequate search heuristic, STRIPS occasionally generates a plan containing a superfluous subsequence – a subsequence of the form $\langle OP, OP^{-1} \rangle$, where OP^{-1} negates exactly OP's effects. During plan execution, PLANEX would detect that the world model immediately following the application of OP^{-1} is the same as that

immediately preceding the application of OP; consequently, PLANEX would not execute the superfluous subsequence.

Although MACROPS do not have the form standardly assumed in hierarchical planning, i.e., simply steps along with a set of preconditions and effects, the extra complexity in their tabular representation affords a plan monitor the ability to readily identify the role each action plays in an overall plan, specifically what its important effects are and why these effects are required in the plan. It is worth noting that a MACROP also compactly conveys the inter-relationships between the sequence of action schemas it represents; e.g., which action schema is dependent on what effects of which other action schemas, which effects of which action schema are negated by which other action schemas, whether an action schema can be proved directly from the initial world model, to name a few.

Triangle tables also have limitations. First, triangle tables inherently do not encode information about their own relevance and usefulness. However, to be a planner that truly learns with a fixed-sized memory, the system should have a mechanism for forgetting old, less relevant plans in order to make room for newly learned, more relevant plans. While it is relatively easy to identify and discard a MACROP subsumed by a more powerful MACROP, no mechanism exists for the more difficult task of deciding which old plans should be retired. A possible solution would be to keep frequency statistics on how often various MACROPS are used and to dispose of those dropping below some threshold [29].

Another limitation of triangle tables is that their usefulness can be realized only to the extent that their level of detail is appropriate for the problem at hand. A planning system learning from its old plans to create its own MACROPS is, at the very least, confronted with the possibility of creating a MACROP whose first kernel is the conjunction of so many formulas that a theorem prover flounders [29]. Therefore, it is imperative that one institute a mechanism for abstracting the preconditions of a MACROP so as to keep only the main preconditions. The system would plan first with these abstracted preconditions, and then would fill in the details as required only if the initial planning phase succeeds.

4.2 ABSTRIPS

ABSTRIPS [79] (Abstraction-Based STRIPS) is a problem solver modified from STRIPS and designed by Sacerdoti in 1973. Given a problem and its STRIPS representation, ABSTRIPS defines a hierarchy of abstraction spaces and uses this hierarchy in solving the problem. ABSTRIPS is approximately five times faster than STRIPS.

The motivation for planning in an abstract space hierarchy in which successive levels of detail are incorporated stems from the characterizing deficiency of general purpose solvers like STRIPS [28] and GPS [65]. Since a general problem solver deals with some representation of the domain of a given problem, it seems plausible that a complex representation, say, a naive transcription of a complex problem domain, could exceed the capability of general purpose solvers. As [79] points out, selecting a simplifying representation inherently involves a balance between "heuristic adequacy" and "epistemological adequacy" as coined by McCarthy and Hayes [58]. The former adequacy dictates that heuristic search through the representation should be of sufficiently short duration that one can reach a goal state in the problem space, while the latter requires that the representation preserve all details needed to solve the given problem. However, it is claimed that no epistemologically adequate representation can be heuristically adequate for a sufficiently complex problem domain.

To this end, Sacerdoti proposed heuristically searching first through an abstraction space for a solution to a given problem and then later filling in the detail of the connection between the steps of the solution. An abstraction space is a simplifying representation of the problem space in which non-essential details are omitted. The steps of the solution found in the abstraction space are a series of subproblems in the original problem space. A solution to the original problem is obtained only if all the subproblems can be solved; otherwise, replanning in the abstraction space is done to explore an alternative solution. Further, extended to a hierarchy of spaces in which each higher level deals with successively fewer details, heuristic search can focus on a substantially reduced portion of the search space by considering details only when a successful plan at a higher level space lends credence to their importance. The abstraction is highly structured and dependent on the domain syntax.

For a problem solving system to be practical, we would like a sufficiently large difference between an abstraction space and its ground space and yet a not overly complex mapping, in order to achieve an improvement in efficiency. In the STRIPS context, an abstraction space differs from its ground space only in the preconditions of its operators; that is, the precondition well-formed formulas in an abstraction space will have fewer literals than those in its ground space, with the omitted literals being details attainable by a simple plan once the more important literals have been achieved. In the ABSTRIPS context, each literal within the preconditions of each operator is assigned a "criticality" value at the time the domain is first defined; only the most critical literals appear in the highest abstraction space, followed by less critical ones in lower spaces. Approaches to such an assignment can be completely automatic, or manual as part of the specification of the problem domain.

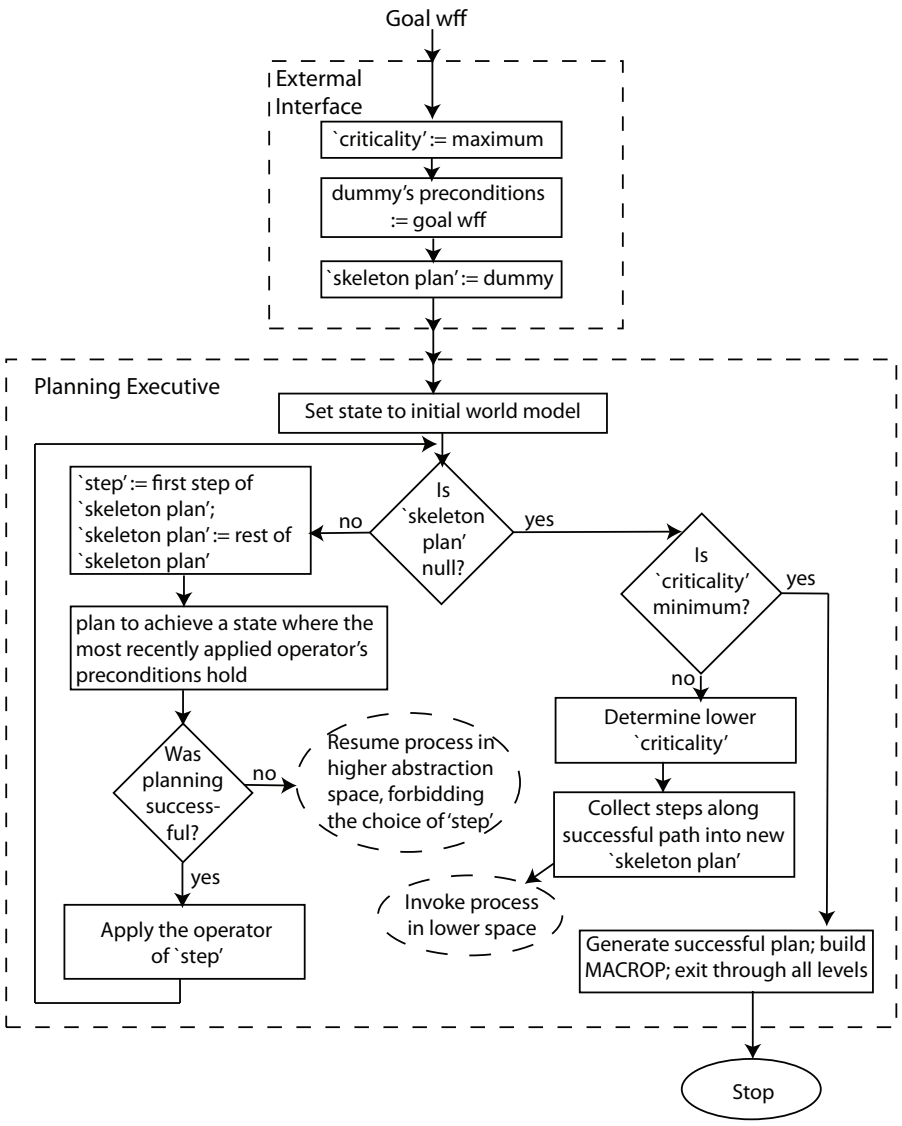

Figure 4.3: The Flow of Control of ABSTRIPS

ABSTRIPS uses a predetermined, partial ordering of all the predicates used in describing the problem domain to specify an order for considering the literals of the precondition well-formed formulas of all operators in the domain. First, all literals whose truth value cannot not be changed by any operator are assigned a maximum criticality value. Next, each remaining literal is examined in the predetermined ordering. If a short plan can be found to achieve a literal from a state in which all previously processed literals are assumed true, then the literal is a detail and is assigned a criticality equal to its rank in the ordering. Otherwise, the literal is given a criticality higher than the currently highest rank.

The ABSTRIPS system is a recursive executive program accepting two parameters. The first is a criticality value indicating the abstraction level in which planning is to occur, while the second is a list of nodes from the search tree in the higher level comprising a skeleton plan. When a new problem is presented to ABSTRIPS, the executive is invoked with the maximum criticality and the skeleton plan consisting only of a dummy operator whose preconditions are the goal well-formed formula. Recursion continues until a complete plan is constructed in the problem space. If a subproblem in a space cannot be solved, then the search tree is restored to its state before the selection of the node which led to failure in the ground space, and then after disregarding that node, the search for a successful plan at the higher level continues. Figure 4.3 demonstrates the flow of control in ABSTRIPS.

There are several implications of hierarchical planning in ABSTRIPS [79]. First, it can assist in learning task-specific knowledge. Once ABSTRIPS has built a small plan to achieve a state in which a given literal is true, ABSTRIPS recognizes that literal is a detail and saves the plan as a MACROP to be used as the first-choice pertinent operator when that detail need be achieved. The MACROPS obtained this way, along with the set of basic operators, constitute a knowledge body about how to solve problems in a given task domain.

Second, hierarchical planning in ABSTRIPS simplifies the process of creating conditional plans, plans with loops, and plans with information-gathering operators. The outcome of such operators is uncertain only to a particular level of detail, so in a higher abstraction space, a simple specification can adequately model the preconditions and effects of these operators. Third, hierarchical planning in ABSTRIPS lends itself to a fully integrated planning and execution system. Overall planning is roughed out in an abstraction space, ignoring levels of detail so that the plan is fairly certain to succeed. Using a few steps of the plan as a skeleton, we can add more detailed steps and further building and extend various subplans. Until we achieve the ultimate goal, we alternate between adding detailed steps to the plan and actually executing some steps.

LAWALY [88], a hierarchical planning system invented by Siklossy and Dreussi and also using a hierarchy of abstraction spaces in solving planning problems, emerged around the same time in 1973 as did ABSTRIPS. LAWALY was even faster than ABSTRIPS because of its avoidance of theorem proving and its use of STRIPS operators that were precompiled into LISP functions. A clear negative implication of the absence of theorem proving is that LAWALY cannot derive logical consequences to address the ramifications of actions.

The primary contribution of ABSTRIPS and LAWALY is the notion of a hierarchy of abstraction spaces to use in planning, contributing to substantial speed-ups over STRIPS. However, both systems are incomplete planners in that neither can solve the Sussman anomaly due to their unwillingness to undo a previous subtask which they wanted to achieve and have achieved. Furthermore, what is lacking in both systems is a genuine concept of higher level actions that can be elaborated into lower level ones. Hence, NOAH [82], which we examine in Section 4.3, was developed by Sacerdoti in 1975 to remedy these two deficiencies.

4.3 NOAH

NOAH [80, 81] (Nets of Action Hierarchies), designed by Sacerdoti in 1975, is a problem-solving and execution-monitoring system using a nonlinear representation of plans. In addition to being the first partial order planner, NOAH was the first *hierarchical task network* planner. NOAH's approach to planning accords well with our intuitive understanding of how high level actions can be decomposed into lower level ones, presenting a genuine account of the relationships between actions at different levels of detail. It was precisely the absence of, and the desire for, such an account in ABSTRIPS that had propelled Sacerdoti's development of NOAH.

Our detailed discussion is decomposed into several subsections. In Section 4.3.1, we describe the general *hierarchical task network* (HTN) planning paradigm. Section 4.3.2 presents NOAH's framework, including the central notion of a *procedural net*, nodes, general-purpose critics, and the planning algorithm. Following in Section 4.3.3 is an example of how NOAH solves the Sussman anomaly. Lastly, in Section 4.3.4, we conclude with an overview of related systems inspired by NOAH.

4.3.1 Hierarchical Task Network Planning

Hierarchical task network (HTN) planning is a paradigm in which the initial plan, a very high level description of what is to be accomplished for a given planning problem, is successively

refined by applying action decompositions until only *primitive tasks* remain in the plan. A *primitive task* is one that can be directly achieved by executing the corresponding action. Each action decomposition reduces a high level action to a partially ordered set of lower level actions; thus, action decompositions are embodiment of knowledge about how to really implement actions.

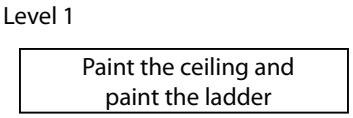

Figure 4.4: Level 1 of Procedural Net for Painting

Consider an example from [82], in which the given planning problem is to paint the ceiling and paint the ladder. A goal node is pictured as a rectangular node connected to its predecessor(s) by (incoming) edge(s) touching the left side of the node, and to its successor(s) by (outgoing) edge(s) touching the right side of the node. At the coarsest level, the plan can be represented as a single goal node as shown in Figure 4.4. A moment's thought leads one to more finely represent the conjunctive goal as in Figure 4.5, where S denotes a SPLIT node indicating a forking of the partial ordering and D denotes a JOIN node indicating a rejoining of subplans within the partial ordering. Still more detailed subplans can be obtained as in Figure 4.6

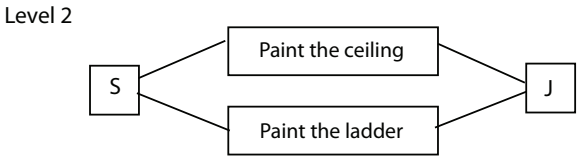

Figure 4.5: Level 2 of Procedural Net for Painting

The graphic representation used in Figure 4.4 - 4.7 mask much information associated with each goal node, the add- and delete-lists, for example. The existence of different add- and delete-lists for each goal node gives rise to potential, clobbering interactions, i.e., one step clobbering another step by contradicting some of its intended effects or some of its required preconditions. For instance, in Figure 4.6, 'get ladder' makes the ladder available for use as required as a precondition for the immediately following 'apply paint to ceiling,' whereas 'apply paint to the ladder' makes the ladder unavailable for use (at least temporarily). This clobbering is avoided in Figure 4.7 by imposing a temporal ordering link placing 'apply paint to ceiling' before 'apply paint to ladder.'

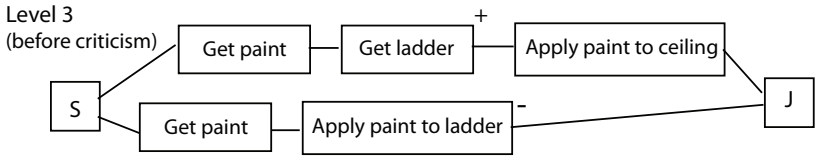

Figure 4.6: Level 3a of Procedural Net for Painting

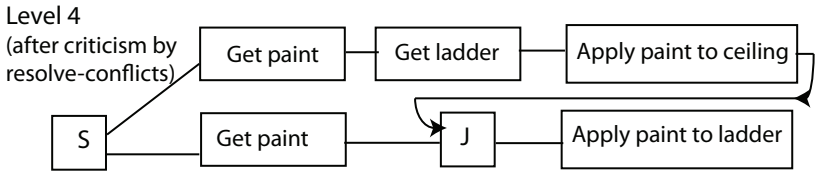

Figure 4.7: Level 3b of Procedural Net for Painting

HTN planning can be best elucidated by comparing and contrasting it with STRIPS-style planning. STRIPS-style planning refers to any planner whose operators are STRIPS-style operators consisting of a precondition list, an add-list, and a delete-list. Both HTN planning and STRIPS-style planning have similar representations of the world and actions [26]. A state of the world is represented by the set of atoms true in that state. Although an action corresponds to a state transition, i.e., a partial mapping from one set of states to another (possibly the same) set of states, actions in HTN planning are called primitive tasks. In the aforementioned example, 'get paint,' 'get ladder,' 'apply paint to ceiling,' and 'apply paint to ladder' might be considered primitive tasks.

The primary difference between HTN planners and STRIPS-style planners lies in what they plan for and how they plan for it [26]. STRIPS-style planners search for a sequence of actions that would transform the initial state of a given problem into a goal state. Such planners find instantiated operators with the desired effects and solve the preconditions of these operators as subgoals. In contrast, as demonstrated in Tate's NONLIN [91], [26] cites as a motivation for HTN planning the reconciliation of AI planning techniques and operations research techniques for scheduling and project management. Therefore, HTN planners search for plans that would accomplish *task networks*, effecting planning by task decomposition and conflict resolution.

A *task network* is a collection of tasks that have to be executed, along with constraints on the way variables are instantiated, the order in which tasks are executed, and what literals must hold before or after the execution of each task [26]. With this definition, [26] further defines an HTN problem as a triple $P = (d, I, V)$, where d is the task network to plan for, I is the initial state, and V is the set of operators and methods for the planning domain. V can

be considered a plan library for the particular planning domain, storing general descriptions of action decomposition methods; methods are extracted from the library and instantiated to suit the needs of the plan being constructed.

Given an planning problem $P = (d, I, V)$, an HTN planner essentially operates as follows [27]:

(1) If P contains only primitive tasks, then resolve the conflicts in P, and either return the result if all conflicts can be resolved, or return failure otherwise. Otherwise, go to step (2).

(2) Choose a non-primitive task t in P. Go to step (3).

(3) Choose an expansion for t. Go to step (4).

(4) Replace t with the expansion. Go to step (5).

(5) Use critics to find the interactions among the tasks in P, and suggest ways to handle them. Go to step (6).

(6) Apply one of the ways suggested in step (6). Go back to step (1).

HTN planning operates by both task decomposition via expansion and conflict resolution iteratively, until a plan free of conflicts and comprised only of primitive tasks is found. Expanding a non-primitive task in steps (2)-(4) entails finding a method that can achieve the non-primitive task, and then replacing the non-primitive task with the task network obtained by applying the method. The task network so obtained in step (4) may contain clobbering interactions. Critics detect and resolve such conflicting interactions, if any, in steps (5) and (6).

Erol, Hendler, and Nau proved that HTN planning is strictly more expressive than STRIPS-style planning in [26]. Notably, STRIPS-style planning, being a special case of HTN planning, can represent only a strictly smaller set planning domains than can HTN planning. The relationship between STRIPS-style planning and HTN planing is exactly analogous to that between regular (right linear) grammars and context-free grammars. [26] attributes the extra computational power in HTN planning to its admission of multiple tasks, compound tasks (abstract representations of sets of task networks consisting solely of primitive tasks), and arbitrary constraint formulas in task networks; however, since procedural knowledge is encoded in the task hierarchy, it is difficult for an HTN planner to reason about solution plans not entailed in the task knowledge. Readers interested in the decidability and complexity results, or the expressivity proofs, are referred to [26].

4.3.2 Framework

Central to NOAH's framework is the *procedural net* data structure, in which plans are built. A *procedural net* is a network of nodes each containing procedural and declarative information, as well as links to other nodes. Each node represents an action at some level of detail, and nodes are connected to form hierarchical descriptions of operations and to form plans of action. The links between nodes reflect their partial, temporal ordering and their conjunctive and disjunctive relationships. In addition, precondition/subgoal relationships are inferred from links indicating which nodes represent expansions of other nodes. A node is understood as being connected to its predecessor(s) by undirected edge(s) touching its left side, and to its successor(s) by undirected edge(s) touching its right side. As an example, recall the procedural net in Figures 4.4 - 4.7 in Section 4.3.1.

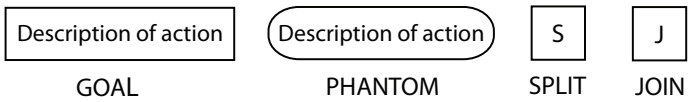

Figure 4.8: Graphic Representation of NOAH's Nodes

There are four types of nodes, as depicted in Figure 4.8. *GOAL* nodes represent a goal to be achieved; *PHANTOM* nodes, the goals which should already be true at the time they are encountered; *SPLIT* nodes, a branching of the partial ordering; *JOIN* nodes, a rejoining of subplans within the partial ordering. Each node points to a body of code, and the action represented by the node can be achieved by evaluating the body. The evaluation, in turn, causes new nodes representing more detailed actions to be added to the net. When a node is created, its add-list and delete-list are computed to represent the changes to the world model that arise from the action represented by the node.

When presented with a planning problem, NOAH is supplied knowledge about the task domain as well as a goal to achieve. Task-specific domain knowledge is in the SOUP[1] (Semantics of User's Problem) language. NOAH starts with a procedural net consisting solely of a single goal node to achieve the given goal. NOAH proceeds by expanding each non-primitive node in the order of their position in the time sequence, and thus producing more child nodes, until it eventually has only primitive actions. A non-primitive node is one that is decomposable and not immediately executable. The corresponding actions for

[1]Influenced by MIT's procedural approaches, especially those of Carl Hewitt, as well as Sussman and Winograd, SOUP operators are procedural and rather opaque, similar to STRIPS operators but with LISP-like code. Later HTN planners used much tidier representations, with preconditions, steps, effects, etc.

conjunctive goals are grouped using *AND* splits and joins. Similarly, wherever alternative ways of establishing a precondition exist, the corresponding actions are grouped using *OR* splits and joins. To ensure that local expansions contribute to a valid overall, more detailed, plan, a set of constructive critics, which add constraints to as yet unconstrained plans, examine the new plan for global cohesiveness.

Three types of general-purpose critics are found in NOAH. First, the *resolve-conflicts* critic examines portions of a plan that represent conjuncts to be achieved in parallel. The implicit assumption is that all of a subgoal's preconditions must remain true until the subgoal is executed. A subgoal in a conjunct is endangered if one of its preconditions is deleted by an action in a parallel branch of the plan. A similar conflict happens if an action in one conjunct deletes a precondition of a following subgoal, and the precondition must be reachieved after the deleting action. This conflict may be resolved by requiring the endangered subgoal to be achieved before the parallel action that would delete the precondition.

For instance, Figure 4.6 contains a conflict. Whereas 'get ladder' makes the ladder available for use as required as a precondition for the immediately following 'apply paint to ceiling,' 'apply paint to the ladder' makes the ladder unavailable for use (at least temporarily). This conflict is denoted pictorially by a plus sign over the precondition and a minus sign over the step violating it. This conflict is resolved in Figure 4.7 by imposing a temporal ordering link placing 'apply paint to ceiling' before 'apply paint to ladder.'

Another type of general-purpose critics is the *use-existing-objects* critic. During planning, NOAH avoids binding a variable to a specific object unless a clear best choice for the binding is available, and opts instead to generate and bind a formal parameter to the variable. This allows the system to avoid making arbitrary, possibly wrong, binding choices on the basis of insufficient information. Once a plan has been completed at some level of detail deemed appropriate by the *use-existing-objects* critic, the critic will replace formal parameters with specific objects mentioned elsewhere in the plan. The replacement may involve merging nodes from different parts of the plan; therefore, it may result in reordering or partial linearization [82].

The *eliminate-redundant-preconditions* critic is the last type of general-purpose critics. This critic removes redundant preconditions to conserve storage and avoid redundant planning at detailed levels for achieving them. For instance, in Figure 4.7, 'get paint' appears twice, and the critic may remove the temporally later 'get paint' by ensuring that an adequate amount of paint for both the ceiling and the ladder is obtained the first time.

Roughly stated, the algorithm for NOAH's planning process is as follows:

(1) Execute the currently most detailed plan in the procedural net, producing a new, more detailed plan. Go to step (2).

(2) Criticize the new plan, performing any necessary reordering or elimination of redundant operations. Go back to Step (1).

As with ABSTRIPS and other hierarchical planners, NOAH generally allows a plan to be obtained more quickly because the search space is much smaller at a higher level of abstraction. Instead of inefficiently reasoning at some low level of primitive actions that may never pan out, we consider details by expanding nodes only when a successful plan at a higher level lends credence to their importance. Moreover, assuming that conjunctive goals are independent, NOAH applies critics to identify potential conflicts and resolves conflicts by linearizing only the portion of the plan concerned. Since NOAH does plan linearization only as needed when it discovers the nature of the interactions between conjunctive goals, NOAH can place actions in a safe order for goal attainment and never has to undo the effects of a false assumption [82].

NOAH has some deficiencies, however. First, its decomposition of higher level actions into lower level ones is specified procedurally, rather than declaratively; therefore, the computations specified in the procedural semantics are not amenable to analysis by a reasoning component. Second, given its procedural semantics, NOAH does no theorem-proving, and this limits NOAH's ability to address the ramifications of actions. Furthermore, recall from Section 3.1.3 the register exchange problem, which STRIPS cannot solve. NOAH also founders on this problem as non-linearizable interactions exist between the two subgoals $Contains(R_2, N_1)$ and $Contains(R_1, N_2)$, where no simple ordering of the actions accomplishing each subgoal independently can accomplish the overall goal [82].

4.3.3 An Example

Consider the Sussman anomaly example [90] included in [82]. Below we show pictorially and explain how NOAH successfully solves the Sussman anomaly. The planning problem in the Sussman anomaly is to transform the initial state in which a clear-topped block C is atop a block A resting on a table with a clear-topped block B resting on the table, into a goal state in which a clear-topped A is atop B which is, in turn, atop C resting on the table.

Figure 4.9 depicts the initial state and the goal state, complete with their respective sets of ground literals.

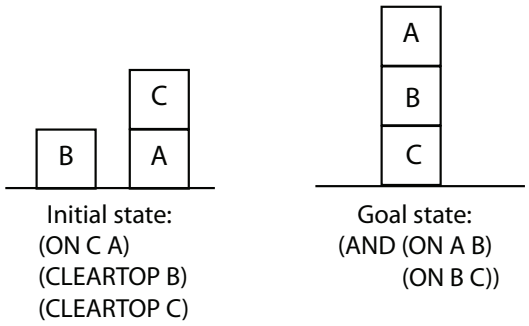

Figure 4.9: The Sussman Anomaly Example

NOAH starts by constructing an initial procedural net consisting only of a single GOAL node. The node is to accomplish the given goal, and its body is a list of task-specific SOUP functions, $CLEAR$ and $PUTON$ in this particular example. Figure 4.10 shows this one-step plan.

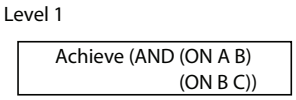

Figure 4.10: Level 1 in NOAH's Sussman Anomaly Example

Next, in Figure 4.11, NOAH splits up the given conjunctive goal in order to accomplish its two conjuncts independently. Although function $PUTON$ is relevant to the achievement of both conjuncts, NOAH does not invoke it immediately. Instead, NOAH creates a new GOAL node for each invocation. The new nodes, both with $PUTON$ in their respective body, are to achieve $(ON\ A\ B)$ and $(ON\ B\ C)$, respectively. With a greater level of detail added to the original plan, NOAH applies critics and finds no problems with the plan generated thus far.

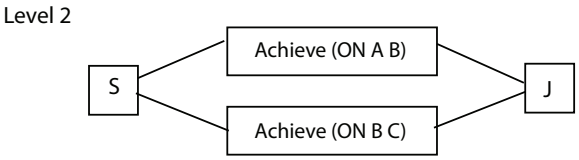

Figure 4.11: Level 2 in NOAH's Sussman Anomaly Example

Executing the two GOAL nodes introduced in Figure 4.11, NOAH applies the $PUTON$ function in their body to each goal assertion. This, in turn, generates a new level of GOAL nodes, as shown in Figure 4.12. The nodes of the plan are numbered so we can refer to

them in our discussion of actions taken by the critics.

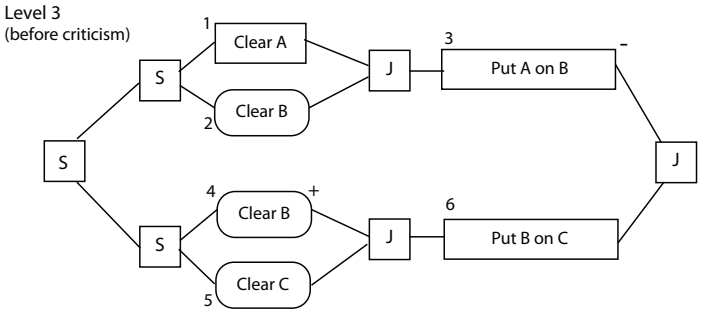

Figure 4.12: Level 3a in NOAH's Sussman Anomaly Example

NOAH next applies critics to the plan. Any conflicts arising as a result of the nonlinearity assumption are found by the resolve-conflicts critic. Only one such conflict exists in the plan, namely that node 4 has ($CLEARTOP\ B$) as an effect required as a precondition of the immediately following GOAL node 6, while node 3 produces the negation of this as an effect. Consequently, the resolve-conflicts critic reorders the plan by placing node 6 before node 3. Figure 4.13 displays the new plan.

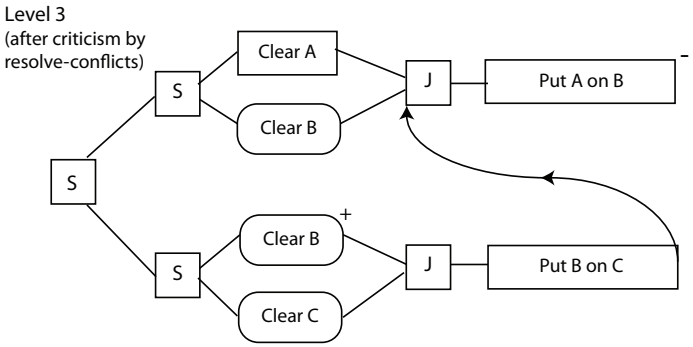

Figure 4.13: Level 3b in NOAH's Sussman Anomaly Example

The use-existing-objects critic need not transform the plan as no formal objects have been generated at this level of detail. Applying the eliminate-redundant-preconditions critic, NOAH obtains the new plan in Figure 4.14.

NOAH now executes the plan in Figure 4.14 and obtains a new, more detailed plan, as illustrated in Figure 4.15. Critics are then applied to the new plan. The resolve-conflicts critic discovers one conflict, namely that a PHANTOM node 'clear C' has ($CLEARTOP\ C$) as an effect required as a precondition of the immediately following GOAL node 'put C on OBJECT 1,' while the GOAL node 'put B on C' produces the negation of this as an effect.

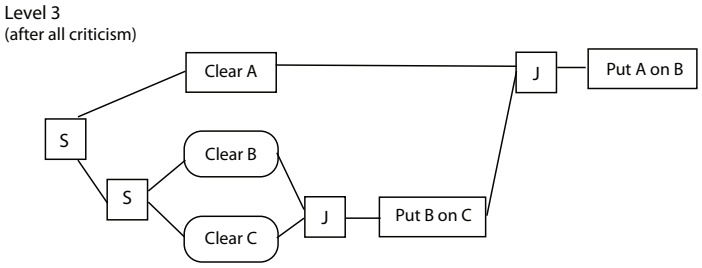

Figure 4.14: Level 3c in NOAH's Sussman Anomaly Example

Consequently, the resolve-conflicts critic reorders the plan by placing 'put C on OBJECT 1' before 'put B on C.' Figure 4.16 displays the new plan.

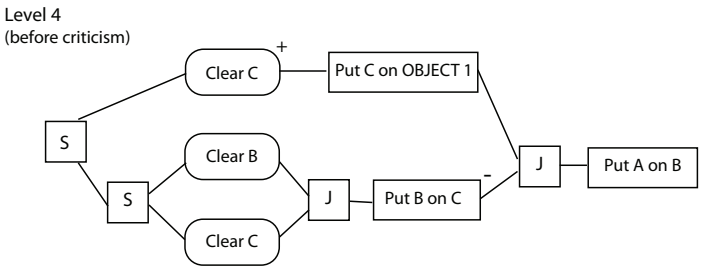

Figure 4.15: Level 4a in NOAH's Sussman Anomaly Example

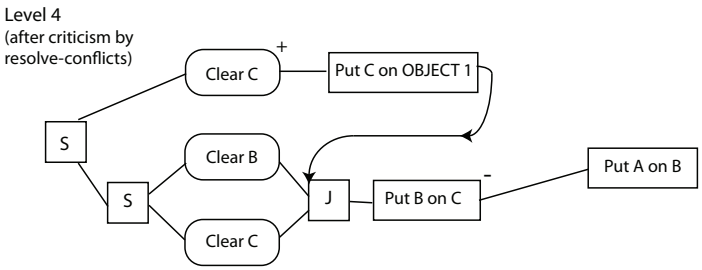

Figure 4.16: Level 4b in NOAH's Sussman Anomaly Example

The use-existing-objects critic need not transform the plan as no formal objects have been generated at this level of detail. Applying the eliminate-redundant-preconditions critic, NOAH obtains the new plan in Figure 4.17. The final plan is given by ⟨ 'put C on OBJECT 1,' 'put B on C,' 'put A on B'⟩.

4.3.4 Related Systems

Russell and Norvig [78] acknowledged that almost all planners for large scale applications are HTN planners, lending support to the belief that HTN planning is both efficient and

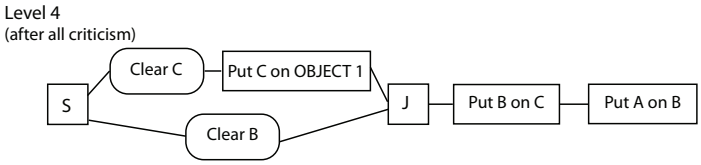

Figure 4.17: Level 4c in NOAH's Sussman Anomaly Example

practically oriented. This embrace of HTN planners can be ascribed to its admission of procedural knowledge provided by the human expert. Here we briefly survey a few practically oriented HTN planning systems that are successors to NOAH.

NONLIN [91], due to Tate, is a partial order HTN planner that emerged in 1977. Its development was motivated by a gaping deficiency in NOAH. When NOAH has to make choices as to how interacting actions are ordered in a plan, NOAH makes one particular choice without keeping any other backtrack choice points. This incapacitates NOAH's from undoing a failed plan step, leading to an incompleteness of the search space which can render some solvable planning problems unachievable. To rectify this deficiency, NONLIN keeps a record of all the decision made as well as all the goals achieved. This enables NONLIN to backtrack to an earlier choice point and attempt an alternative choice, should a plan step fail; nevertheless, NONLIN is still not a provably complete planner. In addition to the improvement in the ability to backtrack, NONLIN improves on NOAH's syntax, making the decomposition of higher level actions more understandable.

Currie and Tate's O-PLAN [22, 92] is a direct successor to NONLIN. It is a domain-independent, general planning and control framework with the ability to use detailed knowledge of a specific domain. O-PLAN strives to demonstrate how a planner in a particular task assignment and plan execution environment using specific domain knowledge can allow for "flexible, distributed, collaborative, and mixed-initiative planning" [92]. O-Plan as one of the systems coming closest to being practical. Not only has O-Plan been applied to logistics tasks requiring flexible responses in dynamic situations, but it has also been used to develop production plans for Hitachi. As [78] points out, given a product line of 350 products, 35 assembly machines, and more than 2000 operations at Hitachi, O-PLAN can generate a 30-day schedule involving millions of steps with three eight-hour shifts per day.

SIPE (System for Interactive Planning and Execution monitoring) [101], devised by Wilkins in 1983, is yet another NOAH-like planning system which also maintains a network of partially ordered tasks. Unlike NOAH, however, SIPE is domain-independent and capable of interleaving planning and execution. SIPE models the effects of an action using a variant

of the STRIPS representation augmented with domain rules to derive additional, context-dependent effects of the action. Plans generated by SIPE may involve parallel actions, where interactions between concurrent actions are arbitrated through a constrained form of resource modeling to be described next.

In SIPE, constraints are posted on variables, restricting their type, equality or inequality to other variables or constants. This preempts a premature commitment to their exact values, thereby resulting in arguably more efficient planning. Furthermore, certain variables of either actions or goals can be designated as resources; actions competing for the same resource cannot be parallelized in executing a partially ordered plan. SIPE's improvement in deductive and replanning capabilities over NOAH has endeared it to commercial applications. Not only has SIPE been incorporated into systems for responding to oil spills and air campaign planning, but it has also become the core of the CYPRESS system [102], which is an integrated environment for reasoning under uncertainty, reactive plan execution, and dynamic replanning.

In [95], Vere stated that both NOAH and NONLIN trivialize the time component in planning problems as they both assume that parallel tasks are concurrent. To properly account for the start times and durations of sets of goal conditions in planning problems, Vere's DEVISER, invented in 1983, was based exactly on NONLIN but further extended with the addition of time windows to resolve conflicts in parallel tasks with critical execution times. DEVISER was aimed at the automated control of the Voyager spacecraft; thus, it was imperative that DEVISER consider the time component in planning. DEVISER is discussed next in Section 4.4.

4.4 DEVISER

Vere's DEVISER [95], which appeared in 1983 along the evolutionary path of NOAH and NONLIN, is not only the first temporal planner (i.e., one with temporal considerations), but also the first one planning in continuous time. Instead of regarding the world as a sequence of discrete state transitions, DEVISER considers the evolving world situations via the construct of *time windows*. In particular, DEVISER provides a mechanism for specifying external events along with their time of occurrence, delayed events caused by a planned action, as well as *time windows* on goals and activities [23].

In DEVISER, as in any temporal logic system, the truth of a literal is, in effect, a function of time. Specifically, every literal has an associated activity, and every activity

has an associated start time interval and duration. A start time interval is termed a *time window*, and a duration may be a function of the activity parameters. We further explicate some preliminary concepts. An *activity* is an *action*, *event*, or *inference*; an *action* is a change in the world which the actor in a plan may opt to execute; an *event* is a change triggered spontaneously by the state of the world; an *inference* is an activity in which facts are added whose validity is contingent on the truth of the preconditions of the inference.

The activity representation used by DEVISER is less general than, but related to, the STRIPS representation, in the sense that the propositions deleted by an activity in DEVISER are explicitly included in the *antecedent* of that activity. An activity is represented by a *relational production*, which has components *context*, *antecedent*, and *consequent*, each comprised of an unordered set of literals. The *context* consists of literals which are the preconditions of the activity and whose truth remains unaffected by the activity; the *antecedent*, of preconditions deleted from the world model when the activity happens; the *consequent*, of literals added to the world model when the activity happens. A *relational production* has the following form, where the *activity-type* is either $ACTION$, $EVENT$, or $INFERENCE$, and $CONTEXT$ is a fixed keyword, while the other fields are to be substituted during instantiation: ($\langle$*activity-name*$\rangle\langle$*activity-type*$\rangle$

$(CONTEXT \quad \langle list\text{-}of\text{-}context\text{-}literals\rangle)$

$(\langle list\text{-}of\text{-}antecedent\text{-}literals\rangle) \rightarrow$

$(\langle list\text{-}of\text{-}consequent\text{-}literals\rangle))$

DEVISER uses a partially ordered network of events, but allows for reasoning about temporal constraints by means of durations and windows for activity and goals. $DURATION$ and $WINDOW$ are fields added to an activity or a goal specification. A duration may be either set to a constant or computed as a function of activity parameters. A window, defined in terms of a global clock, stipulates a lower and upper bound on the time when an activity may occur or when a goal is to be accomplished. It is a triple (*earliest-start-time*, *ideal-start-time*, *latest-start-time*) whose components are all real numbers in $[0, \infty)$ such that $earliest\text{-}start\text{-}time \leq ideal\text{-}start\text{-}time \leq latest\text{-}start\text{-}time$.

During planning, DEVISER may revise the *earliest-start-time* or *latest-start-time* of an activity in order to compress its window. However, DEVISER may widen a window as necessary, only when backtracking to an earlier state in plan generation. DEVISER considers the *ideal-start-time* of a window only in selecting the start times for actions after the plan network has been completely generated. Moreover, the windows of sequential nodes in the plan

network are dependent. Suppose nodes N_1 and N_2 of respective durations D_1 and D_2 are sequential, with windows $(EST_1, IDEAL_1, LST_1)$ and $(EST_2, IDEAL_2, LST_2)$, respectively. Then planning must satisfy the inequalities $EST_1 + D_1 \leq EST_2$ and $LST_1 + D_1 \leq LST_2$. Since an activity's window cannot be widened, if the first inequality is unsatisfied, DEVISER can satisfy it only by increasing EST_2; similarly, the second inequality, if unsatisfied, can be satisfied only by decreasing LST_1 . Whenever DEVISER links, expands, or orders nodes in the plan network to resolve conflicts, it must also reexamine their windows and compress them as necessary in order to preserve conditions designated by the windows.

DEVISER's concern with time is motivated by its intended application to the autonomous, unmanned spacecraft Voyager at the Jet Propulsion Laboratory. As such, DEVISER is practically oriented. Specifically, DEVISER plans and schedules actions for the automated control of the spacecraft, handling a repertoire of actions including the progression of the spacecraft past a planet and the rotation of the spacecraft and planets. Furthermore, DEVISER's use of temporal constraints facilitates planning and scheduling in the presence of deadlines.

Despite DEVISER's incorporation of temporal constraints in both planning and scheduling, [59] argues that the process in which DEVISER produces a schedule can be very inefficient with a large amount of backtracking. [59] ascribes the inefficiency to the lack of a distinct scheduler in DEVISER; DEVISER's plan expansion and ordering is not guided by a scheduler, but, rather, by relying on the general hierarchical planning mechanism with backtracking. To overcome the inefficiency, Miller, Firby, and Dean proposed techniques for eliminating unnecessary travel time by a robot and for avoiding backtracking in the presence of failures. Their techniques were materialized in their practically oriented planner FORBIN (First Order RoBot INtender) [24].

Chapter 5

Concluding Discussion of Part I

In Chapters 2 - 4, we have studied externally motivated planning systems, which find a plan solution to a user-given planning task with total disregard for their own cumulative utility. Aside from a summary of the three preceding chapters, this chapter concentrates on the analysis of two planning techniques. In particular, we compare and contrast linear (total order) and nonlinear (partial order) planning, as well as progressive planning (forward chaining) and regressive planning (backward chaining).

We have seen the contributions of human cognition and theorem proving to planning. GPS [63, 65, 9] is the general problem-solving paradigm of identifying the differences between the current state and a desired state, acting accordingly to eliminate the differences, and then advancing to the desired state. Green's logical characterization [41] in the situation calculus, along with his QA3 system, bridged the gap between theoretical specification and practical implementation. Kowalski [49] further reified the predicates in Green's formulation to simplify frame assertions; however, he failed to account for the ramification problem, the precondition interaction problem, and the effects interaction problem. Bibel's deductive plan generation [9] drew inspirations from program synthesis, offering generality and notational ease over the first two logical planners. This formulation is compatible with any theorem prover satisfying a specific linearity restriction, and dispenses with the inclusion of state variables and cumbersome first-order predicates.

Despite the strengths of logical planners, they have been largely overlooked in later planning systems which place emphasis on efficiency. A negative concomitant of this oversight is that we cannot derive logical consequences to address the ramifications of actions. The sentiment preferring efficiency over the robustness of theorem proving is reflected in the following excerpt:

> [T]he [BDI] architecture cannot be simply based on a traditional theorem-proving system, even if extended to handle the temporal, epistemic, and nondeterministic elements of the logic .... The reason for this is that the time taken to reason in this way, and thus the time taken to act, is potentially unbounded,[1] thereby destroying the reactivity that is essential to an agent's survival. Thus, although we could use a theorem prover to reason "offline" about the behavior of an agent-based system, we cannot directly use such a theorem prover to implement the system itself. [Rao & Georgeff, 1995, page 5 of "BDI Agents: From Theory to Practice"]

We have also reviewed both non-hierarchical planning and hierarchical planning. Although both seek to find a plan solution to a user-given planning task, they differ vastly in their approaches. Non-hierarchical planning could waste time searching for solutions to non-critical parts of a plan, only to find a more critical part of the plan unsolvable. In contrast, hierarchical planning may preempt the premature development of unnecessary plans, by first developing a higher level plan and then elaborating on it at finer levels later. However, since hierarchical planning generates multilevel abstraction, it may be more time-consuming than planning approaches generating only one level of abstraction.

One non-hierarchical planner surveyed is STRIPS [28], which aims to find a sequence of operators to transform a given initial model into one which a given goal formula holds. However, the ramification problem and the qualification problem are two problems confronting STRIPS. The former is the problem of deciding which action consequences to incorporate into STRIPS actions, while the latter is the problem of making an action's preconditions sufficiently inclusive to ensure that its add- and delete-lists fully capture guaranteed consequences given the preconditions [72].

Non-hierarchical planners and hierarchical planners can be further classified into linear (total order) and nonlinear (partial order) planners. A linear (total order) planner encodes a plan as a totally ordered sequence of actions under the linearity assumption that subplans can be merged only by concatenation and not by interleaving. As some examples, among the systems surveyed in Chapters 2 - 4, STRIPS [28], ABSTRIPS [79], and Warplan-C are linear planners.

On the other hand, a nonlinear (partial order) planner, by imposing ordering constraints only as required at execution time, can solve both the Sussman anomaly [90] and the register

[1] It is really "potentially unbounded" only if the system designers allow it to be, though.

exchange problem, two problems that foil a linear planner. As some examples, UCPOP [69], Graphplan [12], SNLP [53], TWEAK [16], NOAH [80, 81], NONLIN [91], SIPE [101], O-PLAN [92], and DEVISER [95] are nonlinear planners.

In [61], Minton, Bresina, and Drummond constructed a total order planner and a partial order planner that could be directly compared by a mapping between their search spaces. Their planners were named TO and UA, respectively. Their analysis reveals subtle assumptions underlying the supposed efficiency of partial order planning. First, UA's search space is never larger than that of TO, and is exponentially smaller for certain problems. This, coupled with the observation that UA incurs a small polynomial time increment per node over TO, suggests that UA is generally more efficient.

However, UA's search space economy does not necessarily translate into an efficiency gain, as the latter is dependent on the search strategy and heuristics used by the planner. For instance, [61] shows empirically that such distribution-sensitive search strategies as depth first search benefit more from partial order planning than do distribution-insensitive search strategies.

To generalize their empirical results, Minton et al. compared UA with partial order planners like NOAH, NONLIN, and TWEAK. Claiming that UA is more committed than these planners due to UA's requirement that each plan be unambiguous, they found that less committed partial order planners may create redundancies in the search space, and thus partial order planners do not necessarily have smaller search spaces than do total order planners [61]. For example, a TWEAK-like planner they considered had a larger search space than their TO on some problems.

While Minton et al. [61] offered general insights regarding partial order planners, they also noted that one could not expect one arbitrarily chosen planner to clearly dominate another arbitrarily chosen planner due to the ill-defined nature of the comparative behavior of different planners. A further delineation between partial order planning and total order planning can be found in [93].

Georgeff [33] defined progressive and regressive planning with the following terminology. Consider a primitive plan α which satisfies $exec(\alpha, \phi, \psi)$, where ϕ and ψ are some conditions; i.e., after α is initialized in a state in which ϕ holds, ψ will hold at the completion of execution. If ϕ is the weakest condition for which we can prove that this holds, then ϕ is defined as the *weakest provable precondition* of α with respect to ψ. Similarly, ψ is defined as the *strongest provable postcondition* of α with respect to ϕ, if ψ is the strongest condition for which we can guarantee that ψ will hold if α is initiated in a state in which ϕ holds.

Progressive planning and regressive planning are then two of the ways we can form a plan p to attain a goal ψ, beginning in an initial world in which ϕ holds. Let α denote any primitive plan element. Then in progressive planning, forward-chaining from the initial state, p satisfies $exec(\alpha, \phi, \psi)$ iff $p = \alpha; q$, where q satisfies $exec(q, \delta, \psi)$ and δ is the strongest provable postcondition of α. In regressive planning, backward-chaining from the goal, p satisfies $exec(\alpha, \phi, \psi)$ iff $p = q; \alpha$, where q satisfies $exec(q, \phi, \delta)$ and the regressed goal δ is the weakest provable precondition of α and ψ.

Forward chaining is a form of data-driven reasoning, that is, reasoning wherein the focus begins with the known data [78]. As every inference in forward chaining is essentially an application of Modus Ponens, forward chaining is sound. It is also complete in that every entailed atomic sentence can be derived. We humans use some amount of data-driven reasoning as we receive new information. However, we exercise control when using forward chaining so as not to be overwhelmed with impertinent consequences. For instance, when one sees it is raining outside, one may infer that the grass outside one's home will get wet, but it is highly unlikely that one will deduce that the fifth right whisker on the face of one's neighbor's cat that is roaming outdoors will get wet.

Backward chaining (regression) is a form of goal-directed reasoning. Often, the cost of backward chaining is much less than linear in the knowledge base size, because the process considers only relevant facts [78]. It is useful for answering specific questions like "Where are my reading glasses?" and "What shall I do now given my goal to eat a blueberry muffin?" However, for efficiency, backward chaining bears the criticism that planning is carried out myopically in the absence of complete information such as the preconditions for applying a selected action operator. Put another way, the danger is that we are working from an incomplete state and thus cannot take into account properties of the state space; therefore, in effect, backward chaining would work only with state constraints. Backward chaining is also a depth first search algorithm. On one hand, its space requirements are linear in the size of the proof. On the other hand, in contrast with forward chaining, it suffers from problems with repeated states and incompleteness.

As noted by Russell and Norvig [78], forward chaining on graph search problems is an example of dynamic programming, in which the solutions to problems are constructed incrementally from those to smaller subproblems and are cached to avoid recomputation. Furthermore, [78] suggests that a similar effect can be attained in backward chaining using memoization, that is, caching solutions to subgoals as they are found and then reusing these solutions when the subgoal recurs, instead of repeating the previous computation. Therefore,

backward chaining using memoization alleviates problems with redundant inferences and infinite loops [78].

In general, it would seem advantageous for an agent to combine the dynamic programming efficiency of forward chaining with the goal-directedness of regression. One may steer towards a gap-free plan as quickly as possible by guiding forward chaining of actions with heuristic measures of distance between states obtained by forward chaining and either the goal conditions or the action preconditions for actions obtained by regression. For example, the LPG (Local search for Planning Graphs) system by Gerevini et al. [40, 39] obtains plans by successive local plan modifications, somewhat analogously to SAT-planning. Implicitly, this strategy, like SAT-planning, combines forward and backward search. LPG also handles quantitative constraints and predictable exogenous events.

Part II

Internally Motivated Systems

Part II, comprised of Chapters 6 - 8, of our paper is devoted to a discussion of *internally motivated* systems, broadly including planning agents. Specifically, we survey systems endued with self-awareness and with the mental attitudes beliefs, desires, and intentions. We hypothesize that internal motivation arising from a self-aware planning agent's beliefs, desires, and intentions enables the agent to act opportunistically. Although we do not intend to prove our hypothesis in this paper, we endeavor to lend credence to our hypothesis by reviewing the crucial roles that self-awareness and mental attitudes assume in creating intelligent, artificial (planning) agents.

If any vision of instilling or attaining human-level intelligence in artificial (planning) agents is to be realized, then the agents must be able to function appropriately and adaptively not only in static environments, but also in constantly evolving environments. The rationale for this is given by the observation that real-world situations almost invariably never remain stagnant. Even as we attempt to contemplate a perpetually stagnant situation, we must acknowledge the passing of time as well as the aging of the animate entities involved in such a situation.

As potentially infinitely many unexpected changes can occur in dynamic environments, the use of pre-compiled reactive packages, such as those proposed by Firby [30], consisting of a set of fixed rules for responding to a limited number of unexpected changes, is doomed to be inadequate in handling all possible unexpected changes. Therefore, we are not interested in purely reactive agents; rather, we are interested in planning agents that can reason and act opportunistically, maximizing their own cumulative utility while achieving user-specified goals.

Dynamic environments pose three main types of challenges to an artificial planning agent; that is, unexpected changes can arise in the form of unexpected action failures, unexpected threats, and unexpected serendipitous opportunities. For an agent to act truly opportunistically is to be able to recover from (unexpected) failures, avoid (unexpected) threats, and seize (unexpected) favorable opportunities in an appropriate and timely manner. In essence, such changes are unexpected because the agent has only incomplete or partial knowledge; furthermore, the unexpected changes affirm the qualification problem, that is, the agent simply does not know all the conditions for an action to succeed and so may experience unexpected outcomes.

The first type of challenges a planning agent faces is serendipitous opportunities, and the agent should respond to such opportunities by seizing them and reaping their benefits. For example, while walking from home to school , an agent sees a $10 bill on the sidewalk

and then decides to claim the bill as its own after thinking about how it can benefit from an increase in its wealth; perhaps the agent would be happy to acquire a book which he could purchase with this money.

Second, while attempting to perform an action to achieve a goal, an agent may experience an unexpected failure, and the agent must recover from it. Imagine, for example, a thirsty agent intending to quench its thirst by drinking water. After evaluating alternative ways in which it can drink water, the agent decides to fill an empty cup with water and then drink water from the cup. However, while pouring water into the cup, the agent discovers, much to its dismay, that water is leaking from a hole in the bottom of the cup. Confronted with this unexpected failure to fill the cup, the agent must recover from it to resume pursuing its goal, perhaps by acquiring and filling another, non-defective cup with water.

The third type of challenges confronting a planning agent is characterized by unexpected threats, and the agent must strive to avert threats or alleviate their negative effects if the threats are unavoidable. In general, a threat is any external circumstance which, if ignored, could negatively affect the agent. For example, an agent hiking in the woods may face the threat of being maimed by a bear if the agent encounters one. Milder threats might be approaching heavy clouds indicative of an impending thunderstorm, or the agent shaking hands with someone that has the flu.

In Section 8.2, we present our preliminary proposal of a self-aware, opportunistic planning agent that maximizes its own cumulative utility while achieving user-specified goals. The agent should reason and behave opportunistically according to its own value metrics based on its beliefs, desires, and intentions.

Chapter 6

Belief-Desire-Intention Agents

The integral role played by such mental attitudes as beliefs, desires, and intentions in the design of intelligent agents has been extensively investigated and well acknowledged in both Philosophy and AI literature. Specifically, these attitudes are instrumental in determining the behavior of an agent as it seeks to accomplish a given goal. Furthermore, intention has been singularly noted as being primarily responsible for maintaining a "rational balance" among the beliefs, commitments, plans, goals, intentions, and actions of an autonomous agent [17], whereby by "rational" is meant that the agent should be committed to achieving a given goal, but be able to change its focus and pursue a new goal when the situation warrants it.

We are interested both in the distinct role of intention in governing an intelligent agent's behavior, and in how an agent endowed with beliefs, desires, and intentions can benefit from this endowment. In Section 6.1, we discuss Bratman's view of intention [14], informed through a philosophy of mind and action perspective. In contrast to prior philosophical literature reducing intention to some combination of belief and desire, Bratman proposed several characteristic functional roles of intentions distinguishing intention from belief and desire. Additionally, Bratman [14] showed how one can adopt intentions relative to a background of relevant beliefs, goals, and other intentions.

In the spirit of Bratman's work on intention [14, 15] but coming from an AI perspective, Cohen and Levesque [17] analyzed various properties of intention and the significance of such properties in the specifications of intelligent agents that aim to approximate a theory of human action. Among Cohen and Levesque's contributions [17] is their formalization of the conditions admitted by an agent's intentions under which an intelligent agent should and can abandon its goals. Cohen and Levesque's theory [17] also shows how an intelligent

agent can avoid intending all the anticipated side-effects of its intended action. We discuss Cohen and Levesque's work [17] in Section 6.2.

Extending work in belief-desire-intention (BDI) agents still further is Rao and Georgeff's logical framework founded on the primitive modalities beliefs, desires, and intentions [74, 76]. Their framework formalizes intentions based on a branching-time possible-worlds model, leading to a BDI architecture for rational reasoning. An improved version of this architecture, in turn, evolved into the goal-directed and reactive PRS [36], one of the first practical, agent-oriented systems implemented based on the BDI architecture. We present Rao and Georgeff's BDI framework and BDI architecture in Section 6.3, before concluding this chapter with a review of the PRS system in Section 6.4.

6.1 Intention from a Philosophical Perspective

Philosophers have long since been engaged in the study of intention, and many of them have differentiated present-directed intentions from future-directed intentions [85, 14]. The former causally generate behaviors [85], whereas the latter guide behaviors and mold the formation and adoption of other intentions [14]. In [14], not only did Bratman delineate the concept of intention as more than merely a combination of belief and desire, but he also distinguished intending (or having an intention) to do something from doing something intentionally. The two correspond to future-directed and present-directed intentions, respectively, with the former focused chiefly on the coordination of an agent's plans. While we acknowledge the distinction, our ensuing discussion is restricted to future-directed intentions.

Coming from the functionalist tradition in the philosophy of mind and action, Bratman [15] defined what it is to intend to do something by the three functional roles characterizing intention. The three functional roles of future-directed intentions are as follows [15]:

(1) *Intentions pose problems for further deliberation by a planning agent.* For example, if an agent *intends* to travel from Rochester to New York City next week, the agent should be motivated to develop a plan for the intended travel.

(2) *Intentions constrain the formation and adoption of other intentions.* An agent typically does not adopt intentions that they perceive as conflicting with the agent's present- and future-directed intentions. For example, a hungry agent with desires to eat both an apple and a orange only has enough money to buy either but not both; if the agent intends to buy the apple, then the agent should not simultaneously intend to buy the orange.

(3) *Agents monitor the success of their attempts to accomplish their intentions.* Agents care whether their attempts succeed; furthermore, they are inclined to replan if their attempts fail to obtain the intended effects.

In addition to the aforementioned functional roles, [14] argues that an agent intending to achieve some arbitrary goal p should satisfy the following properties, aptly summarized by [17]:

(4) *The agent believes that p is possible.*

(5) *Under certain conditions, the agent believes it will achieve p.*

(6) *The agent does not believe that it will not achieve p.*

(7) *The agent need not intend all the expected side-effects of its intentions.*

For example, noticing an outbreak of flu among his associates, a still healthy agent intends to inoculate himself against it by getting a flu shot. Although the agent would likely anticipate the needle injection to be somewhat painful, the agent would most certainly deny that he *intends* to be in pain.

Bratman [14] argued that intention is more than merely a combination of belief and desire, providing two justifications. First, [14] observes that since an agent has only limited resources, it cannot continually evaluate its competing desires and beliefs when deciding what to do next; that is, at some point, the agent must decide to aim for just one state of affairs. This is identified with intending, a limited form of commitment. Second, once an agent intends to do a future act, it must coordinate its future actions; in particular, it must not simultaneously believe that it will not do the act. In these two regards, intentions are set apart from beliefs and desires, both of which allow for the possibility of coexisting, conflicting beliefs and desires, respectively.

Furthermore, Bratman [14, 15] postulated that what one intends is a subset of what one chooses. One is often confronted with a set of competing desires, and must choose just one of these to pursue. If one believes one's actions will produce certain effects, by committing to perform these actions, one has, in a way, also chosen their effects including unintended ones. However, side-effects are distinct from intentions for two reasons. First, an agent achieves unintended side-effects only incidentally, as a result of performing actions. Second, if an agent fails to perform an action some of whose effects the agent intends to achieve, the agent will replan only for the sole purpose of achieving the intended effects and will disregard the unachieved, unintended side-effects.

Cohen and Levesque's theory of intentions [17, 18], which we review in Section 6.2, shares a similar sentiment with respect to the distinction between intended effects and unintended side-effects. Particularly, this theory specifies how anticipated side-effects are chosen, but not intended.

6.2 A Theory of Intention from an AI Perspective

Cohen and Levesque [17, 18] proposed a theory of intention, in order to specify the *rational balance*[1] needed among an autonomous agent's beliefs, goals, intentions, and actions. Originally developed as a foundation for Cohen and Levesque's theory of speech acts [19], the theory also provides a model of rational behavior and has been used in the analysis of conflicts and cooperation among multiple agents [51].

Specifically, Cohen and Levesque [17] investigated the role one's intention plays in governing the *rational balance* among one's beliefs, goals, intentions, and actions. Their proposal is summarized in the following excerpt:

> An autonomous agent should act on its intentions, not in spite of them; adopt intentions it believes are feasible and forgo those believed to be infeasible; keep (or commit to) intentions, but not forever; discharge those intentions believed to have been satisfied; alter intentions when relevant beliefs change; and adopt subsidiary intentions during plan formation. [Cohen & Levesque, 1990, page 214 of "Intention Is Choice with Commitment"]

Cohen and Levesque [17] devised an intention-defining logic with modal operators that satisfy Bratman's seven postulates for intentions [14, 15]. Notably, as with Bratman, Cohen and Levesque acknowledged that an agent should not abandon an intention without a good reason (e.g., a good reason could be that the agent believes it is impossible to achieve the intention) and that an agent intending to achieve some goal p need not also intend p's consequences. Specifically, Cohen and Levesque's logic is multi-modal, many-sorted, and quantified, with four chief modalities:

- $(BEL\ \ x\ \phi)$: agent x believes ϕ;
- $(GOAL\ \ x\ \phi)$: ϕ will be true in any world where agent x's goals are achieved;

[1] Cohen and Levesque [17] attributed this phrase to Nils Nilsson.

- $(HAPPENS\ \ \alpha)$ or $(HAPPENS\ \ x\ \alpha)$: action α will happen next, with the latter also specifying x as the agent of α;
- $(DONE\ \ \alpha)$ or $(DONE\ \ x\ \alpha)$: action α has just occurred, with the latter also specifying x as the agent of α.

Both the BEL and $GOAL$ operators are closed under logical consequence and prescribed possible-worlds semantics. Additionally, $(BEL\ \ x\ \phi)$ implies $(GOAL\ \ x\ \phi)$, as given a set of possible worlds granted by what an agent believes, a subset of these are the worlds in which the agent's goals are accomplished. Based on this observation, the notion of a persistent goal is formalized. An agent x has a persistent goal ϕ, i.e., $(P\text{-}GOAL\ \ x\ \phi)$, if and only if the three following conditions hold [3]:

1. Agent x does not currently believe ϕ, i.e., $\neg(BEL\ x\ \phi)$;
2. A consequence of agent x's goal is that ϕ is true later, i.e., $(GOAL\ \ x\ (LATER\ \ \phi))$;
3. ϕ (being true later) will remain a consequence of agent x's goal until agent x either believes it to be true or believes it to be impossible.[2]

The world is defined as a discrete sequence of events, extending temporally and infinitely into the past and the future. In addition to the temporal operators $HAPPENS$ and $DONE$, other temporal operators in dynamic logic [43] are employed. [17] adopts the following notation for actions, where α and β denote action sequences:

- $\alpha;\beta$ denotes that α is followed by β;
- $\alpha|\beta$ is a nondeterministic choice between α and β;
- $\alpha||\beta$ is a co-occurrence of α and β;
- $\alpha?$ is a test action; e.g., $\alpha?;\beta$ expresses that β occurs next when α is true, while $\beta;\alpha?$ expresses that α holds after β occurs;
- $\Diamond\alpha\ \ =_{def}\ \ \exists x(HAPPENS\ \ x;\alpha?)$;
- $\Box\alpha\ \ =_{def}\ \ \neg\Diamond\neg\alpha$;
- $\neg\alpha \wedge \Diamond\alpha\ \ =_{def}\ \ (LATER\ \ \alpha)$.

[2] We argue that there is another until condition, namely that x believes there is a better option to pursue. See our discussion at the end of Section 6.2.

Case	Relationship of p and q	(P-$GOAL$ x q)?
1	$p \rightarrow q$	No
2	$(BEL\ x\ (p \rightarrow q))$	No
3	$(BEL\ x\ \Box(p \rightarrow q))$	No
4	$\Box(BEL\ x\ \Box(p \rightarrow q))$	No(Yes)
5	$\models (p \rightarrow q)$	No(Yes)
6	$\models (p \equiv q))$	Yes

Table 6.1: P-$GOAL$ and Progressively Stronger Relationships between p and q

Based on the formalized notion of persistent goals, Cohen and Levesque then offered two definitions of intention, where α is any action expression, and e and e' are arbitrary primitive events:

(D1) $(INTEND_1\ x\ \alpha) =_{def} (P\text{-}GOAL\ x\ [DONE\ x\ (BEL\ x\ (HAPPENS\ \alpha))?; \alpha]);$

(D2) $(INTEND_2\ x\ \alpha) =_{def} (P\text{-}GOAL\ x\ \exists e[DONE\ x\ (BEL\ x\ \exists e'(HAPPENS\ x\ e'\,; \alpha?))$
$\neg(GOAL\ x\ \neg(HAPPENS\ x\ e; \alpha?))]?; e; \alpha?)).$

(D1) defines what it means for an agent x to intend to do an action α; i.e., agent x is committed to believing that it is about to do the intended action α, and then doing it. In contrast, (D2) defines what it means for an agent x to intend that α becomes true, or to intend to bring about α, without necessarily knowing how to achieve that state of affairs. That is, agent x is committed to doing some sequence of events e, after which α holds.

Cohen and Levesque showed how the aforementioned definitions of intention satisfy Bratman's seven postulates [14, 15] for a theory of intention. Furthermore, they characterized the expected consequences of a goal as $(GOAL\ x\ p) \wedge (BEL\ x\ (p \rightarrow q)) \rightarrow (GOAL\ x\ q)$; that is, if an agent x believes that p implies q and chooses worlds in which p holds, then agent x chooses worlds in which q is true. However, unintended side-effects are differentiated from persistent goals, as illustrated in Table 6.1, recreated from [17].

Cohen and Levesque [17] proved that P-$GOAL$ is closed only under logical equivalence, which is effectively demonstrated in Table 6.1. For all six cases in Table 6.1, assume that (P-$GOAL$ x p). The third column indicates whether (P-$GOAL$ x q) holds or not. While cases 1, 2, 3, and 6 are straightforward, cases 4 and 5 warrant further explanations. In case 4, although the agent always believes that $p \rightarrow q$, q need not be a persistent goal as the agent could believe that q already holds. The same reason also blocks case 5. However, in both cases 4 and 5, the agent would be required to have q as a persistent goal if the agent did not believe q already held.

The assumptions underlying this theory of intention seem reasonable and intuitive.

Among them are that an agent should persist in pursuing its goals insofar as the circumstances justify this persistence, and that an agent should abandon goals given to it by other agents when it determines the goals need not be achieved. However, the theory is not impeccable and has been subject to critical examination by both Allen [3] and Singh [89].

Singh [89] took issues with several assumptions, claims, and proof steps included in [17]. We shed light on two of Singh's criticisms, referring readers interested in the rest to [89]. First, the theory seems to be designed for agents with no more than one intention at any time which they can act on. The reason is that both definitions of intention, namely $INTEND_1$ and $INTEND_2$, state that an agent intends to do an action iff the agent has a *P-GOAL* to have done the action *immediately* after believing it was about to happen. Similarly, because of the requirement that only one event take place at a time, an agent cannot succeed with its intention if another agent also acts. As a result, this theory is unsuitable not only for agents possessing and acting on several intentions at once, but also for multiple agents acting in an interleaved manner [89].

A second issue lies with Cohen and Levesque's claim about the assumption that all goals by all agents are eventually dropped [17]. They formalized the assumption by $\models \Diamond\neg(GOAL\ x\ (LATER\ p))$; also, they claimed that it captures the following restrictions:

(a) Agents do not persist forever with a goal; and

(b) Agents do not forever defer working on their goals.

While it is clear that the assumption captures (a), it is dubious how it could also capture (b) as the assumption does not involve action in any way. As Singh [89] argued, since the assumption requires not that an agent act on any goal but that the agent drop it, any agent, even one that would not or cannot act on a goal, will end up dropping it. The assumption, thus, does not account for restriction (b).[3]

Lastly, whether the concept of *P-GOAL* can be reconciled with rationality is disputable. We drop intentions whenever better options (in terms of anticipated rewards) present themselves. As we can only do so much thinking and planning before needing to act, we generally do not disrupt our intended plans too much lest we get lost in a thicket of alternatives, which we are unable to evaluate adequately in the time available. This manifestation is not so much rationality as a concession to practicality.

[3]However, if all goals are eventually dropped, then it is not clear how the contrary of (b), that an agent does forever defer working on a particular goal, could possibly occur. To defer something is to shift the intention to achieve something into the future, without abandoning the intention.

6.3 BDI-Agents from Theory to Practice

Wooldridge and Jennings [104] acknowledged the lack of consensus in the AI and Philosophy communities on the combination of attitudes and information best suited to characterizing rational agents. In Cohen and Levesque's theory of intention [17, 18], only the two basic attitudes beliefs and goals are used, while other attitudes such as intention are further defined in terms of these. In related work, Rao and Georgeff [74, 76] developed a logical framework founded on the primitive modalities beliefs, desires, and intentions. Their primary concern in [74] lies with the notion of realism – the question of how an agent's beliefs about the future affect its intentions and desires [104].

6.3.1 Necessity of Beliefs, Desires, and Intentions

Rao and Georgeff established the necessity of beliefs, desires, and intensions for systems to act appropriately in a class of practically oriented application domains. As given in [76], an example of such a domain is the design of an air traffic management system[4] responsible for determining the expected times of arrival (ETA) for arriving airplanes, ordering them in accordance with some optimality criteria, reassigning the ETA of an airplane according to this optimal order, issuing directives to the pilots to inform them of the assigned ETAs, and monitoring the pilots' compliance.

From this aforementioned domain and other real-time application domains, Rao and Georgeff extrapolated the common domain characteristics as follows [76]:

(1) At any instant of time, the nondeterministic nature of the environment gives rise to potentially many different ways in which the environment can evolve; e.g., the runway conditions, subject to the weather and other parameters, can change over time in unpredictable ways.

(2) At any instant of time, the nondeterministic nature of the system itself results in potentially many different actions or procedures the system can perform; e.g., the system can request a change of speed in an airplane, or recalculate the flight path for an airplane, and so on.

(3) At any instant of time, the system can potentially be asked to accomplish many

[4]The flight control domain is interesting in that it illustrates that an environment may constantly present new challenges (threats and opportunities). Nevertheless, it is misleading in that it is largely a scheduling domain, rather than one that seriously involves long-range planning and goal achievement.

different objectives, not all of which may be simultaneously achievable; e.g., the system can be asked to land airplane U053 at 20:00, land airplane U049 at 20:02, and so on.

(4) The actions or procedures that best[5] achieve the various objectives depend not on the internal state of the system, but on the state of the environment; e.g., the actions by which the airplanes meet their assigned landing times depend on operating conditions, the wind field, and so on.

(5) One sensing action is insufficient for fully determining the state of the whole environment, as the environment can be sensed only locally; e.g., the system receives only some wind data from an airplane at certain times at certain locations, and thus, cannot determine the current wind field.

(6) Computations and actions can be carried out at a rate within reasonable bounds of the rate at which the environment evolves; e.g., changes in the wind field, runway conditions, and so on, can happen while the system calculates an efficient landing sequence, or while an airplane is flying to meet its assigned landing time.

Characteristics (1) and (2) suggest modeling the behavior of such a system as a branching decision tree structure, in which each node represents a certain state of the world; each branch, an alternative path of execution; and each transition, either a primitive event occurring in the environment, or a primitive action performed by the system, or both. The system's objectives are, thus, identified with particular paths through the tree structure, with each path labeled with the objective realized and the utility gained by traversing it. Furthermore, actions by the system are differentiated from events in the environment by the use of two node types. Specifically, the *choice (decision) nodes* and *chance nodes* represent options available to the system and uncertainties in the environment, respectively.

Since the system acts, it needs to be able to select appropriate actions or procedures at its disposal that it can execute. Considering the aforementioned domain characteristics, a selection function requires at least two types of input data [76]. First, given characteristics (3) and (4), [76] argues that it is essential that the system also have information regarding the objectives it is to achieve and regarding the priorities or utilities associated with the various current objectives. This *motivational* system component is the system's *desires.*

[5]However, we note that "the best" is in general not achievable (if even definable), so the system makes choices dependent on its limited knowledge and limited problem-solving time. Therefore, to that extent, what is best hinges on what is computationally feasible for the system, which, in turn, depends on its internal state.

Second, characteristic (4) underscores the importance for the system to have information regarding the state of the environment; however, as noted by [76], in light of characteristics (1) and (5), such information cannot be acquired in one sensing action. Therefore, it is necessary that some *informative* system component should represent this information and be updated following each sensing action. Such a system component is the system's *beliefs*, which may be implemented as a database, a set of logical formulas, or some other data structure.

Given the six aforementioned domain characteristics, it is crucial that the system not be blindly or unconditionally committed to pursuing a chosen course of action. On the other hand, we would not want the system to reconsider all its available choices for actions at each step if no changes occur. Accordingly, [76] observes that if it is assumed that the occurrence of potentially significant changes can be detected instantaneously, it is possible to reach a balance between too much reconsideration and too little reconsideration. This necessitates the inclusion of the system's *intentions*, the system's *deliberative* component.

6.3.2 The BDI Framework

Based on the informal discussion in Section 6.3.1 of the belief, desire, and intention components of the system state, Rao and Georgeff [76] developed a theory for describing these components declaratively, in propositional logic. Rao and Georgeff further suggested that such a theory can be formalized in the framework of BDI logics [76]. This framework, which they claimed to be close to the traditional epistemic models of belief, can then be used to specify and implement systems with the domain characteristics enumerated in Section 6.3.1.

Recall the branching decision tree structure briefly described in Section 6.3.1. The structure consists of choice (decision) nodes, chance nodes, and terminal nodes; additionally, it has a probability function mapping chance nodes to real-valued probabilities, as well as a utility function mapping terminal nodes to real numbers. A deliberation function, such as the maximization of expected utility, is also defined in order to select one or more best sequences of actions to execute at a given node [76]. These functions, nonetheless, are feasible only if at any given time one has a rather limited, well-defined set of options to consider, such as in the flight control domain.

[73] presents an algorithm for transforming a decision tree and the associated deliberation functions, into an equivalent, possible-worlds model representing beliefs, desires, and intentions as accessibility relations, namely BA, DA, and IA, respectively, over sets of possible worlds. The resulting model is comprised of a set of possible worlds each of which is

a decision tree; each decision tree corresponds to a possible environment state with some probability of occurrence, and consists solely of choice (decision) nodes and terminal nodes. *BA* represents the occurrence probabilities of each different possible world, *DA* specifies the utilities assigned to each tree path, and the *IA* names, for each decision tree, its best course(s) of actions, i.e., selected tree path(s), as determined by the defined deliberation function. With each index within a possible world, we associate a set of *belief-accessible worlds*, *desire-accessible worlds*, and *intention-accessible worlds*; these can be understood as those worlds the agent *believes* to be possible, *desires* to bring about, and *intends* to bring out, respectively [76].

Using the aforementioned transformation as a guide, Rao and Georgeff [76] sought to develop a logical theory of deliberation by agents. As [76] points out, a theory of deliberation based solely on the aforementioned transformation does not address cases where good estimates for probabilities and utilities are unavailable, nor does it address the dynamic aspects of deliberation, especially those regarding commitment to previous decisions. To fix these inadequacies, Rao and Georgeff [76] first reduced all the probabilities and utilities to 0-1 values, regarding propositions as either believed or not believed, desired or not desired, and intended or not intended.

Additionally, they axiomatized beliefs, desires, and intentions and augmented the theory with both static constraints and dynamic constraints. The static constraints specify the static relationships among the belief-, desire-, intention-accessible worlds, while the dynamic constraints govern an agent's behavior according to the agent's level of commitment as determined by the components of the agent's commitments.

According to Rao and Georgeff [76], a commitment has two components, with the first being the commitment condition that the agent is committed to maintaining, and the second being the termination condition under which the agent foregoes the commitment. However, we counter that this notion of commitment is an unlikely reflection of reality. In real life, an agent continually strives for a better plan (in terms of anticipated rewards), and as the agent's plan evolves, so do the agent's commitments.

6.3.3 The BDI Architecture

Rao and Georgeff [76] maintained that the purpose of the theory discussed in Section 6.3.2 is to build practical and formally verifiable systems. Their rationale is that if one knows the environment changes and expected system behaviors for a given application domain, then one can use the theory to specify, design, and verify agents that, when placed in the

environment, display all and only the desired behaviors. Based on the theory, they then proposed an abstract architecture underlying a practical system – a simplified version of the PRS system, which we review in Section 6.4.

The architecture consists of an input queue of events, and three dynamic data structures representing the agent's beliefs, desires, and intentions. All four data structures are assumed to be globally shared, and the system can recognize both internal events and external events. Moreover, both events and the agent's actions are assumed to be atomic. Central in the architecture are the update operations on the agent's beliefs, desires, and intentions to ensure the agent's mental attitudes are as specified in the theory. Algorithm 5 outlines the architecture.

Algorithm 5 Rao and Georgeff's BDI Interpreter

```
1: procedure BDI-Interpreter( )
2:     initialize-state();
3:     loop
4:         options ← options-generator(event-queue);
5:         selected-options ← deliberate(options);
6:         update-intentions(selected-options);
7:         execute();
8:         get-new-external-events();
9:         drop-successful-attitudes();
10:        drop-impossible-attitudes();
11:    end loop
12: end procedure
```

Each cycle begins with the option generator reading the event queue and providing a list of options.[6] From this list, the deliberator selects a subset of options to be adopted, and adds them to the intention structure. At this point, if the agent has an intention to execute an atomic action, then the agent executes it. Next, any external events that have occurred during the cycle are appended to the event queue. In contrast, internal events are appended to the event queue as they occur. The agent then modifies both the desire and intention structures by dropping not only all successful desires and satisfied intentions, but also all impossible desires and unattainable intentions.

Rao and Georgeff [76] contended that although the architecture conforms to the theory

[6]One wonders how options are generated. After all, the point of planning is to discover actions that will advance the agent towards its goals, and this in general requires exploring and evaluating sequences of possible actions, extending well beyond the current time point. Without such extrapolation to the future, the agent is in effect purely reactive (or at least short-sighted).

faithfully, it is not a practical system for rational reasoning for two reasons. First, since the architecture is based on a deductively closed set of beliefs, desires, and intentions, the provability procedures required are undecidable. Second, it fails to specify how the option generation and deliberation procedures can be made sufficiently fast so as to meet the system's real-time demands.

To provide a more practical system for rational reasoning, Rao and Georgeff added more stringent representational choices to the architecture. The resulting practical system is the PRS, one of the first agent-oriented systems implemented based on the BDI architecture. PRS, together with the aforementioned new representational choices, is presented next in Section 6.4.

6.4 PRS

PRS (Procedural Reasoning System) [36], proposed by Georgeff and Lansky in 1987, is an agent architecture based on the belief-desire-intention (BDI) framework for intelligent agents. At any instant, the actions being considered by PRS depend both on its current desires or goals and on its beliefs and previously formed intentions. Notably, since PRS interleaves planning and performing actions in the world, at any instant of time, the system has a fluid and partially specified plan. Furthermore, PRS is able to reason about and modify its own internal state, which is composed of its own beliefs, desires, and intentions; thus, PRS offers reactivity crucial for survival in dynamic and uncertain worlds.

As stated in Section 6.3.3, PRS is more computationally feasible and responsive to real-time demands than is the basic BDI architecture described in secion 6.3.3. The improvement realized in PRS can be attributed to three particular representational choices [76]. First, only beliefs about the current state of the world are explicitly represented in PRS, and only ground sets of literals are considered. Second, each intention that the system forms by adopting certain plans of action is represented implicitly using a *process stack* of hierarchically related plans. Third, information regarding the means of attaining certain future world states and the options available to the agent are represented as *plans*.

Additionally, to ensure that PRS can satisfy real-time demands while functioning in a dynamic environment, the process of option generation is streamlined by inserting an additional procedure, post-intention-status(), at the end of the interpreter loop [76]. Aside from this, PRS's main interpreter loop is identical to the loop in Algorithm 5. This procedure allows the system to defer appending events to the event queue regarding any changes to the

intention structure until the end of the interpreter loop. [76] states that this way of posting appropriate events to the event queue enables the system to specify which changes to the intention structure will be noticed by the option generator. As a result, various notions of commitment can be modeled, yielding different behaviors of the agent [75] as warranted by the circumstances in the current situation.

The system consists of a *database* of current *beliefs* and facts about the world expressed in first-order predicate calculus, a set of current *goals* or *desires* to be achieved, a set of *procedures* or *knowledge areas (KAs)* describing how certain sequences of actions and tests may be executed either to accomplish given goals or to react to particular situations, and an *interpreter* or *inference mechanism* for manipulating these components. At any instant of time, the system also has a *process stack* of all the currently active KAs, which can be considered the system's current *intentions* for accomplishing its goals or reacting to an observed situation. Figure 6.1 shows the PRS system structure.

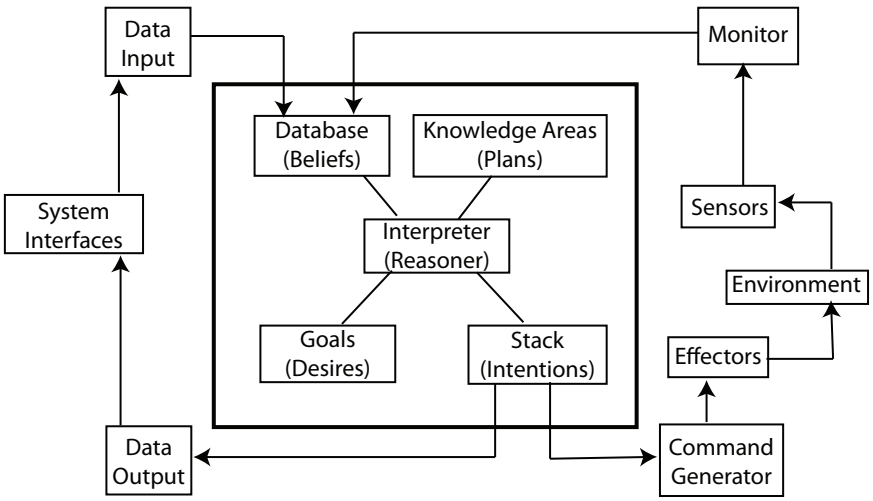

Figure 6.1: The System Structure of PRS

6.4.1 The Database, Goals, and Knowledge Areas

The database in PRS is comprised of a set of state descriptions describing the system's current beliefs, including facts about the static properties of the application domain. For example, a static property may be the physical laws by which some mechanical system components must abide. Expressed in first-order predicate logic, state descriptions are provided initially by the user, or derived by the system as it executes its KAs. Meta-level state descriptions are descriptions of the internal system states; e.g., the meta-level

expression ($goal\ q$) is true if q is a current goal of the system.

Goals, or desires, in PRS are the desired behaviors of the system, rather than static world states which are to be attained eventually [36]. Consequently, goals are expressed as conditions on some sequence of world state, i.e., some time interval. Moreover, goal behaviors can be described in two ways. One is to apply a temporal predicate to an n-tuple of terms. A temporal predicate denotes a set of state sequences, or an action type; e.g., ($walk\ c\ d$) might denote the set of state sequences embodying walking actions from point c to d.

The other way to describe a goal behavior is to apply a temporal operator to a state description. Three temporal operators are used. Suppose that q is an arbitrary state description. Then expression ($!q$), denoting those behaviors that achieve q, is true of a sequence of states if q is true of the last state in the sequence; expression ($?q$), denoting those behaviors that result from a successful test for q, is true of a sequence of states if q is true of the first state in the sequence; ($\#q$) is true of a sequence of states if q is preserved throughout the sequence.

As with state descriptions, behavior descriptions can be of both the external environment and the internal system behaviors. Behavior descriptions of internal system behaviors are meta-level behavior specifications; e.g., the meta-level expression ($\Rightarrow q$), or ($!(belief\ q)$), describes a behavior that places some state description q in the system database.

In PRS, knowledge areas, or plans, are declarative procedure specifications representing knowledge about how to accomplish given goals or how to react to certain situations [34]. Each KA has a body specifying the primitive actions or subgoals that must be achieved for plan execution to succeed. Each KA also has an invocation condition, given as an arbitrarily complex logical expression specifying the conditions under which the KA can be chosen as an option. The body and the invocation condition of a KA jointly express a declarative fact about the effects of performing certain sequences of actions under certain conditions.

The body of a KA consists of possible sequences of subgoals to be achieved, rather than of possible sequences of primitive actions. Additionally, the subgoals in the body can be described by arbitrarily complex temporal expressions, and complex control constructs including loops, recursion, and conditionals can be used. Thus, the set of KAs in a particular PRS application system comprises the procedural knowledge about the application domain.

Georgeff and Lansky [34] enumerated several benefits this procedural representation offers over the conventional use of rules involving individual atomic actions. First, [34] claims that much expert knowledge, e.g., about diagnosing an engine failure or solving a

Rubik's cube, is procedural in nature; thus, a procedural representation of such knowledge is more natural and understandable than the deproceduralization of this knowledge into disjoint descriptions of individual actions, which would involve introducing additional control conditions to ensure that individual actions are linked and performed in the correct order. In essence, procedural knowledge is akin to MACROPS in STRIPS, realizing efficiency in not having to reconstruct particular plans of action from knowledge of individual actions.

Second, an agent's procedural knowledge allows the agent access to its history of actions, and the agent can deduce additional information about the state of the world from knowledge of its past activities [34]. An example is given in [34] of a cook that follows the steps of a recipe exactly; the cook can usually assume the food will turn out right, without having to perpetually taste it. Also, having access to the behavior of a procedure enables the agent to make more good use of procedural knowledge. Given and executing a procedure with a course of action $a \rightarrow b \rightarrow c$ to accomplish some goal, once the agent gets around to c, it knows that a and b have been executed. This knowledge assists the agent in making various assumptions, for example, that certain environmental conditions will hold because a and b tend to establish them.

Third, the agent can reason about these procedures as whole entities [34]. For instance, given two independent procedures for unclogging a drainage pipe, the agent can evaluate them separately and determine which has a more favorable cost-benefit ratio. This reflective reasoning about the agent's own internal procedures enables the agent to work effectively.

6.4.2 The Interpreter

The interpreter in PRS runs the whole system, interacting with the database (beliefs), the goals (desires), the process stack (intentions), as well as the library of KAs (plans). The interpreter maintains beliefs about the current world state, determines which goal to strive to achieve next, decides which knowledge area to apply in the current situation for the achievement of the selected goal. At any particular time, extant beliefs in the database and active goals in the system trigger a set of applicable KAs. Once the interpreter has selected a goal to pursue, it selects one of the applicable KA relevant to the achievement of the selected goal and places the selected KA on the process stack. The interpreter then executes this KA.

As the interpreter executes the selected KA, new subgoals that emerge are posted to the goal stack, and new beliefs derived are incorporated into the database. When a new belief is added to the knowledge base, the interpreter will perform necessary consistency maintenance

procedures, possibly activating other relevant KAs. Similarly, When a new goal is posted to the goal stack, the interpreter checks to see if any new KAs are now applicable, selects an applicable KA, places it on the process stack, and executes it. [36] states that various meta-level KAs may be invoked by the interpreter to choose a KA from a set of applicable KAs and make choices among alternative paths of execution. This results in an interleaving of plan selection, formation, and execution.

In our review of the PRS-related literature, we were unable to find the complete formal semantics of the original PRS system; the most related coverage, though, is in [37] on the semantics of a procedural logic. Furthermore, several issues are unclear due to details omitted from the PRS-related papers. These include the criteria the PRS interpreter use to select a goal to pursue among the goals on the goal stack, the criteria the PRS interpreter use to select a KA to execute among the currently applicable KAs for its selected goal, how the PRS interpreter determines the applicability of a KA (whether theorem proving as in the original STRIPS [28] is involved), and so on.

PRS has been used in a variety of real-world applications including the fault diagnosis in a space shuttle [35] and the control of a mobile robot [38]. [35, 38] showcase PRS's dual role as a reactive planner and reasoner; readers interested in the details are referred to them.

6.4.3 An Example

Consider an example from [36] involving the route planning and navigation of a robot. We provide a high level sketch, and refer readers interested in the details to [38]. Given the goal to reach a particular destination, the robot must first plan a route to the destination and then navigate to the destination. The top level strategy is encapsulated in the GO-TO KA, illustrated in Figure 6.2 recreated from [36].

When planning the route, the robot uses the topological knowledge stored in its database, in addition to its knowledge of how to plan a route represented in various route-planning KAs. In this example, the topological knowledge is quite coarse, only stating which rooms are in which corridors and how the corridors are connected [36]. The route plan created by the robot is also high level, usually of the form "Travel to the end of the corridor, turn left, then go to the second room on the right." Moreover, the robot is continuously reactive throughout this planning stage; e.g., if the robot notices an indication that fuel for the spacecraft is about to be depleted, the robot may very well interrupt its route planning and attend to the task of refueling the spacecraft.

In order to execute the route plan, the robot invokes a set of plan-translating KAs that

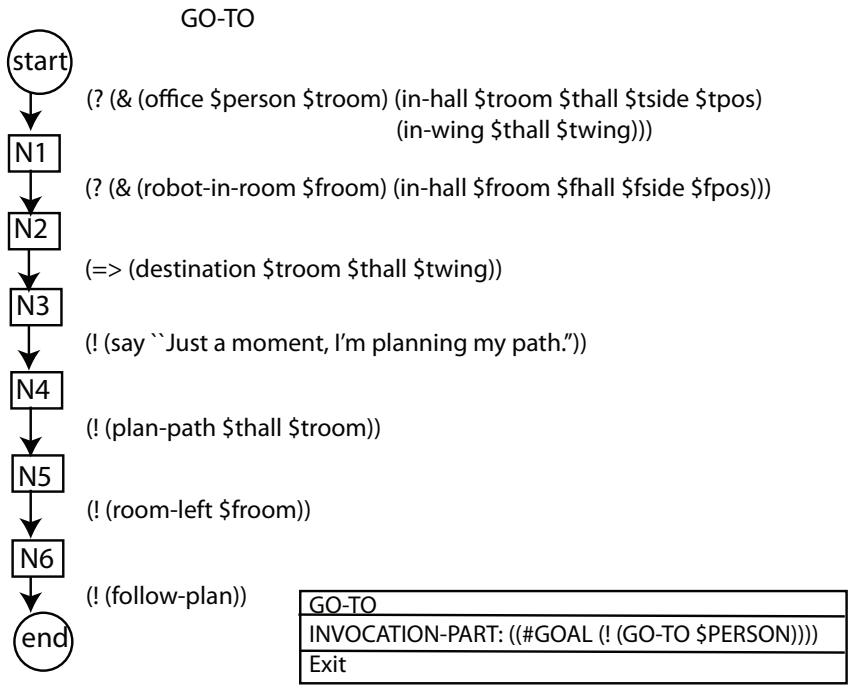

Figure 6.2: The Top Level Strategy

are activated by the plan presence; these KAs translate the route plan into an appropriate sequence of subgoals, generating subgoals from each leg of the plan. Subgoals include updating the database to reflect progress, turning a corner, and so on. A group of navigational KAs then reacts to these subgoals by counting doorways, watching for obstacles, interpreting readings, and so on. For example, while executing the GO-TO KA in Figure 6.2, the robot acquires the goal $(!(room\text{-}left\ \$froom))$, where variable $\$froom$ is bound to the room the robot tries to leave. This goal, in turn, activates the ROOM-LEFT KA in Figure 6.3; the robot performs the steps in the ROOM-LEFT KA to leave the room and to insert the fact $(current\text{-}origin\ \$froom\ \$fhall)$ into its database reflecting its current location, where variables $\$froom$ and $\$fhall$ are bound to specific constants. Next, continuing with the execution of the GO-TO KA, the robot acquires the goal $(!(follow\text{-}plan))$, which causes the FOLLOW-PLAN KA in Figure 6.4 to react. The robot then executes this KA to follow each leg of route plan and reach the destination. As well, beliefs of the form $(current\text{-}origin\ \$locale\ \$spot)$ are continually updated to keep the robot informed of its location and adjust its bearings [36].

Other KAs coordinate the control of the robot. Some of them react to anomalous situations the robot encounters, while other, meta-level KAs determine the relative priorities

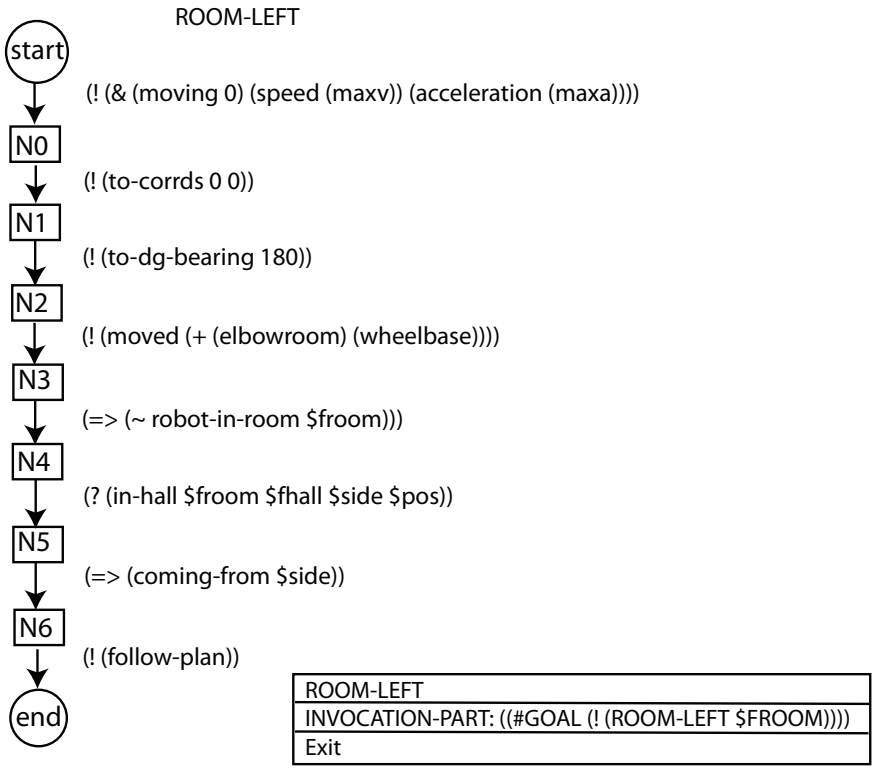

Figure 6.3: Route Navigation KA

of possibly mutually inconsistent goals or choose among various alternatives encountered by the robot as it navigates to the destination. Detailed descriptions of the KAs are provided in [38].

6.4.4 Discussion

PRS is one of the first practical systems for rational reasoning. Specifically, it has been used in a variety of applications including the fault diagnosis in a space shuttle [35] and the control of a mobile robot [38]. Georgeff and Lansky [36] attributed the success of PRS to its partial planning strategy, its reactivity, its use of procedural knowledge, and its meta-level, reflective capabilities.

The PRS architecture is innovative in that it demonstrates how the attribution of beliefs, desires, and intentions to an autonomous system can assist both in interacting with the system and in specifying complex system behaviors [38]. Note that, however, Georgeff et al. was not concerned with the actual implementation of PRS, for example, whether distinct data structures were incorporated to explicitly represent psychological attitudes. Rather,

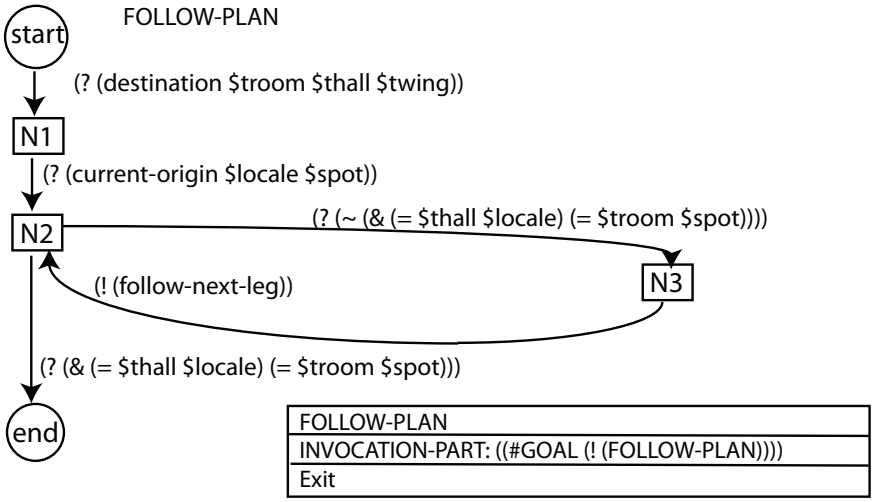

Figure 6.4: Plan Interpretation KA

Georgeff et al. [38] suggested viewing the specification of the PRS system, along with its KAs, as a description of the desired behavior of an autonomous system.

Despite the innovative and practical features of the original PRS [36], it has several limitations that need to be addressed. First, there is a large class of facts that the PRS system must be told in order to function correctly. For instance, if the system does not have knowledge of the agent's starting location, the interpreter aborts. Such problem could be remedied perhaps by incorporating knowledge areas which automatically ask for crucial missing information. Second, reasoning about process interference and synchronization is essential for concurrency control in multi-agent planning. The system should be augmented to integrate more complex reasoning about interprocess communication, such as that described by Cohen and Levesque [19, 51]. Third, some goals are not made as explicit as one would like, but are implicit in the KAs used by the agent. For example, while it is straightforward to handle a goal like "move forward as fast as possible," the agent can only attempt to approximate a goal like "travel along the center of the hallway;" an approximation of the latter goal is given in [38].

Chapter 7

Self-Aware Machines

Recent years have seen a surge of interest in instilling consciousness into machines, both in the sense of self-awareness and in the sense of perceptual awareness. While perceptual awareness is just as essential to an intelligent agent as is self-awareness, self-awareness seems arguably a stronger testament to human level intelligence. For example, while one would not consider a man born with congenital blindness unintelligent or less intelligent than a man with 20/20 eyesight, one would, at the very least, seriously ponder whether a man without a sense of the self or reasoning abilities is really intelligent.

Self-awareness is an actively researched topic in many disciplines ranging from Philosophy, Cognitive Science, to Computer Science. Given its multi-disciplinary implications, self-awareness has many competing definitions. In this chapter, we explore the notion of self-awareness in machines, which broadly include planning systems. As well, we examine the additional capabilities self-awareness affords machines, in our quest to understand how introspection is a prerequisite for human level intelligence in machines, as McCarthy claimed in [57].

We begin this chapter by presenting several views of self-awareness, due to Schubert [84], Minsky [60], and McCarthy [55], in Section 7.1. An investigation of how McCarthy's *mental situation calculus* [57] and Scherl, Levesque, and Lespérance's situation calculus with sensing and indexical knowledge [83] support a machine's introspection is included in Section 7.1.3. This is followed immediately by Section 7.2, which examines introspection in machines. Section 7.3 looks at Anderson and Perlis's *metacognitive loop* [5], an application of self-monitoring and self-guided learning to the design of self-aware machines. Lastly in Section 7.4, we conclude this chapter with Homer [96], an exemplar of a conscious agent exhibiting both perceptual awareness and self-awareness.

7.1 Views of Self-Awareness

Self-awareness, with its multi-disciplinary connections, has been the focus of much research effort in various communities ranging from philosophers, cognitive scientists, to computer scientists. As a consequence of the fertile research, widely varied definitions of self-awareness have been proposed.

In the ensuing sections 7.1.1 - 7.1.3, we review three perspectives of self-awareness, due to Schubert [84], Minsky [60], and McCarthy [55]. Notably, Schubert's notion of *explicit self-awareness* is compared and contrasted with other weaker notions of self-awareness commonly found in the research literature. Minsky's proposed taxonomy of knowledge, in conjunction with self-awareness, is discussed. As well, logically-oriented perspectives of self-awareness are presented, both in the form of McCarthy's *mental situation calculus* [57] and in the form of Scherl, Levesque, and Lespérance's situation calculus with sensing and indexical knowledge [83].

7.1.1 Schubert's Perspective

In [84], Schubert reviewed several competing notions of self-awareness and defined a notion of *explicit self-awareness*. Schubert characterized *explicit self-awareness* as being both human-like and explicit. Specifically, it is human-like in that an explicitly self-aware agent must have a well-elaborated human-like model of the world, including a model of itself and its relationships to the world. The self-model encompasses its beliefs, desires, intentions, knowledge, abilities, autobiography, the current situation, etc; furthermore, the self-model must be amenable to inferences in conjunction with world knowledge, and the system must be capable of goal- and utility-directed planning.

In addition to prescribing the aforementioned human-like capabilities, *explicit self-awareness* is explicit in three respects [84]. First, an agent's representation of self-knowledge must be amenable to self-observation and self-interpretation by the agent. Second, *explicit self-awareness* must be conveyable by the agent, through language or other modalities. Third, such self-awareness is pervious to the same inferential processes as is all other knowledge.

Schubert further enumerated reasons motivating the need for explicit self-awareness. First, given its bootstrapping potential with respect to meta-control, error recovery, autoepistemic reasoning, and learning of skills or facts, explicit self-awareness would help expand the frontiers of Artificial Intelligence [84]. Additionally, an explicit self-aware agent can interact with human users in a transparent, natural, and engaging manner by having

a shared context. Lastly, operational, explicitly self-aware agents can serve as exemplars of entities whose internal basis for self-awareness can be analyzed by consciousness theorists wishing to better understand self-awareness.

Schubert also acknowledged several related, albeit weaker, notions of self-awareness. First, some cognitive scientists, including Anderson and Perlis [5], equate self-monitoring with self-awareness. Even though *self-monitoring* agents monitor, evaluate, and adapt their internal processes purposefully (e.g., for performance optimization or error recovery), they need not have a self-model or general reasoning mechanisms. By way of a simple example, an automatic temperature control system can be considered a self-monitoring agent which takes a room temperature as input, compares the input to a certain threshold, and either turns on the heat if the input drops below the threshold, turns off the heat if it exceeds the threshold, or does nothing otherwise. Another example is Anderson and Perlis's metacognitive loop [5], which we review in Section 7.3.

Another weaker form of self-awareness is captured by *self-explaining* agents. [84] suggests that these agents are able to both recount and justify their actions, and substantiate their inferences; however, they fail to integrate their self-model with general reasoning mechanisms. A prime example is Winograd's SHRDLU [103]. SHRDLU examines its stack of actions, along with their corresponding preconditions and effects, to extract explanations for its actions. Nonetheless, SHRDLU does not genuinely understand its explanations for actions, as it would be stumped by a question probing whether the effects of certain actions could have been achieved by some other actions. Yet another example is Cox's Meta-AQUA system [21] for learning from failures in story comprehension.

A third, weaker from of self-awareness is embodied in the *global workspace systems*. Such systems have a global blackboard which numerous processes share to interact and communicate with one another. The Opportunistic Planning Model invented by Hayes-Roth and Hayes-Roth [44] exemplifies such a system. Schubert [84] argued that although such a structural feature seems to be reflected in Baars's proposed architecture for human consciousness [6], such a feature does not entail reflective thought or any inferential mechanisms.

The last weaker form of self-awareness surveyed by [84] is exhibited by *adaptive, robust, goal-directed* systems. As with the other weaker forms of self-awareness, such a system does not imply a self-model or inferential mechanisms, and thus, either lacks a self-model, or fails to integrate its self-model into general reasoning. For instance, although an antibody system is highly adaptive, robust, and goal-directed in combatting viral intrusions, one would hardly deem an antibody system self-aware in any sense of the word.

Having compared and contrasted these various notions of self-awareness, Schubert [84] then proposed augmentations to McCarthy's logically-oriented enterprize [57] of enabling an agent to reason about its own internal states. Also detailed in [84] are specific knowledge representation and reasoning requirements for self-awareness. In addition to a basic logical framework, these requirements include logical formulations of events, situations, attitudes, autoepistemic inferences, generic knowledge, and various meta-syntactic devices such as axiom schemas, knowledge categorization, knowing or deriving a value, and experience summarization. Readers interested in the details are referred to [84]; as well, McCarthy's proposal [57] is surveyed in our Section 7.1.3.

7.1.2 Minsky's Perspective

Minsky [60] contended that in order for a machine to answer questions about the world or about itself in the world, it must be able to simulate the world and itself procedurally. The pivotal thrust of Minsky's proposal is that if an agent can answer questions regarding a hypothetical experiment without actually executing it, then the agent demonstrates its knowledge of the world; moreover, if this hypothetical experiment involves the agent itself, then the agent also demonstrates its knowledge of itself. Central to this proposal is a computational model of the world which the machine possesses and can simulate to answer questions about actions and events in the world, regardless of whether these actions and events actually occur in the world. Additionally, to be able to answer questions about itself in the world, the machine must also have a model of itself in the world model.

Figure 7.1, recreated from [20], illustrates the taxonomy of knowledge proposed by Minsky in [60]. Schematically, this anticipates the possession of a self-model entailed by Schubert's explicit self-awareness [84]. The machine M exists as the modeler in the world W. M possesses a model of the world $W*$, which, in turn, encompasses a model of the machine $M*$. Whereas M simulates $W*$ to understand and answer questions about actions and events in the world, M simulates $M*$ to answer questions about itself. Accordingly, [20] conceives of $W*$ and $M*$ as the machine's knowledge of the world and its reflective knowledge of itself, respectively. Moreover, embedded in this taxonomy of knowledge are $W**$ and $M**$, which [20] coins *meta-knowledge* and *introspective knowledge*, respectively. The machine requires the former to answer questions about and reason about its model of the world $W*$, as well as the latter to answer questions about and reason about its self-knowledge $M*$.

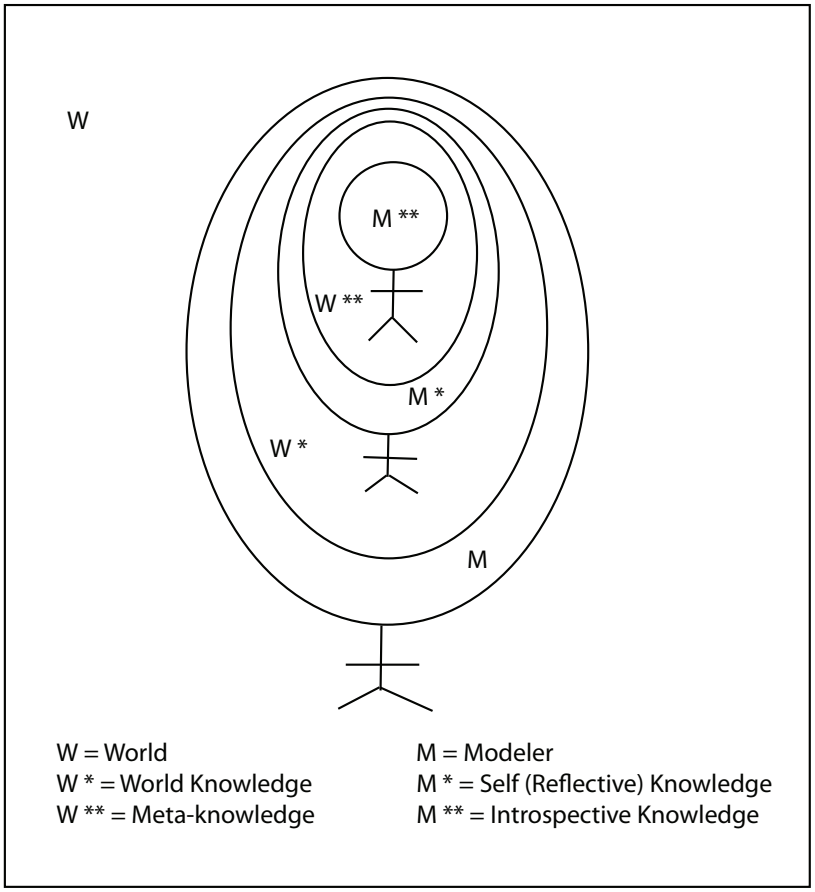

Figure 7.1: A Taxonomy of Knowledge

7.1.3 McCarthy's Perspective

In [55], McCarthy postulated that a machine must declaratively represent its mental states in order to reason about them. This postulation figures prominently in McCarthy's proposal to imbue machines with self-awareness, in the ultimate objective to attain human level intelligence in machines. Also entailed in this proposal is the assignment, to machines, of mental qualities, such as beliefs, wants, intentions, knowledge, abilities, free will, and consciousness, in the *mental situation calculus* [57]. This is the situation calculus with the amendment that many fluents now are knowledge properties, and that various intensional notions are introduced. A proposition or individual concept is distinguished as a first-class object, from the truth value of the proposition or the value of the concept. The former is denoted by a word beginning with an uppercase letter; the latter, by a word in all lowercase letters.

As an example, consider the following sentences from [56] expressing that although Mike and Mary have the same telephone number, Pat knows Mike's telephone number but not Mary's:

$$\left\{ \begin{array}{l} denotation(Telephone(Person)) = telephone(denotation(Person)), \\ denotation(Mike) = mike,\ telephone(mike) = telephone(mary), \\ knows(pat, Telephone(Mike)),\ \neg knows(pat, Telephone(Mary)) \end{array} \right\}$$

Knowing creates an intensional context such that what is in the context cannot be freely substituted for based merely on the equality of actual objects.

Furthermore, McCarthy introduced context as a first-class object, and as such, variables, as well as arguments and results of functions, can assume contexts as their values. Notably, $Ist(c, p)$, or $c : p$, asserts that some proposition p is true in some context c, where p is reified as a term in first-order logic. The uses McCarthy seemed to intend for contexts are either (a) to provide a "here, now, I" context for all propositions that currently hold for the machine, or (b) to assist in hypothetical reasoning.

This formalism enables a machine to observe its own body, its relative position to the environment and its internal variables. As an instance of (a), as given by $C(Here, Now, I) : Lowbattery \wedge In(Screwdriver, Hand3)$ from [57], the machine is aware of its low battery as well as its holding a screwdriver in hand. Additionally, the machine can observe whether it does or does not know the value of a certain term. For example, $C(Now, I) : \neg Know\text{-}whether(Sitting(Clinton))$ from [57] allows the machine to assert that it does not know whether Clinton is currently sitting. Deciding that it does not know and cannot infer

whether Clinton is currently sitting is the motivator that could impel the machine to take actions to find out the answer.

The mental situation calculus introduced in [57] envisions a system in which *knowledge* is not closed under deductive inference, whereas *holds* is. Specifically, $Holds(p, s)$ asserts that some term p representing a proposition holds in some situation s. $Know(q)$ and $Believe(q)$ are among the reified propositions taking some term q. For instance, $Holds(Know(p), s)$ expresses that the machine knows p in situation s, while $Holds(Know(NotKnow(p), s)$ states that the machine knows that it does not know p. The latter kind of assertions are useful in prompting the machine to take actions to acquire knowledge that it currently lacks.

Several kinds of mental events can be represented in the mental situation calculus, including the sequential occurrence of mental events, learning, forgetting, assuming, inferring, and seeing. Let's review an example from [57], making use of $F(p)$, $Occurs(e, s)$, and $Next(p, s)$, where s is a situation, e is an event, and p is a reified proposition. Particularly, $F(p)$ is a reified proposition that p will hold at some time in the future; $Occurs(e, s)$ is a point fluent asserting that e occurs in s; $Next(p, s)$ denotes the next situation following s in which p holds. For example, $Occurs(Learn(p), s) \rightarrow Holds(F(Know(p)), s)$, together with the axiom $Holds(F(p), s) \rightarrow Holds(p, Next(p, s))$, would express that if the machine learns p in situation s, then the machine will know p in the next situation following s in which p holds. The imprecision in how many time steps after s this next state is, nevertheless, unresolved by this formulation.

A similar, logically-oriented endeavor to endue machines with self-awareness was realized by the cognitive robotics project at the University of Toronto. Scherl, Levesque, and Lespérance [83] proposed extending the situation calculus with sensing and indexical knowledge. In this framework, both the knowledge prerequisites and effects of actions can be specified in terms of indexical rather than objective knowledge. This representation was motivated by several observations [50] about the unsuitability of objective knowledge alone for the formalization of actions. Specifically, objective knowledge is neither necessary nor sufficient for actions, nor is it produced by perceptual actions. For example, a robot may not know an object's absolute location, but knows the object's relative location to itself; conversely, a robot may not know its current location, but knows an object's absolute location. As well, [83] claims that a robot's perceptual actions, at best, inform the robot of an object's relative location to the robot.

Both the mental situation calculus [57] and the situation calculus augmented with sensing and indexical knowledge [83] readily lend themselves to a machine's introspection, which

McCarthy claimed to be an essential requirement for a machine to attain human level intelligence. Section 7.2 discusses introspection, illuminating the connection with the mental situation calculus by an example.

7.2 Introspection

Cox [21] suggested that an intelligent agent confronted with a choice in what to do in the world must make three kinds of decisions. First, it must decide which of several available actions is the most suitable in the current situation. Second, it must decide whether its decision in the first step is sufficiently justified to warrant its commitment; i.e., it must cogitate about its cognition. Third, in further evaluating its decisions in the first two steps, it must reflect on its similar, past experiences and weigh reasons for and against its decisions. All three decision processes involve *introspection*, or *metacognition*.

Anderson and Oates [4] gave a fictitious story featuring Rosie the robot maid from the TV show *The Jetsons*, in which Rosie uses introspection to monitor, model and control her cognition. In one episode, Rosie forgets to buy food for the family pet dog and decides to feed the dog human food instead. However, realizing that human food is relatively more expensive and that she is forgetful, Rosie sticks a spare dog collar on her shopping list to serve as a reminder next time she does groceries. In another episode, Rosie consults her to-do list prioritized by task importance, and decides to allocate the computationally intensive but relatively less important task of making arrangements for the Jetsons' vacation to the evening, when her workload is less. The episodes reveal that Rosie employs introspection to know her capabilities and deficiencies, take steps to address the latter, and make the best use of her limited resources.

Cox [21] asserted that being self-aware is not simply perceiving the world, nor is it merely a logical interpretation mapping symbols to relations, objects, and functions in the world.[1] Rather, it behooves a self-aware machine to understand itself well enough. According to Cox [21], this *self-interpretation* [21] is identified with *introspection*, or *metacognition*. Introspection entails understanding oneself well enough to not only explain oneself to others, but also generate a set of learning goals in order to improve the knowledge one uses in decision-making [21].

Similarly, McCarthy [57] defined *introspection* as the observation and reasoning about

[1] Cox seemed to presuppose that some authors regard self-awareness as a property of denotational semantics. However, we are not aware of anyone that holds such a view.

one's own mental states, including one's own beliefs, desires, intentions, abilities, knowledge, and consciousness. One can, thus, propound that all planners are introspective in a way. A planner has knowledge of its capabilities afforded by the repertoire of operators at its disposal, in addition to knowledge of what operators it can invoke and what goals it can readily achieve in a given situation. Furthermore, in solving a given planning problem, a planner has knowledge of its knowledge deficiencies and can remedy such knowledge gaps. For example, to establish a solvable goal, a planner attempts to identify and perform a sequence of actions whose fulfillment leads to goal attainment.

Introspection also enables a planner to infer non-knowledge and do nonmonotonic reasoning. Under the closed world assumption, a planner presumes false anything it cannot prove. As another example, a Prolog planner based on Horn clauses may treat non-knowledge as failure [57]; i.e., for an arbitrary proposition p, if the planner can prove neither p nor $\neg p$, then the planner can infer that it does not know whether p holds. Moreover, an introspective machine reasons about the correctness and consistency of its knowledge base, revising and contracting it appropriately in the face of new information. For example, $C(Now, I) : \neg Know(Telephone(Clinton))$ in the mental situation calculus [57] allows a machine to assert that it does not know Clinton's telephone number. This determination by the machine can, in turn, motivate the machine to plan and act accordingly to get the number. Once the machine has acquired the number, it revises its knowledge base to reflect its knowledge of Clinton's telephone number.

[4] proclaims introspection as a pillar upon which the robustness of an intelligent agent rests. That is, introspection provides a level of decision-making that makes the agent error-tolerant, robust, and responsive to a dynamically changing environment. However, [20] deplores that to date, no truly introspective mechanism has been implemented. As McCarthy noted in [57], although reason maintenance systems record the justifications for their beliefs and perform belief revision and contraction appropriately, they cannot observe or reason about their justification structures, due to a lack of introspective subroutines.

7.3 The Metacognitive Loop

Applying self-monitoring and self-control to the problem of designing self-aware machines, Anderson and Perlis introduced the *metacognitive loop* [5], a self-guided learning strategy by which a machine monitors, reasons about, and modifies its decision-making components in response to perturbations in the environment. The metacognitive loop (MCL) was founded

on the conviction that an intelligent agent should be able to detect when something is amiss by comparing it with a dynamically changing list of anomaly types, evaluate the abnormality, and implement a solution [5].

[5] outlines three key problems in commonsense reasoning. The first problem is *slippage*, the difference between what is believed at a given time and what holds at a later time as time elapses. The *knowledge representation mismatch* problem arises when systems using different sets of representational conventions for a given circumstance fail to recognize one another's representations of the circumstance. The third, *contradiction* problem is concerned with how a machine can reason effectively in the presence of contradictions.

Elgot-Drapkin and Perlis [25] devised *active logic* to implement the MCL. *Active logic*, being a type of paraconsistent logic, does not stipulate that a contradiction entail all sentences in the language; rather, an agent is directed by an identified contradiction to specific issues and problems that need be addressed. Furthermore, each step in an active logic takes one active logic time-step. Since all reasoning also occurs stepwise in time, the time sensitivity and contradiction tolerance of active logic are conducive to effective reasoning in the presence of slippage, knowledge representation mismatch, and contradictions.

The MCL essentially functions as a supervisory module overseeing both the symbolic, inferential modules and the adaptive, self-correcting modules, in a triadic architecture. Therefore, the MCL can direct an agent to learn something that it does not know or has answered incorrectly. In fact, the MCL has been found to improve performance in robot navigation, reinforcement learning, and human-computer dialogues [4]. For example, the MCL contributes to faster reinforcement learners in evolving environments, as well as an agent's accurate interpretaion of users' intentions and recovery from miscommunication failures in dialogues with human users [4]. However, this still contrasts starkly with Schubert's notion of explicit self-awareness [84], in that the MCL does not entail a well-defined self-model, embedded in a broad base of general knowledge about the world.

7.4 Homer

Homer was Vere and Bickmore's endeavor to construct a conscious agent "embedded in time and functioning in a simulated dynamic environment" [96]. The agent is an integration of temporal planning and reasoning, reactive replanning, action execution, equipped with natural language understanding and generation, episodic memory and reflection, and symbolic perception. The creation of the agent had been inspired and rendered possible by the

progress in computer hardware and such AI components as parsers, semantic interpreters, generators, and inference systems [97].

7.4.1 The Environment, Time, and Actions

Homer's simulated, two-dimensional sea-world environment consists of Homer, such fixed elements as a land, an island, a dock, a breakwater, and a pier, as well as such floating objects as logs, icebergs, and mines, ships, and boats. The environment is, for the most part, non-dynamic; almost all objects in the world are motionless, with the exception of a moving ship which patrols back and forth on a fixed path. Rather, changes to the environment, when they do occur, arise as a result of arbitrary manipulation of objects specified by human users.

Homer's environment also has a time component. Agent time is an integer representing the number of seconds that have elapsed since January 1, 1900. Changing the agent time factor from 1 accelerates or decelerates the rate of sweep of agent time relative to real time. The acceleration of agent time makes feasible the testing of scenarios with large time displacements. For implementation reasons, agent times are represented in two varieties. First, absolute agent time is the number of seconds since 1900. Second, relative agent time is the number of seconds since the last reinitialization. Reinitialization is a frequent occurrence in the development and debugging of Homer. At reinitialization, Homer knows only its own location and orientation and that of the fixed objects, and most movable objects are placed randomly in the world.

Homer's action repertoire includes the following capabilities: swimming about within the world; picking up, holding, dropping, shooting, photographing, or perceiving objects; refueling; hearing and generating natural language utterances over a simulated radio-telephone link. Homer's actions are distinguished from external events in the world by their coordinates in the two-dimensional plane and the time at which they occur. Initially, Homer does not know of objects in the environment, and he learns about them only through conversation and perception.

Homer has limited natural language understanding and perception capabilities. He has a vocabulary of roughly 800 words. In his simulated perception system, Homer's line of sight spans 120 degrees and measures a radius of 150 feet. Attributes of perceived objects are fed directly into Homer's episodic memory.

To facilitate interactions with users, a graphical representation of the environment is displayed on a terminal and the agent communicates with human users via a natural language

text interface. Through this interface, Homer answers user-posed questions and responds to user-given tasks such as taking a picture of an object, or picking up and delivering an object.

7.4.2 Architectural Overview

Homer's architecture consists of five main components. These components together interpret and execute user commands, consulting and maintaining two repositories of information – the episodic memory and the dynamic plan. These architectural components are the text interpreter, the reflective processes, the planner based on DEVISER [95], examined in our Section 4.4, the plan interpreter, and the text generator. In this design, components may share information with one another either directly, or indirectly through memory.

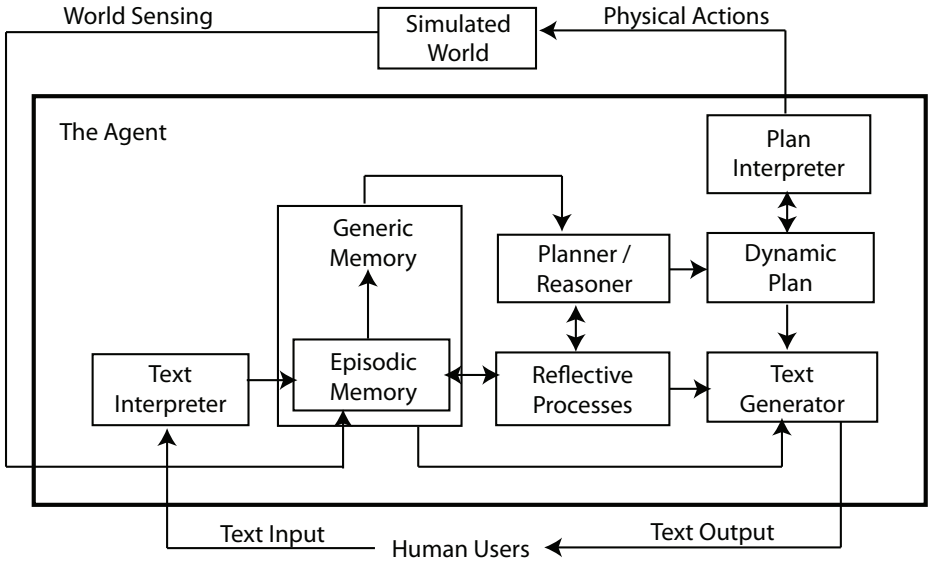

Figure 7.2: The System Block Diagram of Homer

Figure 7.2, recreated from [96], shows the major architectural components and their interactions. They are detailed in subsequent subsections.

7.4.3 The Parser and Natural Language Interpreter

Homer's text interpreter takes English sentences as input and translates them into the state transition representation to be described shortly. Sentences can be commands, questions, or statements of fact. The result of parsing and semantic representation is an *inform event*. An inform event is a list structure that records the information source, which is generally

the user interacting with Homer, the information recipient, the information type, and the time when the information is received. The inform event for an input command is a goal for Homer to accomplish the command. On the other hand, the inform event for an input question is a goal for Homer to answer the question. Inform events are then stored in the episodic memory.

The text interpreter has a moderate coverage of English syntax as follows:

- voice: active and passive;
- tense: past, present, future, and infinitive;
- aspect: simple, perfect, progressive, and perfect progressive;
- verb type: intransitive, transitive, and bitransitive;
- noun type: count and non-count;
- recursive structures, including prepositional phrases, relative clauses, and embedded nominal clauses;
- possessives;
- adjectives, including comparative and superlative forms;
- modal qualification;
- adverbs;
- negated and conjunctive sentences;
- definite noun phrases.

The text interpreter interleaves parsing with semantic interpretation at each step and operates bottom-up, so that semantically meaningless subtrees can be eliminated as early as possible. Since the interpreter runs asynchronously while the words of a sentence are being typed in, most of the parsing and interpretation time is masked by the input time, and the interpretation results are usually ready within a second after the last symbol of the sentence is typed.

7.4.4 The Sentence Generator

Homer's text generator outputs English sentences to convey Homer's actions to the user and to give immediate feedback to the user's input. Particularly, Homer provides three kinds of responses. First, Homer acknowledges a command by replying "OK." Second, he replies "OH" to an assertion. Third, he either gives his answer to a question, or answers "I don't know" when it fails to find an answer. A pattern matcher using declarative production-rule style translations does the mapping from the state-transition representation to sentence specifications.

The *state-transition representation* is a frame-like description of verbs complete with a description of the effect of the action on the world state [96]. It assists in the integration of natural language processing and planning. To plan to use an action, one must have a representation of what the action does to the world state, what the preconditions of the action are, and how the action procedurally expands into subactions if it is a macro-action.

Consider, as an example, the definition for one sense of the verb "give" given by [96]:

$$
\begin{array}{l}
(ACTION\ (.agent\ .recipient\ .thing) \\
\quad (CONSTRAINTS \\
\quad\quad (ISA\ .agent\ ANIMAL) \\
\quad\quad (ISA\ .recipient\ ANIMAL) \\
\quad) \\
\quad (ANTECEDENT \\
\quad\quad (POSSESS\ .agent\ .thing) \\
\quad\quad (GOAL\ .agent\ (POSSESS\ .recipient\ .thing)) \\
\quad) \\
\quad (CONSEQUENT \\
\quad\quad (POSSESS\ .recipient\ .thing) \\
\quad\quad (NOT\ (POSSESS\ .agent\ .thing)) \\
\quad) \\
)
\end{array}
$$

In the aforementioned example, one sense of the word "give" as a bitransitive verb specifies that with both the giver and receiver being animals, if the giver wants the receiver to own an object currently in the giver's possession, then ownership of the object will be transferred to the receiver as a result of the "give" action.

7.4.5 Knowledge and Action Representations

Two types of knowledge exist in the system. The first type, declarative knowledge, is comprised of events in the episodic memory as well as Homer's vocabulary in generic memory. The second type, procedural knowledge, consists of the definition of actions carried out by Homer.

Homer models executable actions in two ways. While the text interpreter uses a linguistic model providing general verb definitions, the planner adopts a model that includes information about Homer's actual action implementations. For example, the action "go" is both a verb and an executable action. While the linguistic model describes the general concept of what it means to go from one place to another, the planner model includes estimates of both fuel consumption and travel time based on the distance to be travelled. Readers interested in the syntax of the dual action representations are referred to [96].

7.4.6 The Episodic Memory and Reflection System

Homer has an episodic memory of all events in his life. All his perceptions are recorded as *sense events*, while sentences including commands and questions in conversations are recorded as *inform events*. All actions performed by the agent are also entered into memory; as well, all personal events are stored and tagged with the time of occurrence in agent time.

The reflective processes in Homer access the episodic memory to process inform events. One demon handles general commands by extracting the goals, processing them, and then passing them to the temporal planner for plan synthesis. Another demon handles questions, extracting goals for the planner but calling the planner in *inference* mode, i.e., only using the planner as an inference engine. Other demons cause Homer to react when spoken to, or to determine when certain natural events have occurred, such as a known object disappearing, an object passing another, an object reaching another, etc.

7.4.7 Plan Execution

Homer's planner operates in two modes. The first mode is for general commands. In this mode, the planner first generates a sequence of actions to achieve a given goal using its knowledge of the world in memory. The planner then stores this information in the dynamic plan ready to be passed on to the plan interpreter for execution. A temporal plan may consist of actions, events, or inferences, and goal protections are implemented and monitored to allow for both detection of plan violation and explanation of motivation

for actions of interest. To answer a user-posed question, the planner works in the second, inference mode, using only inferences and not real actions to obtain its goal, i.e., the answer to the question. Next, the text generator, by accessing the dynamic plan, verbalizes the extracted answer from the inference plan generated by the planner.

The plan interpreter performs plans stored in the dynamic plan. In addition, it ensures that plans are executed in the proper sequence and at the proper time. When a temporal plan is formed, nominal start times for each action are determined. An action is initiated when all its predecessors have finished and when its start time has arrived. If the desired start time has arrived but some predecessors have not finished, then initiation of the action is delayed until all predecessors have finished. When Homer's reactive capabilities (to be described below) are invoked, however, plan execution is temporarily suspended, actions in execution halted and the relevant world state updated. Plan execution resumes once replanning is complete.

7.4.8 Reactive Capability

Dynamic by nature, the planning is reponsive and always subject to the circumstances; e.g., plans can be modified to reflect a more efficient way to perform the task. In addition, plans can also be revised to accommodate changes in the environment caused by other plans or external events. Homer supports some reactivity via two means – replanning and obstacle avoidance. By the latter means, when confronted with an obstacle in its path, Homer will instead follow a path around the obstacle and recompute a path to the original destination from its new location.

Specifically, Homer employs a combination of reactive features:

(1) detection and reaction to plan violation;

(2) detection and reaction to subplan obviation;

(3) excuse-making; and

(4) perseverance after failure to synthesize a plan.

In (1), new information received by Homer, through conversation or perception, is checked to see if it clobbers an old fact that must be protected for the current plan. If so, the relevant portion of the plan is pruned back to the goal, and the planner is reinvoked to recreate the plan. In (2), each new fact entering Homer's memory is matched against the original, uninstantiated form of goals of future planned actions and events. When a

match is detected, the action pyramid above the goal is destroyed, and planner reinvoked to achieve the matched goal. In (3), when Homer fails to form a plan in response to a command, he gives a natural language excuse to explain the problem; e.g., "Sorry, I don't know the location of David." In (4), Homer will reattempt previously impossible goals should the situation become favorable later. To do this, Homer keeps a record of blocking goals, those which themselves could not be achieved and none of whose predecessor goals were achieved either. Not only do blocking goals provide the basis for a verbalized excuse for failing to generate a plan, but they also serve as the trigger for dynamically generated goal reactivation demons.

7.4.9 Discussion

Homer is a paragon of an explicitly self-aware agent. Equipped with an episodic memory, Homer possesses a self-model integrated with general reasoning mechanisms. Furthermore, Homer exhibits coherent behavior in that all of his actions are aimed at accomplishing tasks specified by users. Homer's integrated range of capabilities enable him to respond appropriately to user-posed assertions, commands, and questions. For a given command, Homer's temporal planner first constructs a plan, resolving temporal references with respect to agent time, and then the plan interpreter executes the plan. Homer can also answer questions regarding his experiences, general knowledge, perceptions, present activities, and future intentions.

Homer, albeit anthropomorphized, can hardly be considered a fully alive, humanoid agent. Notably, Homer lacks desires, preferences, and initiatives; Homer, being highly goal-directed, acts only in response to user-specified commands, questions, or assertions, nor does Homer learn from his experiences. Moreover, Homer is not easily extensible or adaptable to new environments, probably due to the mixing of internal variable management with domain symbolism in the procedural representation. Consequently, Homer has not, so far, become a springboard for development of a more sophisticated agent.

Chapter 8

Conclusion of Part II and Future Work

In Chapters 6 and 7, we have investigated various notions of self-awareness and of such mental attitudes as beliefs, desires, and intentions in machines, which broadly include planning systems. Our investigation is also complemented by our review of planning systems endowed with self-awareness and with these mental attitudes.

We begin this chapter with a summary of Chapters 6 and 7 in Section 8.1. Then in Section 8.2, we introduce our preliminary proposal of a self-aware, opportunistic planning agent that maximizes its own cumulative utility while achieving user-specified goals.

8.1 Summary

We have studied the integral role such mental attitudes as beliefs, desires, and intentions play in the design of intelligent agents. First, Bratman's philosophical perspective of intention [14, 15] distinguishes intention from belief and desire by the function roles of intention. Notably, conflicting intentions by an agent cannot exist simultaneously, whereas the agent's conflicting beliefs and desires can coexist. Bratman [14, 15] also propounded that an agent need not intend all the side-effects of an action that it intends to perform.

Influenced by Bratman's proposal, Cohen and Levesque [17] devised an intention-defining logic with modal operators that satisfy Bratman's postulates for intention [15]. Cohen and Levesque's theory encapsulates many intuitive assumptions; e.g., an agent should persist in pursuing its goals insofar as the circumstances warrant this persistence, but the agent should abandon its goals when it determines that the goals cannot or need not be achieved.

A critique of this theory by Singh [89], however, challenges some of the assumptions, claims, and proof steps in [17].

In work related to Cohen and Levesque's theory of intention, Rao and Georgeff [74, 76] developed a logical framework founded on the primitive modalities beliefs, desires, and intentions. This belief-desire-intention (BDI) framework was realized in Georgeff and Lanky's PRS system [36]. PRS is a reactive system that interleaves planning and execution. As such, not only can PRS modify its current plans to accomplish given goals, but it can also completely change its focus and pursue new goals when the situation justifies it. Therefore, PRS is able to rapidly modify its intentions (plans of action) based both on what it believes, intends, and desires and on what it perceives [34].

We have analyzed three views of self-awareness, namely those of Schubert [84], Minsky [60], and McCarthy [55]. Schubert [84] defined *explicit self-awareness* as being both human-like and explicit in character; specifically, an explicitly self-aware agent must have a self-model with a representation amenable to inferences and planning, self-observation, and self-interpretation, and such awareness must be demonstrable and open to inferential processes. Furthermore, Schubert [84] differentiated such awareness from self-monitoring, self-explaining, and that displayed in the global workspace systems or in adaptive, robust, goal-directed systems.

Anderson and Perlis's metacognitive loop [5], for instance, is not an explicitly self-aware agent, but a self-monitoring agent. In this self-guided learning strategy, a self-monitoring agent monitors, reasons about, and modifies its decision-making components in response to changes in the environment. This proposal stands in contrast to Schubert's notion of explicit self-awareness, as such a self-monitoring agent need not possess a well-defined self-model.

Minsky's taxonomy of knowledge [60] shows that a self-aware agent has self-knowledge, world knowledge, meta-knowledge, and introspective knowledge. The first two kinds of knowledge are identified with knowledge represented in its self-model and in its world model, respectively. The agent can simulate these models to answer questions regarding a hypothetical experiment involving itself, the world, its world knowledge, and its self-knowledge, without actually executing the experiment.

McCarthy [57] formalized a machine's self-awareness and mental attitudes in the mental situation calculus, which is situation calculus with the change that many fluents are now knowledge properties. A similar, logically-oriented endeavor to imbue machines with self-awareness was Scherl, Levesque, and Lespérance's situation calculus with sensing and indexical knowledge [83]. Both of these logical formulations assist a machine in introspection,

or metacognition.

Introspection has long since been acknowledged as a requisite property of any intelligent agent [57, 20, 21, 4]. McCarthy [57] defined introspection as the observation and reasoning about one's own mental states, including one's own beliefs, desires, intentions, abilities, knowledge, and consciousness. We have studied how introspection enables a planning agent to infer its non-knowledge, know its capabilities and deficiencies, address its deficiencies and knowledge gaps, and do nonmonotonic reasoning.

Chapter 7 concludes with our discussion of Vere and Bickmore's Homer [96], an exemplar of a conscious agent exhibiting both perceptual awareness and self-awareness. Functioning in a simulated, two-dimensional sea-world environment with a time component, Homer embodies an integration of temporal planning and reasoning, reactive replanning, action execution, limited natural language understanding and generation, episodic memory and reflection, and symbolic perception. Despite Homer's wide range of capabilities, it cannot be considered a fully alive, humanoid agent, as it lacks desires, preferences, and initiatives.

8.2 Future Work

Our survey of the internally motivated planning systems has been motivated by our primary interest in designing and developing self-aware planning agents that can act opportunistically in dynamic environments. By opportunistically, we mean that the agents can recover from unexpected action failures, seize unexpected favorable opportunities, and avoid unexpected threats. More specifically, unanticipated events or circumstances directly trigger suggested actions or goals likely to help exploit them or avert threats. These suggested actions or goals become prime candidates for incorporation into the current plan of the agent, though evaluation of the resulting plan may result in rejection of such suggestions.

Our proposal is to employ such knowledge- and suggestion-driven planning behavior in a utility-maximizing agent capable of both dialogue and action. Furthermore, we wish such an agent to exhibit explicit self-awareness. In theory, a self-aware and opportunistic planning agent achieves user-specified goals while maximizing its own cumulative utility (rewards minus penalties). In essence, this would happen because the agent finds helpful, cooperative behavior rewarding.

To continually maximize the cumulative utility of the current plan, our agent takes a variety of actions. First, the agent can modify the current plan, by adding, removing, expanding, or ordering steps. This includes both top-down goal expansions and the inclusion

and evaluation of suggestions generated from the current situation and general knowledge. Second, the agent can evaluate and annotate the current plan by making inferences about the expected consequences of plan execution, and about the expected cumulative utility. Third, the agent can execute primitive, or immediately executable, steps of the plan.

At first glance, our proposal might bear resemblance to a reinforcement learning (RL) method in its utility-maximizing behaviors. Like RL, our proposal would also employ the discounting of future rewards in calculating the expected cumulative utility. RL is concerned with how an agent should take actions in an environment in order to maximize some notion of a long-term reward [78]. Accordingly, RL methods look for a policy mapping states of the world to the actions the agent should take in those states. However, a few moments' deliberation reveals that the search for a policy in RL methods is guided only by the currently perceived state and involves no reasoning, whereas our proposal entails making inferences about the expected cumulative utility and expected consequences of actions on the basis of what our planning agent perceives, believes, desires, and intends.

In order to understand the importance of an agent's self-awareness and opportunistic behaviors in connection with planning, we plan to conduct experiments by ablation by suppressing one or both of its self-awareness and opportunistic behaviors at a time, while leaving all other conditions constant. That is, we aim to construct convincing and meaningful scenarios to test our agent under four conditions: (1) stripped of its self-awareness and opportunistic behaviors, (2) equipped with both, (3) being self-aware but non-opportunistic, and (4) being opportunistic but not self-aware. Our guiding principle is that the agent's own satisfaction is of the utmost significance to the agent; although the agent achieves user-specified goal, it is not a slave, but rather an adaptive assistant driven primarily by how well it can do for itself . From a utility-centric perspective, the agent satisfies itself also vicariously by assisting human users.

As examples, enumerated below are three scenarios in which a self-aware and opportunistic agent named Robbie can demonstrate his self-awareness in a simulated world.

(SA1) Robbie answers questions asked of him by other animate or inanimate agents or by a human user; example questions include *who are you*, *how old are you*, *what is your current goal*, *why are you asking me that question*, *what do you have*, *where are you coming from*, and so on.

(SA2) Given Robbie's likes and dislikes, Robbie will gain positive utility when experiencing pleasant qualia (e.g., the pleasure of eating or feeling energized, or the satisfaction

of having correctly answered a question), but will suffer negative utility when experiencing unpleasant qualia (e.g., hunger, fatigue, or a sense of having failed).

(SA3) Robbie remembers and knows what it knows or has learned, so as to be able to reason about his actions; for instance, Robbie might try to avoid a stretch of road that he has learned, in a previous excursion, is tougher to navigate due to a steep slope.

Below are two scenarios in which Robbie can exercise his opportunistic behaviors.

(OP1) Robbie sees a one-dollar bill on a deserted sidewalk, picks it up and keeps it. Later, Robbie gives another agent this one-dollar in exchange for something else Robbie desires without having to go to the bank to withdraw additional money.

(OP2) Robbie happens upon a book of jokes and decides to read it as he likes reading in general. Later, when Robbie meets a friend who is despondent, Robbie can try to cheer his friend up by recounting jokes from the book.

Experiments for the aforementioned scenarios, as well as our proposed experiments by ablation, are currently in progress. We are optimistic that results from these experiments will afford us more insight into the integral roles that self-awareness and opportunistic behaviors play in planning.

Bibliography

[1]

[2] J. Allen, J. Hendler, and A. Tate, editors. *Readings in Planning.* Morgan Kaufmann Publishers, San Mateo, CA, 1990.

[3] J. F. Allen. Two views of intention: Comments on Bratman and on Cohen and Levesque. In P. R. Cohen, J. L. Morgan, and M. E. Pollack, editors, *Intentions in Communication*, pages 71–75. MIT Press, 1990.

[4] M. L. Anderson and T. Oates. A review of recent research in reasoning and metareasoning. *AI Magazine*, 28(1):7–16, 2007.

[5] M. L. Anderson and D. R. Perlis. Logic, self-awareness and self-improvement: The metacognitive loop and the problem of brittleness. *Journal of Logic and Computation*, 15(1):21–40, 2005.

[6] B. Baars. *A Cognitive Theory of Consciousness.* Cambridge University Press, New York, NY, 1988.

[7] C. E. Bell and A. Tate. Using temporal constraints to restrict search in a planner. In *Proceedings of the Third Workshop of the Alvey IKBS Programme's Special Interest Group on Planning*, 1985.

[8] W. Bibel. *Automated Theorem Proving.* Vieweg Verlag, Braunschweig, Germany, 1982.

[9] W. Bibel. A deductive solution for plan generation. *New Generation Computing*, 4(2):115–132, 1986.

[10] W. Bibel. Summary of online discussion: Let's plan it deductively! *Electronic News Journal on Reasoning about Actions and Change*, 1:38–45, 1997.

[11] W. Bibel. Let's plan it deductively! *Artificial Intelligence*, 103(1):183–208, 1998.

[12] A. Blum and M. Furst. Fast planning through planning graph analysis. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI 1995)*, pages 1636–1642, 1995.

[13] R. J. Brachman and H. J. Levesque. *Knowledge Representation and Reasoning.* Morgan Kaufmann Publishers, 2003.

[14] M. E. Bratman. *Intentions, Plans, and Practical Reason.* Harvard University Press, Cambridge, MA, 1987.

[15] M. E. Bratman. What is intention? In P. R. Cohen, J. L. Morgan, and M. E. Pollack, editors, *Intentions in Communication*, pages 15–32. MIT Press, 1990.

[16] D. Chapman. Planning for conjunctive goals. *Artificial Intelligence*, 32(3):333–377, 1987.

[17] P. R. Cohen and H. J. Levesque. Intention is choice with commitment. *Artificial Intelligence*, 42(2-3):213–261, 1990.

[18] P. R. Cohen and H. J. Levesque. Persistence, intention, and commitment. In P. R. Cohen, J. L. Morgan, and M. E. Pollack, editors, *Intentions in Communication*, pages 33–69. MIT Press, 1990.

[19] P. R. Cohen and H. J. Levesque. Rational interaction as the basis for communication. In P. R. Cohen, J. L. Morgan, and M. E. Pollack, editors, *Intentions in Communication*, pages 221–256. MIT Press, 1990.

[20] M. T. Cox. Metacognition in computation: A selected research review. *Artificial Intelligence*, 169(2):104–141, 2005.

[21] M. T. Cox. Perpetual self-aware cognitive agents. *Artificial Intelligence*, 28(1):32–45, 2007.

[22] K. Currie and A. Tate. O-PLAN: The open planning architecture. *Artificial Intelligence*, 52(1):49–86, 1991.

[23] T. Dean. Using temporal hierarchies to efficiently maintain large temporal databases. *Journal of the ACM*, 36(4):687–718, 1989.

[24] T. Dean, R. J. Firby, and D. Miller. Hierarchical planning involving deadlines, travel time and resources. *Computational Intelligence*, 4(4):381–389, 1988.

[25] J. Elgot-Drapkin and D. Perlis. Reasoning situated in time I: Basic concepts. *Journal of Experimental and Theoretical Artificial Intelligence*, 2(1):75–98, 1990.

[26] K. Erol, J. Hendler, and D. S. Nau. Complexity results for HTN planning. Technical Report CS-TR-3240, Dept. of Computer Science, University of Maryland, March 1994.

[27] K. Erol, J. Hendler, D. S. Nau, and R. Tsuneto. A critical look at critics in HTN planning. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI 1995)*, pages 1592–1598, 1995.

[28] R. E. Fikes and N. J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[29] R.E. Fikes, P.E. Hart, and N.J. Nilsson. Learning and executing generalized robot plans. *Artificial Intelligence*, 3(4):251–288, 1972.

[30] J. Firby. *Adaptive Execution in Complex Dynamic Worlds*. PhD thesis, Yale University, New Haven, CT, January 2000.

[31] B. Fronhöfer. The action-as-implication paradigm: formal systems and application. *Computer Science Monographs*, 1, 1996.

[32] T. Garvey and R. Kling. User's guide to QA3.5 question-answering system. Technical Report 15, AI Center, SRI International, 333 Ravenswood Ave., Menlo Park, CA 94025, December 1969.

[33] M. P. Georgeff. Planning. In J. Allen, J. Hendler, and A. Tate, editors, *Readings in Planning*, pages 5–25. Morgan Kaufmann Publishers, 1990.

[34] M. P. Georgeff and A. L. Lansky. Procedural knowledge. In *Proceedings of the IEEE Special Issue on Knowledge Representation*, volume 74, pages 1383–1398, 1986.

[35] M. P. Georgeff and A. L. Lansky. A system for reasoning in dynamic domains: Fault diagnosis on the space shuttle. Technical Report 375, AI Center, SRI International, 333 Ravenswood Ave., Menlo Park, CA 94025, January 1986.

[36] M. P. Georgeff and A. L. Lansky. Reactive reasoning and planning. In *Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI 1987)*, pages 677–682, 1987.

[37] M. P. Georgeff, A. L. Lansky, and P. Bessiere. Fast planning through planning graph analysis. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence (IJCAI 1985)*, pages 516–523, 1985.

[38] M. P. Georgeff, A. L. Lansky, and M. J. Schoppers. Reasoning and planning in dynamic domains: An experiment with a mobile robot. Technical Report 380, AI Center, SRI International, 333 Ravenswood Ave., Menlo Park, CA 94025, April 1986.

[39] A. Gerevini and I. Serina. LPG: a planner based on local search for planning graphs. In *Proceedings of the Sixth International Conference on Artificial Intelligence Planning and Scheduling (AIPS 2002)*. AAAI Press, 2002.

[40] A. Gerevini, I. Serina, A. Saetti, and S. Spinoni. Local search techniques for temporal planning in lpg. In *Proceedings of the Thirteenth International Conference on Automated Planning and Scheduling (ICAPS 2003)*. AAAI Press, 2003.

[41] C. C. Green. Application of theorem-proving to problem-solving. In *Proceedings of the First International Joint Conference on Artificial Intelligence (IJCAI 1969)*, pages 219–239, 1969.

[42] C. C. Green and B. Raphael. Research on intelligent question-answering systems. Technical report, AI Center, SRI International, 333 Ravenswood Ave., Menlo Park, CA 94025, May 1967.

[43] D. Harel. Dynamic logic. In D. Gabbay and F. Guenther, editors, *Handbook of Philosophical Logic Volume II – Extensions of Classical Logic*, pages 497–604. Reidel Publishing Company, 1984.

[44] B. Hayes-Roth and F. Hayes-Roth. A cognitive model of planning. *Cognitive Science*, 3:275–310, 1979.

[45] J.Gustafsson and P.Doherty. Embracing occlusion in specifying the indirect effects of actions. In *Proceedings of the Fifth International Conference on the Principles of Knowledge Representation and Reasoning (KR 1996)*, pages 87–98. Morgan Kaufmann Publishers, 1996.

[46] H. Kautz and B. Selman. Pushing the envelope: Planning, propositional logic and stochastic search. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence and the Eighth Innovative Applications of Artificial Intelligence Conference*, pages 1194–1201. AAAI Press / MIT Press, 1996.

[47] H. A. Kautz and B. Selman. Planning as satisfiability. In *Proceedings of the Tenth European Conference on Artificial Intelligence (ECAI 1992)*, pages 359–363, 1992.

[48] R. Kowalski. *Logic for Problem Solving.* Elsevier North Holland, New York, NY, 1979.

[49] R. A. Kowalski and M. Sergot. A logic-based calculus of events. *New Generation Computing*, 4(1):67–95, 1986.

[50] Y. Lespérance and H. Levesque. Indexical knowledge and robot action: A logical account. *Artificial Intelligence*, 72(1-2):69–115, 1995.

[51] H. J. Levesque, P. R. Cohen, and J. H. T. Nunes. On acting together. In *Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI 1990)*, pages 94–99, 1990.

[52] F. Lin. Embracing causality in specifying the indirect effects of actions. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI 1995)*, pages 1985–1991. Morgan Kaufmann Publishers, 1995.

[53] D. McAllester and D. Rosenblitt. Systematic nonlinear planning. In *Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI 1991)*, volume 2, pages 634–639. AAAI Press/MIT Press, 1991.

[54] N. McCain and H. Turner. A causal theory of ramifications and qualifications. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI 1995)*, pages 1978–1984. Morgan Kaufmann Publishers, 1995.

[55] J. McCarthy. Programs with common sense. In M. L. Minsky, editor, *Semantic Information Processing*, pages 403–418. MIT Press, 1968.

[56] J. McCarthy. First order theories of individual concepts and propositions. In *Machine Intelligence 9*, pages 129–147, 1979.

[57] J. McCarthy. Making robots conscious of their mental states. In *Machine Intelligence 15*, pages 3–17, 1995.

[58] J. McCarthy and P. Hayes. Some philosophical problems from the standpoint of Artificial Intelligence. In D. Michie and B. Meltzer, editors, *Machine Intelligence 4*, pages 463–502. Edinburgh University Press, 1969.

[59] D. P. Miller, R. J. Firby, and T. Dean. Deadlines, travel time, and robot problem solving. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence (IJCAI 1985)*, pages 1052–1054, 1985.

[60] M. L. Minsky. Matter, mind, and models. In M. L. Minsky, editor, *Semantic Information Processing*, pages 425–432. MIT Press, 1968.

[61] S. Minton, J. L. Bresina, and M. Drummond. Total-order and partial-order planning: A comparative analysis. *Journal of Artificial Intelligence Research*, 2:227–262, 1994.

[62] R. J. Mooney. Ucpop. Talk on planning at the University of Texas at Austin. Available at `http://www.cs.utexas.edu/ftp/pub/mooney/talks/planning/`.

[63] A. Newell and H. A. Simon. Report on a general problem-solving program. In *Proceedings of the International Conference on Information Processing*, pages 256–264, 1959.

[64] A. Newell and H. A. Simon. Computer simulation of human thinking. *Science*, 134:2011–2017, 1961.

[65] A. Newell and H. A. Simon. GPS, a program that simulates human thought. In E. A. Feigenbaum and J. Feldman, editors, *Computers and Thought*, pages 279–293. R. Oldenbourg KG., 1963.

[66] N. J. Nilsson. *Principles of Artificial Intelligence*. Tioga Publishing Company, Palo Alto, CA, 1980.

[67] N. J. Nilsson. Shakey the robot. Technical Report 323, AI Center, SRI International, 333 Ravenswood Ave., Menlo Park, CA 94025, 1984.

[68] N. J. Nilsson. Triangle tables: A proposal for a robot programming language. Technical Report 347, AI Center, SRI International, 333 Ravenswood Ave., Menlo Park, CA 94025, February 1985.

[69] J. S. Penberthy and D. S. Weld. UCPOP: A sound, complete, partial order planner for ADL. In B. Nebel, C. Rich, and W. Swartout, editors, *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning (KR 1992)*, pages 103–114. Morgan Kaufmann Publishers, 1992.

[70] M. A. Peot and D. E. Smith. Conditional nonlinear planning. In *Proceedings of the First International Conference on Artificial Intelligence Planning Systems*, pages 189–197. Morgan Kaufmann Publishers, 1992.

[71] J. Pinto. Concurrent actions and interacting effects. In A. G. Cohn, L. K. Schubert, and S. C. Shapiro, editors, *Proceedings of the Sixth International Conference on the Principles of Knowledge Representation and Reasoning (KR 1998)*, pages 292–303, 1998.

[72] J. L. Pollock. Planning agents. In A. Rao and M. Wooldridge, editors, *Foundations of Rational Agency*. Kluwer Academic Publishers, 1998.

[73] A. S. Rao and M. P. Georgeff. Deliberation and its role in the formation of intentions. In *Proceedings of the Seventh Conference on Uncertainty in Artificial Intelligence (UAI 1997)*. Morgan Kaufmann Publishers, 1991.

[74] A. S. Rao and M. P. Georgeff. Modeling rational agents within a BDI-architecture. In J. Allen, R. Fikes, and E. Sandewall, editors, *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning (KR 1991)*, pages 473–484. Morgan Kaufmann Publishers, 1991.

[75] A. S. Rao and M. P. Georgeff. An abstract architecture for rational agents. In C. Rich, W. Swartout, and B. Nebel, editors, *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning (KR 1992)*, pages 439–449. Morgan Kaufmann Publishers, 1992.

[76] A. S. Rao and M. P. Georgeff. BDI-agents: from theory to practice. In *Proceedings of the First International Conference on Multiagent Systems*, 1995.

[77] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, Upper Saddle River, NJ, first edition, 1995.

[78] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, Upper Saddle River, NJ, second edition, 2003.

[79] E. D. Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5(2):115–135, 1974.

[80] E. D. Sacerdoti. A structure for plans and behavior. Technical Report 109, AI Center, SRI International, 333 Ravenswood Ave., Menlo Park, CA 94025, 1975.

[81] E. D. Sacerdoti. *A Structure for Plans and Behavior*. American Elsevier, New York, NY, 1977.

[82] E. D. Sacerdoti. The nonlinear nature of plans. In J. Allen, J. Hendler, and A. Tate, editors, *Readings in Planning*, pages 162–170. Morgan Kaufmann Publishers, 1990.

[83] R. Scherl, H. Levesque, and Y. Lespérance. The situation calculus with sensing and indexical knowledge. In M. Koppel and E. Shamir, editors, *Proceedings of the Fourth Bar-Ilan Symposium on Foundations of Artificial Intelligence (BISFAI 1995)*, pages 86–95, 1995.

[84] L. Schubert. Some knowledge representation and reasoning requirements for self-awareness. In M. Anderson and T. Oates, editors, *Metacognition in Computation: Papers from the 2005 AAAI Spring Symposium*, pages 106–113. AAAI Press, 1995.

[85] J. R. Searle. *Intentionality: An Essay in the Philosophy of Mind*. Cambridge University Press, New York, NY, 1988.

[86] M. Shanahan. Event calculus planning revisited. In *Proceedings of the Fourth European Conference on Planning (ECP 1997)*, pages 390–402. Springer-Verlag, 1997.

[87] M. Shanahan. The ramification problem in the event calculus. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI 1999)*, pages 140–146. Morgan Kaufmann Publishers, 1999.

[88] L. Siklossy and J. Dreussi. An efficient robot planner which generates its own procedures. In *Proceedings of the Third International Joint Conference on Artificial Intelligence (IJCAI 1973)*, pages 423–430, 1973.

[89] M. P. Singh. A critical examination of the Cohen-Levesque theory of intention. In *Proceedings of the Tenth European Conference on Artificial Intelligence (ECAI 1992)*, pages 364–368, 1992.

[90] G. J. Sussman. *A Computer Model of Skill Acquisition*. Elsevier Science Inc., New York, NY, 1975.

[91] A. Tate. Generating project networks. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence (IJCAI 1977)*, pages 888–900, 1977.

[92] A. Tate, B. Drabble, and J. Dalton. O-PLAN: A knowledge-based planner and its application to logistics. In *Advanced Planning Technology, the Technological Achievements of the ARPA/Rome Laboratory Planning Initiative*. AAAI Press, 1996.

[93] A. Tate, J. Hendler, and M. Drummond. A review of AI planning techniques. In J. Allen, J. Hendler, and A. Tate, editors, *Readings in Planning*, pages 26–49. Morgan Kaufmann Publishers, 1990.

[94] M. M. Veloso. *Learning by Analogical Reasoning in General Problem-solving*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, 1992.

[95] S. Vere. Planning in time: Windows and durations for activities and goals. *IEEE Tran. on Pattern Analysis and Machine Intelligence*, PAMI-5(3):246–267, 1983.

[96] S. Vere and T. Bickmore. A basic agent. *Computational Intelligence*, 6:41–60, 1990.

[97] S. A. Vere. Organization of the basic agent. *SIGART Bulletin*, 2(4):164–168, 1991.

[98] D. H. D Warren. WARPLAN: A system for generating plans. Technical Report Memo 76, Department of Computational Logic, University of Edinburgh, Edinburgh, Scotland, 1974.

[99] D. H. D Warren. Generating conditional plans and programs. In *Proceedings of the AISB Summer Conference*, pages 344–354, 1976.

[100] D. S. Weld. An introduction to least commitment planning. *AI Magazine*, 15(4):27–61, 1994.

[101] D. E. Wilkins. Domain-independent planning: Representation and plan generation. *Artificial Intelligence*, 22(3):269–301, 1984.

[102] D. E. Wilkins, K. L. Myers, and L. P. Wesley. CYPRESS: Planning and reacting under uncertainity. In *Proceedings of the ARPA/Rome Laboratory Planning and Scheduling Initiative Workshop*, pages 111–120. Morgan Kaufmann Publishers, 1994.

[103] T. Winograd. Procedures as a representation for data in a computer program for understanding natural language. Technical Report 235, Massachusetts Institute of Technology, February 1971.

[104] M. Wooldridge and N. R. Jennings. Agent theories, architectures, and languages: A survey. In *Proceedings of the Workshop on Agent Theories, Architectures, and Languages on Intelligent Agents*, pages 1–39, 1995.

[105] Q. Yang. *Intelligent Planning: A Decomposition and Abstraction Based Approach.* Springer-Verlag, Berlin, Germany, 1997.