How to (plan to) meet a deadline between *Now* and *Then*

Madhura Nirkhe[1] Sarit Kraus[4,2] and Donald Perlis[2,3]

[1]Department of Electrical Engineering
University of Maryland, College Park, MD 20742
[2]Institute for Advanced Computer Studies
University of Maryland, College Park, MD 20742
[3]Department of Computer Science
University of Maryland, College Park, MD 20742
[4]Department of Mathematics and Computer Science
Bar Ilan University Ramat Gan, 52900 Israel

madhura@cs.umd.edu, sarit@bimacs.cs.biu.ac.il, perlis@cs.umd.edu [*]

Abstract

In planning situations involving tight deadlines a commonsense reasoner may spend a substantial amount of the available time in reasoning toward and about the (partial) plan. This reasoning involves, but is not limited to, partial plan formulation, making decisions about available and conceivable alternatives, plan sequencing, and also plan failure and revision. The key observation is that the *time taken in reasoning about a plan brings the deadline closer.* The reasoner should therefore take account of the passage of time during that *same* reasoning, and this accounting must continuously affect every decision under time-pressure. Step-logics were introduced as a mechanism for reasoning situated in time. We employ an extension of them here, called "active logics", to create a logic-based planner that lets a time-situated reasoner keep track of an approaching deadline as he/she makes (and enacts) his/her plan, thereby treating *all* facets of planning (including plan-formation and its simultaneous or subsequent execution) as deadline-coupled. While an agent under severe time-pressure may spend a substantial amount of the available time in reasoning toward and about a plan of action, in a realistic setting the same agent must also measure up to two other crucial resource limitations as well, namely space and computation bounds. We address these concerns and offer some solutions by introducing a limited short-term memory combined with a primitive relevance mechanism, and a limited-capacity inference engine. We propose heuristics to maximize an agent's chances of meeting a deadline with these additional realistic constraints.

[*]This research was supported in part by NSF grant IRI-9311988.

Contents

1 Introduction

Time is the most obvious critical resource in planning with deadline constraints. There is a given moment D in the future by which a goal G must be achieved, and the agent's task is to find a suitable plan to achieve G and enact it before deadline D. This means that both the planning and the enacting of the resulting plan must be take no more than $(D - Now)$ time units, where Now is the initial time at which planning begins. Proper planning often involves "meta-planning", in order to adjudicate between alternative plans, reject infeasible plans, and so on. But that takes time too! Action occurs in the very process of thinking or reasoning, including such meta-reasoning. In [PEDM], it is argued that, traditionally, actions in AI are viewed as separate from the planning process which leads to those actions. Even when the two are intertwined, as in real-time, dynamic or reactive planning, the planning effort is treated as a different kind of beast, not an action itself. Just as it is essential to understand certain features of actions in order to make an intelligent choice of actions in a plan, it is necessary to reflect upon features of planning to make intelligent decisions while planning.

When the reasoning is not carried out *within* but rather only *about* a deadline situation the time for meta-planning does not enter the computation. However, in reality meta-planning often itself must go on as the deadline approaches. To be sure, in some commonly encountered situations the time taken for meta-planning may be very short. But what of highly novel settings in which one cannot *a priori* assign expected utilities to various conceivable options or refinements? Then the planner is forced to decide on utilities and other factors in real time. In these cases it seems unlikely that such meta-planning will always have a modest time cost. Clearly, the emphasis then is not on searching for a theoretically optimal plan, but one which is speculated to work within the deadline. The reasoner must have the flexibility to interleave planning and execution, not only because there may not be enough time to wait until a complete plan is formulated, but because future planning actions may depend upon the outcomes of earlier executions.

The above discussion illustrates the importance of accounting for time of meta-planning as part of overall time of planning and acting. But in general it may be impossible to determine in advance how long meta-planning will take. An alternate perspective, which we explore here, is to simply measure how long planning, meta-planning, and acting are in fact taking, and use this increasing time measure to help decide how to continue in the planning/meta-planning/acting vis-a-vis the approaching deadline.

Thus our approach is not to provide a special technique for precomputing time for meta-reasoning (which we suspect is indeterminate, in general) but rather one in which the reasoning and meta-reasoning are performed together and the time for each is fully accounted for as they occur. We don't precompute how long meta-planning will take; we do some rough estimation of time to perform actions, but chiefly, we track how long planning, meta-planning and acting are taking in real-time, as they occur. Simultaneously with this, we compare the evolving present time elapsed with the approaching dead-line, and this comparison effects decisions about continued planning and acting.

We revise the mechanism of step-logics [DP86a, DP86b, ED88a, EDP90] into so-called "active logics", in solving the fully deadline-coupled planning problem. In contrast to other formalisms for commonsense reasoning, step-logics (and active logics) give the reasoner the ability to recognize that his/her reasoning takes time. What is of special interest to us is not an "ultimate" plan computed in a static world, but a plan which evolves in time in a changing world [Nil83]. Every activity of the reasoner is carried out in fundamental time units called *steps*. The reasoner's thought activity is treated in the same manner as his/her other activities in the outside world. The two are allowed to take place concurrently with each other and with other changes in the world, in particular, with the ticking of a clock. The reasoner has a (largely) declarative inference engine, with some procedural rules[1].

It is worth clarifying how do we use the notion of logic and inference in this paper. A formal logic is often taken to be a semantics to a formal language. While we have done some work on semantical foundations for active-logics described bellow [NKP93], this work is preliminary and not our main focus of this paper. Rather, we are focused on the syntactic rules of inference by which a reasoner proceeds to make and enact plans. Ultimately, of course, one would wish a semantics to match the syntax behavior of the formalism. The formalism described bellow is a logic in the sense that we have a precise notion of axioms and rules of inference and a largely declarative representation. We believe that unlike completely procedural approaches, we have a principled way of incorporating the issues in deadline coupled planning into the knowledge representation. It is also an inference *engine* in the sense that the inference rules are coupled to an external clock that provides a semantics for the crucial predicate "Now".

The paper is organized as follows: bellow we present a sample illustration of a planning problem in which many of the underlying issues surface. In section 2 we present the key technique that allows us to keep track of time as reasoning proceeds, and in section 3 we apply this to deadline planning. Then in section 5 we specialize again to our illustrative example from section 1. In section 6 we take up the issue of resource limitations, introducing short-term memory in our formalism for this purpose, and in section 6.3 we prove that under certain strong assumptions this is no real limitation on the proof-power of the formal logic. In section 6.4 we address heuristics for planning with deadlines. Finally, in section 7 we give related work and in 8 we present conclusions and future work.

An Illustration

To elaborate on the fully deadline-coupled planning problem, we present an illustrative domain, which we call the *Nell & Dudley Scenario*[2]: Nell is tied to the railroad tracks as a train approaches. Dudley must formulate a plan to save her and carry it out before the train reaches her. If we suppose Dudley has never rescued anyone before, then he cannot rely on having any very useful assessment in advance, as to what is worth trying. He must deliberate (plan) in order to decide this, yet as he does so the train draws nearer to Nell. We want to prevent Dudley from spending so much time seeking a theoretically optimal plan to save Nell, that in the meantime the train has run Nell down. Moreover, we want Dudley to do this without much help in the form of expected utilities or other prior computation.

[1]The time taken for executing such procedures, however, is itself accounted for and declaratively represented, and the results of such procedures are also in declarative form. An example is the calculation of WET (*working estimate of time*) discussed later.

[2]This problem was first mentioned in the context of time-dependent reasoning by McDermott [McD78], and more recently discussed in [CL90].

$$\begin{array}{llcl}
0: & & \emptyset & \\
\vdots & & & \\
i: & \ldots & \alpha & \ldots \\
i+1: & \ldots & \alpha \rightarrow \beta, \beta \rightarrow \gamma & \ldots \\
i+2: & \ldots & \beta & \ldots \\
i+3: & \ldots & \gamma & \ldots \\
\vdots & & &
\end{array}$$

Figure 1: Step-logic studies

Thus, he must assess and adjust (meta-plan) his on-going deliberations *vis-a-vis* the passage of time. His total effort (plan, meta-plan and action) must stay within the deadline. He must, in short, reason in time about his own reasoning in time.

The above dramatic life-and-death scenario is deliberately chosen to illustrate a hard deadline setting. Real-life abounds with deadline scenarios. One can envision the use of automated agents in scenarios such as a pilot trying to rescue a wounded soldier before an enemy patrol arrives at the spot or a relief squad trying to deliver aid in the face of an approaching hurricane. A very familiar and interesting hard deadline scenario is *the examination problem* [PEDM]. A student is taking an exam. Initially she spends time planning (i.e., deciding) which problems to attempt first. She may even partially attempt a problem for better assessment. Although this is very useful toward improving her overall performance on the exam, as time ticks, it can not remain so. Here is the basic trade-off: every second spent thinking about strategy is one less second for actually working. A particularly bad outcome is a problem completely worked out in the mind, but no time to write it on paper.

2 How can a logic keep track of time as theorems are proven?

Step-logics [ED88a, EDP90, ED91] were introduced to model a commonsense agent's ongoing process of reasoning in a changing world[3]. They have been extended and renamed as active logics to allow several new features, including limited short-term memory, and the introduction of new expressions into the language over time; only the first of these will be used here.

A step-logic is characterized by a language, observations and inference rules. A *step* is defined as a fundamental unit of inference time. Beliefs are parameterized by the time taken for their inference, and these time parameters can themselves play a role in the specification of the inference rules and axioms. The most obvious way time parameters can enter is via the

[3]Step-logics have also been used for multi-agent coordination without communication using focal points [KR92]. Note that these logics are not "temporal logics" in the usual sense (e.g., [MP92, KL88]), since the notion of present time changes as inferences are drawn.

expression $Now(i)$, indicating the time is now i. Observations are inputs from the external world, and may arise at any step i. When an observation appears, it is considered a belief in the same time-step. Each step of reasoning advances i by 1. At each new step i, the only information available to the agent upon which to base his further reasoning is a snap-shot of his deduction process completed up to and including step $i - 1$.

The agent's world knowledge is in the form of a database of beliefs. These contain domain specific axioms. A number of inference rules constitute the inference engine. Among them may be rules such as Modus Ponens and rules to incorporate new observations into the knowledge base as well as rules specific to deadline-coupled planning such as checking the feasibility of a partial plan or refining a partial plan. Figure 1, adapted from [ED88b] illustrates three steps in a step-logic with Modus Ponens ($\frac{i:\alpha,\alpha\rightarrow\beta}{i+1:\beta}$) as one of its inference rules.

The following features of this framework relate and contrast it to conventional commonsense reasoning systems:[4]

Thinking takes time: Reasoning actions occur concurrently with other physical actions of the agent and with the ticking of a clock. The agent can not only keep track of the approaching deadline as he enacts his plan, but can treat other facets of planning (including plan formulation and its simultaneous or subsequent execution and feasibility analysis) as deadline-coupled. Related to this feature of active-logics is the fact that there is no longer a one final theorem set. Rather, theorems (beliefs) are proven (believed) at certain times and sometimes no longer believed at later times. Provability is time-relative and best thought of in terms of the agent's ongoing lifetime of changing views of the world. This leads to the issue of contradictions below.

Handling contradictions: An agent reasoning with active logic is not omniscient, i.e., his conclusions are not the logical closure of his knowledge at any instant, but rather only those consequences that he has been actually able to draw.[5] Also, since commonsense agents have a multitude of defeasible beliefs, they often encounter contradictions as more knowledge is obtained and default assumptions have to be withdrawn. While a contradiction completely throws an omniscient agent off track (the swamping problem), the active-logic reasoner is not so affected. The agent only has a finite set of conclusions from his past computation, hence contradictions may be detected and resolved in the course of further reasoning.

Inferences are very tightly controlled in our system. Contradictions are handled by a general handling rule in step-logics and by specific rules that manipulate the context sets and plans in the planning framework. Also, even when contradictions do occur, we do not get all possible beliefs, only ones brought in by active logic inferences and these are kept under check and stopped from propagating once the contradiction is recognized. Contradiction discovery drives the reasoning to eliminate default conclusions that no longer hold in the face of new evidence, and subsequently the reasoning ferrets out such inconsistencies.

Nonmonotonicity: Active logics are inherently nonmonotonic, in that further reasoning always leads to retraction of some prior beliefs. The most obvious one is $Now(i)$, which is believed at step i but not at $i + 1$. The nonmonotonic behavior enables the frame-default

[4]This description is necessarily very brief; for details see the various papers by Elgot-Drapkin et al.

[5]Konolige [Kon86], Levesque [Lev84] and Fagin and Halpern [FH88] proposed systems in which the agents are not omniscient. However, the inference time is not explicitly captured in their systems.

reasoning that the commonsense agent must be capable of [MH69].

We remind the reader that these "logics" are combination of formal inference rules and an inference engine and a representational system (a meta interpreter and a meta language) that reasons about the on-going proof process itself by means of external clock. This in fact, is the mechanism that allows time of meta-planning to be factored into overall approach of dead-lines. While it is true that any logic has to function under space and computation limitations, our system has the ability to *reason about its limitations* such as the time taken to make inferences as well as other parameters such as size of the memory or the number of firings in each step or the control heuristics as part of the same framework. Thus reasoning about time, space and computation management is built into the mechanism.

A formal treatment of evolving time makes the knowledge representation issues challenging but interesting because of the novel capability to reason about the reasoning process itself. Another advantage of having as much as possible in declarative form instead of control procedures is the possibility of dynamically changing the parameters or the inference rules either as a result of learning or as a function of the context of reasoning. We note that humans often regard these parameters as dynamic in their reasoning; e.g., people ask for paper and pencil or seek help from other persons or storage aids when it appears that a particular problem has a higher memory requirement and can not ne "solved in one's head".

3 Planning in deadline situations using active-logics

In this section we present a formalism for planning in deadline situations. The approach is deliberately noncommittal with respect to a number of traditional planning issues, such as total or partial order. Indeed, any planning algorithm can be implemented in the active logic framework. In our illustrations we choose to do total order planning to keep the planning as simple as possible while dealing with the temporal aspects. We have not sought to build an optimal planner, not even a state-of-the-art planner; there are many ways to make the planner more sophisticated. Our aim has been first and foremost to couch planning in a fully time-situated framework; further work will be required to incorporate our findings into state-of-the-art techniques for a truly efficient planner. However, in our view, evolving-time is a sufficiently critical issue for real-time deadline coupled planning, that it must be tackled directly (as our effort attempts) no matter what other desirable features may or may not be included such as partial order planning.

We have created a representational language to tackle prototypical variations of the illustrative Nell & Dudley deadline problem. These have been implemented in prolog. A few sample axioms and inference rules can be found in the appendices. We start by providing a brief description of the syntax used in the deadline-coupled planning mechanism based on active logics. A formula $X(S : F, Args)$ consists of a predicate name X which may represent a fluent or an action predicate, with a list of arguments. The first argument denotes the time interval $S : F$ over which the predicate holds and S and F are the first and last end points of the interval, respectively. The other arguments of the predicate follow and are denoted by $Args$ for easy reference. We often wish to express formulas with predicates that hold only over the duration of their interval $S : F$ and do not continue to hold beyond F by persistence. Most of the predicates denoting agent actions (e.g., *Run*, *Shoot*) fall in this category. We denote the time intervals in formulas involving these predicates by $S : \overline{F}$. We use the shorthand S for $S : S$ to denote an instantaneous action over a point time interval. The subscript *obs* indicates that the formula it is attached to is the result of an observation.

Further, we have four different forms of formulas. $X(S : T, Args)$ denotes that X holds over interval $S : T$, $\neg X(S : T, Args)$ denotes that $\neg X$ holds over $S : T$. Further, $X_c(S : T, Args)$ (resp., $\neg X_c(S : T, Args)$) are used to denote that not only does the agent believe in X (resp., $\neg X$) over the interval, but that the agent has reason to believe that the time point S could be a possible point of change of the predicate from $\neg X$ to X (resp., from X to $\neg X$) in the event that $\neg X$ (resp., X) holds in the interval ending in S. For example, $On(2 : 4, floor, ball)$ should be read as the ball was on the floor in the interval 2 : 4 and $On_c(2 : 4, floor, ball)$ is read the ball was on the floor over the interval 2 : 4 and probably wasn't there before 2.

Either of $\{X(S : F, Args), X_c(S : F, Args)\}$ are defined to be in *direct contradiction* with either of $\{\neg X(S : F, Args), \neg X_c(S : F, Args)\}$. A *uniqueness contradiction* exists between formulas $X(S : F, Args1, U, Args2)$ and $X(S : F, Args1, V, Args2)$ if $X(S : F, Args1, U, Args2) \rightarrow \neg X(S : F, Args1, V, Args2)$ whenever $U \neq V$. E.g., $At(5, Dudley, home)$ and $At(5, Dudley, railroad)$ are in uniqueness contradiction.[6] A formula α *du-contradicts* a formula δ, if it is in direct or uniqueness contradiction with δ. The same definitions of contradiction extend to X_c and the negated versions.

We use annotated formulas such as $X(S : F, Args)[\beta_1, \ldots, \beta_k]$ to denote a formula that is derived using the default formulas (projections) $\beta_1, \ldots, \beta_k$ in its proof. Such an annotated formula itself has the status of a default and is as feasible as the weakest default in the annotation as explained in Sections 4.1.1 and 4.1.3.

An action triplet denoted by $[C_A, A, R_A]$ consists of an *action* A preceded and followed, respectively, by a list of *conditions* C_A and *results* R_A. A is a formula with an action predicate and C_A and R_A are lists of formulas[7]. The conditions may need to be true over all or some of the duration of the action. An action may be complex or primitive (atomic). Firing of an inference rule corresponds to a *think* action. Dudley's non-defeasible beliefs are treated as *facts*[8]. Observations incorporate into beliefs in the same time step. Theorems whose premises consist of facts alone are also regarded as facts.

$Now(i)$ denotes Dudley's belief that the time is currently i. A partial plan is a belief $\mathbf{Ppl}(i, p, Triplet\_List)$ denoting a partial plan at step i with the name p. The $Triplet\_List$ is an ordered list of action triplets. We will sometimes use $\mathbf{Ppl_{i,p}}$ to denote the $Triplet\_List$ with respect to i and p.

A special plan with the name *null* is a plan with no actions in it. Dudley simultaneously develops alternative plans towards attaining his goals or subgoals. Each of these partial plans (including the null plan) defines a context within which reasoning can be done about the expected state of the world if the plan were to be carried to completion.

[6]Recall that we use 5 here as a shorthand for 5:5.

[7]Whenever formulas appear in lists such in C_A or R_A and later in beliefs **CS**, **Proj** and **Ppl**, they are in fact treated as if they are "quoted". We omit the quotes to keep the long strings readable. Thus the beliefs of the agent that we will describe shortly are still first order formulas.

[8]Strictly speaking though, the agent only has beliefs, never facts, since even observations are not etched in stone, and may very well change over time. In all the problems that we tackle though, we will treat observations and facts synonymously.

The agent maintains a belief $\mathbf{CS}(i, p, \mathit{Context\_List})$ denoting the **Context_set** for each plan p at each step i. The list *Context_List* consists of quoted formulas (we omit the quotes for readability), and includes all of the facts (observations)[9], formulas corresponding to actions in the plan and formulas that the agent deduces to be true in the state of the world resulting from the successful execution of plan p. We will often use $\mathbf{CS_{i,p}}$ to denote the list *Context_List*. All formulas corresponding to a given predicate X are kept sorted in the list with $S : F$ of each formula as the key. The context set changes with time as the plan undergoes modification and as inferences are made in the context of the plan.

At each step i, the belief $\mathbf{Proj}(i, p, \mathit{Proj\_List})$ denotes the projection that is formed in the context of each partial plan p that is in progress, based on the default of persistence[10]. The i denotes the step number, and *Proj_List* is a list of quoted formulas. We will often use $\mathbf{Proj_{i,p}}$ to denote the list *Proj_List* with respect to i and p.

The belief $\mathbf{WET}(i, p, N)$ denotes the *working estimate of time* for the plan p computed as of step i of reasoning. WET computation is revised at each step by an inference rule and the feasibility of the plan p is continuously checked by making sure that the sum of N and i does not exceed the deadline.

The belief $\mathbf{Goal}(p, G, D)$ is maintained to denote that the plan p is being developed to meet a goal G by a deadline D.

3.1 How long will it take?

A truly time-situated planner must be able to keep track of every unit of time spent, whether it is spent in inferential or physical activity. This also includes the time spent in making estimates of how much time will be spent. Thus, first of all, we need a time-situated estimation mechanism. The WET (working estimate of time) of a plan is a rough estimate of the total time that the plan will consume. It consists of two parts. The PET (*planning estimate of time*) is the (estimated) time spent in reasoning about the plan. This includes plan formulation, refinement, temporal projection and context-based reasoning. The EET (*execution estimate of time*) of the plan is the (estimated) time required to actually execute the actions that have been identified in the plan. Thus, WET = EET + PET.

At each step, the agent works with the partial plan devloped thus far. Since the plan is partial, the planning and acting in fact consists of two components. One works to identify additional actions that must be incorporated to make this a *complete* plan, and the other works to refine and execute the actions that are already part of the current partial plan. We estimate the WET of a plan based on the estimates of the WET's of the actions that are already part of the plan. Thus, for each action (corresponding to a triplet in the plan), we have two estimates. We do not have the mechanism to estimate the WET of the unplanned portion of the plan except for the sliding *Now* which accounts for the time taken to idenfify the remaining portion of the plan. The PET of an action A is computed based on these criteria:

[9]Actually it only consists of the subset of facts that is relevant to the particular partial plan. Section 6 deals with space bounds on the reasoning and proposes a relevance mechanism to keep the reasoning directed to a particular partial plan for a duration of time.

[10]Projections (and persistences) have been studied by numerous authors; see e.g., [Wil83a, Kau86, McD87]. Our treatment is along the lines of time-maps of [DM87].

(1) Whether A is a primitive action that needs refinement.

(2) Whether A has any unbound time variables (whether or not the exact start and finish time of the action is known).

(3) Whether A has any *other* unbound variables in its description.

and (4) Whether A is the first of a sequence of actions or whether its time variables will be bound automatically when the time for a previous action is decided upon.

For a non-primitive action, when (1) is true, we account for atleast one time step in the PET to refine it to the level of primitive actions. If (2) is true, we estimate that it will take atleast one time step to bind the time variables. If (3) is true, we add another step to the PET since there is atleast one step necessary for instantiating the other unbound variables. If according to (4) the action is one of a sequence such that its time variables will be bound whenever those of an earlier action are bound, we substract one from the PET.[11] The examples will illustrate the PET estimation for various actions. We estimate the PET only by a small factor that is an estimate of how long it will take to refine the current action to the level of primitive actions. Basically, for each action that is non-primitive, this adds a constant number of time steps (default is two) that are required to firstly refine the action, and secondly to bind the time variables for actual execution of the action. Thus, we add at least $2n$ to the EET of the plan if there are n non-primitive actions currently in the plan. If a measure of the level of abstraction of an action is available (such as in the representation in the ABSTRIPS planner [Sac73]), that number would reflect the number of steps required to refine the action into primitive level actions, and could be substituted as an estimate in place of the default one step that we currently account for.

The EET for an action is the difference between the start and finish times for the action when it is known. If there is an explicit belief about the EET for a particular action, that number is used to replace this calculation when available.

In our design, the decision to not include an estimate of the (future) meta-planning time[12] into the WET was taken to avoid recursion of meta-meta-meta ... levels of estimation. Time must be spent to choose between alternatives or to adjust the plan to ensure that it does not violate other goals [Wil83a]. Inferencing such as this constitutes the meta-planning that Dudley performs. We do account for the time spent in making these inferences, as they are made. But the WET is restricted to a calculation based on actions in the plan, namely object level actions. This is not a serious disadvantage. We have a uniform approach to treating planning and meta-planning. A meta-level plan will eventually be translated into an object level plan that satisfies the meta-goal. Once at this level, the WET will accommodate the execution time of the meta-plan into the new WET. We give a brief example to illustrate this.[13]

Suppose Dudley has a plan to go out and fetch the newspaper in the morning. However, on a particular morning, it is raining outside. The plan being developed to fetch the newspaper has the ramification that it will cause Dudley to get soaked, and violate the sustenance

[11]The various components, viz. refinement, binding, etc. that constitute the PET of a particular action may be concurrent with those of another action in the plan due to the assumption of unlimited parallelism. In this case, the WET may be estimated to be higher.

[12]Current and past meta-planning time is fully accounted in the sliding **Now** predicate and is factored already into the feasibility analysis.

[13]This example is mentioned in [Wil83a].

goal[14] to keep himself dry. He must then (meta) plan to try and still keep himself dry. The meta-reasoning results in an object level plan to wear a raincoat, which must be merged with the plan to fetch the newspaper. The new WET will continue to reflect only the execution time of the plan to walk outside and fetch the newspaper while the meta-planning proceeds in time. But once the object level plan to wear the raincoat begins to be synthesized, the WET reflects this additional time to look for a raincoat and put it on. As the meta-planning proceeds, time is consumed and is accounted for by the sliding *Now*. In this sense, we have a commonsense formalism for a fully deadline-coupled estimation of the WET. One may argue that the planning time may be too high, and if the WET can not factor that into the computation the estimates will be too low to be useful! That may very well happen if planning continues to introduce only inferential actions into the plan for which no object level estimation is available. With human reasoners, in case the problem involves very complex deliberations that are all inferential, they do not have any idea of how long their reasoning may take. So long as the agent can switch to reasoning about object level actions after a certain amount of thinking, the estimates will not be too low. Between these estimates and the accounting for how much *Now* has changed while making them, we feel that we have a reasonable estimation method for the WET. It has the advantage that it is a simple computation that does not require too much prior knowledge or tedious processing.

Note that the WET is only a rough estimate and hence feasibility conjectures based on it are at best approximate. The agent often tries to estimate an upper bound on the WET, so as to make sure deadlines are met. Deadlines may still be missed because: (1) The WET estimate was not accurate, individual components took longer to execute than expected. (2) The agent experienced sudden unexpected changes that rendered the planning obsolete. (3) Actions in the plan took their estimated time too execute, but, these actions *failed* and did not yield the expected results. (4) A plan had too large WET and was frozen while the dead-line passed.

The following two rules compute the WET of a plan and check for its feasibility at *every* step in the reasoning.[15]

- Computes the WET

$$\frac{i : \mathbf{Ppl}(i, p, \left\{ \begin{bmatrix} C_{A_1} \\ A_1(s_1 : f_1, \ldots) \\ R_{A_1} \end{bmatrix} \cdots \begin{bmatrix} C_{A_k} \\ A_k(s_k : f_k, \ldots) \\ R_{A_k} \end{bmatrix} \right\}), \ldots}{i+1 : \mathbf{WET}(i, p, \sum_{j=1}^{k} EET(A_j) + PET(A_j))}$$

The EET and PET for each action A_j is computed based on the criteria described above. The EET $= (f_j - s_j)$ if known or is obtained from a belief that the agent has at step i regarding the execution time if the difference between start and finish times

[14]A sustenance goal is one which must be preserved during the entire planning process.

[15]We remind the reader that in Active-Logics all rules that are applicable at any step i are in fact applied to draw inferences forming beliefs at step $i+1$. This presupposes unlimited parallelism and is an idealization that is formally convenient, but clearly unrealistic for an implemented agent. In section **??**, we describe work on limited time and space that addresses this problem. Nevertheless, even the idealized version does account for the fact that time is taken in applying inference rules such as calculating WET.

is not known. The PET is computed depending upon the primitiveness and level of instantiation of action A_j taking into account any abstraction level information known about A_j at step i.

The WET is Dudley's calculation of how long his partial plan (formed as of the previous step) will take to refine and execute. This he adds to the current time and compares the result to the deadline to make sure the plan is not hopeless. As long as the WET + *Now* is within the deadline, he declares it **Feasible**, and continues refining and/or putting the partial plan into execution. If the WET computation indicates the plan is not feasible, the plan is frozen (no longer refined for the time being), but maybe used in the future. Our focus here is not to find an optimal heuristic for producing the best plans, but rather to develop the underlining framework for incorporating passage of time into an inference based approach to planning. Within such a framework, numerous experiments contrasting various heuristics can now be undertaken; this is a direction for future work.

- Keeps track of feasibility

$$\frac{i : \mathbf{Ppl}(p, i, \{\ldots\}), \mathbf{Goal}(p, G, Ddl), \mathbf{WET}(i-1, p, \omega)}{i+1 : \mathbf{Feasible}(i, p)}$$

if $\omega + i \leq Ddl$.

4 Categories of actions and time estimates for plans

As Dudley develops a partial plan to save Nell, he continuously refines his estimate of the time needed to carry the plan to completion.[16] In the beginning phase of plan generation, actions are more complex and abstract. Estimates for the execution time for these actions are based on prior experiences with the action or with similar actions and are held as axioms of the form $estimate(<action>, <time\_to\_complete\_action>)$. The estimates may also be acquired as a result of observation. As **Now** changes, and time is spent in planning, the agent substitutes lower level actions into the plan for which closer estimates may be available, up until the level of primitive actions where the estimate is simply the anticipated interval for executing the low level task. In general, the estimate may or may not be separable into individual constituents.

Dudley's database of axioms and rules contains knowledge about actions and their (intended) effects. However, not all actions are the same from the perspective of planning. Especially with regard to planning under time pressure, Dudley may have to estimate differently the time interval for the duration of each action in the plan, depending on the type of the action. We attempt here to formalize some categories of commonly encountered actions from the standpoint of planning.

[16]The WET estimation is one of our concessions to procedural methods: we do not require Dudley to figure out how to do arithmetic but rather allow that he already knows. But we do require him to note the passage of time *during* the execution of the procedure.

- **The *Repeat_until* category**

  (*Repeat* $<$ *action* $>$ *until* $<$ *signaling_condition* $>$) is the form of an action that needs to be performed in a loop. In order to achieve a particular goal, the only known procedure may be to repeat a certain action or sequence of actions. Very often, there is an observation that signals the successful completion of the task. This observation (*signaling_condition*) may or may not coincide with the goal. For example, dialing the telephone repeatedly until a connection is obtained, is a means for establishing contact with someone. Similarly, beating egg whites until stiff peaks appear [Ger90] is a means for beating eggs to the right consistency. An agent must incorporate repeated actions into plans in many day to day situations. The *signaling_condition* is often known to the agent. The inference rules for plan formulation enable the agent to formulate a plan with a *Repeat_until* action, and guide the actual execution of this type of action. A primitive action can be directly acted upon, and is removed from the plan upon its execution. A *Repeat_until* action is a non-primitive action which can be executed, i.e. the repetitive part of it can be executed, but is not removed from the plan unless the *signaling_condition* is observed.

  For most actions that fall in this category, the agent may have an estimate of how long it may take until the *signaling_condition* typically appears. The agent knows what sequence to repeat, but does not know the exact number of times that it must be carried out. For example, in the case of beating egg whites, Dudley may know that this typically takes 3 minutes. If 6 minutes go by and stiff peaks do not appear, it signals a possibility of failure of the *Repeat_until* action[17]. The planning process as well as execution is incremental; the actual number of times the repeat is executed is determined in real-time through execution combined with observation.

  There is a difference in the two examples of *Repeat_until* actions described above. In the case of the egg whites, it is necessary to repeat the action, not because it fails to give the intended result, but more because it is part of a sequence of actions that must be performed in order to achieve the goal. Here, the cumulative effect of the repeats marks the end of the loop. In the example of dialing until a connection is obtained, the agent keeps redialing because the earlier dialings fail. The first successful action marks the end of the loop. However, we will omit this distinction here, and put both examples in the *Repeat_until*[18] category, since, from the planning agent's point of view the plan has the same structure and monitoring must be done to watch out for the *signaling_condition*.

[17]This paper does not offer a formal treatment of plan failure and recovery. An alarm mechanism can be built that signals a potential failure to Dudley in the event that he overshoots his estimate for a *Repeat_until* action by a substantial margin.

[18]Another real-time scenario which illustrates the use of a *Repeat_until* action, is one in which Dudley chases the bad guy in real-time. As the target object moves, Dudley must perform the action of taking one pace in the direction of the current position of the target. At every step 'now' and 'here' must be used as parameters to create a new instance of a pace in the dynamic plan. Dudley may be able to estimate the time required to reach the bad guy from the differential in their respective speeds, and can use it to tailor his plan vis-a-vis the approaching deadline; but the actual number and specification of the paces must match the uncertainties in the changing environment.

- **The** *Conditional_effect* **type actions**

  An axiom for this type of action is often of the form:

  $C \wedge A \rightarrow R$
  ($if < condition(s) > and < action > then < result >$) ,where R is a (sub)goal, and is the result of performing A, given that condition C is satisfied. The condition C must be held true in addition to the list of conditions C_A which are seen as necessary by the agent in order to be able to perform A.

  Within this category, there are two possibilities:

  (a) C is *do-able* under the agent's domain of control, i.e. the agent has an axiom of the form $B \rightarrow C$. The inference rule for planning for this type of axiom is as follows: To make a plan to achieve R, insert A into the plan, and add C along with C_A as the conditions in the triplet corresponding to result R. In our formalism, once inserted, C will also be continuously checked at each step, so that the plan may be altered in the event that C ceases to hold. Further,

  (i) If C is already true in the context of the plan, the agent does not have to plan additionally for it. The estimate of the time for achieving R in this case is not affected by the presence of C.

  (ii) However, if C is not already in the context of the plan, it will be necessary to add action B to the plan to first achieve C and then proceed with A. In this case the estimate should include along with the estimate of A, at least two additional time steps: one is estimated for the addition of B to the plan, and at least one other for executing it. In this case, the estimate is $E + 2$, where $estimate(A, E)$. In subsequent steps, as B is added and refined, a more accurate estimate can be obtained.

  As an example of this category, consider this axiom from the Yale Shooting Problem: $Loaded(T) \wedge shoot(T) \rightarrow \neg alive(T + 1)$. If the goal is to kill Fred and $Load(T) \rightarrow Loaded(T + 1)$ is known, a plan for killing must include along with the conditions for shoot, the condition that the gun must be loaded. Further, in the event that the gun is not already loaded, the $Load(T) \rightarrow Loaded(T + 1)$ axiom suggests additional planning to this end.

  (b) C is not *do-able*, but is merely *observable*, i.e. it is not under the control of the agent. Here there are two cases to consider:

  (i) If C has already been observed and is projected to remain so, planning and time estimation can proceed as in case 2(a)(i).

  (ii) C is an observable condition, but it is not known 'now' whether or not C is true. Then, Dudley must insert into his plan an action to observe whether C is true, and depending on the conclusion, insert A into the plan, in the event that C is indeed true. If several alternatives exist, based on several observable conditions, each suggesting different actions to be undertaken, he must postpone deciding between them for the present. However, he can use the possible list of alternatives to obtain a bound on the estimate, taking the alternative that has the maximum estimate into account.

As an example of 2(b)(ii) consider: Dudley can see that Nell is tied to the tracks, but can not tell from the distance what kinds of knots the bad guy has used in tying the ropes. He knows of the following common kinds of knots from his boy scout days and of particular procedures employed in tying and untying them.

$Clove\_hitch \wedge Untie\_clove\_hitch(T : T+2) \rightarrow \neg Tied(T+2)$
$Timber\_hitch \wedge Untie\_timber\_hitch(T : T+2) \rightarrow \neg Tied(T+2)$
$Unknown\_knot \wedge Cut\_ropes(T : T+8) \rightarrow \neg Tied(T+8)$
$estimate(Untie\_clove\_hitch, 2)$,
$estimate(Untie\_timber\_hitch, 2)$,
$estimate(Cut\_ropes, 8)$

Since Dudley has no control over which knot he will encounter upon arriving at the tracks, he must plan for all contingencies. The decision regarding which action to insert into the plan must wait until the appropriate observation. He could plan to cut the ropes regardless, but that will take the longest time. Thus, by postponing his decision until run-time, Dudley may save on time. He must, however, make sure that the conditions corresponding to all the above alternatives will be satisfied at that point in time, if he wishes to keep the choices to the very end. He must therefore bring a knife to the tracks in case he will have to resort to cutting. He thus creates at this point a kind of pseudo-action (or meta-action) to insert into the plan. This action is the disjunct of all the alternatives. To supplement the plan to take that decision, Dudley inserts an *Observe* action into the plan,[19] which will itself take a time step. The estimate of the pseudo-action is taken to be the maximum of all its disjuncts. The result of the *Observe* action is unknown at the time of planning. At execution time, more will be known, and the pseudo-action can be substituted for the actual action applicable in that context.[20]

- **Actions with a simple formula for an estimate**

  These are the kind of actions for which the agent can determine a time estimate instantly, if an estimate for the *rate* of the action is known, and if an estimate for the *amount* of work to be done is also known. The *Run* action is in this category. Knowing the distance to Nell and his speed of running, Dudley can estimate how long his *Run* will take. But, it is possible that one or both, the distance or the speed may not be known. Dudley must carry out a deliberate observation or calibration to obtain these. E.g., Dudley looks out of the window and sees Nell tied to the rail tracks. He may not know the distance between his house and the tracks. Either he must look it up, ask someone, or do some calibration, such as simple trigonometry. His own running speed, he may know from past experience, or he may need to figure that out too, by running the length of his living room and timing himself as he does so. Such methods

[19]This ties to spatial reasoning, and to aspects of a plan that involve getting more information; for instance Dudley may have to move in order to see whether Nell is tied. This in turn relates to existing work ([KP89], [Dav88]) on ignorance and perception.

[20]This is also linked to the notion of plan commitment. This is a strategy to delay commitment until the last possible instant to allow for more flexibility in planning, of course at the cost of planning for all contingencies and allowing for the time in the on-the-spot decision-making.

for obtaining estimates for actions in this category may be undertaken in circumstances where it is very crucial to obtain these estimates, and further decision making hinges on them. The cost of the more refined methods is obviously the time spent in obtaining them. In our simplified scenario, Dudley knows the distance to the rail track and his speed of running.

- **An action with a fixed interval between its start and finish times**

  This is the simplest category, and includes the kinds of actions which are relatively simple. These actions have a fixed duration. The estimate for an action in this category is the difference between its start and finish times where known. An example of this category of action is 'pull'. It takes one step to pull Nell from the tracks once she is untied.

For each category of actions described , we have described the inference rules for planning for the actions and for the estimation of the time required to carry them to completion. In most cases some form of a time estimate is either known from prior experience, or acquired from observation. In those cases where no known estimates are available to the agent, the unknown estimates are a measure of how much knowledge the agent has regarding the WET of the plan. The currently unknown estimates may potentially be a big drain on time. Dudley keeps a count of how many such unknown estimates exist in each plan. In ongoing work on this front, we are looking at 'agent attitudes' to characterize agents who can use this and other uncertainty information along with perhaps some on-line utility computation, to perform a primitive risk analysis as a basis of choosing between plans which have the same time estimates. An agent who is *risk averse* may choose to go with a plan that is better known even if it has a large WET.

4.1 Temporal reasoning aspects

This section describes Dudley's inference mechanism for temporal reasoning. We describe three inference rules that are crucial for this: temporal projection rule (TP), context set revision rule (CSR) and the restructured modus ponens rule (RMP). In all the active logic scenarios we have underlined new formulas in the context sets or projections, to highlight the differences with the corresponding beliefs at the previous step. In each step the TP rule derives a new projection in the current context and the RMP and CSR rules together deduce a new context set.

4.1.1 Temporal projection rule

The temporal persistence rule (TP) effectively *smoothes* beliefs over time intervals which present gaps in the agent's knowledge. At each step, $\mathbf{Proj_{i,p}}$ holds the results of the temporal projection rule applied to the context set $\mathbf{CS_{i-1,p}}$ of the previous step. Our approach can be best described by a term which we call *parallel projection.* That is, the entire known state of the world at one moment is used to determine the (expected) state at the next moment. Since active-logics are built around the idea of specifying what is known (e.g., proven) *so*

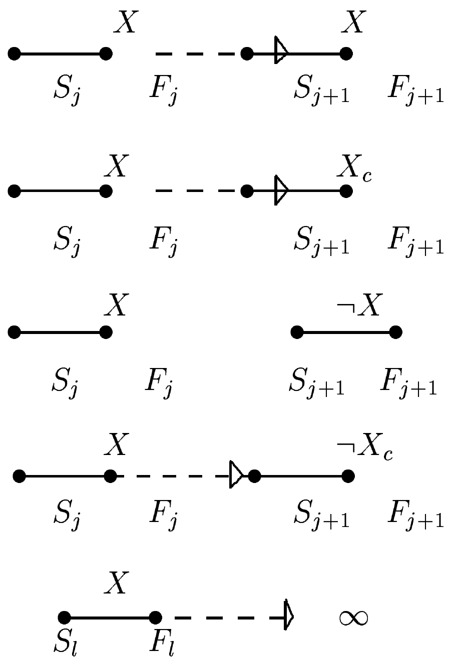

Figure 2: Pictorial description of the TP rule

far, all predicates, and all context sets can be simultaneously reconsidered at each new time step.

Here is a description of the TP rule applied to atomic formulas corresponding to a given predicate X in the context set at step i in order to constitute $\mathbf{Proj_{i+1,p}}$ for the partial plan p. Note that formulas of the form $X(S : \overline{F}, Args)$ are not projected, since they are intended to represent mostly agent actions that do not persist. Formulas $X(S : F, Args)$ for each predicate X are kept sorted in $\mathbf{CS_{i,p}}$ with $(S : F)$ as the key. Only predicates corresponding to *fluents* (i.e. those formulas without the bar on top as in $S : \overline{F}$) are eligible for projection. The formulas in $\mathbf{Proj_{i+1,p}}$ are strictly those that are obtained by persistence of those in $\mathbf{CS_{i,p}}$. Let α_j and α_{j+1} denote consecutive formulas in sorted order in $\mathbf{CS_{i,p}}$ and let α_l denote the last formula in this order. The TP rule can then be described as follows[21]: Figure 2 shows a pictorial description of the TP rule, with the dashed lines denoting the intervals that are filled with the projection.

1. If α_j is of the form $X(S_j : F_j, Args)$ and α_{j+1} is of the form $X(S_{j+1} : F_{j+1}, Args)$ then $\mathbf{Proj_{i+1,p}}$ contains $X(F_j + 1 : S_{j+1} - 1, Args)$ whenever $F_j < S_{j+1}$.

2. If α_j is of the form $X(S_j : F_j, Args)$ and α_{j+1} is of the form $X_c(S_{j+1} : F_{j+1}, Args)$ then $\mathbf{Proj_{i+1,p}}$ contains $X(F_j + 1 : S_{j+1} - 1, Args)$ whenever $F_j < S_{j+1}$.

3. If α_j is of the form $X(S_j : F_j, Args)$ and α_{j+1} is of the form $\neg X_c(S_{j+1} : F_{j+1}, Args)$ then $\mathbf{Proj_{i+1,p}}$ contains $X(F_j + 1 : S_{j+1} - 1, Args)$ whenever $F_j < S_{j+1}$.

[21]For brevity, we only describe the rule for $\alpha_j = X(S_j : F_j, Args)$. The same applies to $\alpha_j = X_c(S_j : F_j, Args)$. The dual form involving $\neg X$ is similar.

4. If α_j is of the form $X(S_j : F_j, Args)$ and α_{j+1} is of the form $\neg X(S_{j+1} : F_{j+1}, Args)$ then $\mathbf{Proj_{i+1,p}}$ does not speculate over the truth or falsity of X over $F_j + 1 : S_{j+1} - 1$. The projection rule will smooth over this interval when further information about a possible point of time where the value of X changes becomes available.

5. If α_l (with the latest interval in $\mathbf{CS_{i,p}}$ corresponding to the predicate X is of the form $X(S_l : F_l, Args)$ or $X_c(S_l : F_l, Args)$ then $\mathbf{Proj_{i+1,p}}$ contains $X(F_l + 1 : \infty, Args)$.

If any of α_j and α_{j+1} in the context set du-contradict, the projection is frozen for the interval in dispute until the contradiction is resolved. In case of a contradiction, it may sometimes be necessary to break up the formulas into two or more parts to identify the extent of the contradiction over some sub interval. e.g., when $X(5)[Y(4)]$ and $\neg X(1 : 6)$ are in a context set, the latter must be split into $\{\neg X(1 : 4), \neg X(5), \neg X(6)\}$ to identify the range of the contradiction.

In case of a contradiction in the context set, between two formulas, α_j and α_{j+1} in the sorted order, Dudley's TP rule can not decide on which of the contradicting formulas to project, until the contradiction is resolved. Often times, one of the contradicting formulas is a fact while the other is weaker since it is only based on projections, and the contradiction is resolved in subsequent steps. The projection is frozen for subsequent α_{j+2}, α_{j+3} , $\ldots$. However, when a fact $\alpha_k = X(S_k : F_k, Args)$ is encountered down the chain, Dudley can resume projection starting with the interval $S_k : F_k$ by applying the above procedure to the remainder of the sorted list with α_k as its first element. Projections are not made in the interval between the contradiction and S_k, until the contradiction is resolved.

Example
Consider a scenario with a bucket filled with oil and a ball lying on the floor. This example illustrates an application of the TP rule to $\mathbf{CS_{15,p}}$ to yield $\mathbf{Proj_{16,p}}$.
$\mathbf{15} : \ldots, \mathbf{CS}(15, p, \{\ldots, \mathit{filled}(0), \mathit{filled}(4), \neg \mathit{filled}_c(7), \mathit{filled}(10),$
$\mathit{on}(0, \mathit{floor}, \mathit{ball}), \neg \mathit{on}_c(5, \mathit{floor}, \mathit{ball}), \ldots\}), \ldots$
$\mathbf{16} : \ldots, \mathbf{Proj}(16, p, \{\ldots, \mathit{filled}(1 : 3), \mathit{filled}(5 : 6), \mathit{filled}(11 : \infty),$
$\mathit{on}(1 : 4, \mathit{floor}, \mathit{ball}), \neg \mathit{on}(6 : \infty, \mathit{floor}, \mathit{ball}), \ldots\}), \ldots$

4.1.2 A restructured modus ponens (resolution) rule

Instead of applying in its familiar form : viz. from α and $\alpha \rightarrow \beta$ deduce β, we choose a representation in clause form and apply a restructured MP in accordance with our philosophy to let earlier defaults play out their effects completely to result in an anticipated state of the world to which later defaults may be applied if necessary. A formula which is a fact has no justification attached to it. All axioms are treated as facts. A formula α which was derived using one or more projections $\beta_1, \beta_2 \ldots$ is only as feasible as the weakest projection, and is itself classified as a default. Such a formula is annotated with the projections used in its derivation and is written as $\alpha[\beta_1, \beta_2, ...]$.

Let $\neg\alpha_1 \vee \neg\alpha_2 \vee \ldots \vee \neg\alpha_n \vee \beta$ be the expression in clause form that appears in the context set $\mathbf{CS_{i,p}}$ of a plan p at step i. This may either be an axiom or an observation. We formulate a rule that adds new atomic formulas (with or without justifications) derived within a context to $\mathbf{CS_{i+1,p}}$. We use the terms, *finished* and *unfinished* to describe a resolution where the results are atomic and non-atomic formulas respectively. The rule only carries over the result of a finished resolution to the context set at the next step[22].

The process of resolution can be outlined as follows:

- All the α_j from $\mathbf{CS_{i,p}}$ which are facts are first used to resolve. If there are no facts that are eligible for resolution, the resolution is not carried out at all. There must be at least one fact among the resolvents for the RMP to fire[23].

- Subsequently, if the resolution is unfinished, members from $\mathbf{CS_{i,p}} \cup \mathbf{Proj_{i,p}}$ which are themselves defaults are next tried. All formulas from $\mathbf{Proj_{i,p}}$ as well as those formulas in $\mathbf{CS_{i,p}}$ that are annotated with projections qualify as defaults. From among those in $\mathbf{Proj_{i,p}}$, those with earlier time parameters are used before the ones with later parameters. For the annotated formulas, the annotation with the latest time parameter that was used in the derivation is used to decide the priority[24].

- The result of the resolution β is then annotated with all the projections used, either directly, or in the annotation of resolving formulas from the context set. The annotations are attached in square brackets to the formulas. This provides the basis for a real-time truth maintenance mechanism which is useful in resolving contradictions.

- If a projection α with a later time than the time of β is used in the RMP application, β is not added to the context set. It is discarded. Thus the axioms are used to derive future conjectures based on projections of current beliefs, but prevented from using future projections to derive past conclusions and from jumping to extreme conclusions.

- If $\beta = X(S : F, \ldots)$ has its time interval $S : F$ such that S is later than the time intervals of all the α_j used in the resolution, then it is marked as $X_c(S : F, \ldots)$ in the context set, to denote that it could be a potential point of inflection in the value of

[22]Since we have the luxury of applying all rules to all formulas at every step, not much is to be gained by adding the results of an unfinished resolution to the context set. We wait to get more information later such as from observations or from further deductions so that an atomic formula can be derived. This serves the purpose of limiting the size of the context set. It is possible to write a version of the RMP that will also add the results of unfinished formulas to the context set, and would effectively function the same but use more space.

[23]The motivation behind this is to reduce the large number of formulas resulting from applying RMP to projections alone, since what can be derived thus is also obtained by the combined effect of RMP and TP. This reduces the actual number of formulas in the context set without loss of any meaningful commonsense conclusions. As an example, consider the axioms $Alive(T) \rightarrow \neg Dead(T)$. Suppose $Alive(0)$ is the only formula in the context set. By TP, the agent would add $Alive(1 : \infty)$ to the projection. and by RMP, $\neg Dead(0)$ to the context-set. Note that $\neg Dead(1 : \infty)$ will subsequently be in the projection, and there is no need to additionally have $\neg Dead(1 : \infty)[Alive(1 : \infty)]$ in the context set. Hence this is a reasonable way to curtail the size of the context set.

[24]In case of a tie, we draw all the conclusions resulting from the use of the projections with identical time intervals, one at a time. This may result in implicit contradictions. In this situation, what the system deduces an expected contradiction.

the predicate X.[25] This marking is of help in deciding whether to project X such as in the TP rule above.

Example
This example illustrates two applications of RMP (in steps 4 – 6) following Dudley's observation of someone dropping a ball into a full bucket. Given the axioms that a ball dropped into a full bucket results in a spill, Dudley concludes that the floor will no longer be dry[26].
Axioms(These are part of the *Context_List* of every context set):
$\neg Filled(T) \vee \neg Drop(\overline{T}, ball) \vee Spill(\overline{T+1})$
$\neg Spill(\overline{T}) \vee \neg Dry(T+1, floor)$

$\mathbf{4}: \mathbf{CS}(4, null, \{\ldots, Dry(0, floor), Filled(0), \underline{Drop(\overline{4}, ball)}\}),$
$\mathbf{Proj}(4, null, \{\ldots, Dry(1:\infty, floor), Filled(1:\infty)\})$

$\mathbf{5}: \mathbf{CS}(5, null, \{\ldots, \underline{Spill_c(\overline{5})[Filled(4)]}, Dry(0, floor), Filled(0), Drop(\overline{4}, ball)\}),$
$\mathbf{Proj}(5, null, \{\ldots, Dry(1:\infty, floor), Filled(1:\infty)\})$

$\mathbf{6}: \mathbf{CS}(6, null, \{\ldots, Spill(\overline{5})[Filled(4)], Dry(0, floor),$
$\underline{\neg Dry_c(6, floor)[Filled(4)]}, Filled(0), Drop(\overline{4}, ball)\}),$
$\mathbf{Proj}(6, null, \{\ldots, Dry(1:\infty), Filled(1:\infty)\})$ [27]

The RMP rule is used in extending the context set. This allows Dudley to compute the extended effects of actions. It also allows him to deduce the future consequences of his planning as it interacts possibly with the actions of other agents or with events observed in the world. It allows for reasoning with the current projection by letting earlier events play out their consequences in an anticipated future before later events. Just as in the Yale Shooting Problem [HM87], there is in principle an un-intuitive outcome, such as that someone emptied the bucket unknown to the Dudley, exactly before the ball was dropped. However, the RMP rule excludes this and other similar outcomes, by dis-allowing later defaults to serve as justifications to form earlier beliefs [NK94]. Thus the conclusions $\neg Filled_c(4)[Dry(6, floor)]$ will not be drawn because $Dry(6, floor)$ is a default with a later time then the potential belief $\neg Filled_c(4)$.

4.1.3 Context set extension and revision rule (CSR)

The CSR rule ensures that the context set is always kept updated to match the most current projection, and the state of the world in which the agent is situated. As explained before,

[25]There is an implicit causality assumption here; earlier events are potential causes in axioms for changes but later events are useful only in explanations of past values, not responsible for changing the past values. Note that we say a *potential* point of change.

[26]This is an extended effect of the spill; we will elaborate later on the significance of this type of reasoning.

[27]Note that there is a contradiction between the **CS** and the **Proj**. Underlying idea is that projections have a default status, and are not really treated as true (unlike facts), and cautiously used in derivations. The agent keeps track of their use and makes necessary adjustments to iron out the inconsistencies in time. In this example the contradiction is resolved in the next step.

formulas are annotated by the projections which are used to support them in future conjectures. In the event that the projections cease to hold as of "now", the formulas that are supported by them are dropped from the context set in the revision process. The revision is a kind of real-time truth maintenance. The CSR rule also plays the important role of resolving contradictions in a time situated manner.

Following is a description of the rule used in deciding the contents of $\mathbf{CS_{i+1,p}}$ based on the contents of $\mathbf{CS_{i,p}}$, $\mathbf{Proj_{i,p}}$ and $\mathbf{Ppl_{i,p}}$. In Part I we decide a set $\mathbf{Candid_{i,p}}$ selected from $\mathbf{CS_{i,p}}$ which are formulas to be considered as candidates for retention. Part II decides which members of $\mathbf{Candid_{i,p}}$ will make it to $\mathbf{CS_{i+1,p}}$.

Part I (Select candidate formulas to inherit):

1. If two formulas α and δ in $\mathbf{CS_{i,p}}$ du-contradict each other, then the following criteria are used in deciding which of them go to $\mathbf{Candid_{i,p}}$[28].
   (a) If α is a fact, while δ is a default (is annotated with a projection), select α and reject δ to go into $\mathbf{Candid_{i,p}}$.
   (b) If α and δ are both defaults, select neither[29] to go into $\mathbf{Candid_{i,p}}$.
2. Formulas that are not part of a contradiction go into $\mathbf{Candid_{i,p}}$.

Part II (Choose among candidate formulas):

1. A formula α from $\mathbf{Candid_{i,p}}$ which is a fact is inherited to $\mathbf{CS_{i+1,p}}$.
2. A formula $\alpha[\beta_1, \beta_2, \ldots, \beta_k]$ which is a default is inherited unless for some $1 \leq j \leq k$, $\beta_j \notin \mathbf{Proj_{i,p}}$. Also, if any of $\beta_1, \ldots, \beta_k$ now appear as facts in $\mathbf{CS_{i,p}}$, they are removed from the annotation.
3. Formulas corresponding to actions that are added to the plan in the previous step are added to the $\mathbf{CS_{i+1,p}}$[30].

We remark here, that there are two more rules which fire to add formulas to $\mathbf{CS_{i+1,p}}$. One is the OBS rule which add new observations that are made in step $i + 1$ to $\mathbf{CS_{i+1,p}}$. The other is RMP, which was described before in detail.

Example

In step 5 in this example, Dudley concludes that there must have been a spill at step 5 based on the projection that the bucket was still filled at step 4. At the same time, however, he is

[28]Note that we do not encounter situations in which facts (direct or indirect descendents of observations alone) contradict each other. Observations with different time intervals involving the same predicate may well disagree, but these are not contradictory.

[29]Where both are defeasible beliefs, a working strategy is to not inherit either of them, and to continue the reasoning to see if one of them will reappear in the face of stronger evidence.

[30]Formulae in the context set are in fact, doubly annotated in the implementation, with the projections if any used in their derivation, and with the action(s) in the plan that are used in their derivation. Should the plan get revised to no longer require any of these actions, the corresponding formula is not inherited in the CS.

told by a reliable observer that the bucket was in fact not filled at step 4 and adopts it as a fact.

The projection catches up at step 6 to no longer believe $Filled(4)$. As a result of CSR, $Spill(\overline{5})[Filled(4)]$ and $\neg Dry_c(6, floor)[Filled(4)]$ are no longer inherited to the context set at step 7. Note that the projection at step 7 already reflects a wet floor. This will also be subsequently revised in step 8 by an application of the TP rule, since $\neg Dry_c(6, floor)[Filled(4)]$ is no longer in $\mathbf{CS_{7,null}}$.

$\mathbf{4}: \mathbf{CS}(4, null, \{\ldots, Dry(0, floor), Filled(0), \underline{Drop(\overline{4}, ball)\}}),$
$\mathbf{Proj}(4, null, \{\ldots, Dry(1:\infty, floor), Filled(1:\infty)\})$

$\mathbf{5}: \mathbf{CS}(5, null, \{\ldots, \underline{Spill(\overline{5})[Filled(4)]}, Dry(0, floor), Filled(0), \underline{\neg Filled(4)}, Drop(\overline{4}, ball)\}),$
$\mathbf{Proj}(5, null, \{\ldots, Dry(1:\infty, floor), Filled(1:\infty)\})$

$\mathbf{6}: \mathbf{CS}(6, null, \{\ldots, Spill(\overline{5})[Filled(4)], Dry(0, floor), Filled(0), \neg Filled(4),$
$\underline{\neg Dry_c(6, floor)[Filled(4)]}, Drop(\overline{4}, ball)\}),$
$\mathbf{Proj}(6, null, \{\ldots, Dry(1:\infty, floor), \underline{\neg Filled(5:\infty)}\})$

$\mathbf{7}: \mathbf{CS}(7, null, \{\ldots, Dry(0, floor), Filled(0), \neg Filled(4), Drop(\overline{4}, ball)\}),$
$\mathbf{Proj}(7, null, \{\ldots, \underline{Dry(1:5, floor)}, \underline{\neg Dry(7:\infty, floor)}, \neg Filled(5:\infty)\})$

$\mathbf{8}: \mathbf{CS}(8, null, \{\ldots, Dry(0, floor), Filled(0), \neg Filled(4), Drop(\overline{4}, ball)\}),$
$\mathbf{Proj}(8, null, \{\ldots, \underline{Dry(1:\infty, floor)}, \neg Filled(5:\infty)\})$
$\vdots$

5 Examples: Dudley, Nell and the rushing train

In this section we present several examples starting a very simple one, to illustrate the notation and operation of inference rules applied to the Nell and Dudley scenario from section 1. After that, we consider slightly more complex version of this scenario.

5.1 A simple case

To give a flavor of the deadline-coupled reasoning, we first consider a very simple scenario and show some steps from Dudley's real-time reasoning.[31] Dudley knows that Nell is a distance of 30 'paces' from him when he first realizes (step 0) that the train will reach her in 50 time units. He begins to form a plan, seen below in step 1 as **Ppl** (partial plan), and refines the plan in subsequent steps. The deadline is 50 in this example, d is Dudley, n is Nell, h denotes home and r the railroad track. Subscripted t's indicate times (step numbers). **Proj** stands for projection; *save*, that appears as argument to **Ppl**, **Proj** and **Feasible** in step 1, is a label naming the plan he is forming. $X(S:T,\ldots)$ denotes that the predicate X

[31] For fuller details see [KNP90, NKP91].

holds over the interval $S : T$. A point interval $T : T$ is written simply as T. The $\bullet\!\!\rightarrow$ as it appears in $X(S : T \bullet\!\!\rightarrow R, \ldots)$ denotes that X is intended to hold beyond $S : T$ up to R (by default). Its use in a result of an action indicates that the result must be preserved for use in a later segment of the plan. The number at the right bottom corner of a triplet denotes its place in the plan sequence.

Bellow we indicate the steps of Dudley reasoning beginning at step 0, and continuing to step 42, when Dudley has both formed and enacted a plan to save Nell.

Step 0:
$\mathbf{CS}(0, \mathit{null}, \{\ldots, \mathit{At}(0, d, h)_{obs}, r - h = 30_{obs}, \mathit{Tied}(0, n, r)_{obs}\})$,
$\mathbf{Proj}(0, \mathit{null}\{\})$,
$\mathbf{Goal}(\mathit{save}, \mathit{Out\_of\_danger}(50, n, r), 50)$,
$\mathbf{Unsolved}(0, \mathit{Out\_of\_danger}(50, n, r))$, ...
(Step 0 represents Dudley's state of mind before planning had begun.)

Step 1:
$\mathbf{CS}(1, \mathit{null}, \{\ldots, \mathit{At}(0, d, h)_{obs}, r - h = 30_{obs}, \mathit{Tied}(0, n, r)_{obs}\})$,
$\mathbf{Proj}(1, \mathit{null}, \{\mathit{At}(1 : \infty, d, h), \mathit{Tied}(1 : \infty, n, r)\})$,
$\mathbf{CS}(1, \mathit{save}, \{\ldots, \mathit{At}(0, d, h)_{obs}, \mathit{Tied}(0, n, r)_{obs}\})$,

$$\mathbf{Ppl}(1, \mathit{save}, \left\{ \begin{bmatrix} \neg\mathit{Tied}(t_1, n, r) \\ \mathit{Pull}(t_1 : \bar{t}_2, d, n, r) \\ \mathit{Out\_of\_danger}(t_2 \bullet\!\!\rightarrow 50, n, r) \end{bmatrix}_1 \right\}),$$

$\mathbf{Proj}(1, \mathit{save}, \{\})$,
$\mathbf{WET}(1, \mathit{save}, 0)$,
$\mathbf{Feasible}(1, \mathit{save})$, ...
(A new plan called "save" is begun and is initially declared to be feasible.)

Step 2:
$\mathbf{CS}(2, \mathit{save}, \{\ldots, \mathit{At}(0, d, h)_{obs}, \mathit{Tied}(0, n, r)_{obs}, \mathit{Pull}(t_1 : \bar{t}_2, d, n, r), r - h = 30_{obs}, t_2 \leq 50, t_1 = t_2 - 1, t_3 = t_4 - 3\})$,

$$\mathbf{Ppl}(2, \mathit{save}, \left\{ \begin{bmatrix} \mathit{At}(t_3 : t_4, d, r) \\ \mathit{Release}(t_3 : \bar{t}_4, d, n, r) \\ \neg\mathit{Tied}(t_4 \bullet\!\!\rightarrow t_1, n, r) \end{bmatrix}_1 \begin{bmatrix} \neg\mathit{Tied}(t_1, n, r) \\ \mathit{Pull}(t_1 : \bar{t}_2, d, n, r) \\ \mathit{Out\_of\_danger}(t_2 \bullet\!\!\rightarrow 50, n, r) \end{bmatrix}_2 \right\}),$$

$10\mathbf{Proj}(2, \mathit{save}, \{\mathit{At}(1 : \infty, d, h), \mathit{Tied}(1 : \infty, n, r)\})$,
$\mathbf{WET}(2, \mathit{save}, 2)$,
$\mathbf{Feasible}(2, \mathit{save})$, ...
(Plan refinements began and now for the first time WET and feasibility are actually computed. Since the plan in step 1 includes only one action: *Pull*. The PET for *Pull* is the time required to bind its time variables. There are no other uninstantiated variables, and it is not part of a sequence. It is primitive action which does not need further refinement. Since it takes one time step, EET for it is 1. Thus WET for *Pull* is 2, which is also the WET for the partial plan *save*. For brevity we suppressed the null plan. The pull action in the partial plan of step 1, is added to the CS of step 2, indicating that the pull action will occur in the context of the plan "save".)

Step 3:

CS$(3, save, \{\ldots, At(0, d, h)_{obs}, r - h = 30_{obs}, Tied(0, n, r)_{obs},$
$Pull(t_1 : \bar{t}_2, d, n, r), Out\_of\_danger_c(t_2, n, r),$
$Release_1(t_3 : \bar{t}_4, d, n, r), t_2 \leq 50, t_1 = t_2 - 1, t_3 = t_4 - 3, t_4 \leq t_1\}),$

$$\mathbf{Ppl}(3, save, \left\{ \begin{bmatrix} At(t_6, d, L) \\ Run(t_6 : \bar{t}_7, d, L : r) \\ At(t_7 \bullet\!\!\to t_3, d, r) \end{bmatrix}_1 \begin{bmatrix} At(t_3 : t_3 + 1, d, r) \\ Release_1(t_3 : \bar{t}_3 + 1, d, n, r) \\ \neg Tied(t_3 + 1 \bullet\!\!\to t_1, n, r) \end{bmatrix}_{2\ldots} \begin{bmatrix} At(t_3 + 2 : t_4, d, r) \\ Release_3(t_3 + 2 : \bar{t}_4, d, n, r) \\ \neg Tied(t_4 \bullet\!\!\to t_1, n, r) \end{bmatrix}_{4\ldots}\right.$$

Proj$(3, save, \{At(1 : \infty, d, h), Tied(1 : \infty, n, r)\}),$
WET$(3, save, 7),$
Feasible$(3, save), \ \ldots$
(Since the consequence of the pull action is that Nell will be out of danger, this is added to the CS of step 3 as a result of applying RMP to the appropriate axiom. WET for the *Pull* is 2 as explained in step 2, that does not change. As far as the *Release* is concerned, its PET is 2 (one to bind the time variables, and another to refine it into primitive actions) and its EET is 3. Thus WET for Release sums to 5. Thus the WET for the plan (as of the previous step) is 7. This is reflected in the WET belief.)

Step 4:

CS$(4, save, \{\ldots, At(0, d, h)_{obs}, r - h = 30_{obs}, Tied(0, n, r)_{obs}, Pull(t_1 :$
$\bar{t}_2, d, n, r), Out\_of\_danger_c(t_2, n, r),$
$Release_1(t_3 : \overline{t_3 + 1}, d, n, r), \ldots, Run(t_6 : \bar{t}_7, d, L : r), \neg Tied_c(t_4, n, r),$
$t_2 \leq 50, t_1 = t_2 - 1, t_3 = t_4 - 3, t_4 \leq t_1, t_6 < t_7, t_7 \leq t_3\}),$

$$\mathbf{Ppl}(4, save, \left\{ \begin{bmatrix} At(t_6, d, h) \\ Run(t_6 : \bar{t}_7, d, h : r) \\ At(t_7 \bullet\!\!\to t_3, d, r) \end{bmatrix}_1 \begin{bmatrix} At(t_3 : t_3 + 1, d, r) \\ Release_1(t_3 : \overline{t_3 + 1}, d, n, r) \\ \neg Tied(t_3 + 1 \bullet\!\!\to t_1, n, r) \end{bmatrix}_{2\ldots} \right\}),$$

Proj$(4, save, \{At(1 : \infty, d, h), Tied(1 : \infty, n, r), Out\_of\_danger(t_2 + 1 : \infty, n, r), \}),$
WET$(4, save, 9),$
Feasible$(4, save), \ \ldots$
(Planning continues as above. The plan in Step 3 consists of three new primitive actions obtained by refining *Release* into its three components. Out of these, the first, namely, $Release_1$ has a PET of 1, which is the step required to bind the time variables, since it is the first of the sequence of the three actions that constitute the *Release*. Once this is bound, the times of the other two are decided automatically. Thus PET for $Release_2$ and $Release_3$ are subsequently zero. The EET for each of them is 1. The *Run* action has a PET of 3 (one to bind the time variables, 1 to refine it, and 1 to bind the other variables). Thus the WET of the plan is now believed to be 9. Also, notice that in this step, the variable L in the *Run* action has been bound to h by looking it up in the projection.)

Step 5:

CS$(5, save, \{\ldots, At(0, d, h)_{obs}, At_c(t_7, d, r), Run(t_6 : \bar{t}_7, d, h : r), r - h = 30_{obs}, Tied(0, n, r)_{obs},$
$Pull(t_1 : \bar{t}_2, d, n, r), Out\_of\_danger_c(t_2, n, r), Release_1(t_3 : \overline{t_3 + 1}, d, n, r) \neg Tied_c(t_4, n, r),$
$t_2 \leq 50, t_1 = t_2 - 1, t_3 = t_4 - 3, t_4 \leq t_1, t_6 = t_7 - 30, t_7 \leq t_3)\}),$

$$\mathbf{Ppl}(5, save, \left\{ \begin{array}{c} \begin{bmatrix} At(t_6, d, h) \\ Pace(t_6 : \overline{t_6+1}, d, h : h+1) \\ At(t_6+1, d, h+1) \end{bmatrix}_1 \\ \begin{bmatrix} At(t_6+1, d, h+1) \\ Pace(t_6+1 : \overline{t_6+2}, d, h+1 : h+2) \\ At(t_6+2t_3, d, h+2) \end{bmatrix}_2 \\ \dots \\ \begin{bmatrix} At(t_6+29, d, h+29) \\ Pace(t_6+29 : \overline{t_6+30}, d, h+29 : r) \\ At(t_7 \bullet\!\!\rightarrow t_3, d, r) \end{bmatrix}_{30\dots} \end{array} \right\}),$$

$\mathbf{Proj}(5, save, \{At(1:\infty, d, h), Out\_of\_danger(t_2+1:\infty, n, r),$
$Tied(1:t_4-1, n, r), \neg Tied(t_4+1:\infty, n, r),\}),$
$\mathbf{WET}(5, save, 38),$
$\mathbf{Feasible}(5, save), \ \dots$
(Because in step 4 it was deduced that Nell will be untied by time t_4, then in step 5 the projection is revised so that Nell is tied only till $t_4 - 1$ instead of infinity as before. The WET at last now takes on significance (becomes large) since the 30 steps that Dudley must run have been included in his calculations, once the difference between the start and finish times of the *Run* is known. The PET of the *Run* with L instantiated is now 2, but its EET is 30. This takes the combined WET to 38. The partial plan is significantly refined in this step to include the paces that he will run.)

Step 6:
$\mathbf{CS}(6, save, \{\dots, At(0, d, h)_{obs}, At_c(t_7, d, r), Run(t_6 : \overline{t}_7, d, h : r), r - h =$
$30_{obs}, Tied(0, n, r)_{obs}, \neg Tied_c(t_4, n, r),$
$Pull(t_1 : \overline{t}_2, d, n, r), Out\_of\_danger_c(t_2, n, r),$
$Pace(t_6 : t_6+1, d, h : h+1), \dots Pace(t_6+29 : t_6+30, d, h : r),$
$Release_1(t_3 : \overline{t_3+1}, d, n, r), t_2 \leq 50, t_1 = t_2 - 1,$
$t_3 = t_4 - 3, t_4 \leq t_1, t_6 = 6, t_6 = t_7 - 30, t_7 \leq t_3)\}),$

$$\mathbf{Ppl}(6, save, \left\{ \begin{array}{c} \begin{bmatrix} At(6, d, h) \\ Pace(6 : \overline{7}, d, h : h+1) \\ At(7, d, h+1) \end{bmatrix}_1 \\ \begin{bmatrix} At(7, d, h+1) \\ Pace(7 : \overline{8}, d, h+1 : h+2) \\ At(8, d, h+2) \end{bmatrix}_2 \\ \dots \\ \begin{bmatrix} At(35, d, h+29) \\ Pace(35 : \overline{36}, d, h+29 : r) \\ At(36 \bullet\!\!\rightarrow t_3, d, r) \end{bmatrix}_{30\dots} \end{array} \right\}),$$

$\mathbf{Proj}(6, save, \{At(1 : t_6 - 1, d, h), \dots, At(t_7+1 : \infty, d, r), Out\_of\_danger(t_2+1 : \infty, n, r),$
$Tied(1 : t_4 - 1, n, r), \neg Tied(t_4+1 : \infty, n, r),\}),$
$\mathbf{WET}(6, save, 37),$
$\mathbf{Feasible}(6, save), \ \dots$

(The start and finish times of the Paces have been bound using Now in this step, since the first action in the plan is a primitive action that can actually be acted upon. The WET estimate is done similarly, it is 2 for the first Pace (PET is 1 for binding the time variables, and EET is 1) and 1 each for all the subsequent. Thus the WET is 31 for the *Pace* actions, and 4 for *Release*, and 2 for *Pull*, totalling to 37.
)

Step 7:
$\mathbf{CS}(7, save, \{\ldots, At(0, d, h)_{obs}, At_c(t_7, d, r), Run(6 : \bar{t}_7, d, h : r), r - h =$
$30_{obs}, Tied(0, n, r)_{obs}, \neg Tied_c(t_4, n, r),$
$Pull(t_1 : \bar{t}_2, d, n, r), Out\_of\_danger_c(t_2, n, r),$
$Pace(6 : 7, d, h + 1 : h + 2), \ldots Pace(35 : 36, d, h + 29 : r),$
$Release_1(t_3 : \overline{t_3 + 1}, d, n, r), t_2 \leq 50, t_1 = t_2 - 1,$
$t_3 = t_4 - 3, t_4 \leq t_1, t_6 = 6, t_6 = t_7 - 30, t_7 \leq t_3)\}),$

$$\mathbf{Ppl}(7, save, \left\{ \begin{array}{c} \begin{bmatrix} At(7, d, h + 1) \\ Pace(7 : \bar{8}, d, h + 1 : h + 2) \\ At(8, d, h + 2) \end{bmatrix}_{1\ldots} \\ \ldots \\ \begin{bmatrix} At(35, d, h + 29) \\ Pace(35 : \overline{36}, d, h + 29 : r) \\ At(36 \bullet\!\!\rightarrow t_3, d, r) \end{bmatrix}_{30\ldots} \end{array} \right\}),$$

$\mathbf{Proj}(7, save, \{At(1 : 5, d, h), \ldots, At(37 : \infty, d, r), Out\_of\_danger(t_2 + 1 : \infty, n, r),$
$Tied(1 : t_4 - 1, n, r), \neg Tied(t_4 + 1 : \infty, n, r), \}),$
$\mathbf{WET}(7, save, 36),$
$\mathbf{Feasible}(7, save), \ \ldots$
(The time variables in the partial plan at step 5 had become bound to constants in step 6. Consequently, the WET of the first *Pace* was decreased by 1, reducing the WET for the plan to 36. Dudley also performs the first *Pace* action.)

Dudley's planning and acting continues. The following gives an idea of the rest of his time-situated reasoning until he has saved Nell:

Step number	First action in Ppl	WET
$Step8:$	$Pace(8:9, d, h+2:h+3)$	$WET(8, save, 35)$
$Step9:$	$Pace(9:10, d, h+3:h+4)$	$WET(9, save, 34)$
$\vdots$	$\vdots$	$\vdots$
$Step34:$	$Pace(34:35, d, h+28:h+29)$	$WET(34, save, 9)$
$Step35:$	$Pace(35:36, d, h+29:r)$	$WET(35, save, 8)$
$Step36:$	$Release_1(t_3:t_3+1, d, n, r)$	$WET(36, save, 7)$
$Step37:$	$Release_1(37:38, d, n, r)$	$WET(37, save, 6)$
$Step38:$	$Release_2(38:39, d, n, r)$	$WET(38, save, 5)$
$Step39:$	$Release_3(39:40, d, n, r)$	$WET(39, save, 4)$
$Step40:$	$Pull(t_1:t_1+1, d, n, r)$	$WET(40, save, 3)$
$Step41:$	$Pull(41:42, d, n, r)$	$WET(41, save, 2)$
$Step42:$	$Null$	$WET(42, save, 1)$

In our research, we have incrementally considered more complex scenarios, so that by abstracting from them we can identify more critical issues and enhance the framework with additional time-situated planning capability. In the next two sections we consider more complex scenarios.

5.2 The knots may be too tight, a knife may be needed

Frequently in planning, a given action has more than one pre-conditions that must be satisfied. Each precondition may be satisfied by performing other actions. In such a case, there is the question in which order those actions should be performed.

Suppose that Dudley thinks that a knife may be required to cut the difficult knots around Nell, and plans for that contingency. He knows of a knife in the house, he projects it to be there when he needs to use it. Requiring a knife corresponds to a compound condition for the action $Cut\_ropes(S:\overline{F},\ldots)$, namely, $At(S:F,d,r) \wedge Have(S:F,d,knife)$. The inference rule whereby Dudley can subsequently formulate two plans, one in which he plans to satisfy $Have(\ldots)$ before $At(\ldots)$ and the other in which this order is reversed, fires. Both conditions, must however hold up to the time they are needed for the Cut_ropes action. This is where $\bullet\!\!\rightarrow$ comes into use. It enables Dudley to notice that when the result of an action is expected to be preserved up to the time when it is to be used, a plan in which it must be undone in order to satisfy the condition for a subsequent action, is in fact inefficient, and can be frozen in favor of another plan.

This inferencing, though domain independent, does not claim to handle every situation involving conjunctive goals. It can be thought of as one heuristic aid used by commonsense reasoners in limited time to help in plan selection. In the second plan, picking up the knife requires Dudley to be at home (the same location as the knife), and this violates his attempt to achieve $At(t_{11}, d, r)$ and preserve it until the time t_4 when he will finish untying Nell. Dudley chooses to proceed with the first plan in favor of the second. We demonstrate below a few key steps in this reasoning.

Step 3:

$\mathbf{CS}(3, save, \{At(0, d, h)_{obs}, At(0, knife, h), Tied(0, n, r)_{obs}\}),$

$$\mathbf{Ppl}(save, 3, \left\{ \begin{bmatrix} At(t_3 : t_4, d, r) \wedge Have(t_3 : t_4, d, knife) \\ Cut\_ropes(t_3 : \bar{t}_4, d, n, r) \\ \neg Tied(t_4 \bullet\!\!\rightarrow t_1, n, r) \end{bmatrix}_{1\ldots} \ldots \right\}),$$

$\mathbf{CS}(3, save, \{\ldots, t_3 = t_4 - 3, \ldots\})$

$\mathbf{Proj}(3, save, \{At(1:\infty, d, h), At(1:\infty, knife, h), Tied(1 : t_4 - 1, n, r),\ \ldots\})\ldots$

$\vdots$

Step 5:

$$\mathbf{Ppl}(5, save1, \left\{ \begin{matrix} \begin{bmatrix} At(t_6, d, L_1) \wedge At(t_6, knife, L_1) \\ Pick\_up(t_6 : \bar{t}_7, d, knife) \\ Have(t_7 \bullet\!\!\rightarrow t_4, d, knife) \end{bmatrix}_1 \\ \begin{bmatrix} At(t_8, d, L_2) \\ Run(t_8 : \bar{t}_9, d, L_2 : r) \\ At(t_9 \bullet\!\!\rightarrow t_4, d, r) \end{bmatrix}_{2\ldots} \end{matrix} \right\}),$$

$\mathbf{CS}(5, save1, \{\ldots, t_6 = t_7 - 1, \ldots\})$

$$\mathbf{Ppl}(5, save2, \left\{ \begin{matrix} \begin{bmatrix} At(t_{10}, d, L_4) \\ Run(t_{10} : \bar{t}_{11}, d, L_4 : r) \\ At(t_{11} \bullet\!\!\rightarrow t_4, d, r) \end{bmatrix}_1 \\ \begin{bmatrix} At(t_{12}, d, L_3) \wedge At(t_6, knife, L_3) \\ Pick\_up(t_{12} : \bar{t}_{13}, d, knife) \\ Have(t_{13} \bullet\!\!\rightarrow t_4, d, knife) \end{bmatrix}_{2\ldots} \end{matrix} \right\}),$$

$\mathbf{CS}(5, save2, \{\ldots, t_4 \geq t_{13}, t_{12} = t_{13} - 1, t_{11} \leq t_{12} \ldots\})$

$\mathbf{Proj}(5, save1, \{At(1:\infty, d, h), At(1:\infty, knife, h), Tied(1 : t_4 - 1, n, r),\ \ldots\})$

$\mathbf{Proj}(5, save2, \{At(1:\infty, d, h), At(1:\infty, knife, h), Tied(1 : t_4 - 1, n, r),\ \ldots\})\ldots$

(Here Dudley has formed two alternative plans, corresponding to two orders: first satisfy *At* (plan save1) and then *Have*, and vice-versa (plan save2).)

Step 6:

$$\mathbf{Ppl}(6, save1, \left\{ \begin{array}{c} \left[\begin{array}{c} At(t_6, d, h) \wedge At(t_6, knife, h) \\ Pick\_up(t_6 : \bar{t}_7, d, knife) \\ Have(t_7 \bullet\!\!\rightarrow t_4, d, knife) \end{array} \right]_1 \\ \left[\begin{array}{c} At(t_8, d, h) \\ Run(t_8 : \bar{t}_9, d, h : r) \\ At(t_9 \bullet\!\!\rightarrow t_4, d, r) \end{array} \right]_{2 \ldots} \end{array} \right\}),$$

$$\mathbf{CS}(6, save1, \{\ldots, t_6 = t_7 - 1, \ldots\})$$

$$\mathbf{Ppl}(6, save2, \left\{ \begin{array}{c} \left[\begin{array}{c} At(t_{10}, d, h) \\ Run(t_{10} : \bar{t}_{11}, d, h : r) \\ At(t_{11} \bullet\!\!\rightarrow t_4, d, r) \end{array} \right]_1 \\ \left[\begin{array}{c} At(t_{12}, d, h) \wedge At(t_6, knife, h) \\ Pick\_up(t_{12} : \bar{t}_{13}, d, knife) \\ Have(t_{13} \bullet\!\!\rightarrow t_4, d, knife) \end{array} \right]_{2 \ldots} \end{array} \right\}),$$

$$\mathbf{CS}(6, save2, \{\ldots, t_{12} = t_{13} - 1, t_4 \geq t_{13}, t_{11} \leq t_{12} \ldots\}) \ldots$$

(Refinement of the two alternative plans continues.)

Step 7:
$\mathbf{Freeze}(7, save2)$, ...

(Since in plan save2, $t_{12} \leq t_4$ and $At(t_{12}, d, h)$ du-contradicts $At(t_{11}...)$ then this plan is judged (rule 12 in Appendix B) to be nonpromising and is frozen.)

5.3 Another alternative: stop the train!

The examples above was kept as simple as possible to elucidate the workings of the active-logic planner. Now suppose we enhance Dudley's set of axioms so that he knows about stopping trains, warning drivers and making telephone calls. Then, as he synthesizes the above obvious plan to 'run to Nell and untie her', he can simultaneously plan for another alternative – he could get the driver to stop the train in time! But how does he establish contact with the driver? One way is to go to the nearest telephone and call the train station. Dudley knows that it is 50 time steps until the deadline. Where is the nearest telephone? His neighbor has one, and the neighbor lives only 5 paces away. How long will it take Dudley until he can get the connection? We assume his previous experience with telephones tells him he must allow 5 steps; he possibly may have to redial several times (perform a *Repeat_until* type of action[32]). We assume it will take him additional 5 steps to warn the train driver. Thus overall he will eventually allow 15 steps, but it takes him time to realize this. Dudley

[32]As a planner, Dudley must reason about various types of actions vis-a-vis his deadline. These include conditional actions, and repeat-until type of actions as discussed in Section 4. A repeat-until type of action is characterized by the appearance of a signaling condition to mark its end. Appropriate rules in Appendix B show Dudley's inference rules to this effect.

can plan for this 'stop the train' alternative in parallel with the 'run to Nell and untie her' plan, but we do not illustrate the parallel "untie" version here (see above).

This plan involves a dimension that we have not alluded to before; it involves the action of another agent. Unlike in the earlier plan, where all actions were under Dudley's control, this plan depends on an action *Stop_train* which has to be performed by an agent other than Dudley, in this case, the train driver. How can Dudley plan for this? Dudley has the following two axioms:

$Stop\_train(T : T+2, driver) \rightarrow Out\_of\_danger(T+2, nell, railtrack)$

$Warn(S : T, dudley, driver) \rightarrow Stop\_train(T : T+2, driver)$

The second axiom hints at an unknown in the plan: Can Dudley trust the driver to stop the train? What if the villain is driving the train himself? Suppose that Dudley does believe, that by warning the driver he can get him to stop the train, then his plan must include the action *Warn*, and his total time estimate must allow for the time taken by the driver in performing the stop. He can not attempt to satisfy the conditions for *Stop_train* since they are not within his control, but he must proceed with his bit of the plan i.e. with warning the driver. The following steps illustrate his formulation of this plan. The abbreviation *dr* denotes the train driver.

Step 0:
$\mathbf{CS}(0, null, \{At(0,d,h)_{obs}, Tied(0,n,r)_{obs}\})$, $\mathbf{Goal}(stop, Out\_of\_danger(50,n,r), 50)$, ...

Step 1:
$\mathbf{CS}(1, null, \{At(0,d,h)_{obs}, Tied(0,n,r)_{obs}\})$, $\mathbf{Goal}(stop, Out\_of\_danger(50,n,r), 50)$,

$$\ldots \mathbf{Ppl}(1, stop, \left\{ \begin{bmatrix} \ldots \\ Stop\_train(\tau_1 : \tau_2, dr) \\ Out\_of\_danger(\tau_2 \bullet\!\!\rightarrow 50, n, r) \end{bmatrix}_1 \right\}),$$

$\{\tau_2 \leq 50, \tau_1 = \tau_2 - 2\}$,

$\mathbf{WET}(0, stop, 0), estimate(Stop\_train(\tau_1 : \tau_2, dr), 2)$,**Feasible**(1,stop), ...

(Dudley now is forming a plan that includes an action to be performed by someone else: Stop-the-train.)

Step 2:

$$\mathbf{Ppl}(2, stop, \left\{ \begin{array}{c} \begin{bmatrix} In\_contact(\tau_3 : \tau_4, d, dr) \\ Warn(\tau_3 : \tau_4, d, dr) \\ Knows\_about(\tau_4 \bullet\!\!\rightarrow \tau_1, n, dr) \end{bmatrix}_1 \\ \begin{bmatrix} \ldots \\ Stop\_train(\tau_1 : \tau_2, dr) \\ Out\_of\_danger(\tau_2 \bullet\!\!\rightarrow 50, n, r) \end{bmatrix}_2 \end{array} \right\}),$$

$\{\tau_2 \leq 50, \tau_1 = \tau_2 - 1, \tau_3 = \tau_4 - 3, \tau_4 \leq \tau_1\}$

$\mathbf{WET}(2, stop, 2), estimate(Warn(\tau_3 : \overline{\tau}_4, d, dr), 5)$,**Feasible**(2,stop), ...

(Since Stop-the-train is not Dudley's action, he does not refine it; but rather he adds an action "Warn" to his plan, which according to one of his axioms will cause the driver to perform the Stop-the-train action, by means of the result "Knows-about" which can be one of the preconditions to Stop-the-train. Dudley now has the estimate of 5 steps to perform the warning, but has not yet included this into the WET, nor has he yet considered the time that will be taken to get to the neighbor's house, nor to establish a phone connection.)

Step 3:

$$\mathbf{Ppl}(stop, 4, \left\{ \begin{array}{c} \left[\begin{array}{c} At(\tau_6 : \tau_7, phone, L_1) \wedge At(\tau_6 : \tau_7, d, L_1) \\ Repeat\_until(\tau_6 : \tau_7, Dial, Get\_connection, d, dr) \\ In\_contact(\tau_7 \bullet\!\!\rightarrow \tau_4, d, dr) \end{array} \right]_1 \\ \left[\begin{array}{c} In\_contact(\tau_3 : \overline{\tau}_4, d, dr) \\ Warn(\tau_3 : \overline{\tau}_4, d, dr) \\ Knows\_about(\tau_4 \bullet\!\!\rightarrow \tau_1, n, dr) \end{array} \right]_{2\ldots} \end{array} \right\}), \ \{\tau_2 \leq 50, \tau_1 = \tau_2 - 1,$$

$\tau_3 = \tau_4 - 3, \tau_4 \leq \tau_1 \ \tau_6 < \tau_7, \tau_7 \leq \tau_3\}$ $\mathbf{WET}(3, stop, 7)$,
$estimate(Repeat\_until(\tau_6 : \tau_7, Dial, Get\_connection, d, dr), 5)$, $estimate(Warn(\tau_3 : \overline{\tau}_4, d, dr), 5)$,
$\mathbf{Feasible}(3, stop)$, ...

⋮

(Dudley adds the Repeat-until action to his plan, in order to make phone contact with train driver. The WET is now 7 (5 + 2). The 5 step estimate for this have not yet gotten into WET.)

---

6 Towards realism: Limited space and computation capacity

We have thus far sketched our original to tackle the fully deadline-coupled reasoning problem, further details of which can be found in [KNP90, NKP91]. The system described thus far has been implemented in prolog. The implementation serves two purposes: it confirms that the inference engine indeed performs the desired sequence of deliberation and execution, and secondly it gives us the opportunity to vary various parameters that are design and environment dependent to observe the behavior of the agent in different settings. The implementation also brought to our attention the glaring need to address other problems that we had set aside in the interest of providing a treatment of time and deadlines within the deadline. We address those shortcomings in this section.

The space problem: As time advances, more knowledge is gathered as a result of observations from the agent's environment and as a result of the deduction processes within. The knowledge base which is continuously expanding could potentially become so formidable that it would be completely unrealistic to assume that the agent could possibly apply all the inferences to this complete knowledge base. Usually, most of this information is not directly relevant either to the development of the agent's current thread of reasoning. Active-logics

and our treatment of deadline-coupled planning in the previous sections have disregarded the space problem in preference to dealing adequately with time-related issues. The space issue deserves serious attention where the original number of beliefs of the agent is large, and where very many new beliefs are added to the agent's knowledge base over time.
Unrealistic parallelism: A step is defined as the time required by the agent to perform one inference or one primitive physical action in the world. Actions can be carried out in parallel if the sensors and effectors permit. For example, an agent can walk and eat simultaneously. Active-logics planners treat 'think' actions within the agent in the same spirit as physical actions and recognize that they sap precious time resources. The original step-logic inference system assumed that during a given step i the agent can apply all available inference rules in parallel, to the beliefs at step $i-1$. There are two problems with this. One is the unrealistic amount of parallelism that potentially allows the agent to draw so many inferences in one time step that the meaning of what constitutes a step begins to blur. Secondly, it is unreasonable to expect that all inference rules would have the same time granularity. For example, it is unlikely that a simple application of Modus Ponens will take just as long to fire as an inference rule to refine a plan or check for plan feasibility, especially as plans become very large. While the representation is uniformly declarative, some rules have more procedural flavor than others, and can be imagined to take more time steps. Just as there is a limit on the physical capabilities of the agent as to how many physical actions can be done in parallel in the same time step, there must be a limit to the parallel capacity of the inference engine as well.

A claim towards fully deadline-coupled reasoning would be a tall one if the model depicts an agent with an infinite attention span and infinite think capacity. In this section we propose an "active logic" extension of the original step-logic formalism to take into consideration space and computation constraints. We revisit the fully deadline-coupled planning problem in the light of this new framework. For each component that is needed to handle the time and space limitation, we present one possible heuristic. The effectiveness of these heuristics can be questioned. As we have mentioned earlier, our concern here is not with optimality, but rather with a time situated framework in which computational limitations can be reasoned about. In future work we will consider alternative heuristics aimed at improve performance.

6.1 A limited span of attention

We propose a solution to the space problem partially based on [EDMP87] as follows. The agent's current focus of attention is limited to a small fixed number of beliefs forming the STM (short term memory), while the complete belief set is archived away in a bigger associative store, namely, the LTM (long term memory). In addition, we use a QTM which is a technical device to hold the conclusions that result in each step since further inferencing with these must be stalled until the next time step. The size of the STM is a fixed number K[33].

In the most simplistic model, the STM could be represented as a queue, in which case the inference/retrieval algorithm reduces to a simple depth first or breadth first strategy

[33]What is a realistic K for a commonsense reasoner? There is psychological basis that suggests that human short-term memory holds seven-plus-or-minus-two 'chunks' of data at one time [Mil56].

depending upon whether new observations and deductions are added to the head or tail of the queue respectively. It seems that choosing the STM elements without focus consideration may lead the reasoning astray quite easily, and also lead to often incomplete threads of reasoning due to thrashing. We propose to maintain a predicate called **Focus**(...) which keeps track of the current line of reasoning. This is dynamically changed by the agent's inference mechanism and is responsible for steering the reasoning back to a particular thread even when a large number of seemingly irrelevant inferences are drawn. Among the agent's inference rules is a set of *focus changing (FC)* rules, which when fired alter the focus. Those K beliefs from the associative LTM which are most[34] relevant to the current focus are highlighted to form the STM.

In short, the framework can be described as follows. The $QTM_{i/i+1}$ is an intermediate store of formulae that are theorems derived through the application of inference rules to the formulae in STM_i (the STM at step i). They are candidates for the STM at step $i+1$, although only K among them will be selected. Thus the results of the inference rules, can be imagined to fall into $QTM_{i/i+1}$ and are available for selection to form the STM at the next step[35]. The $focus$ and Now which are crucial to time-situated reasoning are always accessible to the agent.

FRAMEWORK:

$$\frac{i : STM_i\{...\}, Now(i), Focus(i, ...), LTM_i\{...\}}{i+1 : STM_{i+1}\{...\}, Now(i+1), Focus(i+1, ...), LTM_{i+1}\{...\}}$$

$QTM_{i/i+1}$ holds β if β is an i-theorem. It includes relevant formulae which are retrieved from the LTM using the retrieval rule. Step i concludes by selecting K formulae from $QTM_{i/i+1}$ which are relevant to $Focus_i$ to form STM_{i+1}. LTM_{i+1} is LTM_i appended with $QTM_{i/i+1}$.

The main problem in limiting the space of reasoning is to decide what should be in the focus. In our planning framework, we have developed a mechanism that is at work to limit the focus to a single feasible plan at a given time step. A list of actions, conditions and results from the plan that need further processing (we call it the active list), form a list of keywords in the focus. We describe some details of this mechanism in section 6.4. Heuristic rules are proposed to maximize the probability of finding a solution within the deadline. This would correspond to a sort of best first strategy or a beam search of width K in the general framework. Although these heuristic rules are independent of the instance of the problem in question, they are likely to be different depending upon the category of the problem being solved. A deadline-coupled actor-planner is likely to maintain a much narrower focus than a long-range 'armchair' planner. In section 6.4, we outline some of the specific heuristic strategies employed for the tightly time-constrained planner.

[34]There is then a ranking among the relevant formulae and the K at the top of the list are picked. In our implementation, we select the K formulae at random from the candidate formulae.

[35]This has the feature that all thinking does not pass through the STM unless it is relevant to the focus.

6.2 A limited think capacity

Next, we address the bounded computation resource problem. An intelligent agent can be expected to have a sizable reservoir of inference rules acquired during its lifetime. Firing of an inference rule corresponds to a 'think' action. Without a bound on its inferencing power, the agent could fire all the inference rules applicable (termed in conventional production systems as the conflict set) simultaneously during a time step. We limit the inference capacity of the engine to I. Each inference rule j is assigned a drain factor d_j. This is a measure of the drain incurred by the inference engine while firing an instance of this rule. For instance, Modus Ponens and the more elaborate inference rule for plan refinement, would be given different drain factors to reflect this difference in granularity [36].

Our limited-capacity inference engine fires only a subset of the applicable rules in each time step. Among the various alternatives, it is possible to pick the inference rules either completely nondeterministically up to the engine capacity I, or one could again apply some heuristics to improve the agent's chances. Several parameters, such as agent attitudes, the uncertainty of the environment, or the urgency to act could dictate this choice.

Thus, in effect, during each step, K beliefs are highlighted from the knowledge base (LTM) to constitute the STM. From among the rules applicable to these K beliefs, a subset of rules is chosen such that sum of the drain factors does not exceed the engine's inference capacity I. The results of the inferencing are put in the QTM. Finally, the contents of the QTM are copied to the LTM.

6.3 On the adequacy of the limited memory model

Let $SL(OBS, INF)$ denote a step-logic with an inference function INF, an observation function OBS and unlimited memory described in the original framework. Let $SL_K^{FET}(OBS, INF)$ denote the corresponding active logic with a limited short-term memory of size K and an algorithm FET describing the strategy for fetching elements into the STM.

The following theorem demonstrates that under appropriate conditions, any inference derivable in an active-logic with no memory limitation, can also be derived in a memory limited active logic. The size of the STM (i.e., K) can be as small as two beliefs. One might expect that for larger K's a given inference will occur more quickly. However, this is true up to a certain point only. We discuss this further at the end of the proof.

Theorem 1 Let $K \geq 2$. If all the inference rules in INF are monotonic then it is possible to describe a simple algorithm FET such that any theorem of $SL(OBS, INF)$ will eventually appear as a theorem of $SL_K^{FET}(OBS, INF)$. I.e. if $\vdash_i \alpha$ in SL (α was proven at step i) then $\exists j$ such that $\vdash_j \alpha$ in $SL_K^{FET}(OBS, INF)$.

Note: the requirement of monotonicity in particular entails that the "clock"-rule for *Now*

[36] How to calibrate the inference rules for the assignment of these drain factors is a separate and interesting issue, but we will not address it presently. Also, how thinking actions compare with physical actions is a technical issue that could be resolved by trying to calibrate the system to check on the relative speed of its inference cycle with that of its sensors and motors. We skip this implementation sensitive issue for the present.

is left out. Thus the result applies only to Now-free inferences. We also assume that new observations are consistent with previous facts and derivations.

Proof We begin by showing that the following dovetailing transformation on INF into $Dove[INF]$ yields an equivalent step-logic with unbounded memory in terms of the final theorem set. We then show that $SL(Dove[INF], OBS)$ has the property that $SL_K^{FET}(Dove[INF], OBS)$ has the same final theorem set as $SL(Dove[INF], OBS)$ for the algorithm FET described below.

Let all the rules in $Dove[INF]$ have at most two antecedent formulae. This is achieved by transforming every rule in INF which is of the form:

$$\frac{A_1 \wedge A_2 \wedge \ldots \wedge A_n}{W}$$

into n rules of the form:

$$\frac{A_1 \wedge A_2}{P_3}$$

$$\frac{A_3 \wedge P_3}{P_4}$$

$$\vdots$$

$$\frac{A_n \wedge P_n}{W}$$

This can make the number of rules very large, but it allows us to have a very simple algorithm FET to show that there is no net loss of theorems on account of limiting the size of the STM. Let $InfCh_\alpha$ denote the inference chain used to derive a theorem α. The algorithm FET proposed uses dovetailing to ensure that STM cycles through all possible combinations of beliefs, so that eventually whatever formulas are used in $InfCh_\alpha$ also occur in STM.

We want, first, to ensure that the algorithm has access to all logical axioms, so we feed them in lexicographically. At each step, some more (finitely many) logical axioms are added to the LTM; we feed in all formulas of length $\leq i$ at step i, that use only the first i symbols of the language. FET forces STM to cycle through all combinations of beliefs in LTM, all combinations of two at a time, to allow every rule that could fire to actually fire.

As time goes on, more and more logical axioms are fed into LTM, and also new inference results are being produced and going into LTM. FET is an algorithm that gets every combination of two formulas, including new ones that come in by either inference or feeding. We can conceptualize all formulas (in the entire language) to be already in LTM but only those that occur in $InfCh_\alpha$ to be marked in red. As time goes on, more and more become red, due to inference and feeding. We also mark each formulas with an index (a unique natural number), and bring into STM two at a time, but only red ones, and never repeat a pair already brought in; we can imagine each pair of formulas has a link that become blue when it is brought into STM; so we never bring a blue-linked pair in again. At each step we bring in a non-blue pair and apply all applicable rules to it. One could either bring in the pair with the smallest index-sum, the pair with the largest index sum or pick a pair at random, among many alternatives.

This simple algorithm FET will perform all possible inferences including those in $InfCh_\alpha$ eventually deriving α, although after many more time steps. ▮

We can see now that even for very small K (such as 2) there are in the worst case many combinations of beliefs (e.g., in pairs) to be brought in turn into STM; this is slow. As K increases this only gets *worse* up to a point (when K is roughly half the the size of LTM—the number of such combinations is simply the relevant coefficient of the binomial expansion). As K approaches the size of LTM, the number of combinations reduces and even is better then for very small K; and on average it will also be better in terms of the likelihood of finding an appropriate (useful) combination sooner. These are time-advantages of large K; however, large K has the space-limitation that we addressed above. To keep the worst case of inference-time small and also address the space problem, one can choose K to be the maximal number of antecedents in any inference rule.

The inference rules used for deadline-coupled planning are nonmonotonic. The rules to calculate WET, to refine a plan, to revise the context of the plan and to project the context set of the plan are the main nonmonotonic inferences drawn in this system. In checking if a condition c already appears in the context or the projection of a plan before the plan is refined, the rule for plan refinement is nonmonotonic since c may be absent when checked but present later. The nonmonotonicity of the projections stems from applying the default of temporal persistence. The context set revision rule is nonmonotonic since it withdraws from the context set formulae whose basis is no longer true in the current projection. In $SL(OBS, INF)$ where there is no limit on the memory, partial plans get refined whenever possible, and the context set is revised at every time step, also, the projection in the latest context is recomputed at every time step. With $SL_K^{FET}(OBS, INF)$ however, the order in which these rules will fire depends upon the simultaneous occurrence of the matching formulae in the STM. If the refinement of the partial plan proceeds before the context set revision, it is possible that redundant plans will be developed before the context set and the projection will catch up to let the planner know that something is already true and does not need to be planned for. As an example, consider a plan in which a condition for a certain action requires that a certain high-rise building be pink. Dudley may have an axiom which says that all high-rises are pink, but has not had a chance to apply it to the context set in question to conclude that the high-rise building in question is pink. Hence he formulates a (redundant) plan to paint the high-rise pink. Subsequently, as the context set is revised this condition is already true in the projection and an inference rule needs to be fired to identify and eliminate this portion of the plan.

We note here that redundant plans of this nature may be generated even in the case of unlimited memory, if there is a long inference chain based on the facts required to derive the condition in question. For example, if Dudley does not directly know that all highrises are pink, but infers it from the fact that all highrises are tall structures, that all tall structures are made of concrete, that anything made of concrete is pink. Since bringing in all these axioms and revising the context set may take quite a few steps, it is likely that redundant plans are generated even in the case of unlimited memory. A rule that corrects this situation is useful in both cases.

Lemma 2 If the inference rules for plan refinement, for context set revision and for computing projection compute all possible instances of **PPl**, **CS** and **Proj** instead of working on the *latest* instances alone, then all the partial plans generated by the unlimited step-logic will also be generated by the bounded active logic where $K \geq 2$.

Proof When each instance of the partial plan, context set and projection is kept active, the same combinations that occur in the unlimited step-logic will eventually cycle through the STM, giving the same partial plans, in addition to several other plans generated.

In some cases, where context-set revision precedes planning, in fact efficient plans may be generated since the planner is in fact more informed about extended effects and side effects prior to the planning[37].

6.4 Heuristic strategies for deadline-coupled planning

In the previous section we presented some formal results on the adequacy of an active-logic to generate the required plans even when there is a bound on the size of the STM. The algorithm *FET* was a simple breadth-first strategy used to demonstrate this. In this section we present heuristic algorithms to be used in place of FET to improve the chances that the formulae used in the derivation of the plan will appear sooner in the STM.

6.4.1 Focus and keywords

As a general approach to limiting space, we proposed that beliefs be organized in the LTM by association with some topics or keywords. When one or more of these topics are in the focus, the related beliefs become candidates for retrieval into the STM, as a result of a retrieval rule. Formulae in the STM are not automatically inherited from one step to the next. Only when they are still relevant to the current focus do they become candidates and must compete with other relevant formulae to fit into the limited size STM.

The focus holds the keywords of current interest. It has similarities with the RTM proposed in [EDMP87]. We imagine that in a more general framework the focus would contain keywords arranged in a partial order according to priorities.[38] Beliefs related to high

[37]In the FET algorithm, bringing the lowest sum of indices corresponds to a breadth-first strategy. Using highest sum of indices would correspond to a depth-first strategy. The former will ensure that partial plan refinement will not get too far ahead of the context set revision. Once the partial plan is refined, a context set revision rule must fire since its antecedents were already present in the LTM. In the depth-first method, you will refine a plan as far as possible, then revise CS as much as possible, then project as much as possible, and then alternate. It is interesting to explore how these will interact. In a random strategy that exhaustively brings in all pairs, arbitrary speeds of the three chains need to be considered to see its effects on the planning.

[38]The main question is how to choose the "keywords" that are in the focus at a given time, and how to assign priorities to them. Our ideas presented here are aimed at a commonsense agent engaged in deadline-coupled planning.

priority topics are given preference for being brought into the STM. As mentioned before, for our actor-planner Dudley we restrict the focus to equal priority keywords related to a single plan at a given time step. Non-primitive actions that appear in the triplets of a given plan, that still need to be refined are appropriate keywords for goal-directed retrieval. Also, the results that appear in these triplets serve as keywords to deduce the effects of the plan. These are kept in the focus as the formula $plan\_in\_focus(p, PKWL)$ where p is the name of the partial plan and $PKWL$ is the list of keywords for p.

Observations are put into the current focus at least for a few time steps, since it is possible that they may be important, and may trigger some new threads of reasoning[39]. Current observations are kept in the focus as the formula $obs\_in\_focus(OBL)$ where OBL is the list of observations that serve as keywords. Thus, we treat the focus as a predicate

$$\mathbf{Focus}(i, plan\_in\_focus(p, PKWL), obs\_in\_focus(OBL))$$

When there are multiple options in the STM for achieving a goal, more than one partial plan is spawned. All plans for achieving a certain goal may be given equal priority at first, thus continuing to develop them in a time shared manner and bringing them into focus sequentially. However, in a deadline situation, it may be advisable to commit to a plan (to put it in focus and the others in a background queue for backtracking) and continue with it unless it seems infeasible.

The development of an appropriate plan depends on three aspects: satisfying (pre)conditions, refinement of general actions into more detailed ones, and using the result of the plan for finding its effects. To illustrate, given a triplet in the $\mathbf{Ppl}[C_A, A, R_A]$, the following axioms are used: axioms that produce C_A (i.e. C_A appears in their conclusions), axioms that are used for refinement of A (again A appears in their conclusions) and axioms in which R_A appears in their antecedents (i.e. $R_A \rightarrow e$), to compute the extended effects. Therefore, if we make sure these types of axioms are brought into the STM, we will increase the probability that a plan will be found. Note, that other axioms should be used for finding indirect effects (i.e., $e \rightarrow e1, e1 \rightarrow e2...$) which can't be obtained by the agent's action. Our heuristic brings such axioms to the STM too.

6.4.2 Some inference rules for resource limited reasoning

At each step, the agent reflects on its long term memory reservoir to pick out formulae that are relevant to its current focus of reasoning using a retrieval rule. The LTM is an associative store and hence this retrieval is fast.[40]
Focus directed retrieval rule (FDRR):
$i : ..., LTM\{..., \beta, ...\},$

[39]How to in fact select some crucial observations from all the stray input to the sensors remain unaddressed, but it is not among the problems we will solve at present. A tutor or a human hint to the automated agent that some observations are worthy of more consideration. In our example, Dudley may first start to think about running to Nell to rescue her, when he suddenly sees a telephone. This brings 'calling', and subsequently the related axiom of calling the driver to stop the train into focus. This spawns the generation of a second plan.

[40]The retrieval rule is a weak parallel of the inheritance rule in Elgot-Drapkin's step logics, in the sense that formulae in the STM at the previous step reappear in the STM at the current step provided they are *still* relevant.

$$\frac{\textbf{Focus}(i, plan\_in\_focus(p, PKWL), obs\_in\_focus(OBL)), ..}{QTM_{i/i+1}\{..., \beta, ...\}}$$

if β is relevant to either p or a keyword in $PKWL$ or OBL.

In our work on planning, the **Focus** includes keywords related to a *feasible* plan. A (partial) plan is *feasible* if the sum of *Now* and the plan's working estimate of time is still within the deadline. A list of feasible partial plans is maintained. From among these a subset of plans is selected to work on and is called the interleaving list (IL). Dudley works on each plan in the interleaving list for a *period* number of steps, then goes on to the next plan in the IL in round robin fashion. The interleaving rule (ILR) serves this purpose by periodically selecting the next plan in the IL to put into the focus. This is one of the focus changing (FC) rules in Dudley's inference engine[41]. This rule time-shares between plans and always fires. A separate rule controls the contents of IL.

Interleaving Rule (ILR):

$$\frac{i : Now(i), IL([p_{j_1}, ..., p_{j_n}]), ...}{i+1 : \textbf{Focus}(i+1, plan\_in\_focus(p_{j_1}, ...), ...), IL([p_{j_2}, ..., p_{j_n}, p_{j_1}])}$$

$$\text{if i mod period} = 0$$

When there are two or more plans in the IL, and when it is time to choose between them, a rule fires to narrow the focus to only one plan. We stipulate that the difficult problem of 'when to decide to choose' depends on mental states and attitudes of agents [Sho91]. A more 'cautious' type of agent will skeptically continue to process two alternatives, perhaps risking overshooting the deadline, but a more 'dashing' type of agent will take the risk to pursue just one plan. We have developed a heuristic rule under the following commonsense observation: An agent can continue to work on several plans provided there is *ample* time ahead to try and pursue them one after another in the interest of fault tolerance. For example, even after calling the driver to stop the train, Dudley may want to run to the railroad track and attempt the rescue Nell nevertheless, if there is enough residual time. An agent may do so as a guard against possible failure of his own or other agents' plans, or perhaps as an extra precaution when the plans are not recognized to be mutually exclusive. We look then at the sum of the WET's of all the plans in the IL as a measure of the overhead planning time. When the sum of the WET's and *Now* exceeds the deadline, he drops a plan from the IL. We currently have the simple heuristic of dropping the plan with the highest WET, but recognize that this may very well be the most refined plan as well[42]. Additional bookkeeping is necessary to ensure that two rules do not alter the IL or the focus simultaneously. We skip these implementation details in this description.

Reduce IL rule (RILR):

[41] Other scheduling procedures that were developed by operating systems researchers such as swapping, time-sharing etc. might be useful here, but this is beyond the scope of our paper. We only demonstrate how such procedures can be used *in* time.

[42] If one can find a way to include good estimation of the planning time (and probably decision time) into the WET it seems that more refined plans will have less planning time than other plans. Maybe, the three parts of the WET should not be combined and the decision whether to knock out a plan from the IL should be made using some sort of multi attribute decision rule (i.e., based on executing time, planning time and decision time).

$$\frac{i : Now(i), IL(L), wet\_ordering([p_{j_k}, ...]), ...}{i+1 : IL(L - p_{j_k})}$$

$if \sum_{l \in L} WET_{p_{j_l}} + Now > Deadline.$

An agent may be forced into a decision if two or more plans are ripe for action and the actions are mutually exclusive. The agent must evaluate the relative merits of the plans before making the decision, if acting on one will commit the agent to one plan. Although we do allow planning and acting to be interleaved, we allow the agent to act on a plan if it is the only one in IL. This is to avoid the complex interactions between plans as the result of the changed state of the world following the execution of one plan. We continue to examine this issue in ongoing work.

6.4.3 Capacity of the inference engine

As mentioned earlier, we suggested a limited capacity inference engine that would fire a cumulative set of inference rules to not exceed its inference capacity in each time step. In the simplistic examples that we present, there is a very limited number of rules firing at each step. Furthermore, if the plan length is within a reasonable bound, drain factors of the rules are also quite small and as a first approximation we postulate them to each take roughly the same time and fire in parallel in a single step whenever applicable. It should be noted that the meta rules for resource limited reasoning which were described above fire alongside the other object level inferencing at each step as part of a uniform framework. If we limit the capacity of the engine, the meta rules that are fired will limit the number of planning rules that are fired in each step.

6.4.4 Some illustrations from two plans

Dudley begins to formulate a plan *save* to get Nell *Out_of_danger*. Initially, the focus consists of $\mathbf{Focus}(j, plan\_in\_focus(save, [Out\_of\_danger(...)]), ...)$, and the interleaving list is $IL([save])$. Here, *save* is the name of the partial plan and is used to retrieve formulae related to the plan such as its WET, its context set, projection etc. The list of keywords for this plan contains Out_of_danger. It is used to retrieve axioms from the LTM whose right hand side matches the keyword. Thus, the plan *save* bifurcates into $save_1$ and $save_2$ based on the following axioms which are retrieved from the LTM:
$Pull(T : \overline{T+1}, Y, X, L) \rightarrow Out\_of\_danger(T+1, X, L)$
$Stop\_train(T : \overline{T+2}, driver) \rightarrow \; hspace * 45pt Out\_of\_danger(T+2, nell, r)$

Plan 1: Pull her away from the tracks

$$\mathbf{Ppl}(11, save1, \left\{ \begin{bmatrix} \neg Tied(t_1, n, r) \\ Pull(t_1 : \bar{t}_2, d, n, r) \\ Out\_of\_danger(t_2 \bullet\!\!\rightarrow Deadline, n, r) \end{bmatrix} \right\}), \{t_2 \leq Deadline, t_1 = t_2 - 1\}$$

Plan 2: Stop the train

$$\mathbf{Ppl}(11, save_2, \left\{ \begin{bmatrix} Knows\_about(\tau_1, n, dr) \\ Stop\_train(\tau_1 : \tau_2, dr) \\ Out\_of\_danger(\tau_2 \bullet\!\!\rightarrow Deadline, n, r) \end{bmatrix} \right\}), \{\tau_2 \leq Deadline, \tau_1 = \tau_2 - 2\}$$

$$\mathbf{Ppl}(10, save_1, \left\{ \begin{array}{c} \begin{bmatrix} At(t_6, d, home) \\ Run(t_6 : \bar{t}_7, d, home : r) \\ At(t_7 \bullet\!\!\rightarrow t_4, d, r) \end{bmatrix} \\ \begin{bmatrix} At(t_3 : t_4, d, r) \\ Release(t_3 : \bar{t}_4, d, n, r) \\ \neg Tied(t_4 \bullet\!\!\rightarrow t_1, n, r) \end{bmatrix} \\ \begin{bmatrix} \neg Tied(t_1, n, r) \\ Pull(t_1 : \bar{t}_2, d, n, r) \\ Out\_of\_danger(t_2 \bullet\!\!\rightarrow Deadline, n, r) \end{bmatrix} \end{array} \right\})$$

$$\mathbf{Ppl}(10, save_2, \left\{ \begin{array}{c} \begin{bmatrix} At(\tau_9, d, nh) \\ Run(\tau_9 : \overline{\tau}_8, d, nh : r) \\ At(\tau_8 \bullet\!\!\rightarrow \tau_7, d, nh) \end{bmatrix} \\ \begin{bmatrix} At(\tau_6 : \tau_7, d, nh) \\ Dial(\tau_6 : \overline{\tau}_7, d, dr) \\ In\_contact(\tau_7 \bullet\!\!\rightarrow \tau_4, d, dr) \end{bmatrix} \\ \begin{bmatrix} In\_contact(\tau_3 : \overline{\tau}_4, d, dr) \\ Warn(\tau_3 : \overline{\tau}_4, d, dr, n) \\ Knows\_about(\tau_4 \bullet\!\!\rightarrow \tau_1, n, dr) \end{bmatrix} \end{array} \right\})$$

Figure 3: Pursuing two alternatives within space limitations. Dudley develops two alternative plans in a time-shared fashion until there comes a time when sum of their WET's is no longer within the deadline. The figure shows a snapshot of the two plans at such a time step. Dudley exercises a choice through the rule RILR which reduces the interleaving list to the plan to call the driver of the train. Abbreviations used are: n=nell, d=dudley, r=railroad track, nh=neighbor's house, and dr=driver of the train.

The interleaving list is expanded to contain both $save_1$ and $save_2$ and Dudley continues to work on both feasible plans in a time-shared fashion. The focus thus contains $save_1$ for an interleaving period during which axioms for untying Nell and running to her are progressively retrieved from the LTM. Other facts of no relevance to the plan such as $color\_of\_eyes(\ldots)$ or that are relevant to the other plan such as the axioms about dialing to get a connection are left alone in the LTM. After the period expires, $save_2$ is brought into focus and worked on in a similar fashion. It is not until much later that Dudley realizes that the sum of the WET's of both plans and *Now* is going to overshoot the deadline, and he must restrict the IL using the RILR rule. We show a snapshot of the two plans when this happens in Figure 1.

Using this heuristic, Dudley gives up the plan with the higher WET, which in this case happens to be the one to run to Nell, and executes the plan to go to the neighbor's house to call the driver to stop the train instead. The run to the railroad tracks is longer than the run to the neighbor's house. The sum of the WET's exceeding the Deadline, Dudley starts to run in the direction of the neighbor's house and removes $save_1$ from the IL, still retaining it in the list of feasible plans to be available in case of unanticipated run-time failure.

7 Related work

Age Here we briefly summarize related work and contrast it with our approach. We treat in turn (i) temporal projection, (ii) plan interaction, and (iii) meta-planning.

7.1 Temporal projection

The issue of temporal projection has been extensively studied in the AI literature. In particular, much of effort was devoted to the problem of forward temporal projections, or predictions that are necessary for planning. This is the problem of determining all the facts that will true during a future time period, given a partial description of the facts that are known. Numerous solutions have been proposed to the temporal projection problem [Gel88, Geo87, Hau87, Kau86, Lif87b, Lif87a, Mor88, Pea88, Sho88, Bak89] to mention a few.

Our projection mechanism has commonalities with some of the chronological minimization approaches, notably those of Shoham [Sho88], Lifschitz [Lif87a], and Kautz [Kau86]. In our approach as well as theirs, defaults are applied forward in time, so that earlier events play out their consequences for later ones. However, these approaches specialize in forward temporal projection problems and can handle backward projections, but cannot solve explanation problems or be used by an active agent who may obtain new information while doing projections. Our projection mechanism provides an active agent the capability to revise its conclusions, in light of new observations, to give explanations to previous events, and to use its predictions in planning.

Ginsberg and Smith [GS87] present an approach of reasoning about action and change using possible worlds. The approach involves keeping a single model of the world that is updated when actions are performed. The update procedure involves constructing the nearest possible world to the current one in which the consequences of the actions under consideration hold. There is no explicit notion of time in this approach and the reasoning done by an "outside" reasoner, hence the time of reasoning is not a concern.

Dean and McDermott [DM87] present techniques for temporal database management. They allow two types of prediction in their system: *projection* and *refinement*. Their system is based on a temporal map that can be described by a graph in which the nodes are instants of time associated with beginning and ending events, and the arcs connecting these nodes describe relations between pairs of instants. We use ordered lists as a simpler data structure and in our framework time is associated with events and predicates, and not the other way around as in [DM87];[43] however, the projection is done in a similar way.

As in previous systems we discussed, Dean and McDermott describe their mechanism as "reasoning about time from the outside. It's as though all of what you know about the past, present and future is laid out in front of you." We consider reasoning done by an agent in

[43]Our projection mechanism is similar to that of [DM87], but we distinguish the case where a change occurs but it is not clear when it occurs.

In particular, if α_j is of the form $X(S_j : F_j, Args)$ and α_{j+1} is of the form $\neg X(S_{j+1} : F_{j+1}, Args)$ then $\mathbf{Proj_{i+1,p}}$ does not speculate over the truth or falsity of X over $F_j + 1 : S_{j+1} - 1$. The projection rule will smooth over this interval when further information about a possible point of time where the value of X changes becomes available.

time. It has only the past and the present in front of it, and it takes the passage of time into its reasoning process.

Amsterdam [Ams91] has the only work other than ours that attempts to discuss the issue of who the reasoner is in a given scenario, which we have addressed at length. Amsterdam highlights the advantage which a reasoner has when he/she is at the site of the action, namely, that he/she can observe an action whenever it happens. This allows the agent to utilize the closure property – "if an action is not mentioned then it did not happen". However, here again, there is a meta-reasoner doing the inference. That theory's greatest limitation (and this is said in [Ams91]) is the rigidity of the above mentioned closure principle. There are scenarios where actions can be derived from propositions and hence do not have to be explicitly specified.

7.2 Plan interactions and dependencies

The Sussman anomaly[Sus73] showed that certain planning situations are intrinsically non-linear. Waldinger [Wal75] first suggested the technique of "goal regression" to tackle the problem of conjunctive goals. With INTERPLAN [Tat75], Tate suggested recording a link between the effect of one action and the condition of another. (We have used a similar idea by using the "$\bullet\!\!\rightarrow$" symbol to record the need to preserve a certain effect of an action until a later time). NOAH's procedural nets, and SOUP (Semantics of User's Problem) [Sac75] used critics which are outside advisors that perform decision making regarding non-linearity and plan optimization. It was the first partial-order planner. We have a total-order planner, simply because it turned out to be the simplest kind to build while we concentrated on the time-related aspects. We commit to a sequence of actions, but the actual times at which the action must be executed is bound to the *Now* only at the time of acting. NONLIN [Tat77] could detect interactions and take the necessary corrective action. DEVISER [Ver83] went a step further and handled time limits while performing partial order planning. Planning with conditional operators and iterators has been dealt with in NOAH [Sac75], SIPE [Wil83b] among others.

There have been numerous efforts to improve planning by recognizing goal interactions and dependencies during the planning process, and better representations of actions and plans; we refer the reader to [HTD90] for a general survey of related work on planning we refer to [AHT90] for a collection of papers. These researchers recognize the need to use features of the plan to reason about improving the plan. But this is not done by the planner itself. In our work, although we do not make any attempts to optimize plans, we perform domain-independent meta-level reasoning within the same framework as the object-level planning; and unlike the "critics" [Sac75], the meta-reasoning is an intrinsic part of the planning that also consumes time.

7.3 Meta-planning

One way to implement meta-level decision making is to design two distinct component systems, one for object-level and one for meta-level reasoning. The other way is to design a uniform meta-level architecture where the meta-level problems are formulated using the same language and structures as the base-level problems, using similar methods. This introduces flexible systems, but along with it also introduces the possibility of infinite regress. This is the metareasoning challenge, well-described in [RW91]. An aim of the model by Russell and Wefald is to establish a methodology for applying rational metareasoning to control any object level decision procedure. We try to provide a more axiomatization foundation. Also, Russell and Wefald assume the outcome of each external action is known at the time the agent chooses among them. We by contrast make no such assumption; we argue that in some cases at least, the agent cannot know this and must instead take note of how long an action is taking as it is performed. When there is available information for reliable (or assumed reliable) estimates in advance, our approach can also make use of these.

Reactive systems eschew meta-level planning, and even planning, by considering all contingencies at design time. Typical of this group is the work of Brooks [Bro91, Bro86]. Other efforts that obviate the need for explicit reasoning at execution time are [AC87] and [RK86] and [Kae87]. This is an interesting approach. In the problems that we have dealt with in this paper, which fall in the "unforeseen" category, the fully reactive approach is often believed to lead to brittle and inflexible systems if no real-time deliberation is performed [DF89, Doy88, PR90, IG90].[44]

At middle of the deliberation spectrum, many researchers agree that some form of deliberation is necessary in planning. We mention a few of these here. SIPE [Wil83b] separated execution and generation by allowing the user to guide the planning process (perform the meta reasoning) during execution. The PRS system [GL88] uses metareasoning to recognize the need for additional planning. More recently [IG90] proposed a situated architecture for real-time reasoning. Based on PRS, it provides management representation of metareasoning strategies in the form of metalevel plans, and describes an interpreter that selects and executes them. Their architecture is not *fully* embedded in real-time though, since the time of this interpreter is not accounted for.

Our *fully* deadline-coupled planner has an important qualification that these efforts fail to meet: in addition to performing metareasoning for determining the current time, estimating the expected execution time of partially completed plans and being able to discard alternatives that are deadline-infeasible, it also has a built-in way of accounting for all the time spent as a deadline approaches. This means not only accounting for the time of various segments (procedures in the more usual approaches), but also the time for this very accounting for time! Active logics do this without a vicious circle of "meta-meta-meta..." hierarchies.

An excellent survey of research in *deliberative real-time* AI is available in [GL94]. They categorize real-time systems into *purely reactive* (those that hardwire reactions completely), *combined response* systems (those that have distinct asynchronous components that handle deliberation and reaction) and *integrated* systems (those that have a single architecture that is capable of a wide range of timely responses depending upon the time criticality requirements).

Those in the last category put the time that is available to the best use. These approaches

[44]Partial reactivity to the environment is achieved in our formalism by taking timely note of changes in the environment through observations. However, we simply incorporate the observations into the ongoing deliberative process of reasoning; they do not trigger any special reactive components.

have been collectively characterized by terms such as *flexible computation* [HR91], *deliberation scheduling* [BD94], and *anytime algorithms* [DB88, ZR92]. They spend the resources available to the agent in deciding whether to act, how to act, and when to act. The main differences between our approach and these is the following: (1) they do not account for the time-cost of the *deliberation scheduling algorithms* themselves, only for the cost of deliberation that they consider; while our mechanism is completely situated in time; (2) they require prior complex (meta) knowledge about their reasoning algorithms or procedures themselves, and their characteristics with respect to time; they also require a great deal of knowledge about the domain in the form of probabilities of events and expected utilities of actions that the agent must be aware of; (3) they usually attempt to solve an *optimization* problem in a specific domain, whereas our approach is to come up with a formalism that accounts for all the time spent between *Now* and the deadline while attempting to reason about the *feasibility* of a solution, not to find an optimal solution. Thus, we note that these approaches are not *alternatives* to our time-situated reasoning approach using active logics, but rather that they are suited for a different range of more informed problem solving.

"Anytime algorithms" is a now widely used term, first coined by Dean and Boddy [DB88]. It represents a class of deliberation algorithms which have the following characteristics: (1) They can be interrupted at any time and will produce *some* solution to the problem; (2) given more time they will produce better solutions; and (3) the user of the algorithm has some explicit characterization of the tradeoff between the algorithm's performance and the amount of time that is available to compute a solution.

Anytime algorithms are similar in spirit to the notion of "imprecise computation" commonly used in research in operating systems which divides the task into a *mandatory* part that gives a solution and an *optional* part that refines this solution. The problems that have been attempted using the anytime technique have the flavor of more traditional optimization problems, which by themselves cannot cover the space of planning problems. The feature of "interruptibility" of anytime algorithms is not particularly one of great value in deadline-coupled planning in commonsense scenarios involving hard and non-extensible deadlines. We want Dudley to come up with a feasible solution by the deadline (if possible). We do not care if at any point of time he has found an approximate solution of an inferior quality. Having that assurance is not crucial.

Also, in the anytime approach, the time for computation is not accounted for: "The time required for deliberation scheduling will not be factored into the overall time allowed for deliberation. For the techniques we are concerned with, we will demonstrate that deliberation scheduling is simple, and, hence, if the number of predicted events is relatively small, the time required for deliberation can be considered negligible," [DB88] (page 50). In the model we are employing, there is also a simple metareasoning process (computing WET, deciding among alternative plans, etc); but its time is not always precomputed but rather assessed as it occurs; and our underlying framework provides a general mechanism that measures the time utilized in any computation whatsoever (even if in future work we employ more complicated metareasoning such as utility-calculations, etc.).

Work by Horvitz et al [Hor88, Hor89, HR91] attacks problems that may be classified as "high stakes decision problems". A typical example is in the medical domain where the decision making is complex, but highly informed. Most of the options and quantified information regarding relationships among decisions and propositions is available in the form of *influence diagrams*. Horvitz et al have also addressed the problem of dividing computational resources between meta-reasoning and object-level problem solving, particularly in the case when both are being solved using anytime algorithms [HB90]. By the use of mathematical functions which assume particular forms for the various utilities, they manage to keep the meta-reasoning cost quite small or constant. Our work work by contrast makes no assumptions of highly informed domains or computable utilities. Thus the "expert" planning of Horvitz et al allows the possibility of much greater optimization than does the commonsense "inexpert" planning of Dudley.

In an approach that is inspired by economics, Etzioni [Etz91] addresses the problem for a time-constrained agent using special terms commonly used in economics. When a particular resource is available in limited quantity, it renders competing actions mutually exclusive. He defines an opportunity cost for each action, which is the maximum of the utilities of the other contending actions. He suggests a heuristic to choose the action with the highest marginal utility, without assuming prior knowledge of the utilities. There is a learning mechanism that calculates them through repeated executions. It seems that it would be possible in principle to implement Etzioni's methods within our active logic framework.

Lastly we mention work in the direction of building systems and architectures that exhibit desirable real-time behaviors, although not all components of these systems function in the real-time domain: Guardian [HRWA$^+$92] Phoenix [HHC90] and PRS [IG90]. FORBIN [DFM88] which is a planning architecture that supports hierarchical planning involving reasoning *about* deadlines, travel time, and resources are some examples of such systems. TILEWORLD [PR90] is a simulated dynamic and unpredictable parametrized agent and environment. It is possible to experiment with the behavior of the agent and various meta-level strategies by tuning parameters of the TILEWORLD system. Once again, although all the reasoning here is not performed in real-time, many of their observations, especially regarding the manifestation of agent attitudes through the tuning of parameters could be of use in the development of an active logic where the active logic can self-adjust its parameters to the environment to decide the level of risk or deliberation it can perform.

8 Conclusions and Future work

We have argued for the need for reasoning about time-of-planning to be included in the planning activity itself, especially in deadline situations. We then presented an illustrative scenario and a solution using active logics. We also examined a method for addressing space limitations, by introducing a short-term memory into the logic, and we showed that under certain conditions the resulting logic loses no power when so limited. We also discussed heuristics to improve the performance (but at the possible expense of proof-power).

Our work provides a uniform declarative framework that accounts for the time taken during planning and acting as they occur, allowing therefore meta-decisions about the course of such activity; the time for these meta-decisions is also measured and accounted for, not in advance but rather on-line. There is no infinite regress of meta-meta-meta-reasoning, since there is a built-in clock that is both declarative and procedural: the clock-time (Now) automatically updates the declarative belief base at every step, allowing the agent to maintain an up-to-date assessment of how much time has been taken, how much remains, and where the planning-acting situation stands. Even time taken to decide whether to refine or freeze (temporarily abandon) a plan-alternative is measured by the same mechanisms.

Among the many things that remain to be investigated, we single out two: (i) The planning we have considered is of a very elementary sort, compared to the current state of automated planning research. We have chosen to focus on the total-time-accounting aspect, and that has presented us with many severe challenges. However, we want to bring this line of work up to the level of being able to produce plausible plans in the same domains as other automated planners, with our added feature of total time-accounting. (ii) We want to incorporate our own related work on other (non-planning) problems involving time-accounting, such as a real-time version of the Yale shooting problem, to achieve an integrated formal real-time (in our sense of time-accounting) system for both planning and problem-solving.

A Sample Axioms

Relevant to moving:[45]

- $Run(T_1 : \overline{T}_2, Y, L_1 : L_2) \rightarrow At(T_2, Y, L_2)$,$T_2 = T_1 + (L_2 - L_1)/v_Y$[46]
- $condition(Run(T_1 : \overline{T}_2, Y, L_1 : L_2), At(T_1, Y, L_1))$
- $result(Run(T_1 : \overline{T}_2, Y, L_1 : L_2), At(T_2, Y, L_2))$

Relevant to untying and releasing:

- $Pull(T : T+1, X, L) \rightarrow Out\_of\_danger(T+1, X, L)$
- $condition(Pull(T : \overline{T+1}, X, L), \neg Tied(T, X, L))$
- $result(Pull(T : \overline{T+1}, X, L),\ Out\_of\_danger(T+1, X, L))$
- $Pick\_up(T : \overline{T+1}, Y, X) \rightarrow Have(T+1, Y, X)$
- $result(Pick\_up(T : \overline{T+1}, Y, X), Have(T : T+1, Y, X))$
- $condition(Pick\_up(T : \overline{T+1}, Y, X),$
  $At(T : T+1, X, L) \wedge At(T : T+1, Y, L))$

Relevant to telephones and warning:

- $condition(Warn(S : \overline{T}, X, Y), In\_contact(S : T, X, Y))$
- $result(Repeat\_Until(S : \overline{T}, Dial, Get\_connection, X, Y), In\_contact(X, Y))$
- $condition((Dial, At(S : T, X, L) \wedge At(S : T, phone, L))$

[45]These constitute a part of the current set of axioms and inference rules. [NKP91, KNP90] gives a more comprehensive initial set. Note that we employ a standard quasi-quote notation throughout, allowing a predicate symbol such as Run to appear also inside other predicates, e.g, condition("Run..."), and then suppress the quotes, yielding condition(Run...).

[46]v_Y is Y's speed while running.

B Sample Inference Rules

This section contains a sample subset of domain-independent inference rules for the active logic for deadline coupled planning.

1. The agent makes an observation

$$\frac{i : \mathbf{CS}(i, p, \{\ldots\})\ldots}{i+1 : \mathbf{CS}(i+1, p, \{\ldots, \alpha\}), \ldots}; \alpha \in \mathbf{OBS}(i+1)$$

   An observation is incorporated into the context set of every plan p being processed. In particular the null plan maintains a context consisting of all observations and the theorems that come to be proven in this context. Note that this rule is a modified form of the original step-logic observation rule which did not have to make the distinction between contexts. In the absence of any planning or temporal projection, the context of the null plan bears close resemblance to the belief set of the SL_7 logics of Elgot-Drapkin.

2. Forms the first partial plan(s) by finding a triplet for the goal

$$\frac{i : \begin{array}{l}\mathbf{Now}(i), \mathbf{Goal}(i, G(S:F, \ldots), \mathit{Deadline\_G}), \mathbf{Unsolved}(i, G), \\ \mathbf{CS}(i, \mathit{null}, \{\ldots, \mathit{Result}(A_k, R_{A_k}(S_k : F_k, \ldots)), \mathit{Condition}(A_k, C_{A_k}), A_k \rightarrow G, \ldots\})\ldots\end{array}}{i+1 : \begin{array}{l}\mathbf{Ppl}(i+1, p_k, \left\{\left[\begin{array}{c} C_{A_k} \\ A_k \\ R_{A_k}(S_k : F_k \bullet\!\!\rightarrow F, \ldots)\end{array}\right]\right\}), \\ \mathbf{CS}(i+1, p_k, \mathbf{CS}_{i,null})\mathbf{Proj}(i+1, p_k, \{\}) \\ \mathbf{Feasible}(i+1, p_k)\mathbf{WET}(i+1, p_k, 0)\ldots\end{array}}$$

   if $G \notin \mathbf{Proj}_{i,null} \cup \mathbf{CS}_{i,null}$.

   When Dudley has a goal that is not currently being planned for, he develops the first partial plan(s) for solving it. For every available action A_k (or conjunction of actions) that solves the goal he generates a new plan and calls it by a name p_k. In short, corresponding to every axiom with the consequent G he performs backward reasoning to deduce the actions that must be done to achieve G. The time of the action is is linked the deadline by the "$\bullet\!\!\rightarrow$" symbol which denotes that the result of the action must be protected until the deadline. We give a simple example to illustrate this rule.

   $\mathbf{5} : \mathbf{Now}(5), \mathbf{Goal}(5, \mathit{At}(10, \mathit{dudley}, \mathit{home}), 10),$
   $\mathbf{Unsolved}(5, \mathit{At}(10, \mathit{dudley}, \mathit{home})),$
   $\mathbf{CS}(5, \mathit{null}, \{\ldots, \mathit{Result}(\mathit{Walk}(T1 : T2, \mathit{dudley}, \mathit{garden} : \mathit{home}, \mathit{Walk\_speed}),$
   $\mathit{At}(T+3, \mathit{dudley}, \mathit{home})),$
   $\mathit{Condition}(\mathit{Walk}(T : T2, \mathit{dudley}, \mathit{garden} : \mathit{home}, \mathit{Walk\_speed}),$
   $\mathit{At}(T, \mathit{dudley}, \mathit{garden})),$
   $\mathit{Walk}(T : T2, \mathit{dudley}, \mathit{garden} : \mathit{home}, \mathit{Walk\_speed}) \rightarrow \mathit{At}(T2, \mathit{dudley}, \mathit{home}), \ldots\})$

$$\mathbf{6}: \mathbf{Ppl}(6, walk\_it, \left\{ \left[\begin{array}{c} At(T, dudley, garden) \\ Walk(T:T2, dudley, garden:home, Walk\_speed) \\ At(T2, dudley, home) \end{array} \right] \right\}),$$

3. Adds an action to the plan to satisfy a condition

$$\frac{i: \mathbf{Ppl}(i, p, \left\{ \ldots \left[\begin{array}{c} \{\ldots, C\} \\ A \\ R_A \end{array} \right] \ldots \right\}), \mathbf{CS}(i, p, \{\ldots, Q \to C\})}{i+1: \mathbf{Ppl}(i+1, p, \left\{ \ldots \left[\begin{array}{c} C_Q \\ Q \\ R_Q \end{array} \right] \left[\begin{array}{c} \{\ldots, C\} \\ A \\ R_A \end{array} \right] \ldots \right\})}$$

   if $C \notin \mathbf{Proj}_{i,p} \cup \mathbf{CS}_{i,p}$

   For every condition C in the condition list of an action that is not projected to be true, if there is an axiom for satisfying it, Dudley adds the corresponding action to the plan. If there is more than one axiom for satisfying the same condition, Dudley formulates a plan for each possibility, and indexes the name of the partial plan with a new suffix to distinguish the new plans.

4. Refines a non-primitive action

$$\frac{i: \mathbf{Ppl}(i, p, \left\{ \ldots \left[\begin{array}{c} C_A \\ A \\ R_A \end{array} \right] \ldots \right\}), CS(i, p, \{\ldots, Q_1 \wedge \ldots \wedge Q_k \to A\})}{i+1: \mathbf{Ppl}(i+1, , p, \left\{ \ldots \left[\begin{array}{c} C_{Q_1} \\ Q_1 \\ R_{Q_1} \end{array} \right] \ldots \left[\begin{array}{c} C_{Q_k} \\ Q_k \\ R_{Q_k} \end{array} \right] \ldots \right\})}$$

   provided every condition $C_A \in \mathbf{CS}_{i,p} \cup \mathbf{Proj}_{i,p}$.

   The active logic planner is hierarchical. Abstraction is embodied in the way the axioms encode the knowledge about actions. Skeleton plans at upper levels first synthesized by using higher level actions. These are then broken into more primitive actions by rules such as the action refinement rule described above. As the refinement progresses, better estimates of the execution time of the plan become available. The context set maintains the actions reasoned about at all levels. Further, these actions are used to annotate any reasoning based on them. Lower level actions are annotated by the higher level action that they refine (see the context set rule from Chapter 4.1.3). In the event replanning becomes necessary, this provides the mechanism to revise a plan by substituting an action and all the actions below it in the hierarchy when required. Our design allows for the concurrent processing of levels, and for concurrent refinement of multiple partial plans[47].

[47]More on restricting parallelism in Section 6.

5. Includes a *Conditional* action in the plan

$$\frac{i : \mathbf{Ppl}(i,p,\left\{\ldots \begin{bmatrix} \{\ldots, C_{A_k}\} \\ A \\ R_A \end{bmatrix} \ldots\right\}), \mathbf{CS}(i,p,\{\ldots, C \wedge Q \rightarrow C_{A_k}\})}{i+1 : \mathbf{Ppl}(i+1,p,\left\{\ldots \begin{bmatrix} \{C\} \cup C_Q \\ Q \\ R_Q \end{bmatrix} \begin{bmatrix} \{\ldots, C_{A_k}\} \\ A \\ R_A \end{bmatrix} \ldots\right\})}$$

if $C \notin \mathbf{Proj}_{i,p} \cup \mathbf{CS}_{i,p}$

When an axiom $C \wedge Q \rightarrow C_{A_k}$ is found to be a way to satisfy a (sub)goal C_{A_k}, the action Q itself is not sufficient for C_{A_k}, C has to be true in the projection as well. This is taken care of by adding C in addition to the conditions for Q in the action triplet introduced in the plan.

6. Executes an action

$$\frac{i : \mathbf{Ppl}(i,p,\left\{\begin{bmatrix} C_A \\ A(i:T,\ldots) \\ R_B \end{bmatrix}_1 \ldots\right\}), \mathbf{CS}(i,p,\{\ldots, C_A\})}{i+1 : \mathbf{Ppl}(i+1,p,\{\ldots\})}$$

This inference rule executes an action when its start time has been bound to the current *Now* by the agent. The time for some actions is decided right when they are inserted into the plan, for others it must be decided by a specific inference rule. In our present implementation, we execute a primitive action as soon as its conditions are satisfied.

7. Includes a *Repeat-until* action in the plan with *signaling-condition* SC_A

$$\frac{i : \begin{array}{l} \mathbf{Ppl}(i,p,\left\{\ldots \begin{bmatrix} C_B \\ B \\ R_B \end{bmatrix}_j \ldots\right\}), \\ \mathbf{CS}(i,p,\{\ldots, \mathit{Repeat\_until}(S:T,A,SC_A,\ldots) \rightarrow C_B, \mathbf{condition}(A, C_A)\}) \end{array}}{i+1 : \mathbf{Ppl}(i+1,p,\left\{\ldots \begin{bmatrix} C_A(S:T,\ldots) \\ \mathit{Repeat\_until}(S:T,A,SC_A,\ldots) \\ C_B \end{bmatrix}_{j-1} \ldots\right\})}$$

The above inference rule adds a repeat-until type of action to the plan. The condition of the repeat-until action is the condition for the repeated part, but maintained over the entire duration of the outer loop.

8. Executes of a *Repeat-until* action in the plan

$$\frac{i : \mathbf{Ppl}(i,p,\left\{\ldots \begin{bmatrix} C_A \\ \mathit{Repeat\_until}(i:T,A,SC_A,\ldots) \\ C_B \end{bmatrix}_1 \ldots\right\})}{i+1 : \mathbf{Ppl}(i+1,p,\left\{\begin{bmatrix} C_A \\ \mathit{Repeat\_until}(i+1:T,A,SC_A,\ldots) \\ C_B \end{bmatrix}_1 \ldots\right\})}$$

if $SC_A \notin \mathbf{CS}_{i,p} \cup \mathbf{Proj}_{i,p}$

9. Completes execution of a *Repeat-until* action when a *signaling-condition* appears

$$\frac{i : \mathbf{Ppl}(i,p,\left\{\begin{bmatrix} C_A \\ \mathit{Repeat\_until}(S:T,A,SC_A,\ldots) \\ C_B \end{bmatrix}_1 \ldots\right\}), \mathbf{CS}(i,p,\{\ldots,SC_A\})}{i+1 : \mathbf{Ppl}(i+1,p,\{\ldots\})}$$

10. Spawns the generation of multiple plans on encountering a compound condition[48]

$$\frac{i : \begin{array}{c} \mathbf{Ppl}(i,p,\left\{\ldots \begin{bmatrix} C'_A(R:S,\ldots) \wedge C''_A(V:W,\ldots) \\ A \\ R_A \end{bmatrix}_j \ldots\right\}), \\ \mathbf{CS}(i,p,\{A' \rightarrow C'_A, A'' \rightarrow C''_A\}) \end{array}}{i+1 : \begin{array}{l} \mathbf{Ppl}(p_1,i+1,\left\{\ldots \begin{bmatrix} \ldots \\ A' \\ C'_A(P:Q \bullet\!\!\rightarrow S) \end{bmatrix}_{j-2} \begin{bmatrix} \ldots \\ A'' \\ C''_A(T:U \bullet\!\!\rightarrow W) \end{bmatrix}_{j-1} \ldots\right\}) \\ \mathbf{Ppl}(p_2,i+1,\left\{\ldots \begin{bmatrix} \ldots \\ A'' \\ C''_A(T:U \bullet\!\!\rightarrow W) \end{bmatrix}_{j-2} \begin{bmatrix} \ldots \\ A' \\ C'_A(P:Q \bullet\!\!\rightarrow S) \end{bmatrix}_{j-1} \ldots\right\}) \end{array}}$$

This inference rule encodes the linear planning strategy of our planner. Clearly, a total ordering such as this will cause the generation of non-optimal and sometimes even redundant plans. We have some heuristic inference rules that identify some obvious redundancies in a planner and identify the presence of loops. In general though, we have not focussed on plan optimization. One heuristic rule that identifies a redundant plan and rejects in favor of a better one is described below.

11. Freeze a plan when it is found to be inefficient

[48]This rule can be easily generalized to more than two conjuncts in a condition.

$$\frac{i : \mathbf{Ppl}(i, p, \left\{ \dots \begin{bmatrix} \dots \\ A_j \\ R_{A_j}(P : Q \bullet\!\!\rightarrow S) \end{bmatrix} \dots \begin{bmatrix} C_{A_k}(V : W, \dots) \\ A_k \\ \dots \end{bmatrix} \dots \right\})}{i + 1 : \mathbf{Freeze}(i, p)}$$

if $P : S$ and $V : W$ overlap, and R_{A_j} and C_{A_k} are in direct or uniqueness contradiction

References

[AC87] P. E. Agre and D. Chapman. Pengi: An implementation of a theory of activity. In *Proceeding, AAAI-87*, pages 268–272, 1987.

[AHT90] J. Allen, J. Hendler, and A. Tate. *Readings in Planning.* Morgan Kaufmann Publishers, Inc., 1990.

[Ams91] J. Amsterdam. Temporal reasoning and narrative conventions. In *Proc. of KR-91*, pages 15–21, 1991.

[Bak89] Andrew B. Baker. A simple solution to the Yale Shooting Problem. In *Proceedings, KR89*, pages 11–20, 1989.

[BD94] M. Boddy and T. Dean. Deliberation scheduling for problem solving in time-constrained environments. *Artificial Intelligence*, 1994. To appear.

[Bro86] Rodney Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1):14–23, 1986.

[Bro91] R. Brooks. Intelligence without reason. In *Proceedings of IJCAI-91, Prepared for Computers and Thought*, 1991.

[CL90] P. Cohen and H. Levesque. Intention is choice with commitment. *Artificial Intelligence*, 42:213–261, 1990.

[Dav88] D. E. Davis. Inferring ignorance from the locality of visual perception. In *Proceedings, AAAI-88*, St. Paul, Minnesota, 1988.

[DB88] Thomas Dean and Mark Boddy. An analysis of time-dependent planning. In *Proceedings, AAAI-88*, pages 49–54, St. Paul, Minnesota, 1988.

[DF89] B. D'Ambrosio and M. Fehling. Resource bounded-agents in an uncertain world. In *Proceeding, AAAI symposium on limited rationality*, Stanford, California, 1989.

[DFM88] Thomas Dean, R. James Firby, and David Miller. Hierarchical planning involving deadlines, travel time and resources. *Computational Intelligence*, 4:381–389, 1988.

[DM87] T. Dean and D. McDermott. Temporal data base management. *Artificial Intelligence*, 32(1):1–55, 1987.

[Doy88] J. Doyle. Artificial intelligence and rational self-government. In *TR CS-88-124, CMU, Pittsburgh*, 1988.

[DP86a] J. Drapkin and D. Perlis. A preliminary excursion into step-logics. In *Proceedings SIGART International Symposium on Methodologies for Intelligent Systems*, pages 262–269. ACM, 1986. Knoxville, Tennessee.

[DP86b] J. Drapkin and D. Perlis. Step-logics: An alternative approach to limited reasoning. In *Proceedings of the European Conf. on Artificial Intelligence*, pages 160–163, 1986. Brighton, England.

[ED88a] J. Elgot-Drapkin. *Step-logic: Reasoning Situated in Time*. PhD thesis, Department of Computer Science, University of Maryland, College Park, Maryland, 1988.

[ED88b] J. Elgot-Drapkin. *Step-logic: Reasoning Situated in Time*. PhD thesis, Department of Computer Science, University of Maryland, College Park, Maryland, 1988.

[ED91] J. Elgot-Drapkin. Step-logic and the three wise men problem. In *Proceeding, AAAI-91*, Anaheim, California, 1991.

[EDMP87] J. Elgot-Drapkin, M. Miller, and D. Perlis. Life on a desert island: Ongoing work on real-time reasoning. In F. M. Brown, editor, *Proceedings of the 1987 Workshop on The Frame Problem*, pages 349–357. Morgan Kaufmann, 1987. Lawrence, Kansas.

[EDP90] 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.

[Etz91] Oren Etzioni. Embedding decision-analytic control in a learning architecture. *Artificial Intelligence*, 49:129–159, 1991.

[FH88] R. Fagin and J. Halpern. Belief, awareness, and limited reasoning. *Artificial Intelligence*, 34:39—76, 1988.

[Gel88] Michael Gelfond. Autoepistemic logic and formalization of common-sense reasoning. In *Proceedings of the second international workshop on nonmonotonic reasoning*, 1988. Munich 88.

[Geo87] Michael P. Georgeff. Many agents are better than one. In F. M. Brown, editor, *Proceedings of the 1987 Workshop on The Frame Problem*, pages 59–75x. Morgan Kaufmann, 1987. Lawrence, Kansas.

[Ger90] M.T. Gervasio. Learning general completable reactive plans. In *Proceeding, AAAI-90*, pages 1016–1021, 1990.

[GL88] M. Georgeff and A. Lansky. Reactive reasoning and planning. In *Proceedings AAAI-88*, pages 677–682, 1988.

[GL94] A. Garvey and V. Lesser. A survey of research in deliberative real-time ai. *Journal of Real Time Sytems*, 6:317–347, 1994.

[GS87] M. L. Ginsberg and D. E. Smith. Reasoning about action I: A possible worlds approach. In F. M. Brown, editor, *Proceedings of the 1987 Workshop on The Frame Problem*, pages 233–258, Lawrence, KS, 1987. Morgan Kaufmann.

[Hau87] Brian A. Haugh. Simple causal minimizations for temporal persistence and projection. In *Proceedings of the sixth national conference on artificial intelligence*, pages 218–223, 1987.

[HB90] Eric J. Horvitz and John S. Breese. Ideal partition of resources for metareasoning. Technical Report KSL-90-26, Knowledge Systems Laboratory, Stanford University, March 1990.

[HHC90] Adele E. Howe, David M. Hart, and Paul R. Cohen. Addressing real-time constraints in the design of autonomous agents. *Journal of Real-Time Systems*, 2(12):81–97, 1990.

[HM87] S. Hanks and D. McDermott. Nonmonotonic logic and temporal projection. *Artificial Intelligence*, 33:379–412, 1987.

[Hor88] E. J. Horvitz. Reasoning under varying and uncertain resource constraints. In *Proceeding, AAAI-88*, pages 111–116, St. Paul, Minnesota, 1988.

[Hor89] Eric J. Horvitz. Reasoning about beliefs and actions under computational resource constraints. In T.S. Levitt L.N. Kanal and J.F. Lemmer, editors, *Uncertainty in Artificial Intelligence 3*. Elsevier Science Publishers, 1989.

[HR91] Eric J. Horvitz and Geoffrey Rutledge. Time-dependent utility and action under uncertainty. In *Uncertainty in Artificial Intelligence 6*, 1991.

[HRWA$^+$92] B. Hayes-Roth, R. Washington, D. Ash, A. Collinot, A. Vina, and A. Sevier. Guardian: A prototype intensive-care monitoring agent. *Artificial Intelligence in Medicine*, 4:165–185, 1992.

[HTD90] J. Hendler, A. Tate, and M. Drummond. Systems and techniques: AI planning. *AI Magazine*, 11(2):61–77, 1990.

[IG90] F. Ingrand and M. Georgeff. Managing deliberation and reasoning in real-time ai systems. In *Proceedings of 1990 DARPA workshop on Innovative Approaches to Planning, Scheduling and Control*, pages 284–291, San Diego, CA, November 1990.

[Kae87] L. Kaelbling. An architecture for intelligent reactive systems. In M. Georgeff and A. Lansky, editors, *Reasoning about Actions and Plans*. Morgan-Kaufmann, 1987.

[Kau86] H. Kautz. The logic of persistence. In *Proceedings, AAAI-86*, pages 401–405, 1986.

[KL88] S. Kraus and D. Lehmann. Knowledge, belief and time. *Theoretical Computer Science*, 58:155–174, 1988.

[KNP90] S. Kraus, M. Nirkhe, and D. Perlis. Deadline-coupled real-time planning. In *Proceedings of 1990 DARPA workshop on Innovative Approaches to Planning, Scheduling and Control*, pages 100–108, San Diego, CA, 1990.

[Kon86] K. Konolige. *A Deduction Model of Belief.* Pitman, London, 1986.

[KP89] S. Kraus and D. Perlis. Assessing others' knowledge and ignorance. In *Proc. of the 4th International Symposium on Methodologies for Intelligent Systems*, pages 220–225, Charlotte, North Carolina, 1989.

[KR92] S. Kraus and J. Rosenschein. The role of representation in interaction: Discovering focal points among alternative solutions. In *Decentralized Artificial Intelligence, Volume 3*, Germany, 1992. Elsevier Science Publishers.

[Lev84] H. Levesque. A logic of implicit and explicit belief. In *Proc. of AAAI-84*, pages 198–202, Austin, TX, 1984.

[Lif87a] V. Lifschitz. Pointwise circumscription. In *Readings in Nonmonotonic Reasoning*. Morgan Kaufmann, 1987.

[Lif87b] Vladimir Lifschitz. Formal theories of action. In *The Frame Problem in Artificial Intelligence*, pages 35–57, Los Altos, CA, 1987. Morgan-Kaufmann.

[McD78] D. McDermott. Planning and acting. *Cognitive Science*, 2:71–109, 1978.

[McD87] D. McDermott. Nonmonotonic logic and temporal projection. *Artificial Intelligence*, 33:379–412, 1987.

[MH69] J. McCarthy and P. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence*, volume 4, pages 463–502. Edinburgh University Press, 1969.

[Mil56] G. Miller. The magical number seven plus or minus two. *The Psychological Review*, 63:81–97, 1956.

[Mor88] P. H. Morris. The anomalous extension problem in default reasoning. *Artificial Intelligence*, 35:383–399, 1988.

[MP92] Z. Manna and A. Pnueli. *Temporal Logic of Reactive and Concurrent Systems.* Addison Wesley, 1992.

[Nil83] N. Nilsson. Artificial intelligence prepares for 2001. *AI Magazine*, 4(4):7–14, 1983.

[NK94] M. Nirkhe and S. Kraus. Formal real-time imagination. *Fundamenta Informaticae, special issue on Formal Imagination*, 1994. accepted for publication, in press.

[NKP91] M. Nirkhe, S. Kraus, and D. Perlis. Fully deadline-coupled planning: One step at a time. In *Proceedings of the sixth international symposium on methodologies for intelligent systems*, Charlotte, NC, 1991.

[NKP93] M. Nirkhe, S. Kraus, and D. Perlis. Situated reasoning within tight deadlines and realistic space and computation bounds. In *Proceeedings of the Second Symposium On Logical Formalizations Of Commonsense Reasoning*, 1993.

[Pea88] Judea Pearl. On logic and probability. *Computational Intelligence*, 4:99–103, 1988.

[PEDM] D. Perlis, J. Elgot-Drapkin, and M. Miller. Stop the world! – I want to think! Invited paper, *International J. of Intelligent Systems*, special issue on Temporal Reasoning, K. Ford and F. Anger (eds.), vol. 6, 1991, pp. 443–456.

[PR90] M. E. Pollack and M. Ringuette. Introducing the tileworld: Experimentally evaluating agent architectures. In *Proceedings, AAAI-90*, pages 183–189, 1990.

[RK86] S.J. Rosenschein and L.P. Kaelbling. The synthesis of digital machines with provable epistemic properties. In *Proceedings of Workshop on Theoretical Aspects of Knowledge*, Monterey, CA, 1986.

[RW91] S. Russell and E. Wefald. *Do the Right Thing.* MIT Press, Cambridge, MA, 1991.

[Sac73] E. D. Sacerdoti. Planning in a hierarchy of abstraction spaces. In *Proceedings of the 3rd Int'l Joint Conference on Artificial Intelligence*, Palo Alto, California, 1973.

[Sac75] Earl Sacerdoti. The non-linear nature of plans. In *Advance papers for IJCAI – 75*, 1975.

[Sho88] Yoav Shoham. Chronological ignorance: Experiments in nonmonotonic temporal reasoning. *Artificial Intelligence*, 36:279–331, 1988.

[Sho91] Yoav Shoham. Agent0: A simple agent language and its interpreter. In *Proc. Ninth National Conference on Artificial Intelligence*, pages 704–709. American Association for Artificial Intelligence, 1991.

[Sus73] G.A. Sussman. A computational model of skill acquisition. Technical Report AI-TR-297, MIT AI Lab, 1973.

[Tat75] A. Tate. Interacting goals and their use. In *Proceedings of the 4th Int'l Joint Conference on Artificial Intelligence*, pages 215–218, 1975.

[Tat77] A. Tate. Project planning using a hierarchical non-linear planner. Technical Report Report 25, Dept. of Artificial Intelligence, Edinburgh University, 1977.

[Ver83] S. Vere. Planning in time: Windows and durations for activities and goals. In *Proceedings, IEEE Trans. on Pattern Analysis and Machine Intelligence*, pages 246–267, 1983.

[Wal75] R. Waldinger. Achieving several goals simultaneously. Technical Report Technical Note 107, SRI AI Center, 1975.

[Wil83a] R. Wilensky. *Planning and understanding.* Addison Wesley, Reading, Mass, 1983.

[Wil83b] D.E. Wilkins. Representation in a domain-independent planner. In *Proceedings of the 8th Int'l Joint Conference on Artificial Intelligence*, pages 733–740, 1983.

[ZR92] S. Zilberstein and S. Russell. Constructing utility-driven real-time systems using anytime algorithms. In *Proceedings of the IEEE workshop on imprecise and approximate computation*, pages 6–10, 1992.