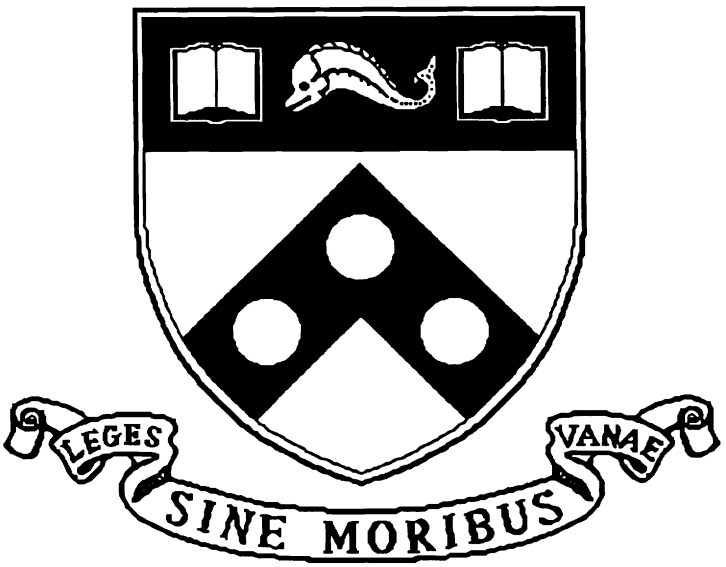
Doing What You're Told: Following Task Instructions In Changing, but Hospitable Environments

MS-CIS-92-74
LINC LAB 236

Bonnie Webber
Norman Badler
F. Breckenridge Baldwin
Welton Becket
Barbara Di Eugenio
Christopher Geib
Moon Jung
Libby Levison
Michael Moore
Michael White



University of Pennsylvania
School of Engineering and Applied Science
Computer and Information Science Department
Philadelphia, PA 19104-6389

September 1992

Doing What You're Told: Following Task Instructions in Changing, but Hospitable Environments*

Bonnie Webber Norman Badler F. Breckenridge Baldwin
Welton Becket Barbara Di Eugenio Christopher Geib
Moon Jung Libby Levison Michael Moore Michael White

1 Introduction

The AnimNL project has as its goal the automatic creation of *animated task simulations* from *natural-language instructions*. The agents participating in these task simulations are animated human figures, as shown in Figure 1. AnimNL is intended to support advanced human factors analysis in what has come to be called *virtual prototyping*, enabling users of computer-aided design tools to simulate people's interactions with the artifacts they are designing and thereby to notice design flaws that might otherwise only become apparent after the artifacts enter the workplace. In this type of human factors analysis, designers usually assume that agents and artifacts are in hospitable environments that the agents exercise control over, while not necessarily knowing everything about. The point is to see if and how a task can be carried out under realistic base conditions.

The AnimNL project builds on an animation system, *Jack*™, that has been developed at the Computer Graphics Research Lab at the University of Pennsylvania. *Jack* provides articulated, animated human figures capable of realistic motion through model-based behaviors [6, 7, 45]. In addition, *Jack* agents can be anthropometrically sized and given different "strengths", so as to vary their physical capabilities. Different spatial environments can be constructed and modified at will, so as to vary the situations in which tasks are carried out. Such flexibility enables designers to explore a wide range of usage situations.[1]

To simulate agents carrying out tasks in response to instructions, one must provide a way of mapping from *high-level task specifications* (specifying a structure of related goals to be achieved and constraints on achieving them) to *plausible physical behaviors* performed

*The authors would like to thank Mark Steedman, for his comments on earlier drafts of the paper, and Xinmin Zhao, for the picture of our agent carrying his coffee urn (Figure 1). This research is partially supported by ARO Grant DAAL03-89-C-0031, including participation by the U.S. Army Human Engineering Laboratory, Natick Laboratory, TACOM, and NASA Ames Research Center; U.S. Air Force DEPTH contract through General Dynamics Convair F33615-91-C-0001; NSF CISE Grant CDA88-22719, and DARPA grant N00014-90-J-186.

[1] In discussing agents, we will use the pronoun "he", since we will be using a male figure in our illustrated example – i.e., a figure with male body proportions. The *Jack* animation system provides anthropometrically sizable female figures as well.

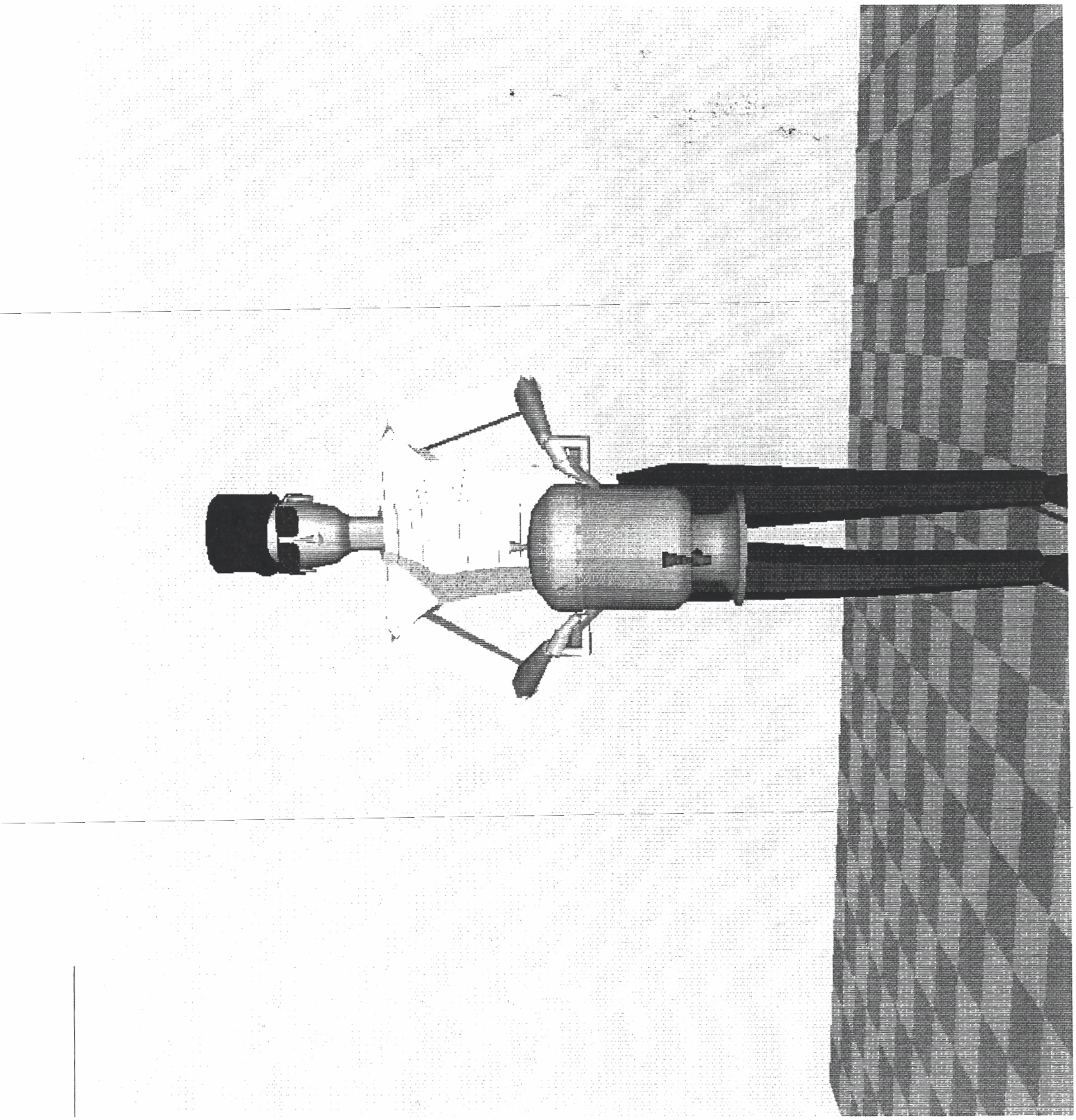


Figure 1: An Animated Agent

veridically. Generating and animating the behavior of highly articulated agents carrying out tasks in a physically veridical manner, in an environment about which they have only partial knowledge, raises a number of problems that have not received serious consideration in planning research or in robotics, although our solutions draw upon important recent work in Natural Language semantics [29], planning and plan inference [1, 39, 46], philosophical studies of intention [11], reasoning about knowledge and action [35, 42, 43] and subsumption architectures for autonomous agents [12].

As for the structure of this paper, we start with an overview in Section 2 of the various processing used in producing an animated simulation starting from an instruction step. Subsequent sections then focus on separate processes (or groups of processes) and show how expectations and generalized abilities facilitate agents' interactions with environments of which they have only partial knowledge.

2 Overview of AnimNL

The AnimNL system architecture (Figure 2) reflects processes that we have so far found necessary to implement, to enable an agent to understand and act in accordance with purposeful instructions and to enable an animation system to simulate and animate that behavior. (We do not believe that this current architecture will remain sufficient, as this work continues.)

The first thing to notice about the diagram in Figure 2 is that it consists of two relatively independent sets of processes. The first set produces commitments to purposeful activity, which we call *action directives* – e.g.

- goto(door1, open(door1)) – "go to door1 for the purpose of opening it"
- grasp(urn1, carry(urn1)) – "grasp urn1 for the purpose of carrying it".

The second set determines how the agent should *move* in order to fulfill these commitments.

We should explain here why action directives (and other task actions in AnimNL as well) are annotated with the purpose they are intended to serve. This is because we have found it impossible to make appropriate decisions about how an agent should act – even to the point of how it should *move* – without taking into account what its actions and movements are intended for.

Intention has been identified as a modulating factor in rational behavior by various researchers (e.g., [11, 14, 46]). However, the role of intentions in interpreting instructions has not yet been fully explored. Chapman [14] does stress that an instruction such as "use the knife" can only be carried out after it has been interpreted *in terms of the situation at hand.* However, for Chapman, intentions are not purposes that an agent adopts, but rather features of the current situation that make a particular course of action sensible: in the context of Chapman's video-game, a knife could be used either to kill a monster or to jimmy a door. The situation itself – e.g. a monster is threatening the agent – will make clear what actions are sensible and determine the "right" sense of "use the knife".

As we hope this paper will make clear, intentions impact almost every aspect of processing in AnimNL, from language understanding and plan expansion, to the lowest levels of object manipulation and locomotion.

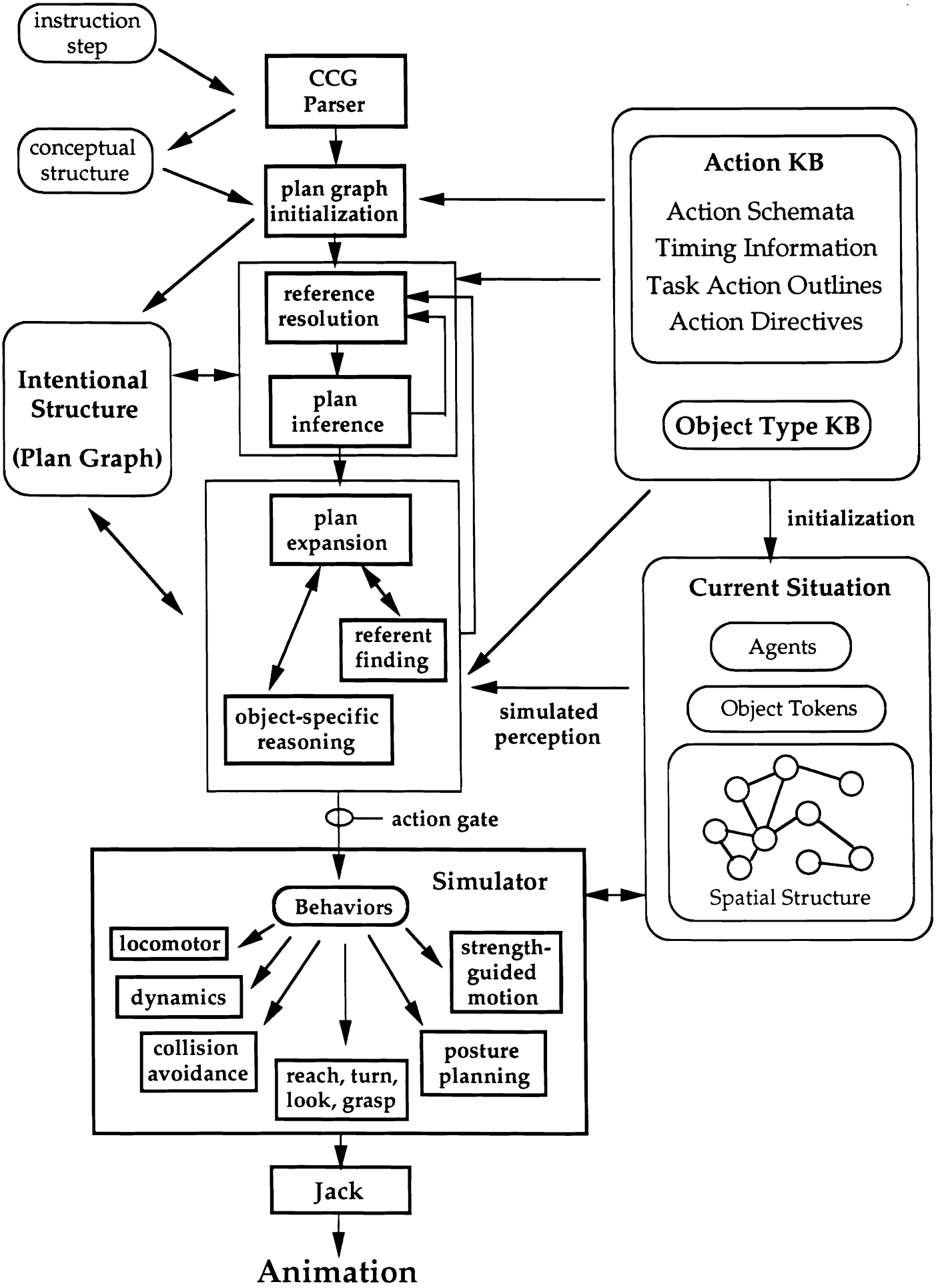


Fig 2: AnimNL System Architecture

To explain the main features of the current AnimNL architecture, we will now walk through the steps from instructions to initial commitments to act, and how actions once embarked upon allow further commitments to be made and acted on. (While our discussion here is in terms of single-agent procedures, it can be extended to multi-agent procedures with the addition of communicative or coordinating actions - cf. [25, 37, 51].)

Instructions are given to AnimNL in *steps* consisting of one or more utterances. Such multi-clause instruction steps are common in maintenance and assembly instructions - e.g.

> "With door opened, adjust switch until roller contacts cam and continuity is indicated at pins A and B. Verify positive switch contact by tightening bottom nut one additional turn." (Air Force manual T.O. 1F-16C-2-94JG-50-2, p. 5-24)

While there are no firm guidelines as to what a single instruction step should encompass, often steps are organized around small coherent sub-tasks (such as adjusting a switch). Such a step may specify several actions that need to be performed together (possibly in some partially-specified order) to accomplish a single subtask, or several aspects of a single complex action (e.g. its purpose, manner, things to watch out for, appropriate termination conditions, etc.). The agent must develop some degree of understanding of the whole step before starting to act.

A "step", for AnimNL, not only defines a subtask, but also specifies behavior that the agent must attend to continuously: while carrying out a step, attention must be maintained on the task at hand. The agent cannot stop until he finishes or hits a snag in the current step. In other words, a step defines the sequence of the instructions that must be processed as a whole, *before* the agent begins to act on them. (With some reflection on one's own past confrontations with new instructions, it is easy to recall situations where one has started to act after reading a few sentences, only to find the resulting disaster could have been avoided by reading a few sentences more. It is not always obvious when one should begin to act, although good instructions often demarcate such steps using devices like paragraphing. In AnimNL, we get around this problem for now by having the designer/user formulate the steps.)

The example instruction step that we will focus on and illustrate in this paper is "Go into the kitchen to get me the coffee urn". While it contains only two clauses, it does illustrate a surprisingly large number of interesting points.

Steps are processed by a parser that uses a combinatory categorial grammar (CCG) [52] to produce an action representation based on Jackendoff's *Conceptual Structures* [29]. For example, from the input instruction "Get me the coffee urn", the parser produces:

$$[\text{CAUSE}(i, [\text{GO}_{\text{Sp}}([\textsc{urn-of-coffee}]_j, k)])]_\beta$$

$$\left[\begin{array}{l} \text{FROM}([\text{AT}(j)]) \\ \text{TO}(me) \end{array}\right]_k$$

This can be glossed as *the agent causing the coffee urn to go from where it is now to where the speaker is.* (Following [58], indices are used to indicate different instances of a conceptual type.)

One reason for using Jackendoff's Conceptual Structures is that they can reveal where information may be missing from an utterance and have to be provided by inference. Here

the representation makes explicit where the coffee urn must be moved from, namely "AT([j])" – where the urn is now. (If the sentence had been "get me the coffee urn from the kitchen", the kitchen would fill this argument.) These Conceptual Structures are discussed in more detail in Section 3.3.3.

The indexed conceptual structures corresponding to a single instruction step are incrementally developed into a *plan graph* that represents the structure of the animated agent's intentions, via processes of *reference resolution*, *plan inference*, *plan expansion*, *referent finding* and *object-specific reasoning*. (These processes are discussed more in Sections 3–6.) The leaves of the plan graph are *action directives*, such as those shown at the start of this section.

The incremental nature of plan graph development derives, in part, from limits on agent knowledge. In particular, we assume that an agent cannot have up-to-date *knowledge* of any part of its environment that is outside its direct perception, where its direct perception is limited by a *portal assumption*: Portals like doors connect opaquely-bounded three-dimensional sub-spaces, which may be adjacent to or embedded within one another (e.g., boxes inside boxes). Agents can only see into, and hence only know the contents of, spaces that have a *portal* open into the space the agent occupies, which we call the *active space*. Thus agents have to open doors, lids, etc., to explore other spaces. Within the *active space*, we make the simplified assumption that agents can see everything up to closed portals. (This simplified model of perception is facilitated by the Object Token Data Base and the Spatial Structure shown in Figure 2. Eventually, we will drop this simplification when it becomes interesting to do so, since the space that is directly visible to the agent is actually modelled in *Jack* [48].)

An action directive will be gated down to the set of simulation process when

- it is "executable" (see Sections 4–6).
- all actions temporally prior to it have been committed to. (Previous actions need not have been completed: an agent can be, and usually is, doing several things at one time.)
- its purpose has been determined.

Gated actions are sent to the Simulator (described in more detail in Section 7) which produces a "program" of *behaviors* to be executed (simulated) in parallel and animated to produce a visual display of what's going on. Actions change the world, as well as the agent's knowledge. Such changes trigger further elaboration of the agent's intentional structure (i.e., the plan graph) and further commitments to action.

We now describe the type of processing required for instruction understanding and response in more detail.

3 Instruction Understanding

Usually one thinks of instructions in terms of what they explicitly tell one to do or to avoid doing. Such strictures may be conveyed explicitly or implicitly. Part of our research [6, 17, 18, 19, 20, 21, 31, 32, 56, 57], as well as that of others [3, 14, 53], has been on this aspect of instructions. In order to understand and respond appropriately to our example

instruction "Go into the kitchen to get the coffee urn", one should also understand another role that instructions can play – that of providing agents with specific, useful *expectations* about actions and their consequences.

We noted in our discussion of instruction *steps* in the previous section, that steps may specify several actions that need to be performed together, often under partial ordering, to accomplish a single sub-task. Agents must develop some level of understanding of an entire step before beginning to act. This is the most obvious sense in which instructions set up agent expectations: by assuming that an instruction step reliably describes a way of accomplishing a sub-task, an agent will expect that the conditions needed for accomplishing subsequent parts of the specified sub-task will hold when they need to. The agent will not expect that he needs to do anything else in order to make these conditions hold. (Of course, if the instructions are poorly defined or otherwise incomprehensible, any trust in the expectations they raise may be misplaced. We hope that recent attempts to characterize what makes for good document design [49, 50] will lead to improvements in the quality of instructions.)

Besides these general expectations, instructions may engender more specific types of expectations that agents can use in carrying out tasks, such as

- expectations about how long a process will take to come to some desired state;
- expectations about the intended consequences of an action;
- expectations about where objects are to be found.

These are discussed in turn in the next three subsections.

3.1 Expectations about Processes

In some earlier work designed for creating animations from recipes, Karlin [31, 32] analysed a range of temporal and frequency adverbs found in instructions. One particular construction she analysed is the following:

> Do α for < duration > or until < event >
> e.g. "Steam 2 minutes or until mussels open."

Karlin notes that this is not a case of logical disjunction, where the agent can choose which disjunct to follow: rather, the explicit duration suggests the usual amount of time that it will take the mussel-steaming process to effect the desired change in the mussels' state. The desired state is that all the mussels that were closed when they were put into the pot (already open ones having been discarded as dead, prior to this point) are now open. If they are not open after two minutes, the agent should wait a bit longer. Those that have not opened after another short wait should then be discarded, since they contain nothing but mud.

The usefulness of this expectation of how long a process will take to come to some desired state comes from the *cost* of sensing. In the case of steaming, cooking is usually done in a closed, opaque cooking pot. Every time the lid is removed to check the state of the contents, the steam used to cook the contents escapes, setting the process back. The result of sensing

too often, is that the mussels become tough through over-cooking. The expectation can therefore be used by the agent to gauge how long he can safely wait before beginning to make costly sensing tests.

3.2 Expectations about Consequences

In a recent paper [56], two of the current authors describe another type of expectation raised by instructions – that is, an expectation about the intended properties of objects produced by actions that can have any of several results.

Consider for example, the action of mixing flour, butter and water. Depending on the relative amounts of these three ingredients and the porosity of the flour (different for different types of flour and for winter and summer wheat), the result may be anything from a flakey mass to a viscous batter. Instructions may tell an agent what the intended result is, so that he or she can augment the amount of one ingredient or another, if the immediate result doesn't satisfy the intended description. (It is almost impossible to specify exact amounts in cooking, since individual variations in ingredients mean results are always somewhat unpredictable.) For example,

a. Mix the flour, butter and water, and ***knead*** until smooth and shiny.
b. Mix the flour, butter and water, and ***spread*** over the blueberries.
c. Mix the flour, butter and water, and ***stir*** until all lumps are gone.

Here the verbs "knead", "spread" and "stir" convey the expected viscosity of the resulting mixture.

3.3 Expectations about Locations

Because multi-clause instruction steps may evoke more than one situational context [56], part of an agent's cognitive task in understanding an instruction step is to determine that situation in which he is meant to find a referent for each of its referring expressions. This is the process of *grounding* referring expressions. Because actions can effect changes in the world or in what is visible to an agent, certain referring expressions in an instruction step might refer to the current world and what the agent is aware of within it, while other expressions might refer to objects that may only come into existence, or whose existence the agent may only become aware of, in the future.

To see this, compare the two instructions

a. Go into the kitchen to get me the coffee urn.
b. Go into the kitchen and wash out the coffee urn.

In the first instruction, "the coffee urn" will generally be taken to denote an urn currently in the kitchen, one that the agent, when given the instruction, may not even be aware of. When given this instruction (or similar ones such as "Open the box and hand me the yellow block", which one of the authors (Webber) has publically tested on several individuals), agents appear to develop an expectation that *after* they perform the action, they will be in a context in which it makes sense to try to ground the expression and determine its referent.

The second instruction ("...and wash out the coffee urn") is different: if the agent sees a coffee urn in the current context, prior to acting, he will happily ground the referring expression "the coffee urn" against that object. If he doesn't see a coffee urn prior to acting, he develops the same expectation as in the first example, that when he gets into the kitchen, he will be able to ground the expression then. The fact that agents will look around when they get to the kitchen if a coffee urn isn't immediately visible, opening cabinets until they find one, shows the strength of this expectation and the behavior it leads to.

This decision as to the context to use in grounding a referring expression is based on distinguishing the information (and assumptions) used to *resolve* a referring expression from that used to *ground* it. Reference *resolution* precedes reference *grounding* (cf. Figure 2) and involves using information from the interpretation of the current utterance (i.e., the explicit description), information from the previous discourse (i.e., the existence of a salient discourse entity with that description in the agent's discourse model) [26, 54, 55], and hypotheses about the intended relationship between actions.

To understand how AnimNL comes to different assumptions in using this information, we must first describe the Action Library (or *Action KB*) and the plan graph data structure.

3.3.1 The Action Library

AnimNL's Action Library contains simple plans that represent common sense knowledge about actions. Components of these plans are expressed in terms of Jackendoff's semantic primitives. To discuss the characteristics of these plans, we will refer to the *move*-action entry shown in Figure 3, which might be described as follows: go to where j is, get control over it, then take it to l[2]. Actions have a *header* and a *body*, where the relationship between the two is more constrained than found elsewhere: in particular, the complex of actions specified in the body *generates* the action specified in the header. (For further discussion of the *generation* relation, see [8, 17, 46].)

The representation does not employ preconditions, one reason being that we have found it difficult to draw the line between what is a precondition and what is part of the body of an action. One could say that *having control over* the object to be moved is a precondition for a *move*-action. However, if the object is heavy, the agent will start exerting force to lift it, and then carry it to the other location. It is not obvious whether the lifting action is still part of achieving the precondition, or already part of the body. (Other reasons for eschewing preconditions are given in Section 4.) Therefore, we do not have preconditions, but only actions which are substeps in executing another action, that is, they may belong to a body that generates a header. The *annotations* on the body specify the relations between the subactions such as partial temporal ordering and enablement.

From the planning tradition, we retain the notions of *qualifiers* and *effects*. Qualifiers are conditions that make an action relevant: for example, *unplug* x is relevant only if x is plugged in. (If qualifier conditions do not hold, an action is not considered further.)

[2]This do-it-yourself method is but one way to move something from where it is to somewhere else. Other methods would be listed separately in the Action Library.

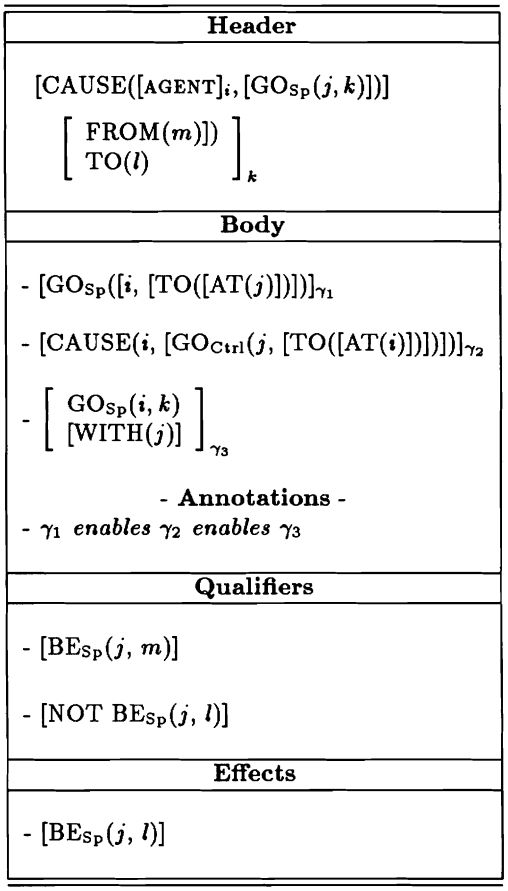


Figure 3: A *Move Something Somewhere* Action.

3.3.2 The plan graph

As mentioned in Section 2, the *plan graph* represents the structure of the intentions that the agent adopts as a response to the instructions. It keeps track of the goals the agent is pursuing, of the hierarchical relations between the goals and the actions whose execution achieves them, and of various relations between the actions. It also helps interpret the instructions that follow. In Figure 4, we show the plan graph structure built after interpreting the instruction "Go into the kitchen to get me the urn of coffee". (This will be further elaborated down to the level of annotated task actions during the course of processing.)

A node in a plan graph contains the Conceptual Structure representing an action, augmented with the consequent state achieved after the execution of that action[3]. The arcs

[3]In Figure 4 the labels on the nodes are only mnemonics, and not their real contents.

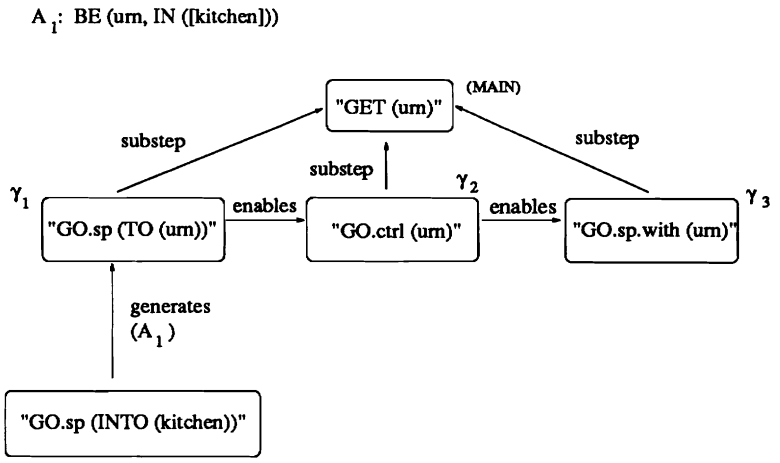


Figure 4: The Initial Plan Graph

represent relations between actions. Those relevant to our example in Figure 4 are: *enablement*, *generation*, and its generalization – *substep*, used when α belongs to a sequence of more than one action that generates β. (Temporal relations, such as *precedes*, are also used in the plan graph.)

There may also be assumptions associated with a plan graph, such as the expectation that the urn of coffee is in the other room. In Fig. 4, such assumption, A1, is shown associated with the arc relating the two actions "GO.sp(INTO(kitchen))" (from the NL input), and GO.sp(TO(urn)), which is a substep of "get".

The plan graph is built by an interpretation algorithm that takes as its input the logical form constructed by the parser. The algorithm works by keeping track of the *active nodes*, which include the goal currently in focus, and the nodes just added to the tree. The algorithm embodies various inference processes, that can be characterized either as *planning*—e.g. plan expansion, subgoaling— or as *plan inference*—e.g. inferring assumptions, inferring the more abstract goal some actions are supposed to achieve.

3.3.3 Computing Expectations

We will now show how the expectation of finding the urn in the kitchen is made while processing the instruction to hand, *Go into the kitchen to get me the coffee urn.*

The process begins with the following representation constructed by the parser, where the FOR-function (derived from the *to*-phrase) encodes the relation holding between the *go*-action α and the *get*-action β:

$$\left[\begin{array}{l} \mathrm{GO_{Sp}}([\textsc{agent}]_i, [\mathrm{TO}([\mathrm{IN}([\textsc{kitchen}])])]) \\ \mathrm{FOR}(\beta) \end{array}\right]_\alpha$$

$$[\mathrm{CAUSE}(i, [\mathrm{GO_{Sp}}([\textsc{urn-of-coffee}]_j, k)])]_\beta$$

$$\left[\begin{array}{l} \mathrm{FROM}([\mathrm{AT}(j)]) \\ \mathrm{TO}(me) \end{array}\right]_k$$

Given the presence of the *to* phrase (indicated as FOR(β)), we know that α may be part of a sequence of actions that generate β — see [17]. To pursue this hypothesis, we begin by looking up β in the action KB. β matches the general *move*-action shown in Figure 3 if the object to be moved j is bound to *the urn of coffee*:

$$j \quad = \quad [\textsc{urn-of-coffee}]$$

Next we try to match α with some subaction γ of β. α matches the first action γ_1 in β if we take m and [IN([KITCHEN])] to be the same place. m is a parameter that appears in the qualifier

$$[\mathrm{BE_{Sp}}(j,\ m)]$$

Substituting [IN([KITCHEN])] for m in the qualifier, the correct assumption is derived:

(1) $[\mathrm{BE_{Sp}}(j,\ [\mathrm{IN}([\textsc{kitchen}])])]$

Once the instruction is understood in this way, the two actions may be incorporated into the plan graph as shown in Fig. 4. (Of course, assumption (1) could be wrong: what the agent might find in the kitchen is someone who will get the coffee urn for him from some other location.)

The difference between this example and the second example *Go into the kitchen to wash the coffee urn*, where a currently visible urn would be taken as the most likely referent of "the coffee urn", comes out as follows:

As in the case of the *get*-action, the *go*-action would match the following subaction of *wash*:

$$[\mathrm{GO_{Sp}}([i,\ [\mathrm{TO(m)}]])]_\gamma$$

However, this time the relevant qualifier from the Action Library entry for *wash* would be

$$[\mathrm{BE_{Sp}}(\textsc{washing-materials},\ m)]$$

Therefore, the assumption (1) would not be derived, permitting the possibility of the urn being in the current room.

3.4 Summary

In this section, we have described some ways in which instructions provide agents with useful expectations. With experience, human agents internalize general expectations and do not have to rely on instructions for them. For AnimNL, this would mean incorporating expectations into its action schemata (Section 3.3.1) and using them in elaborating the plan graph. We hope to do this in the near future.

4 Expectations and Future Intentions

Here we discuss how the relative hospitality of the environment conspires with the agent's limited knowledge to both constrain and enhance the agent's ability to plan. Constraints come in the form of bounds on how far into the future the agent is prepared to look (Section 4.1), while enhancements come in the form of time available to the agent to examine his intentions and choose actions that will allow him to satisfy current goals while positioning himself to easily satisfy future goals (Section 4.2). This allows the agent to locally optimize his behavior and reduce the effort required to achieve his aims. Without this, we have found ourselves unable to achieve veridical behavior in our animations.

4.1 Bounding the Planning Horizon

As noted in Section 1, classical planning is insufficient for agents with limited knowledge. Agents cannot know all of the effects of an action until after it has been performed, and thus they cannot know all the additional actions that will be required to accomplish their goals.

One of the characteristics of hospitable environments is that they allow agents to have *expectations* about the results of actions, even though they cannot know for sure what will happen until the action is performed. However, the further ahead an agent looks, the fewer expectations he can allow himself to have (or the less confidence he can have in their reliability). This suggests that agents should not try to maintain a complete model of anticipated states of the world against which to plan. While an agent could maintain such a model, the model could not be guaranteed to be accurate enough to make planning from it worthwhile. The further into the future that a plan based on such a model extends, the more likely it is to contain errors that would render it useless.

This constraint is counter-balanced by agents' obvious need (or desire) to plan activities and thereby derive the benefits of having plans. As Bratman points out [11], there are simply too many possible actions that an agent could perform at any given moment for him to be constantly reconsidering what to do: plans and intentions provide part of the stability that agents need in order to interact with and react to the environment.

While agents use the stability of plans and intentions in order to act in a rational and coherent manner, limitations of their current and future knowledge make planning too far into the future unreasonable. Therefore in the AnimNL project, we have attempted to take a middle course between the extremes of not planning and complete planning, and instead plan incrementally.

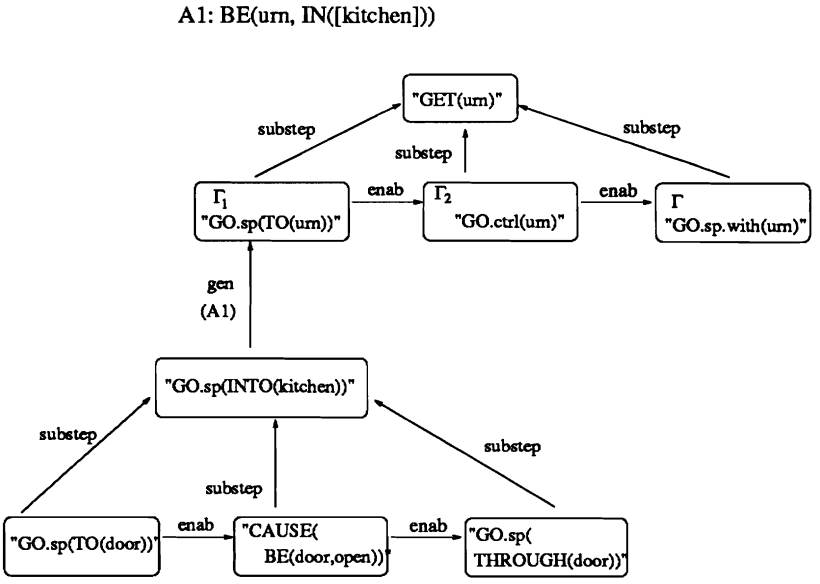


Figure 5: Expansion of Initial Plan Graph

In AnimNL, agents use situated reasoning in planning to achieve the intentions specified in the initial plan graph. The expansion of the plan graph is performed by the Intentional Planning System (ItPlanS), a hierarchical planner much along the lines of McDermott's NASL [39]. When the planning process reaches the level of leaf nodes (*annotated task actions*), ItPlanS must decide on a single action to be taken by the agent and commit to it. It then assumes that this action will succeed and its goal achieved, and goes on to compute the next action, using the same plan. ItPlanS then commits to performing this next action, assumes its success, and continues. (A complete description of ItPlanS can be found in [22]. Motivation for ItPlanS' approach to pre-conditions can be found in [23].)

To bound how far ahead ItPlanS continues this process, there must be a limit to the number of task actions that can be assumed to be successful. This is determined as follows: In considering the agent's next action, ItPlanS must identify the objects in the world that it will employ (described in Section 3 as *grounding* referring expressions). We have used limitations on the agent's ability to do this, to bound the planning process.

For commitment, all of the objects to be used in a task action must be bound to objects within the *active space* (cf. Section 2). (This is not true of higher-level actions, or we would not be able to handle instructions like "find the urn" or "go in the kitchen to get me the urn".) If the objects in question cannot be so bound, then the action cannot be committed to and the planning process stops. As the actions that have been committed to are executed, the active space changes (either its identity or its features), hopefully in such a way that it is eventually possible to bind the arguments of the actions and continue the planning process.

For example, consider our example of going into the kitchen to recover a coffee urn. The

initial plan graph (Figure 4) specifies going into the kitchen. Given the portal structure of the given environment, ItPlanS expands this intention into three substeps (Figure 5): going to the kitchen door, opening the door, and passing through the door. (If there was no door between the two rooms, or the door was not closed, this would not be necessary: the inital part of the agent path could be computed from the intention of going into the kitchen to get control of the urn.

"The kitchen door" in these substeps is a generic door object (suitable as a portal) that must be bound to something in the environment before the steps can be carried out. Since there is something describable as "the kitchen door" in the active space and hence visible to the agent, ItPlanS can commit to performing all these substeps. It now considers the next leaf node of the plan graph (Figure 5) — the leaf node labelled *GO.ctrl(urn)*. Unfortunately, there is no object that "the coffee urn" can be bound to in the active space; therefore, the agent cannot commit to performing the action. The planning process then halts and waits, pending a change in the portal space that allows the binding of the argument. Thus, the agent's limited knowledge bounds the planning process. Obviously this single method of using changes in the active space caused by opening known (visible) portals to bound the planning process, will not suffice for all planning environments, and in the future, we will be exploring how other limitations of the agent's knowledge will work similarly to halt the expansion of plans.

4.2 Taking Advantage of a Bounded Horizon

The relatively hospitality of the AnimNL environment allows agents time to improve their plans. However, as discussed in Section 4.1, the planning process is bounded, so improvements must be local in scope; they cannot rely on knowledge of the complete plan.

It is for this reason that ItPlanS has been designed to use a bounded "lookahead" in making planning decisions. This lookahead has been arbitrarily set at one plan step. That is, as the planner descends through the plan graph, it considers the next action in the structure (i.e., the right sister or aunt of the current node), as well as the current node. For example in Figure 5 when the planner is considering opening the door, it looks at the next step – passing through the door – to see if there is a possible improvement to be made by considering the two at the same time.

ItPlanS improves plans through two forms of what Pollack calls *intentional overloading* [47]. In intentional overloading, agents choose ways of achieving their intentions such that a single action will achieve multiple intentions. ItPlanS performs this kind of reasoning between the current intention being expanded and the next intention to be achieved.

ItPlanS recognizes two forms of intentional overloading. First, it notices conditions where two intentions can share a single action – that is, where the final action used to achieve one goal can be the same as the initial action used to achieve the next goal. Second, two intentions might have similar effects, so that, if the correct action is used, the effect of satisfying the first intention might also satisfy the second intention. For example, suppose a left-handed agent can only crush objects in his right hand. If this agent has the intention to pick up an object near either hand, he will do it with his left hand. On the other hand, if he has an intention of crushing an object, he will pick it up with his right hand. Now suppose, the agent has two separate intentions, first to pick up a beer can to his left-side

and then to crush it. An efficient agent will pick up the can with its right hand rather than its left, thus reducing the number of actions required.

Both of these kinds of intentional overloading are sought by ItPlanS and used whenever possible to improve agent behavior. Of course it is only because the environment is relatively benign that AnimNL agents are able to respond in this manner. If the environment demanded a faster reaction time, there would be less time available for computing these optimizations and they would occur less often.

5 Planning in Anticipation of Acquiring Knowledge

Work on reasoning about knowledge and action [42] has described knowledge preconditions for performing actions. In AnimNL, situations may arise in the process of carrying out instructions where the required knowledge is not available to the agent, such as in the case of going to the kitchen for the coffee urn: before carrying out this instruction, the agent does not know where (if anywhere) in the kitchen the urn will be. To address these situations, *search plans* can be used in the elaboration of the plan graph, to drive the agent's search for the needed knowledge. Successful execution of a search plan will provide the required knowledge and permit performance of the instructed behavior.

5.1 Background

Although considerable work has already been done on reasoning about knowledge and action by Moore [42], Morgenstern [43], Lesperance [35] and others, there is a problem still facing us in AnimNL – what to do *after* an action taken to acquire knowledge. If looking at some site L in the *search space* fails to reveal the location of the sought-after object, the agent needs to keep looking elsewhere until he finds it or gives up. Search plans in AnimNL avoid this problem by specifying a nondeterministic selection of what action to take after an informative action.

Looking elsewhere requires knowing where one has already looked, and so AnimNL agents are assumed able to keep track of their previous actions. We also assume that an object being sought will *remain where it is throughout a search.* With the agent's memory and this type of environmental feature, once a site is excluded as the object's location, the agent need never reconsider it. Thus if the search space is such that it will not be exhausted before the object is found and the object can be reached in a finite number of steps, an agent's search will eventually terminate successfully.

The next things to consider are how the notions of *search space*, the agent's *strategy* for probing it, and the relationship between the two are operationalized in AnimNL. As mentioned in Section 2, we assume that the three-dimensional space in which agents move is divided up into opaquely bounded sub-spaces connected by *portals* such as doors. An agent is assumed able to see, and thus know the location of, everything within his *active space.* The result is that agents only need to search outside the active space in order to find an object.

The problem is that the portal structure of an AnimNL environment is such that, as soon as a closed portal is introduced, an agent will know neither what *objects* lie beyond

the portal nor what what additional *spaces* there are. Thus the agent cannot make use of any search strategy that relies on global knowledge of the search space, as binary search does. It is only *local* structure that an agent can take advantage of. Currently, agents are assumed to use a strategy that minimizes major motion. Thus an agent will first explore those closed portal spaces (e.g., boxes, cabinets, etc.) that are currently within reach, and only walk to the next, nearest site of closed portal spaces if all reachable closed portal spaces have been probed (i.e., opened), and success has not yet been achieved. (We have not yet incorporated any "commonsense" checks on what spaces are explored, such as that the space be big enough to contain the object, or that it be a likely place for it.) This version of the "British Museum algorithm", i.e., exhaustive search, seems to mimic common human behavior in situations in which there is no probabilistic structure of the environment that makes certain sites more likely locations for the object being sought than others.

These ideas about search spaces and search strategies are embodied in the *search plans* invoked by ItPlanS (Section 4), when an agent is blocked from further intended action by not knowing the location of an object specified in the plan graph as a result of the given instruction step. (We have not yet considered the problem of object specifications being introduced into the plan graph through plan expansion, which turn out not to be groundable in the active space: the question is whether this should lead to a serarch for such an object or rejection of this expansion, followed by re-planning.

We now describe how search plans are structured and implemented in AnimNL.

5.2 Search Plans

Consider the problem of the agent locating the coffee urn in the kitchen after being told, "Go into the kitchen to get me the coffee urn." If the coffee urn is stored in a closed cabinet, the agent will be unable to proceed beyond entering the kitchen until he forms a search plan to locate the urn.

Figure 6 shows a search plan P for finding a coffee urn (urn). The predicates $go(e, c)$ and $open(f, c)$ are predicates on events (e and f) that constitute steps in exploring the search space. The variable c ranges over closed portals. The plan P consists of three disjuncts – the first, a success condition, the second, a failure condition, and the third, a continuation condition, to carry on with the search.

The success condition is that the urn is perceived to be in some location, x. When this condition holds, the plan continues with the action described here as win. Perceiving a predicate currently involves a test performed on the Object KB (cf. Figure 2) as to whether the specified arguments (which must be in the *active space*) stand in the specified relation. The subtleties of spatial predication noted by Herskovits [27] have not yet been incorporated into the current system.

The failure condition is that the search space is exhausted: there are no closed portals to spaces that have not not already been searched. Formally, this is expressed as the failure for there to exist a combination of two events (e, f) and a portal (c) such that e consists in a novel move of the agent to the portal and f consists in a novel opening of that portal. The requirement of novelty on these two events is expressed in the definitions of go and $open$ as the agent having no memory of doing such an event. When the failure condition holds, the plan proceeds to appropriate actions, specified here as $lose$.

$$P \iff \left(\begin{array}{l} (\exists x\ (\mathbf{Perceive}(in(urn, x)))\ ;\ win) \vee \\ (\neg\exists e, f, c\ (go(e, c)\ \wedge\ open(f, c))\ ;\ lose) \vee \\ \left(\begin{array}{l} \exists e, f, c \left(\begin{array}{l} (go(e, c)\ \wedge\ open(f, c)\ \wedge\ \neg\exists x\ (\mathbf{Perceive}(in(urn, x)))) \\ \mathbf{Occurs}(e)\ ;\ \mathbf{Occurs}(f) \end{array} \right) \end{array} \right) \ ;\ P \end{array} \right)$$

$$go(e, c) \stackrel{def}{=} \left(\begin{array}{l} move(e)\ \wedge\ agent(e, I)\ \wedge \\ destination(e, c)\ \wedge \\ portal(c)\ \wedge \\ \neg memory(\mathbf{Occurs}(e)) \end{array} \right)$$

$$open(e, c) \stackrel{def}{=} \left(\begin{array}{l} opening(e)\ \wedge\ agent(e, I)\ \wedge \\ patient(e, c)\ \wedge \\ portal(c)\ \wedge \\ connects(c, here, d)\ \wedge \\ subspace(d, kitchen)\ \wedge \\ \neg memory(\mathbf{Occurs}(e)) \end{array} \right)$$

Figure 6: Example Search Plan

When the search is incomplete (neither success nor failure hold), then a sequence of novel *go* and *open* actions is taken and the search plan repeats. The expression **Occurs**(e) in P is an indexical version of Allen's [2] relation between an event and a time, interpreted as event e occurring now. The occurrence of e and f is followed in sequence by P, which is equivalent to the entire search plan. This restatement of the search plan indicates that novel actions will be selected to explore the search space until the search space is exhausted or the urn is found.

This use of search plans in AnimNL is a somewhat simplified implementation of a more formal theory of search plans being developed by co-author Moore and described in more detail in [41].

5.3 Conducting a search

Search plans are developed in response to an inability of the agent to act. In AnimNL, this is detected when the planner has produced a task action which is blocked from commitment by the planner's inability to find a binding for one (or more) of its object descriptions to objects in the active space. Without such objects, the agentd cannot commit to and perform the action. So instead, a search is undertaken to locate the required objects.

A search plan for finding an object mentioned in the current instruction step is constructed from information obtained during the process of reference resolution. Reference resolution provides a description of the object and suggests the space in which that object may be located 3. The object description is used to construct the condition for successful termination of the search. In the example above, the object description consists of the simple predicate *urn*. (We are ignoring the well-known semantic problems of noun–noun

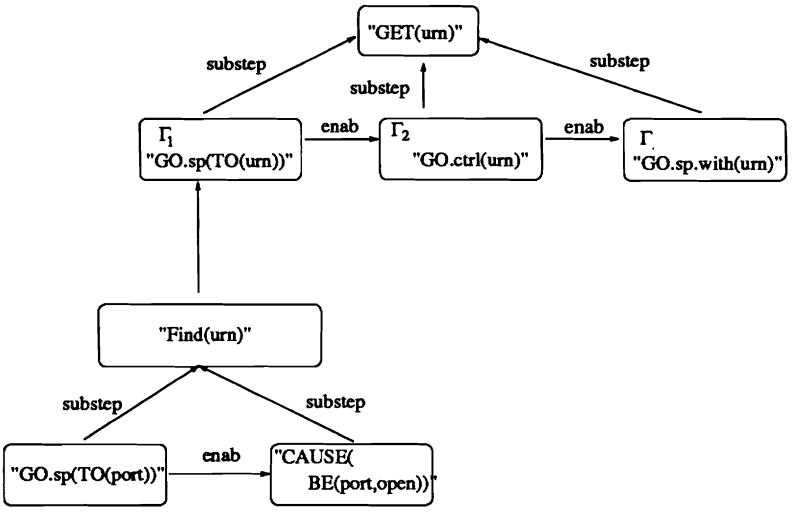


Figure 7: Example of a Plan Graph expanded with a Search Plan

modification by treating coffee-urn as a simple noun.) The portal space description is used to constrain the search space. In this case, the instructions only inform us that the urn is in the kitchen. Inference is required to elaborate a plan to search in the kitchen into a plan to search the closed spaces in the kitchen. These two pieces of information constitute a sufficient description of the search plan and are communicated to the planner, which constructs a *find* node, which is then elaborated with a search plan (cf. Figure 7).

From the standpoint of ItPlanS, search plans are treated in the same way as any other high level action, with the exception that, the first expansion of the find intention is not performed by ItPlanS but rather by the search plan system. An action or sequence of actions is selected which has the search plan success condition as one of its nondeterministic effects. In effect, the planner constructs a plan from the current state to the search plan success condition. The process of search plan elaboration is informed by facts about the world which the agent perceives and the agents internal memory. Details about this model of interactions between perception and memory can be found in [41].

Consider our previous example. Once the agent has entered the kitchen the intention to be in the kitchen has been satisfied. However the agent still has an intention to "GO.sp(TO(urn)", that is, the agent has an intention to be at the urn. If the agent is not at the urn and cannot see the urn within its portal, a find node is inserted into the plan graph. Immediately, the search planning system is called upon to provide an expansion for this node. This expansion is then placed in the plan graph (Figure 7). Since actions used in a search plan resemble others found in the plan graph, ItPlanS may be required to carry out further plan expansion on them, including calls to object–specific reasoning (Section 6).

The plan in this example would call for going to the nearest unopened portal and opening it. This process would continue until the urn is found or there are no more portals to search.

6 Geometric and Functional Planning

6.1 Introduction

Instructions may tell an agent to open a door, but how he actually moves in response will depend on his own physical abilities (Section 7), the geometry of the door and his reason for opening it. For example, he will move differently to open a sliding door than to open a hinged door, to move through the doorway than to let someone else through.

The AnimNL system currently reflects differences in agent behavior arising from differences in object geometry through what we call an *Object Specific Reasoner* (OSR). This OSR bridges the gap between task action commands and *action directives* formulated in terms of known agent behaviors (Section 7). This transformation currently takes account of

1. Task action definitions
2. Object type information
3. Object token Information

Eventually, the OSR will be elaborated to take account of the agent's current intentions as well. This section describes how Object Specific Reasoning is currently performed.

6.2 Task Action Definitions

The OSR receives a committed task action (e.g., **open(door,gothru)**) as input. In order to take account of variations in behavior due to object geometry, each task action is defined in terms of an *action outline* – an underspecified list of all the steps that the agent *might* have to perform, in carrying out the action on an object. The entries in the action outline are either available simulated behaviors (**reach arm**) or task actions (**move object**).

For example, to open an object, the object or a part of it must move. This requires that the agent must have control over the moving part, and that the moving part has no constraints restricting its movement. The action outline for **open** is shown in Figure 8. Outline entries that are themselves task actions will be further expanded. Entries that are irrelevant to the particular object or type of object will be eliminated as the process proceeds.

```
open (obj):   get-control-of obj-part
              break-init-mvmt-constraints obj-part
              move obj-part
```

Figure 8: Action Outline for "open"

open (BARROOM-DOOR): get-control-of DoorPanel
rotate DoorPanel

Figure 9: Partial Plan for Opening a Barroom Door

6.3 Object Type information

Object Type information is categorical information about artifacts in the agent's world. It is organized into a hierarchical, object-oriented database to allow attribute inheritance from higher-level types to lower-level types. For example, SLIDING DOOR and SWINGING DOOR are types of DOOR. They inherit from DOOR the attribute of a movable part. However, their degree of freedom attribute is only specified at this lower level – a SLIDING DOOR will *translate*, while a SWINGING DOOR will *rotate*.

Object Type information is used to convert the action outline to a *partial plan*. An object's attributes lead to elimination, refinement or expansion of steps in the action outline. For example, BARROOM DOOR is a type of SWINGING DOOR, with no constraints on its initial movement. For this type of door, "move obj-part" in the action outline for open will be refined to "rotate DoorPanel", while the "break-init-mvmt-constraints" step will be eliminated. This is shown in Figure 9. (DoorPanel is part of a DoorUnit assembly, comprising a DoorFrame, a DoorPanel and a Lock. A DoorPanel has a subpart Handle.) Note that this structure allows the agent to be told to "go into the kitchen", without having to be told whether or what kinds of door-opening actions might be needed.

6.4 Object Token Information

By virtue of considering an action outline (associated with a task action) in conjunction with the object type information (of its current arguments), an agent will have a partial plan for opening a door. This partial plan can only be further specified by considering the specific instance of door that the agent will open: e.g., the door in front of him — a swinging door hinged on the left, that rotates clockwise around its y-axis, with a handle, but no lock. The agent can only know such information if the door is in the current *active space*. Otherwise, this elaboration of the partial plan will have to wait. In the example case, the plan for opening the door will be as shown below in Figure 10.

(Note that in addition to this general knowledge of actions and objects, one more factor should be taken into account in deciding how an agent should move in carrying out some task action: that is the *intention* associated with the task action, as we indicated earlier in Section 2. While the mapping to action directives does not yet reflect agent intention, it is contained in the task action annotation for use in the future.)

open (KitchenDoor6): grasp Knob67
rotate-cw DoorPanel68

Figure 10: Plan for Opening a Particular Kitchen Door

6.5 Mapping to Action Directives

The Object Specific Reasoner converts the task actions output by the Plan Graph into a set of action directives which are passed on to the simulator. These directives comprise a set of parallel behaviors for the agent to perform. (The issue of determining their duration is discussed is [36].) Because the simulator resolves symbolic names, action directives need only be described in terms of symbolic sites.

The final step of the algorithm is a mapping, via a lookup table, from action(object, intention) terms to <action-site, grasp-site, grip-site, focal-pt> tuples. This information generates the action directives. We cannot guarantee that this set of action directives will be optimal, because we have no theory of when particular low-level actions are unnecessary. For example, a human agent may not look at a light switch when turning it on or off. We currently do not know how to avoid doing so.

6.6 Summary

The OSR can be seen as a hybrid system, incorporating elements of both classical and reactive planning. We use action outlines and partial plans to limit the number of possible plans, but final details – the agent's missing knowledge – is filled in only at the last moment, as the motions are about to be performed. This appears to be a useful model of how an agent can operate in a world of which he has only partial knowledge.

7 Veridical Simulation

The production of action directives by the OSR signifies that the agent has the underlying abilities to carry out the instruction. In a hospitable environment there ought not be any unexpected impediments to progress, but there are still significant motor tasks and decisions to be made. Action directives are formed into parallel inputs to the *Jack* simulation system, which uses a biomechanically reasonable human model as its execution agent [40]. Rather than trivializing the actions into mere state changes, *Jack* must use its repertoire of behaviors to perform the required set of motions. Since the input specification is still at a qualitative, symbolic level, *Jack* must resolve this specification against the exact geometry of the environment and the agent's body size and capabilities. The behavioral vocabulary accessible to the agent includes (but is not at all limited to) walking, stepping, looking, reaching, turning, grasping, strength-based lifting, and collision-avoidance posture planning. Each of these behaviors is environmentally reactive in the sense that *incremental computations during the simulation are able to adjust the agent's performance to the situation without further involvement of the higher level AnimNL processes* unless an exceptional failure condition is signaled.

Recent research in autonomous agent construction and computer graphics animation has found that a control architecture with networks of functional behaviors is far more successful for accomplishing real-world tasks than traditional methods. Like Brooks [13], our approach couples sensors[4], effectors, and behaviors such that independent behaviors interact to achieve

[4] "Sensor" is a generalized term meaning a probe or query into the geometric and symbolic attributes attached to object models.

more sophisticated *emergent behavior*. Such an approach has been proposed under a variety of names including: *subsumption architecture* [12], *reactive planning* [24, 30], *situated activity* [1], and others. Of particular interest to us, however, are those motivated explicitly by animal behavior: *new AI* [13], *emergent reflexive behavior* [4], and *computational neuroethology* [10]. The individual behaviors have the following properties:

- they are grounded in perception.
- they normally participate in directing an agent's effectors.
- they may attempt to activate or deactivate one-another.
- each behavior by itself performs some task useful to the agent.

In the AnimNL world, the agent's behavioral repertoire is programmed rather than learned, but the result is nonetheless a reasonable simulation of the action directives.

If given a **go-to** action directive with appropriate qualifiers, locomotion behavior will be invoked [7, 33]. Typically, the goal action site differs from the present position of the agent, so the locomotion behavior must first determine if the agent needs to walk or step at all. For example, there is no need to stand up to answer the phone if it is within reach. If a goal action site is not immediately reachable then a posture change behavior, e.g. from sitting to standing, may be invoked and simulated first. Proceeding incrementally, the agent re-orients toward the action site and begins one or more steps that carry the body forward. Final re-orientation steps are computed at the action site, to accommodate the goal direction since it may differ from the tangent to the walking path.

During locomotion, if the action site is several steps away, a locally linear path is attempted. If there are obstacles present, they are sensedf by measuring their distance from the agent. Distance-sensitive behaviors to "attract" or "repel" the agent are invoked: the former can keep the agent on a generalized path (such as a sidewalk), while the latter can avoid obstacles. Curved walking paths with appropriate banking and turning motor actions result; pre-computed motion paths are not needed. (In a hospitable environment, we assume that perverse "dead ends" and local minima in the distance field are not going to stop the agent's progress. Were this to happen, control might have to revert back to the planning level to compute alternative actions.) The agent's motion is purposeful, biomechanically reasonable, and environmentally responsive.

The "look-at," "touch," "turn," and "grasp" action directives are achieved by *Jack*'s inverse kinematics procedure. The approach to the target site is mediated by *Jack*'s inverse kinematics procedure which manages goal achievement and satisfies other constraints [44]. The "grasp" action in addition invokes a hand motion behavior that takes into account the current world location and orientation of the grasped site. The hand itself closes around the object in a context-sensitive fashion: the grip type is a function of the intention, the approach is a function of the task, and the fingers close around the object until contacts are detected.

If lifting an object is required, *Jack* uses a behavior that takes into account the goal position and the mass of the lifted object. As with locomotion, the motion path is not predetermined; rather the local direction of motion is dependent upon the strength available to the agent's joints and the discomfort level to be tolerated [34].

One of the most difficult problems facing the simulated agent is something we call "posture planning." Given a reach goal and an action site, the actual geometry of the situation forces postural choices on the agent that result in achieving the required goals while simultaneously avoiding negative situations such as collisions with nearby objects. (Failures at this level must be propagated back to higher levels as task failures.) Though analogous to the curved path obstacle avoidance problem, the issue here is compounded by both the inherent three-dimensionality of the task and the articulation of the body itself. With 88 degrees of freedom in the standard *Jack* human figure, classical configuration state approaches [38] to the collision avoidance problem are simply intractable. We are currently solving this problem in two ways: one method groups degrees of freedom into meaningful chunks to reduce the inherent exponential complexity to humanly-tolerable time scales [15]; the other method uses qualitative reasoning to "think about" potential postures and their consequences [28]. In either case, stepping and turning adjustments may be generated that move the agent away from the supplied action site.

The simulator is described in more detail in [9].

8 Conclusion

AnimNL did not begin as a planning project, or a study of rational agency, or a study in agent-environment interaction. But in trying to link everyday instructions to the animation of reasonable human behavior, it has had to become all three. Thus the focus so far has been on vertical integration, rather than on broad coverage of instruction forms or types of actions. No doubt the course of broadening AnimNL's coverage will reveal other new and interesting cases that await discovery.

References

[1] Agre, P. and Chapman, D. Pengi: An Implementation of a Theory of Activity, *Proceedings of the AAAI-87 Conference*, June 1987, pp. 268–272.

[2] Allen, J. Towards a general theory of action and time. *Artificial Intelligence*, 23:123–154, 1984.

[3] Alterman, R., Zito-Wolf, R. and Carpenter, T.. Interaction, Comprehension, and Instruction Usage. Dept. of Computer Science, Center for Complex Systems, Brandeis University. 1991, CS-91-161.

[4] Anderson, T.L. and Donath, M. Animal behavior as a paradigm for developing robot autonomy, In Pattie Maes (ed.), *Designing Autonomous Agents*, Cambridge: MIT Press, 1990, pp. 145–168.

[5] Badler, N. and Webber, B. Task Communication through Natural Language and Graphics: Workshop Report. *AI Magazine* 11(5), Winter 1990, pp. 71-73.

[6] Badler, N., Webber, B., Kalita, J. and Esakov, J. Animation from Instructions. In N. Badler, B. Barsky and D. Zeltzer (eds.), *Making Them Move: Mechanics, Control and*

Animation of Articulated Figures, Los Altos CA: Morgan Kaufmann Publishers, 1990, pp.51-93.

[7] Badler, N., Phillips, C. and Webber, B. *Simulating Humans: Computer Graphics, Animation and Control.* New York: Oxford University Press, 1993.

[8] Balkanski, C. *Modelling act-type relations in collaborative activity.* Technical Report TR-23-90, Center for Research in Computing Technology, Harvard University, 1990.

[9] Becket, W. and Badler, N. I. *Integrated Behavioral Agent Architecture. Proc. Workshop on Computer Generated Forces and Behavior Representation*, Orlando, FL, March 1993.

[10] Beer, R.D., Chiel, H.J. and Sterling, L.S. A biological perspective on autonomous agent design. In Pattie Maes (ed.), *Designing Autonomous Agents*, Cambridge: MIT Press, 1990, pp. 169–186.

[11] Bratman, M. *Intentions, Plans and Practical Reason.* Cambridge: Harvard University Press, 1987.

[12] Brooks, R. A robust layered control system for a mobile robot. *Robotics Journal*, April 1986, pp. 14-23.

[13] Brooks, R. Elephants don't play chess. In Pattie Maes (ed.), *Designing Autonomous Agents*, Cambridge: MIT Press, 1990, pp. 3–18.

[14] Chapman, D. *Vision, Instruction and Action.* Cambridge: MIT Press, 1991.

[15] Ching, W. and Badler, N.I. Fast motion planning for anthropometric figures with many degrees of freedom. *IEEE Intl. Conf. on Robotics and Automation* May 1992.

[16] Cohen, P.R. and Levesque, H. Persistence, intention, and commitment. In M. Georgeff and A. Lansky (eds.), *Reasoning about Actions and Plans, Proceedings of the 1986 Workshop.* Los Altos CA: Morgan Kaufmann, 1986, pp. 297–340.

[17] Di Eugenio, B. Understanding Natural Language Instructions: the Case of Purpose Clauses. *Proc. 30th Annual Conference of the Assoc. for Computational Linguistics*, Newark DL, June 1992.

[18] Di Eugenio, B. A Study of Negation in Instructions. *Penn Review of Linguistics*, Vol. 17, 1993

[19] Di Eugenio, B. Speaker's Intentions and Beliefs in Negative Imperatives. *Proceedings of the ACL Workshop on Intentionality and Structure in Discourse Relations*, Columbus, Ohio, 1993

[20] Di Eugenio, B. and Webber, B. Plan Recognition in Understanding Instructions. *Proc. 1st. Int'l Conference on Artificial Intelligence Planning Systems*, College Park MD, June 1992.

[21] Di Eugenio, B. and White, M. On the Interpretation of Natural Language Instructions. *1992 Int. Conf. on Computational Linguistics* (COLING-92), Nantes, France, July 1992.

[22] Geib, C. *Intentions in Means/End Planning.* Technical Report MS-CIS-92-73, Dept. Computer & Information Science, University of Pennsylvania, 1992.

[23] Geib, C. A Consequence on Incorporating Intentions in Means-End Planning. In *AAAI Spring Symposium Series: Foundations of Automatic Planning: The Classical Approach and Beyond*, Working Notes. Stanford CA, March 1993.

[24] Georgeff, M. and Lansky, A. Reactive Reasoning and Planning. In J. Allen, J. Hendler and A. Tate (eds.), *Readings in Planning.* Los Altos: Morgon Kaufmann Publishers, 1990, pp. 729–734.

[25] Grosz, B. and Sidner, C. Plans for Discourse. In P. Cohen, J. Morgan & M. Pollack, *Intentions in Communication.* Cambridge MA: MIT Press, 1990.

[26] Heim, I. *The Semantics of Definite and Indefinite Noun Phrases.* PhD dissertation, University of Massachusetts, Amherst MA, 1982.

[27] Herskovits, A. *Language and Spatial Cognition.* Cambridge: Cambridge University Press, 1986.

[28] Jung, M. *Human Body Motion Planning in Work Spaces.* PhD Dissertation, Computer and Information Science, University of Pennslvania, Philadelphia PA, 1992 (to appear).

[29] Jackendoff, R. *Semantic Structures.* Cambridge MA: MIT Press, 1990.

[30] Kaelbling, L.P. An Architecture for Intelligent Reactive Systems, In J. Allen, J. Hendler and A. Tate (eds.), *Readings in Planning.* Los Altos: Morgon Kaufmann Publishers, 1990, pp. 713–728.

[31] Karlin, R. Defining the Semantics of Verbal Modifiers in the Domain of Cooking Tasks. *Proc. 26th Annual Meeting, Association for Computational Linguistics*, SUNY Buffalo, June 1988, pp. 61-67.

[32] Karlin, R. *SEAFACT: Semantic analysis for animation of cooking tasks.* Technical Report MS-CIS-88-04, Dept of Computer & Information Science, Univ of Pennsylvania, January 1988.

[33] Ko, H. and Badler, N.I. Human locomotion on a curved path. Technical Report, Dept. of Computer and Information Science, University of Pennsylvania, Philadelphia PA, 1991.

[34] Lee, P., Wei, S., Zhao, J. and Badler, N.I. Strength Guided Motion. *Computer Graphics* 24(4), 1990, pp. 253-262.

[35] Lesperance, Y. *A Formal Theory of Indexical Knowledge and Action.* PhD thesis, Computer Systems Research Institute, University of Toronto, 1991. (Also Technical Report CSRI-248.)

[36] Levison, L. Action Composition for the Animation of Natural Language Instructions. Technical Report, MS-CIS-91-28, Dept of Computer & Information Science, University Of Pennsylvania, August, 1991.

[37] Lochbaum, K. An Algorithm for Plan Recognition in Collaborative Discourse. *Proc. 29th Annual Meeting, Association for Computational Linguistics*, Berkeley CA, June 1991, pp. 33-38.

[38] Lozano-Perez, T. Spatial Planning: A Configuration Space Approach. *IEEE Transactions on Computers* c-32(2), 1983, pp. 26–37.

[39] McDermott, D. Planning and Acting. *Cognitive Science* 2, 1978, pp. 71-109.

[40] Monheit, G. and Badler, N. A Kinematic Model of the Human Spine and Torso. *IEEE Computer Graphics and Applications* 11(2), 1991, pp. 29-38.

[41] Moore, M.B. *Search Plans.* PhD Dissertation proposal, Department of Computer and Information Science, University of Pennsylvania, May 1993.

[42] Moore, R.C. A formal theory of knowledge and action. In R.C. Moore and J. Hobbs (eds.), *Formal Theories of the Commonsense World.* Ablex Publishing, Norwood NJ, 1984.

[43] Morgenstern, L. Knowledge preconditions for actions and plans. *Proceedings IJCAI*, pages 867–874, Milano, Italy, 1987.

[44] Phillips, C., Zhao, J. and Badler, N.I. Interactive Real-Time Articulated Figure Manipulation using Multiple Kinematic Constraints. *Computer Graphics* 24(2), 1990, pp. 245-250.

[45] Phillips, C. and Badler, N. Interactive Behaviors for Bipedal Articulated Figures, *Computer Graphics* 25(4), 1991, pp. 359-362.

[46] Pollack, M. *Inferring domain plans in question-answering.* PhD thesis, Department of Computer & Information Science, Technical Report MS-CIS-86-40. University of Pennsylvania, 1986.

[47] Pollack, M. Overloading Intentions for Efficient Practical Reasoning. *Nous* 25, pp.513-536, 1991

[48] Renault, O., Magnenat-Thalmann, N. and Thalmann, D. A vision-based approach to behavioral animation. *J. Visualization and Computer Animation* 1(1), 1990, pp. 18–21.

[49] Schriver, K. Plain Language through Protocol-aided Revision. In E. R. Steinber g (ed.), *Plain Language: Principles and Practice.* Detroit MI: Wayne State University Press, 1991, pp. 148-172.

[50] Schriver, K. Teaching Writers to Anticipate Readers Needs: A Classroom-Evaluated Pedagogy. *Written Communication*, 9(2), 1992, 179-208.

[51] Sidner, C. Using Discourse to Negotiate in Collaborative Activity: An artificial language. *Proc. AAAI Workshop on Cooperation among Heterogeneous Agents*, San Jose CA, July 1992.

[52] Steedman, M. Gapping as Constituent Coordination. *Linguistics and Philosophy* 13, 1990, pp. 207-263.

[53] Vere, S. and Bickmore, T. A basic agent. *Computational Intelligence.* 1990, 6, pp. 41-60.

[54] Webber, B. Discourse Model Synthesis: Preliminaries to Reference. In A. Joshi, B. Webber & I. Sag (eds.), *Elements of Discourse Understanding*, Cambridge: Cambridge University Press, 1981.

[55] Webber, B. So What Can We Talk about Now? In M. Brady and B. Berwick (eds.) *Computational Models of Discourse*, Cambridge MA: MIT Press, 1983, pp. 331-371.

[56] Webber, B. and Baldwin, B. Accommodating Context Change. *Proc. 30th Annual Conference of the Assoc. for Computational Linguistics*, Newark DL, June 1992.

[57] Webber, B. and Di Eugenio, B. Free Adjuncts in Natural Language Instructions. *Thirteenth International Conference on Computational Linguistics (COLING)*, Helsinki Finland, August 1990, pp. 395-400.

[58] Zwarts, J. and Verkuyl, H. An Algebra of Conceptual Structure: an investigation into Jackendoff's conceptual semantics. Accepted for publication in *Linguistics and Philosophy*, 1991.