The Institute For Research In Cognitive Science

Understanding Natural Language Instructions: A Computational Approach to Purpose Clauses (Ph.D. Dissertation)

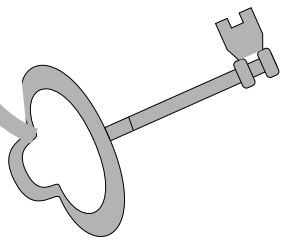
by

Barbara Di Eugenio

University of Pennsylvania
Philadelphia, PA 19104-6228

December 1993

Site of the NSF Science and Technology Center for Research in Cognitive Science



P E N N

University of Pennsylvania
Founded by Benjamin Franklin in 1740

IRCS Report 93-52

UNDERSTANDING NATURAL LANGUAGE INSTRUCTIONS: A COMPUTATIONAL APPROACH TO PURPOSE CLAUSES

Barbara Di Eugenio

A Dissertation
in
Computer and Information Science

Presented to the Faculties of the University of Pennsylvania
in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy

1993

Ai miei genitori, Mario e Adelaide

Acknowledgements

First and foremost, my thanks go to my advisor, Bonnie Lynn Webber: her enthusiasm, her ability to highlight the important issues, her knowledgeability, her support, have been invaluable to me.

The members of my committee each provided the right input at the right moment: Aravind Joshi and Mark Steedman provided the necessary feedback on formal issues; Ellen Prince taught me that you have to look at real data and gave me the instruments to analyze them; James Allen asked the right questions about the Knowledge Representation formalism.

Heartfelt thanks go to the NLP faculty at Penn: besides Bonnie Webber, I would like to thank Aravind Joshi, Mitch Marcus and Mark Steedman in Computer and Information Science, Mark Liberman, Tony Kroch and Ellen Prince in Linguistics. They lead a Computational Linguistics group that is supportive of students both from a scientific and a personal point of view, a real community to which I feel privileged I belonged.

My ideas greatly profited from the weekly meetings of *AnimNL*, the *Animation from Natural Language Instructions* project, directed by Bonnie Webber and Norm Badler. Among the members of the group, I would like to mention first of all Norm Badler, who keeps all of us Computational Linguists with our feet to the ground of producing real animations, and then Chris Geib, Libby Levison, Mike Moore, and Mike White.

Other insights came at different times from Ellen Hayes, Michael Niv, Massimo Poesio, Martha Pollack, Marcel Schoppers, Donia Scott, Rich Thomason, Lyn Walker, professors and students in the Natural Language group in Torino, Italy, and members of the research groups directed by Wolfgang Wahlster in Saarbrücken, Germany. I had other valuable discussions, whether on my thesis or on various Computational Linguistics issues, with Breck Baldwin, Dan Hardt, Jamie Henderson, Beryl Hoffman, Christine Nakatani, Charlie Ortiz, Ron Rymon, Robert Rubinoff, and many other members of the CLiFF group. The financial support for my research was provided by grants DARPA no. N00014-90-J-1863, ARO no. DAAL 03-89-C-0031, NSF no. IRI 90-16592, and Ben Franklin no. 91S.3078C-1.

During my time at Penn the IRCS (Institute for Research in Cognitive Science) was established, under the direction of Aravind Joshi and Lila Gleitman. The exposure I've had to Cognitive Science through the many visitors to IRCS and the activities IRCS organizes is invaluable. Besides, IRCS provided an office space that was ideal, as it allowed students to have a common space in which to discuss Computational Linguistics and Cognitive Science issues, among themselves and with postdocs and visitors. Through these discussions, I have also found some of my closest friends.

There are so many people I would like to thank for their friendship and support over the years: Caroline Heycock, Megan Moser, Lyn Walker, Raffaella Zanuttini, who shared with me so many happy and unhappy moments; Jana Košecká, with whom it was so pleasant to watch late night TV and discuss what had happened during the day; Miloš Žefran, who shared with me that last stressful but also very rewarding phase of writing a thesis; those from "3401" who were companions in deep discussions and chit chat, lunches and dinners, movies and laughter, first of all Michael Niv and Giorgio Satta, and then David Bennett, Anuj Dawar, Bob Frank, Young-Suk Lee, Sandeep Prasada, Owen Rambow, Phil Resnik, Ramesh Subrahmanyam. Moving down to the third floor of our building, the "graspees" Luca Bogoni, Ulf Cahn von Seelen, Jim Gee and Sang-Wook Lee always kept good company. A bit further away, in New Jersey, Dawn Cohen and Kumar Vadaparty. At Rochester, my old pal Massimo Poesio, with whom I must have been discussing Computational Linguistics and the rest for almost ten years, both in Italy and in the States, and whose pointed questions on my work have always helped me. And moving even further away, among my friends in Italy a special thanks goes to my two faithful pen pals, Felice Cardone and Celeste Gallo, who have been so close to me, whether by e-mail or by air-mail.

I would also like to thank professors and students at the Dipartimento di Informatica in Torino, Italy: Pietro Torasso, who first taught me about Computational Linguistics; Leonardo Lesmo, who leads the Torino NLP group with good humor and insight; Paolo Terenziani and Enzo Lombardo, who often brought me a taste of home with their visits to Philadelphia.

And last but certainly not least, my family: my parents, Mario and Adelaide, and my sister Cristina. Their love and support (and my sister's humorous but affectionate remarks) never failed me, even with an ocean in between, and even if probably they didn't image that I would move so far away from home and for such a long time when they suggested, "Why don't you go and spend some time in the US?"

Abstract

Human agents are extremely flexible in dealing with Natural Language instructions. I argue that most instructions don't exactly mirror the agent's knowledge, but are understood by *accommodating* them in the context of the general plan the agent is considering; the accommodation process is guided by the *goal(s)* that the agent is trying to achieve. Therefore a NL system which interprets instructions must be able to recognize and/or hypothesize goals; it must make use of a flexible knowledge representation system, able to support the specialized inferences necessary to deal with input action descriptions that do not exactly match the stored knowledge.

The data that support my claim are Purpose Clauses (PCs), infinitival constructions as in *Do* α *to do* β, and Negative Imperatives. I present a pragmatic analysis of both PCs and Negative Imperatives. Furthermore, I analyze the computational consequences of PCs, in terms of the relations between actions PCs express, and of the inferences an agent has to perform to understand PCs.
I propose an action representation formalism that provides the required flexibility. It has two components. The *Terminological Box* (TBox) encodes *linguistic* knowledge about actions, and is expressed by means of the hybrid system CLASSIC [Brachman *et al.*, 1991].

To guarantee that the primitives of the representation are linguistically motivated, I derive them from Jackendoff's work on Conceptual Structures [1983; 1990]. The Action Library encodes *planning* knowledge about actions. The action terms used in the plans are those defined in the TBox.

Finally, I present an algorithm that implements inferences necessary to understand *Do* α *to do* β, and supported by the formalism I propose. In particular, I show how the TBox classifier is used to infer whether α can be assumed to match one of the substeps in the plan for β, and how expectations necessary for the match to hold are computed.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Thesis Statement

Consider an agent, whether human or artificial, faced with the following two possible instructions:

(1.1a) *Place a plank between two ladders.*

(1.1b) *Place a plank between two ladders to create a simple scaffold.*

In both (1.1a) and (1.1b), the action to be executed is *place a plank between two ladders*. However, (1.1a) would be correctly interpreted by placing the plank *anywhere* between the two ladders: this shows that in (1.1b) the agent must be inferring the proper position for the plank from the expressed goal *to create a simple scaffold*. How is this further constraint on *place a plank between two ladders* computed? What is the relation between the action described in input and its more constrained version? The problem I address in this thesis is how constraints on actions arising in interpreting complex action descriptions are computed.

My primary claim is that:

Most instructions, especially complex ones, don't exactly mirror the agent's knowledge, but are understood by accommodating them in the context of the general plan the agent is considering; the agent's accommodation process is guided by the goal(s) that s/he is trying to achieve. The concept of goal itself is pervasive in NL instructions: a NL system which interprets instructions must be able to recognize and/or hypothesize goals; it must make use of a flexible knowledge representation system, that facilitates computing the description of the action to be performed; and it may need to execute some specialized inferences to perform such computation.

- To show that my claim is *justified*, I will illustrate my views on the agent's inference processes by means of the analysis of naturally occurring data, *purpose clauses* and *negative imperatives*.

I will also show how the inferences necessary to understand such data are directed by the notion of *goal*.

- To show that my claim is *valid*, I will present a computational model of instruction understanding composed of a representation formalism that addresses some of the requirements posed by the need to represent both *linguistic* and *planning* knowledge about actions. Such formalism provides support to perform the inferences I will discuss.

A caveat to the reader before proceeding: claiming that agents' goals pay an important role in understanding NL text is not very novel, as it has long been recognized in work on plan inference [Schank and Abelson, 1977; Wilensky, 1983; Allen and Perrault, 1980; Litman and Allen, 1990]. However, in such work goals are generally used to infer actions or motivations not mentioned in input (e.g. inferring that *John was hungry* from *John went to a restaurant*), or to disambiguate between different "global" interpretations, such as whether *Harry carried a gun* in order to *rob the bank* or *go hunting*; however, the problem of inferring how features of the action to be executed are affected by the surface form of the input has not been addressed. Consider:

(1.2) *Open the box and hand me the red block.*

I suspect that none of the systems above would be able to generate the expectation associated with (1.2), namely, that the red block is in the box: it is the interaction between the surface form and the knowledge about *open* that gives rise to such expectation.

Turning now to systems that interpret and execute NL instructions, they have not given agents' goals as much prominence as plan inference work has; when goals are used, they are considered as global constraints on permissible activities [Vere and Bickmore, 1990], or as guide for repair when a known plan for action fails [Alterman *et al.*, 1991], rather than as originating constraints on the interpretation and refinement of surface forms. In general, I claim that the model proposed by researchers such as Winograd [1972] or Vere and Bickmore [1990] is not sufficient to compute constraints deriving from complex instructions. In such model, the logical form is mapped onto planning knowledge to produce the plan for action to be executed or simulated: in general, such mapping is performed by a more or less sophisticated pattern matcher. What the pattern matcher lacks is the capacity of dealing in a principled way with action descriptions more or less specific than the stored knowledge, and with non standard modifiers such as temporal clauses, extent, manner, purpose — such as the clause *to create a simple scaffold* in (1.1b) — and so forth.

1.2 Contributions

I see the contributions of this thesis as falling into three categories: a pragmatic analysis of purpose clauses, and, to a lesser degree, of negative imperatives; a more flexible computational model of instruction understanding; and a practical application of my work in the context of the *Animation from Natural Language Instructions* project.

Pragmatics. I provide a pragmatic analysis of both purpose clauses and negative imperatives.

Purpose clauses have been examined mainly from a syntactic / formal semantics point of view [Hegarty, 1990; Jones, 1991; Green, 1991]; the only pragmatic work on purpose clauses I know of is by [Thompson, 1985], and, in a more computational setting, by Balkanski [1992a; 1992b; 1993]. I will show what pragmatic functions such clauses perform, what kind of relations between actions they express, and what kind of inferences an agent has to do to understand them. Moreover, my analysis, by showing that purpose clauses express *generation* or *enablement* between the actions described in the main clause and in the purpose clause respectively, lends support to the proposal, made in [Pollack, 1986; Balkanski, 1992a; Balkanski, 1993], that such two relations should be used in modeling actions.

My analysis of negative imperatives is much more limited in scope: I have classified negative imperatives into two subtypes, and shown that they correlate with different discourse functions. The analysis of the computational consequences of negative imperatives is left for future work.

Instruction understanding.

1. As mentioned above, instruction understanding systems assume a direct map between a logical form — built by a parser if the system is actually accepting NL input, or otherwise built by hand — and the knowledge about actions stored in the system's Knowledge Bases. I challenge such assumption, and provide mechanisms to make the mapping more flexible.
2. Action representation formalism. The formalism I will present tries to address the different demands posed by the need to represent both linguistic and planning knowledge about actions. My view is that hierarchical relations between action descriptions are necessary to provide the flexibility required to interpret NL descriptions of actions; however, as the space of such action descriptions is infinite, and therefore they cannot be all known in advance, hierarchical relations need to be computable on demand. The classification algorithm provided by hybrid systems [Brachman *et al.*, 1983b] proves to be useful to perform some of these inferences; therefore, I use one such hybrid system [Brachman *et al.*, 1991].

   To guarantee that the primitives of the representation are linguistically motivated, I use ones derived from Jackendoff's work [1990] on the semantic representation of verbs and actions; such linguistic action types are then combined into *commonsense plans* about actions using notions derived from the planning literature.
3. Plan inference. As mentioned above, the necessity of taking *goals* into account, and of performing *plan inference*, has long been recognized by researchers operating in NL discourse understanding; however, as I will show in Chapter 3, the focus of the research has not been on how to account for the linguistic form of the utterance with all its variability, and on how to map it to the plan knowledge. My work addresses this issue.

AnimNL. The *Animation from Natural Language (AnimNL)* project at the University of Pennsylvania has as its goal the automatic creation of animated task simulations from NL instructions, which are fed to the *AnimNL* system, and simulated by means of an animation system [Webber *et al.*, 1992]. The program I have implemented provides the interface between the parser and the plan inference / planning processes: it builds the first pass of the *plan graph*, namely, the structure of the intentions the performing agent adopts.

1.3 Thesis Outline

First of all, a note on terminology: I will refer to the agent understanding and executing the instructions as *agent* or *hearer* (H for short), and with masculine pronouns; the instructor will be referred to as *instructor* or *speaker* (S for short), and with feminine pronouns.

In Chapter 2, I provide the context for my work: after some general considerations on understanding instructions, and on some necessary inferences, I define what I mean by *accommodating instructions*, deriving the notion of *accommodation* from [Lewis, 1979]; I draw some preliminary conclusions on accommodation inferences, and on the important role that the goals play in directing such inferences; and I discuss the need for some *reference semantics* for instructions.

My work relates to various areas of investigation, such as psychology, theories of actions, plan inference: in Chapter 3, I report on relevant literature from the various fields, with particular prominence given to plan inference work on the one hand, and to computational theories of instruction understanding on the other. As a side remark, notice that I will use the term *plan inference* throughout, while in the literature both *plan inference* and *plan recognition* are used interchangeably.

Chapter 4 is devoted to a pragmatic analysis of both *purpose clauses* and *negative imperatives*. After reviewing some relevant literature, I examine each construct from the point of view of the discourse functions it performs.

Chapter 5 opens the computational part of the thesis. I start by examining the computational consequences that data like *purpose clauses* have: I describe the possible relations between the two actions described respectively in the matrix and in the purpose clause, and some inferences necessary to process such data.

In the following two chapters, Chapter 6 and 7, I describe my computational model of instructions: in Chapter 6 I describe the action representation formalism; in Chapter 7, I discuss the algorithm that implements the inference processes that build the plan graph. The application context in which my work is taking place, namely, the *Animation from Natural Language* project at the University of Pennsylvania, is described in Appendix D.

Chapter 8 is devoted to conclusions and directions for future research.

Chapter 2

On instruction understanding

Instruction understanding, like all other intelligent activities, requires performing inferences. It sounds reasonable, and in fact quite obvious, to assume that such inferences rely on an agent's knowledge. For example, research on *user modeling* tries to assess how the knowledge level, or expertise of an agent, affects the exchange of information in interfaces to computer systems [Kobsa and Wahlster, 1989].[1] However, another component to the inferences that an agent makes is frequently overlooked: that is, the input to the inference process — the surface form that the agent is presented with. This may sound surprising, given that much work has been done on understanding Natural Language instructions, and therefore on interpreting their surface forms — see for example [Winograd, 1972; Chapman, 1991; Vere and Bickmore, 1990; Alterman *et al.*, 1991]. What I mean by saying *the surface form has not been given adequate attention* is that little work has been done on how modifiers and adjuncts to an action description modify its interpretation — specifically, I will concentrate on adjuncts. Reconsider the example from Ch. 1:

(2.1a) *Place a plank between two ladders.*

(2.1b) *Place a plank between two ladders to create a simple scaffold.* [2]

As discussed in Ch. 1, in (2.1b) the agent must be inferring the proper position for the plank from the expressed goal *to create a simple scaffold*. The interpretation of the adjunct clause is therefore constraining the interpretation of the action to be executed.

[1] [Prince, 1981] points out that a recipe whose description is several pages long in a cookbook for the general public, is reduced to one line in a cookbook for expert chefs.

[2] Although this example does not appear in my corpus per se, it is a paraphrase of a real one expressed by a Means Clause: *Create a simple scaffold by placing a plank between two ladders.*

In (2.1b), the goal constrains the interpretation of the prepositional phrase as the target location of the *placing* action; in other examples, the goal can be seen as constraining other features of the action. Consider:

(2.2a) *Cut the square in half.*

(2.2b) *Cut the square in half to create two triangles.*[3]

As in (2.1b), in (2.2b) the goal *create two triangles* constrains the interpretation of the action description *cut the square in half*: however, in this case the constraint is on the path of the cut, namely, *along the diagonal.* The computation of restrictions on action descriptions is one kind of inference I will deal with in this thesis.

Another kind of inference deriving from the same surface form is illustrated by the following examples:[4]

(2.3a) *Go into the kitchen to get me the coffee urn.*

(2.3b) *Go into the kitchen to wash the coffee urn.*

In both cases, it appears that a basic expectation $\mathcal{E}$ is developed about the location of the action β described in the purpose clause. [5] However, the interpretation of (2.3a) yields a further expectation $\mathcal{E}_1$ that the coffee urn is in the kitchen, while the interpretation of (2.3b) does not. Notice that such expectations don't derive simply from knowledge about β. For example, neither (2.4a) nor (2.4b) convey much information about the locations of the coffee urn or of the washing site:

(2.4a) *Get me the coffee urn.*

(2.4b) *Wash the coffee urn.*

Moreover, such expectations can't simply be attributed to *world knowledge*. For example, one explanation of the oddness of (2.5) is that one doesn't normally find washing equipment in the living room. As such, (2.5) indeed appears to convey the expectation $\mathcal{E}$ that *the washing site is the living room*, which goes against world knowledge:

(2.5) *Go into the living room to wash the coffee urn.*

What I will argue is that it is the relation $\mathcal{R}$ between α and β, combined with specific knowledge about both α and β, that can give rise to the expectations about the location

[3] Also this example is a paraphrase of a really occurring one, this time expressed by a gerundive adjunct: *Cut in half on the diagonal creating two triangles.*

[4] These are not really occurring examples, but they are of interest as we are trying to animate them in *AnimNL*. While my analysis of the discourse functions of purpose clauses is indeed based on a corpus of naturally occurring examples, I will illustrate my inferences on examples such as (2.2b) and (2.3a) and (2.3b), that are either paraphrases of naturally occurring ones or are made up. I am confident the reader will be sympathetic to the view that really occurring examples bring in too many diverse issues, and that to test a computational model it is necessary to simplify the examples to a certain degree in order to focus on the issues of interest.

[5] As we will see in Ch. 5, $\mathcal{E}$ may concern the location of some substeps belonging to the decomposition of β, rather than β seen as a whole.

of objects involved in an action. The surface form, by employing a *to* construction, constrains what $\mathcal{R}$ really is. Clearly, there is a *m:n* correspondence between surface form and inferences: we have already seen that the presence of an infinitival *to* stimulates at least two different kinds of inference; likewise, some of these same inferences would be performed in case other surface forms, such as *in order to, so as to, by*, were used instead of *to*, as we will see in Sec. 4.1.4.

There are three major points I would like to make:

1. The inferences I am interested in *constrain* the description of the action the agent should perform by *augmenting* it.
2. The accommodation process is directed by the *goal* the agent is pursuing.
3. Action description interpretation should be grounded in performance.

1. All the inferences I have described so far in a sense *constrain* the description of the action the agent should perform by *augmenting* it. This is consistent with Suchman's observations [1987] that instructions are irremediably incomplete, and that

> the problem of the instruction-follower is viewed as one of turning essentially partial descriptions of objects and actions into concrete practical activities with predictable outcome. ... Instructions rely upon the recipient's ability to do the implicit work of anchoring descriptions to concrete objects and actions.

This quote also supports the view that, in the vast majority of cases, instructions assume a certain basic set of skills and abilities of the agent and just specify how they are to be related to one another. This point of view is shared for example by [Chapman, 1991]:

> Routine activity is by far the more common. We spend most of our lives engaged in activities like making breakfast, driving to work, reading the paper; ... these rarely involve novel elements. Even creative work is mainly routine: it's rewording sentences or painting the background of the scene.

While Chapman is mainly concerned with how much you can do with just these basic skills, I am more concerned with how you can understand instructions by assuming basic building blocks and relating them as the instructions suggest — therefore I am not looking at instructions as a means of learning to do something radically new, but rather, of exploiting preexisting knowledge to perform tasks. Following directions to reach a certain place is a good example of the kind of instruction processing I am thinking of: known actions, such as *go straight until ..., turn left at ..., cross over ...*, are related in a way that teaches the agent something new, namely, how to reach an unknown place.

Even in following directions, finding relations between basic actions does not simply amount to mapping reference points onto their referents in the environment and then executing the command. Directions, like all other instructional text, may involve complex imperatives, that require complex inferences, such as

- Purpose clauses, as in *to get into the building, turn left at the first intersection and ring the bell at the second door on your right*: in this case, the minimal inference the agent should perform is to understand what the surface form leaves implicit, namely, that after ringing the bell, one is supposed to open the door and go in when the door release buzzer sounds.

- Negative commands, as *turn left, but don't take the first left because it is a blind alley*. In this case, the negative imperative rules out the default interpretation of *turn left* as *turn left at the first opportunity*.

- Warnings, as *take the first left, but be careful if you're driving because it is very steep*. In this case, the warning will impose constraints both on the execution of *turn left*, and on the actions that will follow *turn left*.

The following example, presented above as (2.3a), also illustrates what I mean with *relating basic blocks to each other*:

(2.6) *Go into the kitchen to get me the coffee urn.*

I assume that the agent doesn't have a particular plan for *getting someone a coffee urn.* He will have a generic plan for *get x to y*, which he will adapt to the instructions given to him.[6] Some of this adaptation task is quite easy, namely, parameter instantiation, in this case binding the patient parameter of *get* to an individual satisfying the description *coffee urn*. But the rest is not so direct: the agent has to find the relation $\mathcal{R}$ between *go into the kitchen* and *get me the coffee urn.* If the agent's plan for *get x to y* contains a requirement that the agent move to the place where the object to be "gotten" is located, the agent can derive that the (most direct) connection between these two actions is that *go into the kitchen* fulfills this requirement, under the expectation that the referent of *coffee urn* is in *the kitchen*.

My view is that the agent's task includes both *understanding* the role that an input instruction plays in a plan to do something, and *adapting* his own knowledge of that plan to the instruction he has to understand. (2.1b) and (2.2b) are examples of adapting an instruction to the plan, in this case simply specified by the goal contained in the instruction itself. Examples of adapting the stored plan to the input instructions are (2.3a) and (2.3b), in which the plans for *get x to y* and *wash x* are augmented by expectations, as illustrated above.

I would claim that this process of adapting the agent's plan to the instructions and vice versa is what normally happens when people interpret instructions. I will call this process *accommodation*, borrowing the term from [Lewis, 1979]. In Sec. 2.1, I will discuss *accommodation*, first as described in [Lewis, 1979], and then as related to instruction understanding.

[6] Actually the agent may have more than one single plan for *get x*, in which case *go into the kitchen* may in fact help to select the plan the instructor has in mind — I will address this issue in Sec. 5.2.1.2.

2. The second issue I want to raise is that the accommodation process is directed by the *goals* that the agent is pursuing. In the examples above, we have seen that an infinitival *to* construction implies that the action it describes is the *goal* to whose achievement the action described in the main clause contributes; we have also seen that the goal affects the interpretation of the action to be performed. This is true even for very simple instructions, such as:

(2.7) *Open the door.*

The execution of this action will vary according to whether the goal of opening the door is letting the cat in or out — a small opening is sufficient; letting a guest in or out — the door has to be open wider than for the cat; letting in or out movers carrying bulky furniture — the door is best propped open.

That NL phenomena have to be interpreted in context is a truism by now. That goals should be taken into account is widely acknowledged in *plan recognition* work [Litman and Allen, 1990; Schank and Abelson, 1977; Wilensky, 1983], as I will discuss in Sec. 3.2. However, it has not been acknowledged as much in work on understanding NL instructions — see Sec. 3.3. Even in the case of plan recognition work, however, *goals* generally have been considered as global constraints on interpretation, not as affecting features of the input action description.

Quite naturally, giving prominence to goals in instruction interpretation would seem to presuppose a precise definition of what a goal is. I will discuss this issue in Sec. 2.1.2.

3. Finally, my interest in the constraining function of *goals* on logical forms also derives from the need to *ground action in performance.* In theories that are not concerned with action performance, equating the interpretation of an instruction to deriving its logical form is enough; however, if one's research interest is in how actions are actually mapped into performance, and what such performance amounts to, then assuming that the semantics of an instruction is equivalent to its logical form is not sufficient. I will discuss the issue of *grounding action in performance*, and the related notion of *reference semantics*, in Sec. 2.3.

2.1 Lewis on accommodation

Lewis in [1979] uses the term *accommodation* to refer to the process by which *conversational score does tend to evolve in such a way as is required in order to make whatever occurs count as correct play.* By conversational score, Lewis means the state of the conversation, described by various components, such as sets of presuppositions.

The rule of accommodation reads:

(2.8) If at time t something is said that requires component s_n of conversational score to have a value in the range r if what is said is to be true, or otherwise acceptable; and if s_n does not have a value in the range r just before t; and if such-and-such further conditions hold; then at t the score-component s_n takes some value in the range r.

Lewis uses the concept of accommodation to deal with different linguistic phenomena, among which

Presuppositions. If I say *All of Fred's children are asleep*, and there has been no mention of Fred's children so far, the presupposition that Fred has children is readily added to the set of current presuppositions.

Definite descriptions. I will actually describe the way that Heim in [1982] exploits *accommodation* to deal with *novel definites*, because her treatment is more perspicuous.

In the file-change semantics Heim develops for NPs, the usage of a definite NP is felicitous only if there is already an appropriate card in the file describing the referent of that NP. However, there are some usages of definite NPs that do not fit with this condition, for example the inferable usage ("the author" in "John read a book about Schubert, and wants to write to the author"). Heim rewrites her Novelty-Familiarity-Condition by postulating accommodation as

> an adjustment of the file that is triggered by a violation of a felicity condition and consists of adding to the file enough information to remedy the infelicity.

Planning. Lewis also talks about planning, albeit briefly. Discussing a conversation about a plan to steal plutonium from a nuclear plant, Lewis says:

> Much as some things said in ordinary conversation require suitable presuppositions, so some things that we say in the course of planning require, for their acceptability, that the plan contains some suitable provisions. If I say "Then you drive the getaway car up to the side gate", that is acceptable only if the plan includes provision for a getaway car. That might or might not have been part of the plan already. If not, it may become part of the plan just because it is required by what I said.

In the context of plans, therefore, Lewis's accommodation refers to a global process, very similar to the one occurring for presuppositions, of making room for new things in the plan if they are not already there. I am talking about a similar form of accommodation, that allows an agent to understand instructions that don't exactly mirror his knowledge. To see how the inferences I discussed above can be seen as *accommodation*, refer to rule (2.8): in the case of (2.2b), augmenting the description *cut the square in half* with *along the diagonal* can be thought of as setting the right value of the proper component s_n of the conversational score; computing the proper expectation in (2.3a) and (2.3b) would have an analogous effect on the value of the proper s_n.

More recently, [Thomason, 1990] discusses accommodation as a fundamental strategy for pragmatic inferences, and defines it as a combination of *plan recognition* and *cooperative goal adoption*, and therefore, as a special type of obstacle elimination. Thomason mentions that the principle behind accommodation is to adjust the conversational record to eliminate obstacles to the detected plans of one's interlocutor: stretching the analogy a bit, the inferences I propose could be seen as eliminating the obstacles to the detected plan of the instructor, namely, producing meaningful instructions that result in the agent correctly carrying them out. As we will see in Sec. 3.2, the point of view that accommodation is related to plan recognition is supported by the strong ties between plan recognition and the kinds of inferences I propose.

2.1.1 The notion of `goal` as guide to accommodation

Two questions that Lewis does not address, but that are relevant to the role that accommodation plays in planning, are: why does the agent accept a certain instruction and accommodate it in his plan to do something? What supports and justifies accommodation?

My claim is that the answer to the latter question is the notion of *goal*. In Lewis's example above, the accommodation necessary to account for the *getaway car* is readily performed because there is a global goal for the plan in question, which is *steal plutonium from a nuclear plant*; and a subgoal of this plan is *escape safely*. It is not blind accommodation, then: it happens because there is something that justifies it. Notice that mentioning the function of the car, *getaway*, is crucial to the accommodation process: if the instruction to be accommodated had been *Then you drive the car up to the side gate*, and if no car had been mentioned so far, it would have been much harder to find its role in the global plan.

In general, there is not a single goal according to which an instruction has to be interpreted and executed, but a set of them. Consider this instruction step in making a stuffed toy:

(2.9) *To attach arms, use doubled white carpet thread and push needle in one side of the body and out the other side at points indicated by dots on the pattern.*

The action *push needle in one side of the body ...* has to be performed *to attach arms*. In turn, the action *attach arms* has to be performed in the context of making a doll: this puts some requirements on *attaching arms*, e.g. where the arms have to be attached. These requirements "filter down" to the interpretation and execution of *push needle in one side of the body ....* Notice that the modifier *at points indicated by dots in the pattern* is not sufficient to establish where to attach the arms. The pattern contains many dots, corresponding to where arms, legs, head should be attached: therefore, the goal *to attach arms* does affect the interpretation of the prepositional phrases *in one side of the body and out the other side.*

Goals are very often hierarchically organized; however, it may happen that an action achieves two independent goals, as in

(2.10) *Line the tub with cardboard to protect the finish and to prevent debris from clogging the drain.*

There are at least two possible objections to my proposal of using goals in instruction interpretation:

1. **The function of goals is only that of giving the agent the reason why he should perform a certain action.** This is a common view, but naturally occurring purpose clauses show that there is something else going on.

   Going back to Exs. (2.1b) and (2.2b), we saw that the goal, *create a simple scaffold* in the former pair, *create two triangles* in the latter, isn't simply used to tell the agent why he should perform α, but also to help him execute the action correctly.

2. **Goals are significant only at execution time: while the concept of goal is definitely relevant, it has to be taken into account only at the moment of executing an action.** This is the view embodied in [Alterman *et al.*, 1991]. While this is true for example for (2.7) above, there are various reasons why one would think that goals affect the *interpretation* of instructions too.

   First of all, it is very often NL itself that specifies the goal that provides the context for instruction interpretation, as in Exs. (2.1b) and (2.2b). Therefore a NL system that interprets instructions has to be able to recognize and actively use goals as they arise from NL specifications of actions, to compute the descriptions of the actions to be executed.

   Secondly, the goal may affect actions that need to be executed prior to the intended action: in (2.2b), the goal *make two triangles* will affect how the agent picks up the square, namely, which part of the square he grasps, and how he orients it — grasping site and orientation will be different in the case the agent has to cut the square in half along the perpendicular axis.

   Thirdly, sometimes instructions point to general *policies* that have to apply in specific situations, as in

   (2.11) *To prevent stains, wipe up spills as soon as they happen.*

   Clearly in this case no immediate execution is meant. One interesting question is how the policy is stored so that the agent becomes aware of it at all and only the right times: it is possible that the way policies are indexed is by means of both the goal to be achieved and the description of the situation in which the policy applies.

   In conclusion, what I want to point out is that I am not claiming that the *computation* of the action to be executed necessarily happens at interpretation time; what I do want to stress though is that, given his basic abilities, the agent can understand what is required simply from the NL instruction. These are the kind of inferences I'd like to account for; and this is the sense in which I am saying that the *goal* affects not just execution, but also interpretation of instructions.

2.1.2 What are goals?

I have been talking a lot about goals, and the important role they play in understanding instructions: such prominence given to goals would seem to presuppose an understanding of *what goals really are*. However, as we will see, it is not easy to go beyond the intuitive definition that a goal is *The purpose toward which an endeavor is directed; objective* (The American Heritage Electronic Dictionary, 1991).

In the planning literature, a *goal* is any desirable state. According to [Hobbs, 1990, p.445],

> a goal is simply a proposition, or rather a logical expression representing a proposition, that the agent tries to make true. The proposition can be anything, not just actions on the part of the agent; there are no restrictions on its predicates or arguments. Just as `pick-up(I, BLOCK3, NOW)` is a possible goal,

> so are `push({JOHN,I}, CAR5, NOW)` and `win(S.F.GIANTS, World-Series, 1996)`. The goals can represent events in which I myself am the agent and am able to perform directly, events I can bring about in concert with others, and events I have little control over. The goals can be immediate, like the first two, or they can be long-term, like `write(I, Great-American-Novel, Before 2002)`.

For others [Schank and Abelson, 1977; Wilensky, 1983], the notion of goal is left informal; they talk about *actors having goals* and they classify possible goals according to several dimensions, but they don't give any formal definition of what goals are — see Sec. 3.2.1.4.

Clearly, goals and intentions are closely related; according to [Hobbs, 1990, p.446], intentions can be viewed as goals, provided that we don't take the use of the English word "intend" too seriously. Future-directed intentions [Bratman, 1990] are high-level goals expressing conditions in the future, while intentions-in-action are low-level goals to be executed now.

These definitions still seem to beg the question, or are too broad, as in the case of planning where any proposition can be a goal. More precise definitions of *goal* may be found in the context of logical formalisms. For example, the notion of goal proposed in [Cohen and Levesque, 1990] is embedded in the context of a modal logic whose model theory is based on a possible worlds semantics. Their logic has four primary modal operators: BELief, GOAL, HAPPENS (which event happens next), DONE (which event has just occurred). BEL is defined by means of the usual Hintikka-style axiom schemata [Halpern and Moses, 1985], and goals are defined on top of beliefs.

An agent has many, possibly conflicting, desires. From these he chooses his goals, namely, those desires he wants most to see fulfilled. Goals are modeled as propositions true in all of the agent's chosen worlds. The following proposition holds:

(2.12) $\models$ *(BEL x p)* $\rightarrow$ *(GOAL x p)*

namely, worlds compatible with an agent's goals must be included in those compatible with his beliefs: given a set of worlds that are possible given what an agent believes, a subset of these are the worlds in which the agent's goals are achieved.

Notice that Cohen and Levesque's notion of goal is rather counter-intuitive, as noted in [Allen, 1990, p.73]:

> the formula *(GOAL x p)* does not assert that the agent x has p as a goal in the intuitive sense of the word. Rather, it says that p will be true in any world where the agent's goals are achieved. In some ways, the term "consequence-of-goals" might be a better term name for the operator.

By means of the definition of goal, Cohen and Levesque define the concept of *persistent goal*, and on top of it, the concept of intention.

Persistent goal, *(P-GOAL x p)*. An agent x has a P-GOAL p if he has a goal p that he believes currently to be false and that he will continue to choose – at least as long as a given condition q remains valid, where q concerns, among others, the achievability of p.

Intention, *INTEND(x, p, q)*, is defined as a persistent goal on the part of agent x either to do an action p, believing one is about to do it, or to achieve some state of affairs, believing one is about to achieve it. This goal persists as long as condition q does not hold. q's becoming true permits x to drop his intention to do p.

For a critique of Cohen and Levesque's formalism, see [Allen, 1990].

My own view of what a goal is is rather informal: in my opinion, any action that can be seen as achieving a certain result can be considered as a goal. The analysis of NL instructions shows that any action description, even one concerning bodily movements such as *To stretch your back,* [7] *lean over from the waist first to your right and then to your left* can be seen as a goal that an agent intends to achieve. At this point the question may then become, *what is an action?* I will adopt the informal view expressed in [Jackendoff, 1983], that an action can be the answer to the question *what did you do?*.

As we will see in Ch. 6 and Ch. 7, actions will be defined as events having an animate intentional agent; and goals in terms of their relations to other actions in the *plan graph*, which represents the intentions the agent adopts.

2.2 Speaker's expectations and Hearer's choices

Instructions are meant to affect an agent's behavior. While interpreting instructions, an agent is continually faced with an infinite number of choices; to interpret even a simple instruction such as *wash your hands* an agent has to "decide", among other things, where to wash one's hands, whether to use soap and how much soap to use, and so on. However, the vast majority of these choices are not consciously considered, because they don't matter in the situation at hand. One task an instructor is faced with, then, is to identify which choices matter, and to constrain the alternatives.

Another function that goals perform is one of constraining such alternatives; going back to goals expressed by a purpose clause, as in Exs. (2.1b), (2.2b), and (2.3a), we saw that the action described in the main clause admits an infinite number of possible interpretations, and that the goal constrains the choice among such alternative courses of action.

Negative Imperatives are another kind of data that I have examined, and that show how the possible choices of the agent affect how the instructor builds her instructions: negative imperatives, by telling the agent what he should *not* do, are one explicit way of identifying and pruning such choices. More specifically, they can be shown to identify either a choice point that the agent is likely to overlook, or a wrong alternative that the agent could choose when recognizing a certain choice point.

In fact, from the point of view of an instructor, a negative imperative is produced when

[7] A made-up example that nonetheless describes a perfectly plausible goal to achieve in an aerobics class.

- the speaker (S) expects the hearer (H) to overlook a choice point. The choice point is sometimes identified through a side effect that the wrong choice will cause. For example, in

  (2.13) *To hang the border, begin at the least conspicuous corner. The work will go much faster if you have someone hold the folded section while you apply the border to the wall.* **Take care not to drip paste onto the wall**.

  S expects H not to realize that there are different ways of performing the action *hanging the border*, some of which result in the side effect of dripping paste on the wall; and therefore S alerts H to take steps in order to prevent that from happening.

- S *expects* H to choose the wrong alternative among many — possibly infinite — ones.

  (2.14) *After washing the tile, rinse it thoroughly to remove detergent film; then wipe with a soft, dry cloth. For stubborn dirt, scrub tiles with a white, cleansing powder.* **Don't use a cleaner containing bleach;** *this can pull the color out of the colored grouts.*

  If not cautioned against using a cleaner containing bleach, the agent may choose one such.

Although I haven't examined negative imperatives as thoroughly as purpose clauses, and in particular the analysis of the necessary processing inferences is missing, in Sec. 4.2 I will discuss a corpus study of negative imperatives, and some preliminary but suggestive conclusions on the discourse functions of such constructs.

2.3 Grounding action in performance

As I mentioned above, my interest in the kind of inferences illustrated earlier derives from the necessity of *grounding action in performance*, which in turn stems from the *Animation from Natural Language (AnimNL)* project in whose context my work is situated, and in which NL instructions indeed get executed, namely, animated in a simulated environment. Over the years, Penn's Graphics Laboratory has developed an extensive model-based animation system. The system embodies anthropometric, kinematic and dynamic models, so that agents of different builds and strengths can be animated to perform tasks such as *grasp, look at, stand up, sit down* etc.

The idea of interacting with such animated agents by means of NL instructions stems from the observation that, given such model-based animation, it makes sense to envision a system where agents are given goals to achieve, or instructions to perform. Such a system could be used, among other things, to instruct human agents on how to perform a task, or as an aid to designers, e.g. to check that the product is designed correctly for maintenance and repair. Given the wide variety of possible users and applications for such a system,

the most suitable and flexible language for interacting with the animated agent is Natural Language [Webber *et al.*, 1992].

Animating actions, namely, executing them in a simulated environment, requires some kind of *reference semantics*, as has been advocated in different areas of research, including [Miller and Johnson-Laird, 1976; Jackendoff, 1983; Fillmore, 1985; Herskovits, 1986; Suppes and Crangle, 1988; Schirra, 1990]. As [Chapman, 1991] mentions,

> [these researchers] argue for grounding pragmatics, semantics, and some syntactic phenomena in perception, bodily space, and activity. They propose, for instance, that the meanings of many words and perhaps some syntactic constructions ground out in spatial reasoning and representation.

Also [Suchman, 1987] notes that

> instructions rely upon the recipient's ability to do the implicit work of anchoring descriptions to concrete objects and action.

This kind of semantics is termed *reference semantics*, and it is particularly appealing for computer systems which focus on agents, such as robots or, in our case, animated figures, that perform tasks either in the world or in a simulated environment: in such systems perception and vision play an important role.

To take a concrete example, [Suppes and Crangle, 1988] describes an approach to the interpretation of NL commands in which the context of the utterance is brought to bear on the interpretation of words used in the command. The aspects of context they stress are: the perceptual situation in which a command is given, the cognitive and perceptual functioning of the agent being addressed, and the immediate linguistic surround.

They contrast *result* and *process* semantics, which are both untenable, although for different reasons [p.321]:

> When Susan says to an agent "Bring me the book on the table", we naturally tend to think that the command's satisfaction is evaluated just in terms of the result ... No context to worry about. No ambiguity. The same is true of the command "Bring me the book" if only the 'result-semantics' consisting of the pair (brought the book, did not bring the book) is considered. An ideal 'process-semantics' of this command should ... consist of all possible acceptable paths of movement to the table and back, together with a probabilistic measure of their likelihood of occurrence. But this is hard enough to do for the simple, idealized models of classical statistical mechanics.

They conclude that procedural or operational denotations are most appropriate for the language of actions: this is not a conclusion I necessarily agree with, but I certainly share their concern that result-semantics, namely, model-theoretic semantics, doesn't seem to tell you much if you want to account for action execution.

Clearly, looking for a *reference semantics* should not be taken as meaning that there is a 1:1 mapping between an instruction and a specific sequence of actions that corresponds to its execution. Again, the correspondence between instructions and sequences of actions that implement them is *m:n*: a given instruction may be mapped onto zero, one or more sequences of actions; conversely, any sequence of actions can be expressed by means of an infinite number of NL instructions.

Negative imperatives are a type of instructions that doesn't directly map onto a sequence of actions. Consider Ex. (2.13), repeated here for convenience:

(2.15) *To hang the border, begin at the least conspicuous corner. The work will go much faster if you have someone hold the folded section while you apply the border to the wall.* **Take care not to drip paste to the wall.**

S expects H not to realize that there are different ways of performing the action *hanging the border*, some of which result in the side effect of dripping paste on the wall; and therefore S alerts H to take steps in order to prevent that from happening. *Take care not to drip paste onto the wall* is not mapped onto an independent action, but rather, it constrains the whole process of hanging the border.

2.4 Summary

To conclude, my point of view on understanding instructions is as follows:

1. The actions an agent has to perform when he is given instructions have to be *computed* from the descriptions given in the instructions themselves, as opposed to simply *extracted* from such descriptions. Examples such as (2.2b), (2.3a) and (2.3b) show that the logical form produced by the parser must be further constrained. Such inferences can be seen as *accommodation* processes.

2. The goals that an agent is trying to achieve guide accommodation; many of these goals are explicitly stated in the instructions themselves. (2.1b) and (2.2b) in particular are evidence for the constraining task that goals perform.

   Clearly, if one attributes such an important role to *goals*, one should first define what a goal is: in Sec. 2.1.2, I have presented various approaches to such thorny issue, and my own perspective on it.

3. The necessity of computing action descriptions also relates to the need of developing a *reference semantics* for instructions.

In the next chapter, I will relate my own work to various areas of investigation, such as psychology, plan inference and computational systems for understanding instructions: I will report on relevant literature from the various fields, with particular prominence given to plan recognition work on the one hand, and to computational theories of instructions on the other.

Chapter 3

Literature review

In this chapter, I would like to explore the two areas of research closely related to my own, namely, work on plan inference, and on computational approaches to understanding instructions. I will examine such work in light of the following three tenets that underlie my own work, and that I repeat here for the reader's convenience:

1. The actions agents have to perform when they are given instructions have to be *computed* from the descriptions given in the instructions themselves, as opposed to simply *extracted* from such descriptions. Borrowing the term from [Lewis, 1979], I collectively call these inference processes *accommodation*.

2. The goals that an agent is trying to achieve guide accommodation; many of these goals are explicitly stated in the instructions themselves. (2.1b) and (2.2b) in particular are evidence for the constraining task that goals perform.

3. Action descriptions found in instructions can't be expected to exactly match the knowledge that an agent has about actions and their characteristics. Therefore, to model an agent interpreting instructions we need a flexible action representation, and inference mechanisms that can deal with action descriptions at various levels of specificity.

Before describing work on plan inference, and on computational approaches to understanding instructions, I will briefly report on some psychological experiments that support the idea that goals help constrain the interpretation of the actions to be performed.

3.1 Psychological evidence

The point of view that goals affect action interpretation and execution is supported by some psychological evidence. Although I haven't done an in-depth research of the psychological literature, I did find some interesting results in [Dixon, 1987a; Dixon, 1987b; Dixon *et al.*, 1988].

In [1987a], Dixon distinguishes two different kinds of information contained in directions for carrying out tasks: *component step* information, namely, the specific enumeration of actions needed to perform the task, and *organizational* information, that indicates how the component steps are related to each other. His examples are of drawing tasks such as *You can make a wagon by drawing a long rectangle with two circles underneath:* the organizational information concerns the object that will be obtained as a result of drawing, while the component step information describes the individual figures to be drawn.

Dixon's first finding is that single sentence directions are read faster when they begin with organizational rather than with component information. This is consistent with his assumption that the role of organizational information is to provide a framework or schema for interpreting and relating the component step information. The most natural way of constructing a mental plan would be first to use the organizational information to set up a plan schema, and then to incorporate the component steps in the plan. Dixon suggests two possible explanations for the slow-down observed when the component step is presented first:

1. Guessing strategy: readers try to guess how the component steps are organized and related, rather than wait for the organizational information in the sentence. These guesses might be based on conjectures about what the overall configuration is supposed to look like, or default assumptions about how the components should be drawn. Presumably, these guesses can be corrected later if they turn out to be incorrect.

2. Buffering strategy: the component step information would be retained in a relatively uninterpreted form until the organizational information is found — where *relatively uninterpreted* means that decisions about features such as the relative size, placement, and orientation of the components have not been made.

The two accounts make different predictions about whether the extra time involved in processing components-first sentences is due to added difficulty in processing the component step information, or added difficulty in processing the organizational information. If the guessing account is right, the extra time would be due to processing the component step information, because that would be when the guessing process is taking place; if the buffering account is right, then the extra time would be due to processing the organizational component.

Recall that preliminary results show that single sentence directions are read faster when they begin with organizational rather than with component information. Dixon presents three experiments to examine when the slow down occurs, namely, while processing the component step or the organizational information, and as a consequence, whether the guessing or the buffering account seems correct. The results in general show that the slower reading rates occur primarily with the component information, suggesting that subjects need extra time to guess the relationship of the components when they are not told the object at the outset. In the first experiment, the component information and the organizational information are contained in two separate sentences, such as *This will be a picture of a wagon. Draw a long rectangle with two circles underneath.* In the second

experiment, the two different kinds of information are contained in a single sentence which is composed of a main clause and either a Purpose Clause or a Means Clause [1] — *To make a wagon draw a long rectangle with two circles underneath* or *You can make a wagon by drawing a long rectangle with two circles underneath.* In the first experiment, there are two different orders in which sentences can be presented; in the second experiment, there are four variations, according to whether the organizational information or the component information is contained in a subordinate clause, and to whether the subordinate clause is preposed or postposed. Finally, in the third experiment, no organizational information is present — the rationale for this will be explained shortly.

Subjects were asked to press a button each time they were done with the different parts of the sentence: in the first experiment, the first sentence would disappear after the subject had pressed the button, and only then the second sentence would appear; likewise in the second experiment, where subjects were presented only one sentence, the two parts of the sentence would not be present on the screen at the same time. Subjects were also asked to actually execute the instructions. Errors were recorded: in the first experiment, there were significantly more errors when the components information was presented first [p.28]:

> The pattern of error rates supports the hypothesis that subjects generated guesses when the components sentences were presented first. There were no differences between the two sentence orders in subjects' ability to carry out the component steps. However, there was a significant difference in how well the drawing resembled the intended object. These object errors might occur if subjects failed to correct an erroneous guess about the nature of the drawing. The drawing would then reflect the initial faulty interpretation of the components sentence.

For example, a correct interpretation of *This will be a picture of a wine glass. Draw a triangle on top of an upside down T* will result in an "upside down" triangle on top of the T. Instead, some subjects who read the directions in reverse order *Draw a triangle on top of an upside-down T. This will be the picture of a wine glass.* produced a drawing in which the triangle is in its "canonical" orientation on top of the T — see Fig. 3.1. According to Dixon, presumably the triangle was drawn that way because the subject had no indication that the triangle should deviate from the typical orientation when the components sentence was first read. From my point of view, this is a very revealing kind of mistake, as it shows how the goal to be achieved (the organizational information) does indeed constrain the interpretation of the action to be performed, by providing the information about features that the surface form leaves unspecified — similar to my claim for (2.2b) above.

The results of the second experiment, in which the subjects were presented with one single sentence, were consistent with those of the first as far as reading times are concerned. However, there were far fewer errors: according to Dixon, the errors in the first experiment may have occurred because subjects sometimes may not have retained the specific wording and content of an initially presented components sentence, and therefore were not able to

[1] Following [Balkanski, 1992b], a Means Clause expresses the means by which an action is performed, and it is in general built out of a *by* plus a gerund.

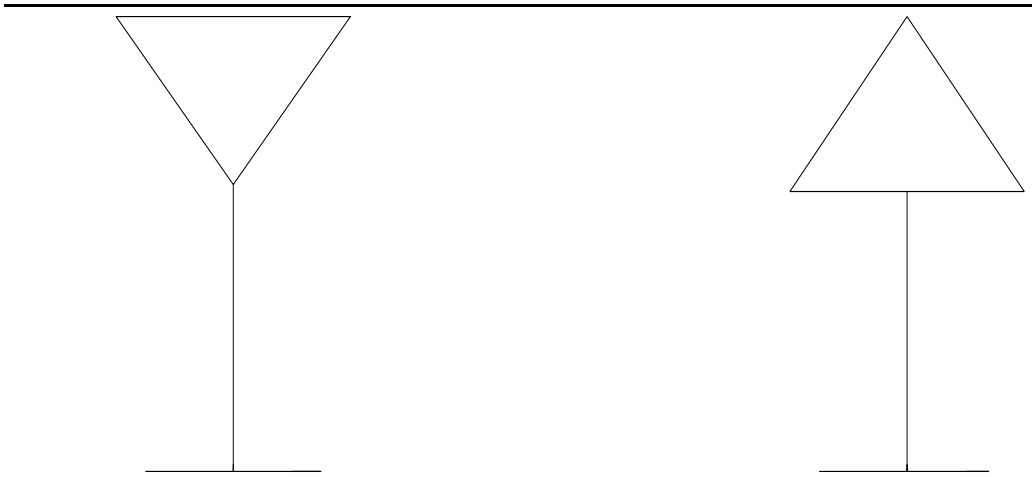


Figure 3.1: Correct and incorrect interpretation of the component step information in Dixon's experiments

use the object sentence to correct erroneous interpretations. The lack of such an effect in Experiment 2 suggests that the specific wording was more often available when the object and component steps were in the same sentence.

Finally, no organizational information was provided in the third experiment, which was meant to test the following: according to the guessing account, the size of the information-order effect should be related to how difficult it is to generate guesses about how the component steps are related. For those sentences with easily interpretable component steps, the order of the two types of information should not matter much; but if the component steps are difficult to interpret by themselves, having the organizational information first should be a great advantage. So in the third experiment subjects were presented only with the component steps, and told that such drawings should form the picture of a common object: subjects were asked to draw and identify the object. The hypothesis was borne out: not surprisingly, the number of object errors in Experiment 3 was substantially larger than it was in Experiment 1.

In the other two papers [1987b; 1988], Dixon and his collaborators present other experiments that support the three assumptions that

1. mental plans consist of a hierarchy of action schemas;

2. the hierarchy is built by beginning with the schema at the top of the hierarchy;

3. plan construction goes on in parallel with other reading processes.

3.2 Plan inference

There are several reasons why *plan inference* is relevant to my work: [2]

1. in the same spirit that plan inference has in input an impoverished description of a certain sequence of events, and tries to infer a richer description of what happened, so I am trying to account for inferences that provide a richer description of the action to be performed;

2. at a higher level, plan inference can be seen as part of accommodation. As I mentioned in sec. 2.1, [Thomason, 1990] expresses the view that accommodation consists of adjusting the conversational record to eliminate obstacles to the detected plans of one's interlocutor, therefore supporting the view that plan inference is actually part of the accommodation process;

3. plan inference work stresses the importance of taking into account the goals the agent is pursuing in order to make sense of his actions: however, I will show that the work that has addressed this issue has not explored the importance of goals as constraints on interpretation;

4. as will become clearer in Chapters 6 and 7, there are many common points between my approach to action representation and to the implementation of the required inferences, and similar issues addressed in plan inference work.

I will start with a general definition of plan inference, not restricted to understanding NL [Kautz, 1990, p.105]:

> One is given a fragmented, impoverished description of the actions performed by one or more agents, and expected to infer a rich, highly interrelated description. The new description fills out details of the setting, and relates the actions of the agents in the scenario to their goals and future actions. The result of the plan inference process can be used to generate summaries of the situation, to help (or hinder) the agent(s), and to build up a context for use in disambiguating further observations.

It is quite uncontroversial that inferring such *rich, highly interrelated description* is relevant to understanding NL — from [Cohen *et al.*, 1990]:

> Much of communication depends about agents' ability to recognize one another's intentions and plans [p. 3] ... Researchers in AI were the first to bring out clearly the necessity to treat discourse as planned activity, requiring theoretical and empirical treatment of plan structure and plan inference. [p. 10]

[2] A reminder to the reader that in the literature the terms *plan inference* and *plan recognition* are used interchangeably, and that I will use only the former.

It is clear that instructional text, like any other form of communication, requires plan inference as well. In fact, the necessity of performing plan inference also affects knowledge representation: we can think of plans as repositories of information about the relation between a goal to be achieved and the actions that achieve it; such knowledge is clearly relevant to computing how goals constrain the actions that contribute to their achievement. Therefore, in the following I will give an extensive overview of the plan inference literature as related to NL understanding: I will first talk about Allen's general description of plan inference processes in NL, and then I will turn to examining various extensions and variations to this basic model, as provided by researchers such as [Pollack, 1986; Litman and Allen, 1990; Grosz and Sidner, 1990; Lambert and Carberry, 1991; Schank and Abelson, 1977; Wilensky, 1983].

3.2.1 Plan inference and NL understanding

Allen in his textbook [1987, p. 378] gives the following definition of a *plan-based* analysis of text:

> a plan system actually attempts to construct a plan by piecing together the actions, states and goals that explain the story or conversation. Two pieces of a plan may be put together by adding an explicit causal relationship between them. For instance, a system might put two actions together using a substep-whole relation (one is part of the decomposition of the other), or an action and a state might be put together using a result relation (the state is an effect of the action), or an enable relation (the state is a precondition of the action).

The simplest form of plan inference is *decomposition chaining*, which consists of matching logical forms to action definitions; when a match has been found, the matching process is then repeated on the newly derived action description to see if it is part of the decomposition of yet another action. This process continues until either no further actions can be derived, or the system finds a connection to a previous sentence.

According to Allen, to connect sentences systems must use the analysis of the previous sentences to affect the processing of the next sentence. Therefore, plan inference systems must maintain a set of *expectations*, which are the plans that have been constructed as possible explanations of the previous sentences. A new sentence is then analyzed using the decomposition chaining technique; however, at each stage the action introduced is matched against the expectations to check for a connection.

For example, if a story started with the sentence *Jack went to the ticket clerk*, the technique of decomposition chaining would check which actions have GOTO as an action step: in the domain of taking trips by train, there may be two such actions, TAKE-TRIP and PURCHASE. The first possibility is eliminated since the LOC role of the GOTO must be a location of a train. The GOTO step of the PURCHASE action is consistent with the input, and an instance of PURCHASE — say, P_1 — is created with the proper parameter bindings. Only one action (TAKE-TRIP) has a PURCHASE action as a step. An instance

T_2 of this action is in turn created, thereby stating that the object being purchased (in P_1) is a ticket for whatever train is being taken (in T_2).

The basic decomposition-based techniques can be extended by adding the action-effect-based reasoning. This involves allowing goal states as expectations, which will be called the *expected goals*. The extended plan inference algorithm must not only search from actions to other actions via decompositions but also search from actions to expected goals via the actions' effects, and vice versa. More complex effects of action-effect-based reasoning arise when two actions are related, not by both being steps of some other action, but by one action enabling the other to occur (that is, one action's effects satisfy the preconditions of the other).

Allen describes a plan inference algorithm — see Fig. 3.2 — that I will use as the "base case" against which other approaches are evaluated. Allen's algorithm is stated in three parts: one for dealing with sentences describing an action, one dealing with sentences describing a state, and one dealing with sentences describing a goal.[3] The E-plan in Fig. 3.2 is the plan structure that serves as the expectation. The algorithm assumes that there is only one E-plan.

Allen notices that the matching process can be helped by syntactic clues, that can prune the possible relationships between actions that the algorithm checks for. In particular, he mentions that ACT_1 *in order to* ACT_2 suggests that ACT_1 is a substep of, or enables ACT_2; and that ACT_1 *by, or by means of,* ACT_2 suggests that ACT_2 is a substep of, or enables ACT_1. As I will show in Ch. 5, my analysis of purpose clauses supports the move of exploiting the surface form to restrict the search for possible connections between actions; also, through my analysis the notion of an action *being a substep of*, or *enabling* another can be refined.

A final observation. Given that inferring the plans underlying discourse is necessary, a question arises as regards the kind of plan — the E-plan in Fig. 3.2 — being inferred, namely, domain or communicative plan. Some of the initial work on plan inference in NL [Cohen and Perrault, 1979; Perrault and Allen, 1980] concentrated on inferring communicative plans, namely, understanding which speech acts a certain utterance realizes; later work [Allen, 1983; Sidner, 1985; Carberry, 1985; Litman, 1985] attempts to infer both the communicative and the domain plans underlying an utterance. In the following, I will briefly discuss work by Litman and Allen [1990] and by Lambert and Carberry [1991; 1992], as representative of the research on inferring different kinds of plans; I will then review [Pollack, 1986], who challenges some assumptions underlying plan inference in discourse.

3.2.1.1 Domain plans and discourse plans

In [Litman, 1985; Litman and Allen, 1990], a plan inference model for task-oriented dialogue understanding is presented, that explicitly separates discourse intentions from commonsense plans that are the objects of such intentions. Two types of commonsense plans are defined: *domain plans*, used to model the tasks, and *discourse plans*, domain independent plans that can be generated from execution or discussion of domain plans.

[3] Allen does not give precise definitions for what *actions, goals, states* are.

To integrate an action A:

- Part 1: Find the possible matches into the E-plan:
  1. Match A directly against any actions in the E-plan.
  2. If step 1 failed, match the effects of A against any expected goals in the E-plan and the preconditions of A against any states in the E-plan.
  3. If step 2 failed, match A's preconditions against the effects of the actions in the E-plan.
  4. If step 3 failed, match A into the decompositions of all actions known to the system. For each action X for which this succeeds, recursively attempt to integrate X into the E-plan.
- Part 2: Integrate the action:
  1. If one of the steps in part 1 succeeded, add the equality assertions — namely, variable bindings — needed to integrate A into the E-plan, and then add all the steps of A's decomposition into the E-plan.
  2. If none of the preceding steps succeeded, add A to the E-plan without any connections to other actions in the E-plan.

To integrate a state S:

1. Match S against the effects and preconditions of each action in the E-plan.
2. If a match was found in the previous step, add the equalities needed to integrate state S into the E-plan.
3. If no matches were found, add S to the E-plan.

To integrate a goal state G:

1. Match G against the effects of any actions in the E-plan.
2. If a match was found in the previous step, add the equalities needed to integrate goal G into the E-plan.
3. If no matches were found, add G to the E-plan.

Figure 3.2: Basic plan inference algorithm for NL understanding

> Our discourse plans represent a new source of commonsense knowledge, mediating between task plans and an important class of discourse intentions. Our approach has several important advantages. First, the incorporation of discourse plans into a theory of task plan inference allows us to account for a wider variety of task-oriented dialogues (for instance, dialogues with dynamically generated subdialogues such as clarifications and corrections) than previously considered. Second, the discourse plan formulation provides a clear computational model. Plans are a well defined method of representation, but, more importantly, the techniques of plan inference can be adapted to identify appropriate discourse plans from utterances. [Litman and Allen, 1990, p.367]

Discourse intentions are purposes of the speaker, expressed in terms of both the *domain* plans used to model the tasks, and the domain independent *discourse* plans that can be generated from execution or discussion of domain plans. The discourse intentions underlying an utterance such as (3.1), told by a Person to a Clerk at a ticket booth

(3.1) *I'd like to buy a ticket to New York. Here's 5 dollars.*

are roughly (1) that Person intends Clerk to believe that Person would like to buy a ticket to New York (and maybe that Person intends Clerk to believe that (3.1) was a request for a ticket), and (2) that Person intends Clerk to believe that Person is giving Clerk 5 dollars to buy the ticket. In contrast, the commonsense knowledge underlying (3.1) corresponds to the task plans to which these discourse intentions refer, such as BUY-TICKET and GIVE-MONEY.

Notice that discourse intentions not necessarily refer to any or every step in the associated commonsense plan.

The task of the plan recognizer is to recognize a domain plan structure, and any discourse plans generated by this structure, to which the discourse intentions of an input utterance refer. That is, the plan recognizer's task is not only to infer a domain plan, but also to infer any plans generated by this domain plan that are relevant to the discourse intentions. The plan recognizer has at its disposal a library of domain plan schemas varying with the domain and a library of discourse and speech act plan schemas, as well as the representation of the parse of the input utterance. The plan recognizer will use this information to associate each utterance with an intention referring to a domain plan, either directly or through a series of generated discourse plans. If the interpretation is ambiguous, a discourse intention and a set of commonsense plans are created for each interpretation.

The ideas put forward by Litman and Allen are extended in Lambert's work [1991; 1992]. She distinguishes three levels necessary to model intentions: the domain level (with domain goals such as traveling by train), the problem-solving level (with plan-construction goals such as instantiating a parameter in a plan), and the discourse level (with communicative goals such as expressing surprise). Lambert claims that her process model has three major advantages over previous approaches [Lambert and Carberry, 1991, p.47]:

1. it provides a better representation of user intentions than previous models and allows the nuances of different kinds of goals and processing to be captured at each level;

2. it enables the incremental recognition of communicative goals that cannot be recognized from a single utterance alone;
3. it differentiates between illocutionary effects and desired perlocutionary effects.

Other work based on distinguishing between different types of plans in understanding discourse is by Ramshaw [1989a; 1989b; 1991].

3.2.1.2 Pollack's model of plans as mental phenomena

In her influential work [1986; 1990], Pollack challenges the traditional approach to plan inference in discourse understanding, which amounts to directly reasoning about the object of an actor's plan, and to constructing a recipe-for-action using a library of simpler recipes that are assumed to be mutually known to the actor and to the inferring agent. In fact, if one considers the algorithm in Fig 3.2, it is clear that there is no explicit representation of beliefs: basically the observer (the system) attributes its own beliefs to the agent, and the fact that such beliefs could be incompatible with the agent's is not taken into account.

Pollack's main points are that:

1. It is necessary to specify the internal structure of plans: Pollack argues that plans should be analyzed as particular configurations of beliefs and intentions.
2. It is necessary to provide a model that makes it possible to handle situations in which there are discrepancies between the beliefs of the various agents involved in the communication situation. She claims that a model of plan inference adequate to support a theory of cooperative communication must concern itself with the structure of the complex mental attitude of having a plan, as well as with the structure of the objects of that attitude.

Pollack's alternative model of plans is composed of beliefs and intentions. The *belief* component of plans is necessary because for an agent to have a plan to do β, which consists in the doing of a collection of acts Π, it is not necessary that the performance of Π actually lead to the performance of β. What is necessary is that *the agent believes* that its performance will do so.

The *intention* component of plans is necessary because beliefs by themselves are not sufficient to guarantee that doing Π is the agent's plan to do β. In particular, for the agent's plan to do β to consist in the agent's doing Π, the agent must intend to execute each of the acts in Π. For example, an agent may believe that calling a friend on the phone would entail talking to her; however, if the agent intends to wait till he sees her face to face, calling the friend on the phone does not count as a plan for talking to her. Furthermore, in order for Π to count as the agent's plan to β, not only must the agent intend to execute Π, he must also intend it *as a way* of doing β: for example, Π may achieve both β and γ. However, if the agent only intends to achieve β, even if as a consequence of executing Π γ is achieved, Π does not count as a plan of the agent to achieve γ.

Definition P_0 [1990, p.90] An agent A has a plan to do β that consists in doing some set of acts Π, provided that :

1. A believes that he can execute each act in Π.
2. A believes that executing the acts in Π will entail the performance of β.
3. A believes that each act in Π plays a role in his plan.
4. A intends to execute each act in Π.
5. A intends to execute Π as a way of doing β.
6. A intends each act in Π to play a role in his plan.

The reader is referred to [Pollack, 1986; Pollack, 1990] for the reasons justifying the presence of each component in the previous definition, and for its formalization. I would just like to point out that Pollack models the notion of *doing one act* **as a way of** *doing another* by means of the *generation* relation, first introduced in [Goldman, 1970]: this formalization of generation and its usage in modeling actions is one of the major contributions of her work. I will discuss the notion of *generation* in depth in Ch. 5 and Ch. 6.

Turning now to invalid plans, an agent has an *invalid* plan if and only if he has a set of beliefs and intentions as in P_0, and one or more of these beliefs is incorrect — incorrect with respect to the inferring agent's knowledge, of course. Invalidities can regard clauses 1, 2 and 3 in Def. P_0. Finally, in a communicative situation, the inferring agent will have to attribute to the performing agent an *Explanatory Plan*, namely, a plan which includes *explanatory beliefs* — the inferring agent may then detect that some of these explanatory beliefs are incorrect.

3.2.1.3 Grosz and Sidner's model of collaborative activity

Grosz and Sidner [1990] approach the problem of plan inference in discourse from the point of view of modeling collaborative activity. They drop the *master-slave* assumption, namely, the assumption that S and H have fixed and not equatable roles, and that therefore only one agent (S) produces utterances, while the other agent (H) attempts to infer from these utterances S's goals and plans. They also drop the assumption that H's knowledge of actions and of the relations between them constitutes a correct and complete description of what agents can do.

To model collaboration, they resort to the notion of *SharedPlan*, which is based on Pollack's definition of *simple plan* — see def. P_0 above. They modify P_0 by requiring all beliefs to be mutual beliefs, and by allowing different agents to perform different actions. Most relevant to the plan inference issue is Lochbaum's work [1991a; 1991b] on how each agent identifies the role that a certain Act of type γ described in the other agent's utterance plays in the SharedPlan. The algorithm assumes that: Act is an action of type γ; G_i is the agent who communicates *Prop(Act)*, namely, the utterance in which Act is contained; G_j is the other agent, namely, the one being modeled; G_i and G_j have a partial SharedPlan at time T_1 to perform an action of type **A** at time T_2, SharedPlan*(G_i, G_j, **A**, T_1, T_2).

The algorithm is as follows [Lochbaum, 1991b, p.4]:

1. As a result of the communication, assert MB(G_i, G_j, BEL(G_i,Prop(Act))).
2. Search own beliefs for BEL(G_j,Prop(Act)).
3. Ascribe BEL(G_i,Contributes(γ,**A**)).
4. Search own beliefs for Contributes(γ,**A**) and where possible, more specific information as to how γ contributes to **A**.
5. If steps 2. and 4. are successful, signal assent and MB(G_i,G_j,Contributes(γ,**A**)).
6. If step 2. or step 4. is unsuccessful, then query G_i or communicate dissent.

In step 4, G_j is trying to determine why agent G_i has mentioned an *Act* of type γ: given the action formalism they adopt [Balkanski, 1990] this is equivalent to G_j trying to identify the role G_i believes γ will play in their SharedPlan. The algorithm Lochbaum develops tries to relate γ to **A**, namely, it examines all the known *recipes* for **A** that G_j knows, to see whether γ is contained in such recipe. If yes, it computes possible constraints that may have to hold for the particular relation $\mathcal{R}$ between γ and **A** to hold. I will come back to Lochbaum's algorithm in Sec. 7.3.

The important contribution of Grosz and Sidner's work is that they drop the master-slave assumption. For the time being, I will hold onto it in my work; however, nothing hinges on it, and it could easily be dropped. Although I assume H has correct knowledge, I drop the assumption he has complete knowledge: in fact, my accommodation inferences integrate the knowledge the agent has with that provided by the instructions, for example by computing expectations about object locations.

3.2.1.4 Scripts and Plans

Another thread of research on plan inference starts with [Schank and Abelson, 1977], and continues with [Wilensky, 1983] and [Dyer, 1983], among others. In [1977], Schank and Abelson (S & A for short) present a theory of story understanding based on the usage of scripts and plans, and in which goals play a prominent part. In terms of the algorithm in Fig. 3.2, S & A's theory offers a much richer representation of the knowledge about actions and how they are organized and attempts to classify all the goals that a rational agent may have. However, this richer representation makes the inferring processes much more complicated, and in the end it is not clear how S & A's algorithm really works.

A script is a *predetermined, stereotyped sequence of actions that defines a well-known situation.* A script is made up of slots and requirements on what can fill those slots. The structure is an interconnected whole, and what is in one slot affects what can be in another. Therefore, a script describes a stereotyped sequence of events, which prescribes the order in which things happen, and the people and objects participating in the action. There are scripts for eating in a restaurant, riding a bus, watching and playing a football game, participating in a birthday party, and so on. Scripts are responsible for filling in the obvious information that has been left out of a story.

However, scripts are not sufficient, as they do not represent intentionality: from a scriptal point of view, each event occurs next simply because it is the next event in the script.

Although script-based programs know that characters initiate the RESTAURANT script to satisfy hunger, they do not know why any specific event within such script, such as *tipping the waiter*, occurs.

Plans are used to encode knowledge about intentionality, and are made up of general information about how actors achieve goals. They explain how a given state or event was prerequisite for, or derivative from, another state and event. According to S & A, people use plans to make sense of seemingly disconnected sentences. By finding a plan, an understander can make guesses about the intentions of an action in an unfolding story and use these guesses to make sense of a story.

According to S & A, the process of plan inference has two main parts: first, understand the goals of the actors in the story; second, understand which particular method is being used to realize each operating goal.

> This implies having a set of methods that we know will realize a goal, or at least being able to recognize the actions of an individual as a possible method for realizing a goal. Methods for realizing goals almost always involve chains of instrumental goals, i.e., necessary partial accomplishments along the path to the main goal.

Certain goals, called *delta goals*, or **D**-goals for short, are general building blocks in many planning processes, and can be characterized as changes in common states. **D**-goals have no value in themselves, but they are subordinate to higher level main goals. Some common **D**-goals are D-PROX, the goal of *going to an intended location*, which can be characterized as a change in the state of proximity (to something); D-CONT, the goal of *gaining control*, which means to change physical control of some object from the present holder to the actor; D-KNOW, the goal of *acquiring knowledge*. **D**-goals point to the set of possible actions that can enable the actor to achieve that particular goal: these sets of possible actions are called *planboxes*. A planbox includes: a key action that will accomplish the goals of the planbox; different kinds of preconditions; and a result. For instance, one of the planboxes for D-KNOW is ASK X Y Q, with X the agent, Y the respondent, and Q the proposition whose truth value is being questioned. ASK has, among its preconditions, that Y knows Q and wants to communicate Q to X; as act, X communicating Q? to Y; as result, Y communicating Q to X.

According to S & A, the process of understanding plan-based stories is as follows:

1. Determine the goal.
2. Determine the **D**-goals that will satisfy that goal.
3. Check whether the input conceptualizations for realize one of the planboxes that are called by one of the determined **D**-goals.

To determine the *goal(s)* in a story, S & A postulate a GOAL MONITOR,

> an interrelated bundle of processes which recognizes when goals are triggered, interprets their nature, keeps track of their fate, and makes predictions about goal-related events. ... When the life of a goal is ended, the goal monitor outputs a history of its fate. Several summaries of a story can be created by coordinating goal fates: one can look at the interlocking histories of all goals, all important goals, all the goals pertaining to a particular character, etc. Summarization devices of this kind we refer to as GOAL FATE GRAPHS.

S & A classify goals as follows:

S-goal (Satisfaction Goal). A recurring strong biological need, which, when satisfied, becomes extinguished for a time — e.g. S-HUNGER, S-SEX, S-SLEEP.

E-goal (Enjoyment Goal). An activity which is optionally pursued for enjoyment or relaxation — E-TRAVEL, E-ENTERTAINMENT, E-EXERCISE.

A-goal (Achievement Goal). The realization, often over a long term, of some valued acquisition or social position. Examples are: A-POSSESSIONS, A-POWER POSITION, A-SOCIAL RELATIONSHIPS.

P-goal (Preservation Goal). Preserving or improving the health, safety, or good conditions of people, position, or property.

I-goal (Instrumental Goal). Any goal which, when achieved, realizes the precondition in the pursuit of another goal, but does not in and of itself produce satisfaction. Instrumental goals occur in the service of **S-, P-, A-,** and **E-**goals, as well as possibly being nested such that an **I-**goal serves another **I-**goal.

D-goal (Delta Goal). To what was said above one can add that a **D-**goal is similar to an **I-**goal, except that general planning operations instead of scripts are involved in its pursuit. Under each **D-**goal is organized a set of planboxes.

On the basis of this theory, a number of computer programs were written: in particular, I would like to mention PAM, by Wilensky [1982; 1983]. PAM processes non-stereotypical stories by tracing character's goals, plans, and actions. PAM attempts to explain each input as a goal for some character, as a step in a known plan, or as a substep of some intermediate plan for a known goal.

According to Norvig [Shapiro, 1992, p.1570-1571], PAM suffers from brittleness:

> because it had knowledge of the character's goals and reactions, PAM could handle a simple story like "Pat hit Kim. She cried". But because the story involves no animate characters and hence no goals, it could not handle a superficially similar story like "The wind blew the vase off the table. It broke".

Again in Norvig's analysis, some of these problems were overcome in BORIS [Dyer, 1983]:

> BORIS combined rules for dealing with different knowledge structures at the same time ... BORIS searched for an explanation at each of four levels: scriptal, goal / plan, thematic, and role. ... BORIS can be seen as an attempt to deal with all the types of knowledge structures that were proposed by [Schank and Abelson, 1977] as well as several others that were introduced since then.

BORIS's problem is that each knowledge type, and each possible interaction between knowledge types, was activated by means of rules: the rules were too *ad hoc* and the interactions among the rules too complicated to allow the rule base to scale up gracefully.

There is a general comment I would like to make on this approach: although [Schank and Abelson, 1977] is valuable in identifying many important themes, it leaves too much of a gap between the surface form and the complex Knowledge Base composed of scripts and plans, so that it is not clear how the former is mapped onto the latter — this mapping remains obscure, even if both the logical form produced by the parser and the entire Knowledge Base are expressed in terms of Conceptual Dependency theory [Schank, 1975]. The various plans are activated and goals recognized by some more or less refined form of pattern matching: in SAM, a routine APPLY takes care of understanding which scripts to apply by using a pattern matcher between the conceptualization of the input sentence produced by the parser, and scripts; for neither PAM nor its successor FAUSTUS is this part of the mapping described in much detail in [Wilensky, 1983]: frames are invoked because they have been indexed under a component that occurs as an input, or because they are explicitly associated with another invoked frame, and then *instantiated*, and *elaborated*, i.e. their empty slots are filled.

As an example, consider an instance of a purpose clause, that is contained in one of the simple stories for which S & A provide a *goal fate graph* [p. 102]:

(3.2) *Professor Stifle came to town to buy a house.*

In this example, one of the inferences I am dealing with — namely, developing the plausible expectation that *the house sought by Prof. Stifle is in town*[4] — needs to be made: as in (2.3a), a primary expectation is developed, that the transaction site is the town; and on the basis of that primary one the secondary expectation regarding the house arises too. In S & A's representation, *come* is mapped to a PTRANS, while *buy* has a conceptual representation made up of two conceptualizations that cause each other, one an ATRANS of money, the other an ATRANS of the object being bought.[5] After the sentence has been parsed, and its conceptualization produced, I would assume that it is recognized as an instance of one of the achievement goals, A-POSSESSIONS. However, how the program gets from such **A-**goal to the proper **D-**goals, which presumably include D-CONT(STIFLE, HOUSE); how the proper planbox among those associated to such goal is activated; and how the PTRANS representing *Professor Stifle came to town* is connected to such planbox, is almost a mystery to me.

[4] Another expectation, that *the real estate agent is in town* is possible. See below.

[5] PTRANS is the primitive act of transferring the physical location of an object; ATRANS is the primitive act of transferring an abstract relationship such as possession, ownership or control.

From the traces of the computer programs at the end of the book, it seems to me that reference resolution simply amounts to binding a referent to an object in a script as soon as possible: therefore, I would assume that *town* is bound to the parameter of a D-PROX, maybe associated to the planbox activated by the goal A-POSSESSIONS, and *house* to the parameter of D-CONT, but it is not clear to me how the following inferences are made, if they are indeed made:

1. there is a relation between the house sought by Prof. Stifle and the town, namely, that the town is where such house is located. [6]

2. *house in town* is simply an expectation, that could be overridden by a continuation such as *The house he wants is in a secluded locality, but the real estate office is in town.*

To conclude, it seems to me that in all these programs the mapping between the surface form and KBs is not clear at all, and the different expectations that different surface forms raise are totally disregarded: this is actually the reason why different surface forms can be interpreted, not because their similarities and dissimilarities are really accounted for.

3.2.2 Summary

To sum up, there are three issues emerging from plan inference research that are relevant to my concerns:

1. The importance of *goals*. Their relevance is emphasized in particular in [Schank and Abelson, 1977] and [Wilensky, 1983]; however, the hierarchical organization of actions in Allen's domain plans [1987] can be seen as implicitly relating the sequence of actions that achieve a goal to the goal itself; also in [Pollack, 1986] the notion of *intentions*, and of doing an action *by* doing another, is related to *goals*, as we saw in the previous chapter when I discussed [Hobbs, 1990] and [Cohen and Levesque, 1990].

   However, even if the importance of goals is stressed, they are still seen as providing the global context for interpretation, and not also as constraints on computing action descriptions.

2. The view put forward by Pollack that plans must be seen as collections of beliefs and intentions, a view then adopted by Grosz and Sidner. Such concepts are sometimes implicit in others' work; however, Pollack was the first to bring them to the forefront and explicitly use them in her formalization of plans. I agree with Pollack's point of view, but I will not explicitly represent the beliefs of the speaker. Notice however that I do relax the assumption that H has complete knowledge, in that his knowledge may be augmented by the input instruction.

3. Finally, I will compare my algorithm, described in Ch. 7, to the basic algorithm in Fig. 3.2.

[6] Notice that no distinction is made in their reference resolution process between attributive and referential readings.

3.3 Computational approaches to instruction understanding

The other body of work relevant to my research concerns computational approaches to instruction understanding, where the emphasis is on the action representation and algorithms necessary to interpret instructions; most of this research has been applied in implemented systems. The need to have a computer understand instructions — whether the system is a data base NL query system, a robot or an animated agent — often results in the main research effort being spent on the system architecture, that has to integrate many different modules, rather than on a broad coverage of NL phenomena and / or in an interest in the NL surface form per se. The AnimNL project is different, in that it tries to achieve vertical integration without disregarding the richness of the NL input.

For each approach that I describe, I will try to highlight the following points:

- what an instruction is mapped into — a logical form, a procedure, a hierarchical representation incrementally built on the basis of the previous instructions, etc;
- how the instruction relates to the goals that the agent may already have, and vice versa, how such preexisting goals can affect the interpretation of the instruction itself;
- whether the instructor has total control over the agent's actions;
- whether the communicative situation in which an instruction is issued and the performance situation in which it is executed coincide.

The latter two points are not directly related to my concern, but they affect the theories of instructions that such system implement, and are therefore worth exploring.

The reader should be aware that there is also a sizable body of literature on instruction generation, including among others: Dale's EPICURE system, that generates recipes [1989; 1992]; Vander Linden's work on generating purpose expressions in instructional text [1992a; 1992b; 1993b], that I will briefly describe in Sec. 4.1.2.1.1; Delin, Scott and Hartley's work on multilingual instructions [1993]; generation of instructional text accompanying graphics or animation, for example the COMET [McKeown *et al.*, 1992] and WIP projects [Wahlster *et al.*, 1989; Wahlster *et al.*, 1991]. Although some of the issues addressed by work on generating instructions are relevant to understanding instructions too, the central concerns of the former are different from those of the latter: therefore I won't discuss this body of literature in depth, but I will refer to it when the need arises.

3.3.1 Basic approaches

In approaches of this kind, understanding an instruction consists in forming a complete plan corresponding to the input command: such plan may be complex, in that it may for example include actions necessary to achieve preconditions of the action described in the input. Subsequently, the plan is executed by simply traversing its substeps, which correspond to *basic actions*.[7] In an implementation context, such basic actions may be

[7] See [Goldman, 1970] and [Pollack, 1986] for a discussion of *basic* actions.

procedural interpretations of VPs and NPs as in the by now venerable SHRLDU [Winograd, 1972]; or simple behaviors of animated agents, such as arm reaches and head orientation changes [Esakov and Badler, 1990; Badler *et al.*, 1990].[8]

I call these approaches *basic* because they implement a less complex model of an agent: he has no other goals apart from fulfilling the input command, the instructor has total control over the goals that the agent adopts, and the communicative situation and the performance situation coincide. Moreover, the world is seen as entirely under the agent's control. The term *basic* is not intended to detract from this kind of work, which has indeed addressed complex facets of the process of instruction interpretation, such as identifying the referents in the world of complex NPs [Winograd, 1972].

3.3.2 The situated approach

Chapman in [1991] proposes a model of instruction interpretation that is concerned with reacting to the situation at hand, and that heavily relies on perception. The instructor gives instructions or offers suggestions, while watching an agent engaged in an activity — in Chapman's thesis, the activity is playing a video game. Each suggestion can be adopted as a goal and executed: whether and when the agent takes the suggestion depends on the current situation, and the decision process is modeled by means of an arbitration network.

Chapman's views instructions as similar to perceptual stimuli:

> When Sonja [the agent playing the game] is given an instruction, it registers the entities the instruction refers to and uses the instruction to choose between courses of action that themselves make sense in the current situation. An instruction can fail to make sense if it refers to entities that are not present in the situation in which it is given, or if the activity it recommends is implausible in its own right. ... Instructions recommend courses of action; Sonja's arbitration network takes account of such recommendations. ... Because instructions can only influence arbitration when they make sense in terms of activities Sonja could engage in autonomously, their role in Sonja is one of *management* only. [Chapman, 1991, p.76]

An instruction then identifies one action among the ones that belong to the system repertoire, and gives an additional reason why the agent should choose it. There is a repertoire of 25 or so understandable instructions: each instruction is associated with a buffer whose valid buffer the instruction sets or resets; each buffer may have one or more associated fields used to register the entities mentioned in the instruction. The contents of the instruction buffers, together with other inputs, such as those deriving from perception, are input to the arbitration network, which will then "decide" what to do next.

The communicative situation and the performance situation coincide as in the "Basic" approach, modulo *making sense* of the situation, e.g. *turn left* does not mean *turn left immediately*, but *turn left in the first place where you can do so*; also notice that the

[8] The system described in [Esakov and Badler, 1990] is the "ancestor" of AnimNL.

performance situation is defined in perceptual terms — for example (from the quote above) an instruction may fail to make sense if it refers to entities that are not present in the situation in which it is given. Chapman's approach seems more concerned with making (immediate) decisions about behavior, than with deriving knowledge from instructions and then acting.

3.3.3 Vere and Bickmore's Homer system

Vere and Bickmore in [1990] describe the architecture of *a complete integrated agent*. Their agent, a little submarine named Homer, is able to navigate in a harbor full of other objects, to go to places, to deliver objects from one place to another. The submarine can take part in dialogues that include questions, assertions and commands.

> A command is a special case of an INFORM event in which the information transmitted is that the informer has a goal and wants the informee to achieve it for him. ... There is a demon in the reflective processes which reacts to general commands by extracting the goals, processing them, and sending them to the temporal planner for plan synthesis. [p.43]

A command is mapped onto a logical form which is an instantiation of a frame-like description of the verb, augmented by a description of the effects of the action on the world state — they call such semantics *state transition semantics*. Such representation is then sent to the temporal planner for plan synthesis; when a temporal plan has been synthesized, it is passed on to the plan interpreter for execution.

Homer accepts simple commands from the instructor and adopts them as its own goals: then it reasons about whether it can achieve them. One interesting point is that there can be time constraints attached to commands, concerning either when they should be performed — *Drop the package at the barge next Sat. at 9 p.m.*, or when they are relevant — *If you see an animal tomorrow, photograph it*: therefore the utterance situation and the performance situation don't coincide.

The system allows for *replanning*, which is used to deal with new declarative information which changes the parameters under which a plan had been formulated, and with additional goals, that may have to be achieved before another goal for which planning had already taken place. Homer does accept negative commands, which are considered as global constraints on the activities it can perform. For example, if Homer is told *Don't leave the island today*, and later, the same day, *Take a picture of the Codfish*, it will say it cannot comply with the latter instruction, because to do so it would be necessary to leave the island.

Although Homer's abilities are quite impressive, it is not clear whether replanning and the capacity of dealing with negative commands amount to something more than propagating temporal constraints.

3.3.4 Instructions as a way of solving impasses

A different point of view on understanding instructions is advocated in [Alterman *et al.*, 1991]. The stress is on extending stored knowledge to new but similar situations, as for example when knowledge about using a certain device — a home phone — is applied to the usage of a new, but similar device — a pay phone.

An agent has a basic set of skills and plans to solve certain problems. Instructions are used as a resource only when the stored knowledge about plans cannot be adapted to the situation at hand, namely, when the agent is blocked in mapping the goal to action: instructions come in to specify the appropriate action(s) that will satisfy the goal in that particular situation. The agent has to integrate these actions in his knowledge and to understand how they relate to the goal he was trying to achieve: this is the only paper I know which assumes a rich relation between the instructions and the preexisting goal(s). It is crucial that the communicative situation and the performance situation are the same: communication comes in to repair a breakdown in performance. From a certain point of view, Chapman's approach is not so different, in the sense that for Chapman as well instructions come in as an additional source of information about the course of action to follow, when the agent doesn't have enough information to choose one course of action over another.

3.3.5 Summary

The following table summarizes the approaches outlined above, as well as the one that I am taking. The column *direct mapping* refers to whether an instruction is *directly* mapped into some kind of logical form, which then is fed to an inference engine that interprets it; the two last columns are included for completeness although I will not be addressing the corresponding issues any further in this thesis:

Approach	direct mapping	relation between instructions and other goals	instructor in control	utterance sit. necessarily = performance sit.
Basic	Y	None	Y	Y
Chapman	NA	Y	N	Y
Vere, Bickmore	Y[a]	Limited	Y	N
Alterman	Y	Rich	N	Y
Di Eugenio	N	Rich	Y	N

[a] Apart from negative commands.

The NA in Chapman's entry stays for *Not Applicable:* in fact, instructions in Sonja are simply mapped into setting or resetting a buffer, and binding some associated parameters; it is hard to see how setting buffers could qualify as a logical form.

One conclusion that can be drawn from this table is that in general researchers have not worried about computing the description of the action to be performed; and that, apart from [Alterman *et al.*, 1991], not much attention is paid to the mutual interaction between the current instruction and preexisting goals, interaction that can affect not just planning as in [Vere and Bickmore, 1990], but the action to be performed itself.

The approach I am going to propose instead relies in an active way on such mutual interaction, and tries to make the agent's knowledge flexible enough so that the same instruction uttered in the context of different goals will result in different behavior.

Chapter 4

The linguistic data and their discourse functions

In this chapter, I will discuss the data I have analyzed, *Purpose Clauses* and *Negative Imperatives*, from a pragmatic / discourse processing point of view: for each of the two classes, I will describe the corpus, talk about some relevant literature, and discuss their respective discourse functions.

The connection between the two kinds of data is a theoretical one, in that both the goal expressed in a purpose clause, and a negative imperative can be seen as constraining the choices that an agent will be faced with when interpreting and / or executing instructions. Besides, not surprisingly goals affect the interpretation of negative imperatives as well. There is also a more direct connection, in that, purpose clauses may describe goals such as *prevent, avoid* etc. that bear a direct relation to negation.

As I mentioned in Ch. 2, my analysis of Purpose Clauses is much more developed than the one of Negative Imperatives; in particular, I haven't examined Negative Imperatives from the point of view of the inferences necessary to understand them yet. In the next chapter, I will present the rest of my analysis of Purpose Clauses, that regards the computational issues of action representation and inference.

4.1 Purpose Clauses

4.1.1 Corpus

First, a note on terminology. In the syntactic literature [Hegarty, 1990; Jones, 1991; Green, 1991], the term *purpose clause* is used to refer to an infinitival *to* clause attached as a daughter of VP, such as [1]

(4.1) *Mary$_i$ bought* [*a book$_j$*] [**e$_i$ to read e$_j$ on the plane**]

Here, I use the term *purpose clause* in a more informal way, to refer to clauses that express the purpose of the action described in the main clause: besides infinitival *to* constructions, other clauses, such as those introduced by *so that, such that, so as to*, [2] may also perform the same function. I have concentrated on infinitival clauses; the ones I will discuss are attached as daughters of S, not of VP, and are termed *rationale clauses* in syntax: all the data I will discuss belong to a particular subclass of such clauses, subject-gap rationale clauses. A use of the term *purpose clause* similar to mine is found in [Thompson, 1985].

My analysis is based on one hundred and sixty-four consecutive occurrences of purpose clauses, which I collected almost entirely from two how-to-do books on placing tiles and hanging wall covers [Hallowell, 1988a; Hallowell, 1988b], and from two craft magazines [CC, 1988; CWP, 1989]; four derive from opportunistic collection. These data can all be seen as instantiations of the patterns *Do* α *to do* β or *To do* β *do* α, where α is an action, or sequence of actions, and β is an action: as we will see shortly, whether the PC is pre- or postposed has some impact on interpretation. However, in the following I will often refer to the pattern underlying purpose clauses only as *Do* α *to do* β; the reader should keep in mind that both orders are possible, and that sometimes α stands for a sequence $\alpha_1, \alpha_2, \ldots, \alpha_n$.

Although purpose clauses can be introduced by *in order to*, I found no such instances. I collected seventy-three further instances of infinitival *to* constructions, but I didn't include them in my corpus. This set of examples includes:

- Fifty-six occurrences of the pattern *Use x to do* β, [3] such as

  (4.3) *Use a sponge dipped in lukewarm water* **to remove excess adhesive before it dries.**

  Although such examples can be seen as purpose clauses, as in [Thompson, 1985], I haven't included them in my corpus because α = *Use x* doesn't per se describe an

[1] Actually the infinitival clause in (4.1) is ambiguous between a purpose clause, which is daughter of VP, and an infinitival relative clause, which is daughter of NP: *Mary$_i$ bought [a book$_j$ [e$_i$ to read e$_j$ on the plane]].*

[2] And sometimes even *and*.

[3] To be precise, in three of them the verb *need*, not *use*, occurs:

(4.2) *You'll need a wide, soft paint roller or pasting brush to apply paste to each strip.*

action to do in order to achieve β, but rather the instrument with which to perform either β, or an action γ that generates or enables β.

- Seventeen occurrences of miscellaneous infinitival constructions. Most of them are purpose clauses in the syntactic sense, namely, they are daughters of VP, rather than of S. An example is [4]

  (4.4) *Before installing the panels, paint* [*a stripe*$_i$ *on the wall between the joints*] [**e**$_i$ **to blend with the color of the panel groove**].

  Others are complements of degree phrases, such as

  (4.5) *Remove any excess backing or adhesive from the floor until the surface is smooth and deep enough* **to install the new tile.**

  And finally, in some of these, the main clause is not in an imperative form:

  (4.6) *The outer edges of wood tiles in an exterior doorway should be carefully sealed before the threshold is replaced,* **to minimize water penetration.**

  This latter kind of examples could be included in my corpus by simply turning the main clause into the active voice; however, I decided to exclude them for the time being, as I want to base my analysis on instances in which the main clause is in the imperative form.[5]

4.1.2 Purpose Clauses in the literature

Purpose clauses have received attention almost exclusively from a syntactic point of view – see for example [Hegarty, 1990; Jones, 1991; Green, 1991].

Very little attention has been paid to purpose clauses in the semantics literature. Jones in [1985; 1991] gives a syntactic and semantic account of purpose clauses in a government and binding framework, concerning himself in particular with control and thematic relations; Jackendoff in [1990] briefly analyzes expressions of *purpose, goal, or rationale, normally encoded as an infinitival, in order to-phrase, or for-phrase*. He represents them by means of a subordinating function FOR, which has the adjunct clause as an argument; in turn, FOR plus its argument is a restrictive modifier of the main clause. However, Jackendoff's semantic decomposition, as Jones's, doesn't go beyond the construction of the logical form of a sentence.

In the pragmatics literature, Thompson addresses the problem of the difference in meaning between *pre-* and *post*posed purpose clauses; her findings have been applied to NL generation in [Vander Linden *et al.*, 1992a; Vander Linden *et al.*, 1992b; Vander Linden, 1993b]. Purpose clauses are among the adjunct clauses generated in the work reported in [Huettner *et al.*, 1987].

[4] (4.4) could also be interpreted as an infinitival relative.

[5] I do include in my corpus examples in which the main clause is not in imperative form in a grammatical sense but does have imperative force, e.g. because of modals: *Within 4 hours of the time the tile was laid, you must roll it to set the tile fully and to level any pieces that have popped up.*

In a computational setting, the most extensive work on purpose clauses has been done by Balkanski [1992a; 1992b; 1993]: she mainly concentrates on the intentions and beliefs that an agent adopts after hearing a purpose clause. I will discuss Balkanski's work in Sec. 4.1.2.2.

4.1.2.1 Thompson's pre- and postposed Purpose Clauses

In [1985], Thompson addresses the problem of whether initial and final purpose clauses perform different discourse functions; her answer is that in fact they are two different constructions, not just the same clause type appearing in two different positions. She claims that the *initial* purpose clause —IPC— states a problem within the context of the expectations raised by the preceding discourse, while the *final* purpose clause —FPC— has a much more local role of stating the purpose for which the action named in the immediately preceding clause is performed.

She counts as purpose clauses infinitival constructions introduced by *in order to*, or by *to* paraphrasable as *in order to*. [6] In Thompson's analysis, an IPC provides the framework in which to interpret the main clause, and it does this by its role as a link in an expectation chain:

1. The environment, including the text itself, as well as the knowledge which the reader brings to it, creates a set of expectations.
2. Within this set of expectations a problem arises.
3. The purpose clause names this problem and raises further expectations about its solution.
4. The following material fulfills these expectations by providing the solution.

The FPC instead has simply a local role of stating the purpose for which the action named in the preceding clause is undertaken. The scope of the FPC is restricted to its immediately preceding main clause, which must name an action performed by a volitional agent.

An example that nicely supports her analysis is:[7]

(4.7) *Brendan was rushing madly farther and farther out to sea.* **To slow her down** *we streamed a heavy rope in a loop from the stern and let it trail in the water behind us to act as a brake, and, hopefully, to smooth the worst of the wave crests. From the stern also dangled a metal bucket; only twenty-four hours earlier we had been using it* **to cook an excellent meal of Irish crabs.** *Now it clanked mournfully every time a wave broke against it.*

Thompson's analysis is as follows [p. 62]:

[6] What she means is that she counts only rationale clauses, excluding other infinitival *to* constructions.
[7] From *The Brendan Voyage.* Timothy Severin, 1978, New York: McGraw-Hill.

> The text preceding the first purpose clause **to slow her down** creates a set of expectations, namely that the crew are in a dangerous situation ... The initial purpose clause **to slow her down** names a problem compatible with this set of expectations ... Naming the problem in turn raises expectations about its solution. Providing the solution is the function of the text following the purpose clause.
>
> For contrast, let us compare the final purpose clause ... **to cook an excellent meal of Irish crabs.** Here there is no set of expectations created by the environment; nothing in either the preceding text or the knowledge which the reader brings to the text suggests anything about cooking Irish crabs. Further, there is no PROBLEM to be solved ... The clause **to cook an excellent meal of Irish crabs** simply provides, in a very local way, the purpose for which the metal bucket was used.

Thompson's conclusions also make sense from the point of view of *given / new* information [Clark and Haviland, 1977; Prince, 1981]. Clark and Haviland in [1977] first observe that focal stress always falls on an element in the constituent that conveys new information; then they note that in general given comes before new, since simple English sentences with normal intonation have their focal stress at or near the end of the sentence.

Therefore, the (given) expectations that arose from the preceding text can be expressed by means of an IPC, while the (new) method to achieve that goal is expressed by means of postposed main clauses.

Although Thompson's analysis is plausible, I would contend that sometimes an FPC can also serve as a text organizer, that it may name a problem to be solved, and that it may encompass more than simply the previous main clause. Consider [8]

(4.8) *Use premixed vinyl adhesive* **to install fabrics laminated to paper backing.** *Trim all selvages before hanging. Then spread paste on the fabric's backing and hang, butting edges. To prevent staining, keep adhesive off the fabric surface.*

The FPC *to install fabrics laminated to paper backing* does indeed express the purpose of *using premixed vinyl adhesive*; however, the rest of the paragraph still belongs to the subdiscourse referring to the "problem" of *installing fabrics laminated to paper backing*. The same holds for the example we are animating in AnimNL, although it is a constructed one:

(4.9) *Go into the kitchen* **to get me the coffee urn**. *Before you pick it up, be sure it is unplugged. When you bring it back here, carry it with both hands.*

The goal *get me the coffee urn* names the common goal, or problem, in the context of which the whole paragraph has to be interpreted.

[8] I am using here an instance of the construction *Use x to do β* because, although I exclude such examples from my PC corpus, Thompson does include them in hers.

It seems to me that the position of the clause in the sentence is a function of both the *given / new* distinction and of the relation between α and β. I will come back to this topic in Sec. 4.1.4.

An interesting finding of Thompson's is that in what she calls procedural text — i.e., what I call instructional text — the ratio of IPC's to FPC's in higher than in narrative text, being 40% and 12% respectively. I haven't examined non-procedural text; however, in my corpus the percentage of IPCs to FPCs is about 52% (see Table 4.1), thus higher but still compatible with the percentage reported by Thompson. She explains this by noticing that procedural texts are more strongly organized in terms of the types of expectation chains that give rise to initial purpose clauses than are non-procedural texts. That is, when one is describing the method for accomplishing a certain end, a convenient way to do this is to state at the outset what the intended goal is, and then describe how one reaches it. She concludes that this is precisely what the use of initial purpose clauses accomplishes.

4.1.2.1.1 IMAGENE I would like now to spend a few words on Vander Linden's work on generating purpose expressions [1992a; 1992b; 1993a] — discussion of his work is relevant at this point because he relies on Thompson's findings.

Vander Linden is interested in generating purpose expressions, which include, besides purpose clauses, *for nominalizations*, as in "Follow the steps in the illustration below, *for desk installation*" [1992a, p.185,ex.3b]; and *for gerunds*, as in "The OFF position is primarily used *for charging the batteries*" [1992a, p.185,ex.3c]. Purpose expressions are generated by traversing two decision networks, [9] the *Purpose-Slot* network, that determines the position of the purpose expression in the sentence, and the *Purpose-Form* network, that is concerned with the form of the expression — purpose clause, for nominalization, or for gerund.

The Purpose-Slot network places the purpose clause / expression in final position in most cases, unless

1. There is more than one action that the purpose pertains to — this is consistent with Thompson's findings, in that Thompson notes that the initial purpose clause raises expectations about the solution to the problem it names, and this solution may often take a number of clauses or even sentences to describe.

2. The purpose is optional — however, I am not sure how *optionality* is determined, as Vander Linden leaves this notion as rather vague.

3. The purpose is contrastive with respect to the previous text.

The Purpose-Form network determines the grammatical form of the purpose expression. Clearly, there are interactions with the Purpose-Slot network: if the purpose expression's scope includes more than one action, and therefore the Purpose-Slot network has determined that the purpose expression should be preposed, the *to* infinitive form is used.

[9] To be precise, they are system networks from systemic functional grammar.

> The remainder[10] of the **Purpose-Form** sub-network is based on the observation that instructional text is oriented toward reader actions. Because purpose clauses / phrases are not reader actions, they tend to be demoted to phrase status whenever possible. Thus, the **purpose-nominal-Form** system will realize a prepositional phrase with a nominalization as the complement whenever possible. [...] Even if a nominalization exists, however, it still may not be used depending upon the determination of the **nominal-complexity** system. This system, based on the examples in our corpus, restricts nominalizations to a single, non-complex argument. [1992b, p.11-12]

Vander Linden has implemented his results, that are derived form a corpus study of instructional text regarding phone installations, in a generation system called IMAGENE. IMAGENE's goal is to produce an instructional text generation system that is capable of taking the output of a planner and producing the appropriate text. IMAGENE's focus is on the construction of realistic rhetorical structures and the generation of the appropriate syntactic structure corresponding to them.

Some observations on Vander Linden's work:

1. I disagree with the statement quoted above that *purpose clauses / phrases are not reader actions*, as all my 164 purpose clauses do express reader actions. This of course may depend on the different corpora we have looked at, but I suspect that by *reader action* Vander Linden means *basic actions* only, namely, those that are actually physically executed. Also, from his own analysis — Table 1 in [Vander Linden *et al.*, 1992b, p.11] — it results that only in the 40% of the cases a purpose is expressed by means of a phrase: this makes the statement *they tend to be demoted to phrase status whenever possible* not totally convincing, given that for one reason or the other in the majority of cases, 60%, purposes are *not* demoted to phrase status.

2. They distinguish the *Precondition* and *Result* relations from *Purpose*, and they don't use infinitival *to* constructions to express the former two. This distinction is linked to the corresponding relations in Rhetorical Structure Theory, but it seems to me that purpose expressions can also express *Precondition* and *Result*. In fact [Vander Linden, 1993a, p.141] explicitly identifies purpose with the expression of the *generation* relation. However, in Sec. 5.1, I will show that PCs can express relations other than generation, and that therefore they could be used also to encode the *Precondition* and *Result* relations.

3. Finally, and more importantly for me, they don't address the issues of the purpose constraining and augmenting the description of the action described in the main clause: from the point of view of generating instructions, this would amount to deciding what can be left out of an action description contained in a main clause because it can be "easily" inferred from the associated purpose clause.

[10] "Remainder" in the sense of "what is not dependent on the Purpose-Slot network".

4.1.2.2 Balkanski's model

Balkanski in [1992a; 1992b; 1993] examines the Speaker's (S) and the performing agent's (G) beliefs and intentions concerning means clauses and rationale clauses, and the relations between actions expressed in such clauses. Means clauses (MeaCs) express the means by which an action is performed, as in (4.10a)

(4.10a) *John dirtied the carpet* **by walking across the room.**

(4.10b) *John walked across the room* **to dirty the carpet.**

As I mentioned in Sec. 4.1.1, rationale clauses are a superset of the PCs I have been talking about, and, to avoid generating confusion, I will talk about PCs in the following — (4.10b) contains such a clause.

In my opinion, Balkanski's important contribution is her analysis of S's and G's beliefs and intentions in the presence of this kind of data, which she mainly collects from narrative text — the bulk of her corpus is derived from Associated Press data. The main claim Balkanski puts forward in [1993] is that an account of the meaning of complex utterances about multiple actions requires consideration of mental states; and that the meaning of utterances with rationale clauses and means clauses consists not only of action descriptions, but also of beliefs and intentions of the agents involved, regarding these actions and relations among them.

Before reporting her conclusions, some observations on her notation: α refers to the generating action and β to the generated action — see the forthcoming discussion of *generation* in Sec. 5.1.2. Therefore, utterances containing a MeaC can be schematized as *Do* β *by doing* α, namely, the generating action α is described in the MeaC, and the generated action β in the main clause; the converse is true for a clause containing a PC, that can be schematized as *Do* α *to do* β.

[Balkanski, 1992b] notes that:

1. In both types of utterances, S believes that α occurred (depending on tense, is occurring, or will occur) — in fact, in both (4.10a) and (4.10b), S believes that John walked across the room.

2. In utterances with MeaCs, but not necessarily in those with PCs, S believes that β occurred (is occurring, will occur) — so in (4.10a) S believes that John dirtied the carpet, but this is not necessarily the case for (4.10b).

3. In utterances with MeaCs, but not necessarily those with PCs, S believes that α *generates* β. In fact, the relation between α and β in an utterance containing a PC may be the more general *contributes*, as Balkanski mentions in [1992a], and as I will show in the following.

4. In utterances with PCs, but not necessarily those with MeaCs, S believes that α was (is, will be) intended on the part of G — in fact, *but he didn't even realize it* is a perfectly good continuation for (4.10a), but slightly puzzling for (4.10b).

5. In utterances with PCs, but not necessarily those with MeaCs, S believes that β was (is, will be) intended on the part of G.

6. In utterances with PCs, but not necessarily those with MeaCs, S believes that α was (is, will be) intended on the part of G *as a way* of generating β. In the PC case, therefore, but not necessarily in the MeaC case, S believes that G expects (at the start of action time) the relevant generation-enabling conditions to hold.

Balkanski also presents a processing framework composed of

- logical forms that reify actions; for example, the LF for (4.10a) is
  $\exists\ x_1,\ x_2,\ dirty(x_2) \wedge agt(x_2,John) \wedge obj(x_2,\ carpet) \wedge past(x_2) \wedge walk(x_1) \wedge loc(x_1,Room) \wedge by(x_2, x_1)$

- standard axioms about beliefs and intentions;

- interpretation rules that define the meaning of PCs and MeaCs, in terms of the beliefs and intentions of the speaker and actors. For example, the rule for MeaCs is
  $LF1:\ BEL(S,t,\ by(\beta,\alpha) \rightarrow GEN(\alpha,\beta) \wedge Occur(\alpha))$

  > The LF1 rule maps the LF representation of means clauses to the generation relation, capturing the fact that the speaker of an utterance with a means clause believes that the occurrence of the main clause action follows from that of the adjunct clause action. [1992b, p.300]

In my opinion, Balkanski's processing framework is sufficient to support her claims, but not to shed light on the issues that I am mostly interested in, namely, computing the constraints that β imposes on α in an utterance containing a PC. Consider her example (4.10b): from such an utterance, we would like to infer that John walked across the room **on the carpet**; this is an inference analogous to constraining the interpretation of *between two ladders* in (2.1b). However, nowhere in her framework does she provide the means to account for this kind of inference.

I will return to her work in Secs. 5.1.2 and 5.1.3, when discussing generation and enablement.

4.1.3 Discourse functions of Purpose Clauses

There are several dimensions along which PCs may be viewed. One is the general functions they perform in discourse. Another is the kinds of actions they describe; still another is what kind of relations between actions they express.

Regarding the functions that PCs perform in discourse, the reader should by now be convinced that S uses a purpose clause to explain to H the goal β to whose achievement the execution of α contributes. Thompson finesses this by attributing different functions to purpose clauses according to their position in the sentence. As I showed above, such

different functions are after all not so clear cut; besides, I think they fall out of the possible relation holding between α and β, as I will show in Sec. 5.1.4.2.

As I already mentioned in previous chapters, purpose clauses perform another very important function, that of expressing that the goal β *constrains* the interpretation of α. This effect is very clear in examples such as (2.1b) — *Place a plank between two ladders to create a simple scaffold* — and (2.2b) — *Cut the square in half along the diagonal to create two triangles*, discussed in Ch. 2. However, it is present in basically every example, in that the purpose clause often provides the degree or the temporal extent to which α has to be performed, as in (4.11), where the quantity of heat applied and the rate of the heating process are affected by the goal *to soften the adhesive*:

(4.11) *If the tiles don't come up easily, warm them up* **to soften the adhesive**.

As far as the actions α and β are concerned, given that a purpose clause expresses the goal to which the action described in the main clause contributes, there is a sense that α is in some kind of hierarchical relation to β. In fact, purpose clauses relate action descriptions at different levels of abstraction, such as a physical action and an abstract process, or two physical actions, but at different levels of granularity; in (4.12) [11] *vacuum* and *dust-mop* can be seen as specializations of *clean*:

(4.12) *Vacuum or dust-mop your parquet* **to clean it.**

Once one has defined one's corpus to be limited to *Do* α *to do* β — and of course *To do* β, *do* α — where both α and β are actions, it is useful to have a closer look at what kinds of goals can appear as β. Table 4.1 shows the three classes of goals I found.

	bring about event		prevent event	affect H's knowledge	
	create	others			
IPCs	3	63	7	12	85
FPCs	5	46	14	14	79
	8	109	21	26	164

Table 4.1: Distribution of Purpose Clauses

Bring about event. Most examples —117, namely, 71%— are occurrences of the most obvious type of goal, namely, causing an event to happen in the world.

[11] (4.12) is a constructed example.

Among the examples of β describing events happening in the world, I singled out the few (only 8, which amounts to 4% of the total, and 6% of this type) that pertain to the creation of an object [12] — they are labelled as *create* in the table. From a discourse processing point of view, interpreting purpose clauses of this kind affects the discourse model by introducing new referents. This happens when the effect of α is to create a new object, and β identifies it. Verbs frequently used in this context include *create, make, form.*

(4.13) *Join the short ends of the hat band* **to form a circle**.

Similarly, in Ex. (2.2b) the discourse referent for the triangles created by cutting the square in half is introduced.

Prevent event. The purpose clause may inform H that the world should *not* change, namely, that a given event should be prevented from happening — I have 21 such examples, corresponding to 12% of the total:

(4.14) *Tape raw edges of fabric* **to prevent threads from raveling as you work**.

Verbs used are *prevent, avoid*. These example are interesting in that they make possible side-effects of actions explicit, and are therefore related to certain kinds of *negative imperatives* — see Sec. 4.2.

Affect H's knowledge. In 26 examples out of 164 —16% of the total— the PC expresses a change that will affect H's knowledge, rather than the world: by executing α, H can change the state of his knowledge with respect either to the value of a certain entity or of a certain proposition.

There are 17 examples, or 65% of this class, of the first kind, in which the goal describes ascertaining the value of an entity:

(4.15) *You may want to hang a coordinating border around the room at the top of the walls.* **To determine the amount of border,** *measure the width (in feet) of all walls to be covered and divide by three. Since borders are sold by the yard, this will give you the number of yards needed.*

Notice that in (4.15) the referent for *amount of border* is introduced. These examples have an effect on the discourse model similar to those belonging to the *create* subclass: in fact, although the NP object of *determine, establish* etc. is generally definite, in only one case does it refer to a referent already established in the discourse model.

[12] I found very few examples in which an object is destroyed, and they generally regard stains or spills to be removed. This may be a characteristic of the kind of instructional text I have examined. Of course even in examples such as (2.2b) the square doesn't exist any more after the cutting action, but it is still possible to refer to it.

The 9 examples that concern ascertaining the value of a proposition involve verbs such as *check, make sure* etc. followed by a *that*-complement describing a state ϕ. The use of such verbs has the pragmatic effect that not only does H check whether ϕ holds, but, if ϕ doesn't hold, he will also do something so that ϕ comes to hold.

(4.16) *To attach the wires to the new switch, use the paper clip to move the spring type clip aside and slip the wire into place. Tug gently on each wire* **to make sure it's secure.**

4.1.4 Expressing purpose with different connectives

In this section, I will discuss an informal experiment that I ran to get a first handle on some questions related to the pragmatic functions of PCs. The results are very preliminary, but I report them because they suggest some interesting research directions.

• A couple of observations that I found in [Huettner *et al.*, 1987, p.208-209] seem to me to deserve further consideration — notice that in the following quote the terms *Rationale Clause* and *Purpose Clause* are used in their syntactic sense:

> Easily confused with the purpose clause is the rationale clause (RatC), also known as an "in order to" clause or result clause. ... A RatC reading may always be paraphrased with *in order* ... Finally, RatC are daughters of S, and not of VP, and may therefore be preposed alone ((4.17b)) or otherwise isolated from the VP ((4.17c)):[13]
>
> (4.17a) *Helga carries a hat pin to protect herself.*
>
> (4.17b) *To protect herself, Helga carries a hat pin.*
>
> (4.17c) *What Helga does to protect herself is carry a hat pin.*

The two questions arising from the previous quote are then:

1. Whether it is true that a PC (RatC) can always be paraphrased with *in order to*.
2. Whether there is any pragmatic difference between pre- and postposed PCs: the discussion on Thompson's findings in Sec. 4.1.2.1 already pointed out that this is the case, even if syntactically (4.17a) and (4.17b) are both acceptable.

• Another hypothesis I wanted to test is the following: it had been brought to my attention [14] that PCs could be paraphrased with *so as to* and with *in order to*, but that not both paraphrases were always acceptable. This observation is rather surprising, given that for instance in The American Heritage Electronic Dictionary, 1991, the two are considered interchangeable: *so as to* is defined as *in order to*, and *in order to* as *for the purpose of; so that*.

[13] My numbering.

[14] Mark Steedman, p.c.; Sheila Rock, p.c.

• Finally, another question is whether a paraphrase with a Means Clause — MeaC for short, a gerund introduced by *by* — is felicitous, modulo Balkanski's observations about the different beliefs and intentions conveyed by the different connectives.

I ran an informal experiment in which I asked 11 native speakers to judge the pragmatic felicity of various possible paraphrases on 10 of my naturally occurring examples; the judgements were on the scale [0-3], with 0 totally unacceptable, and 3 totally felicitous. 8 informants answered, although one of them didn't answer all the questions.

I distinguished final and initial purpose clauses. As far as FPCs — *Do α to do β* — are concerned, I proposed 6 examples, each with 4 possible paraphrases: one paraphrase each for *so as to* and *in order to*, as in (4.18b) and (4.18c), and two for the *by* paraphrase — one corresponding to the original order, the other that locates the adjunct in the preferred postposed position — as in (4.18d) and (4.18e). (4.18a) is the naturally occurring example.

(4.18a) *Line the tub with cardboard* **to** *prevent debris from clogging the drain.*

(4.18b) *Line the tub with cardboard* **so as to** *prevent debris from clogging the drain.*

(4.18c) *Line the tub with cardboard* **in order to** *prevent debris from clogging the drain.*

(4.18d) **By lining** *the tub with cardboard prevent debris from clogging the drain.*

(4.18e) *Prevent debris from clogging the drain* **by lining** *the tub with cardboard.*

I proposed 4 examples of initial PC — *To do β do α* — and for each of them three paraphrases:

(4.19a) **To** *hang the wallpaper, follow the steps on page 47.*

(4.19b) **So as to** *hang the wallpaper, follow the steps on page 47.*

(4.19c) **In order to** *hang the wallpaper, follow the steps on page 47.*

(4.19d) *Hang the wallpaper* **by following** *the steps on page 47.*

The results are plotted, by connective, in Figs. 4.1, 4.2, 4.3; and by final / initial PC in Fig. 4.4. In all plots, each point represents the cumulative number of judgements, summed across all informants, for that level of judgement: the total number of judgements is 48 for FPCs (actually it is 47 for the paraphrase with a MeaC in initial position), and 29 for IPCs. The data of the experiment are reported in Appendix B.

Although the numbers here are too small, I would like to point out some suggestive, if not definitive, conclusions.

As far as the connectives go, the results suggest:

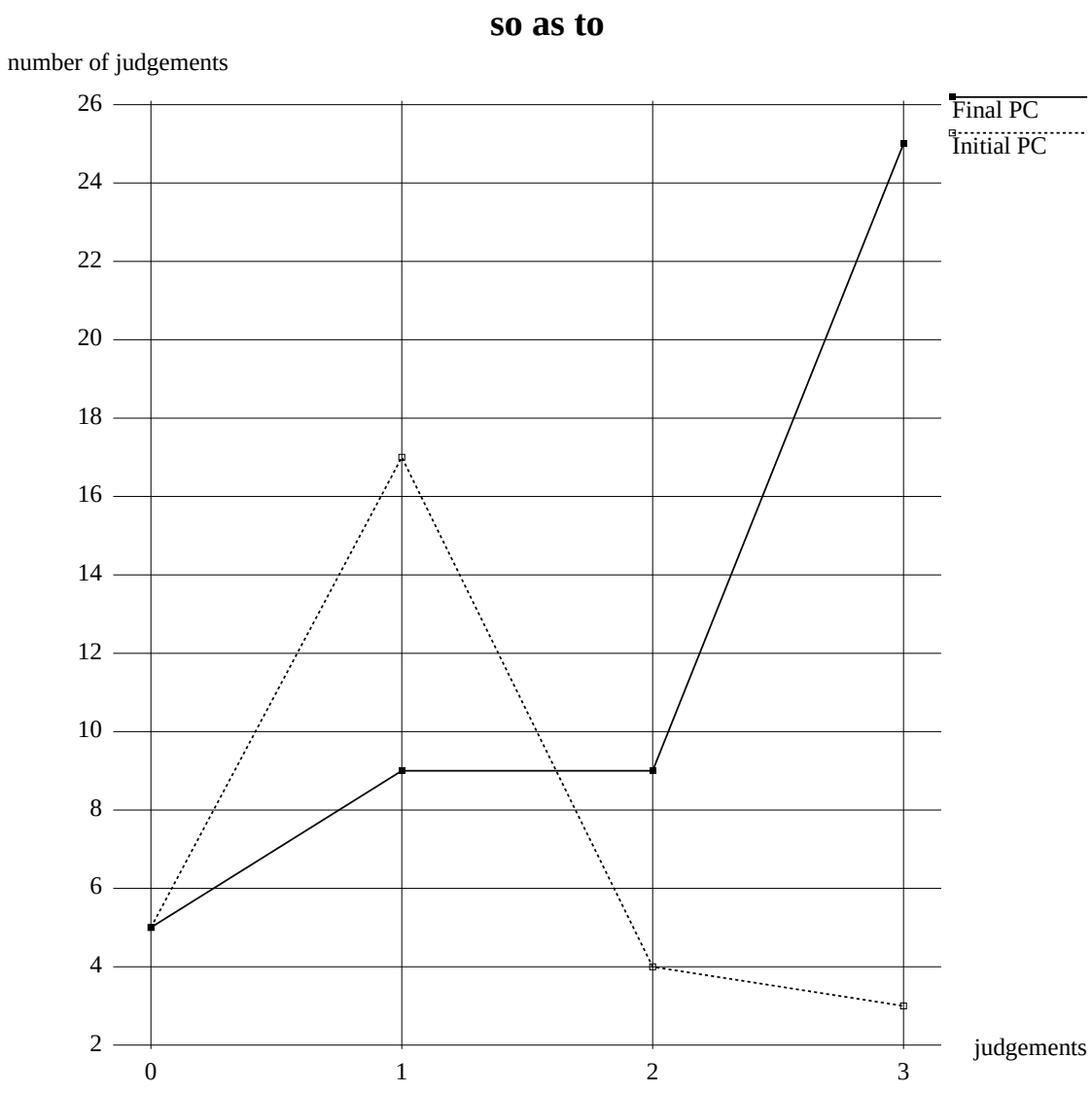


Figure 4.1: Distribution of *So as to*

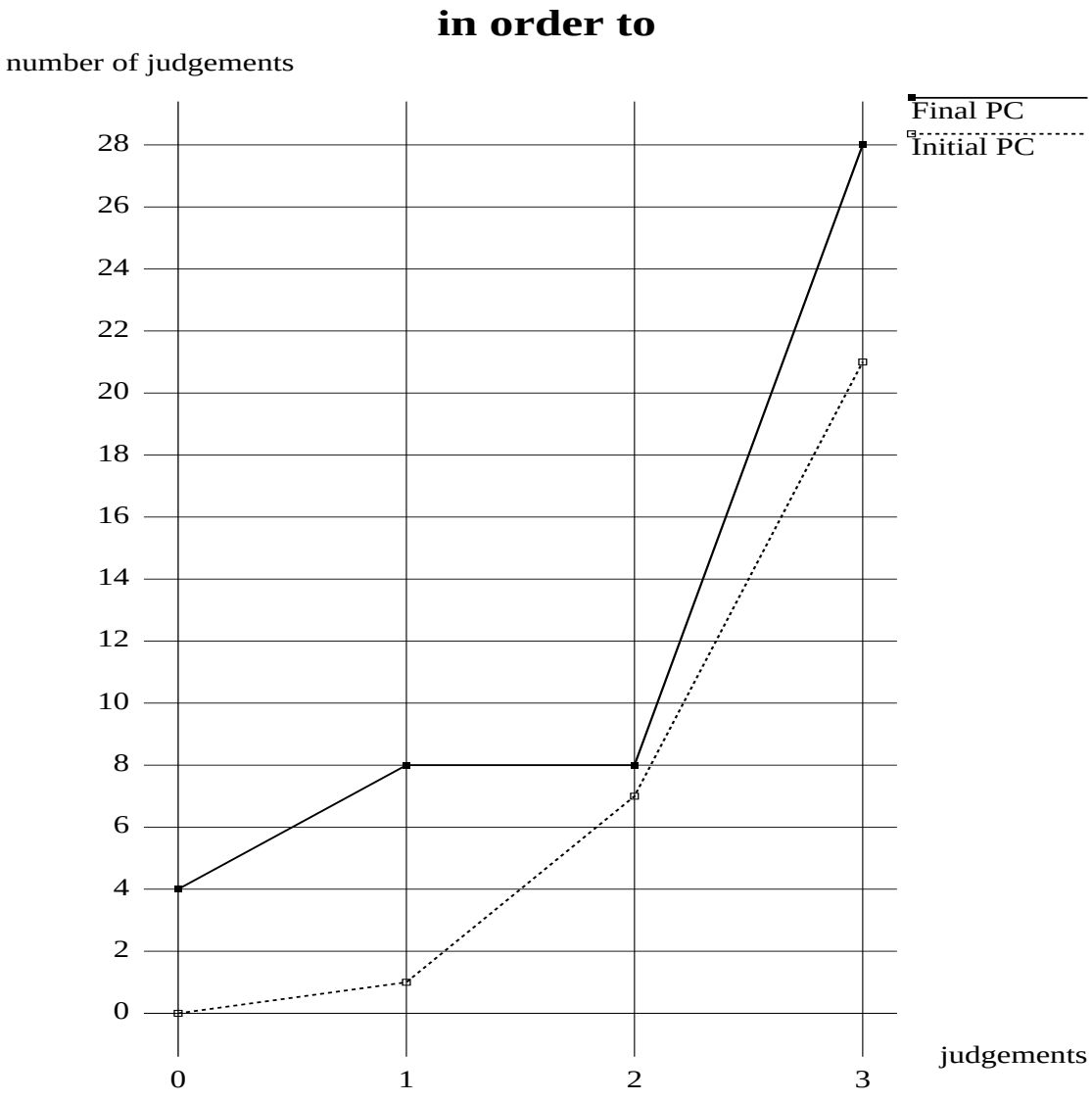


Figure 4.2: Distribution of *In order to*

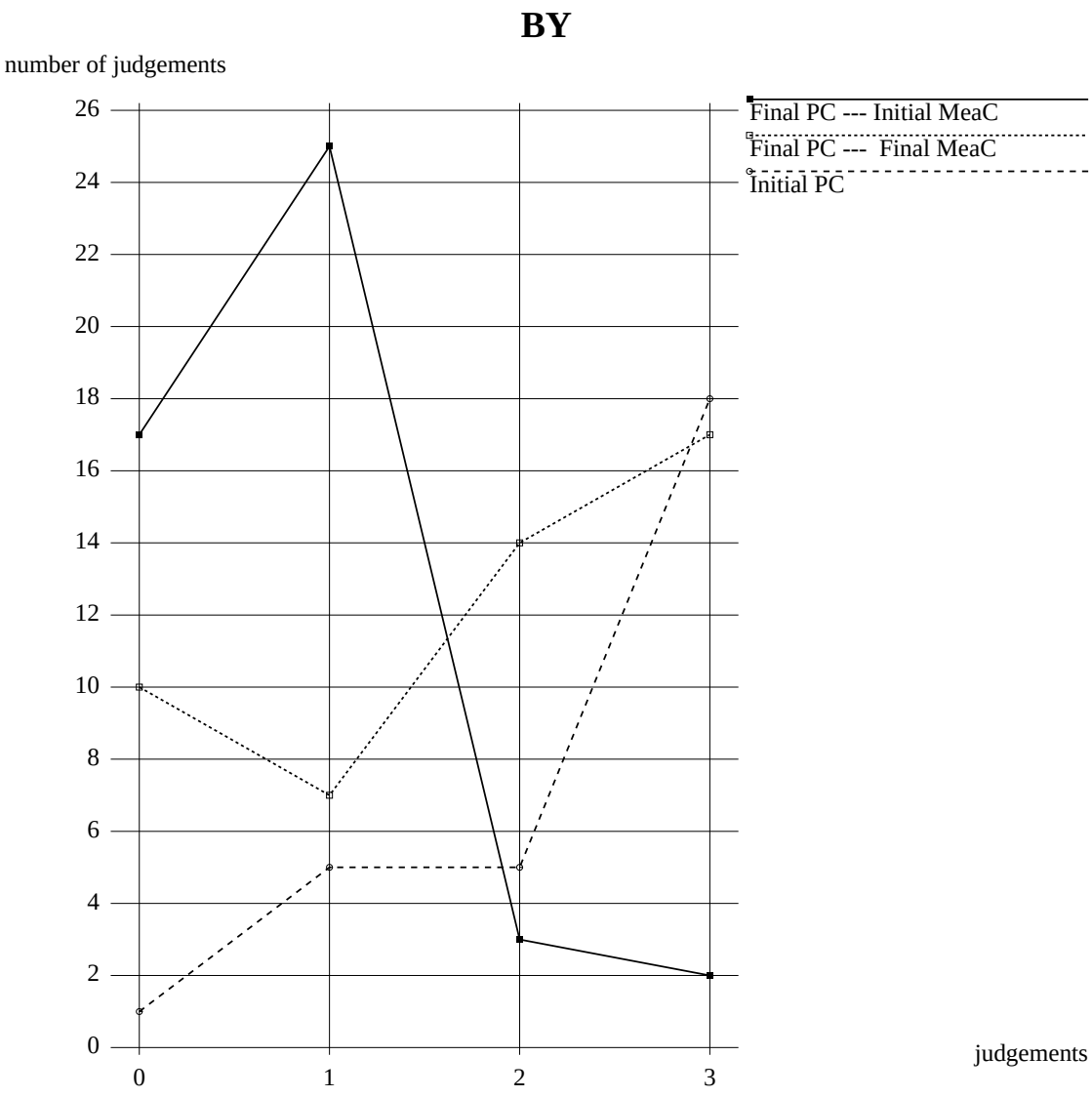
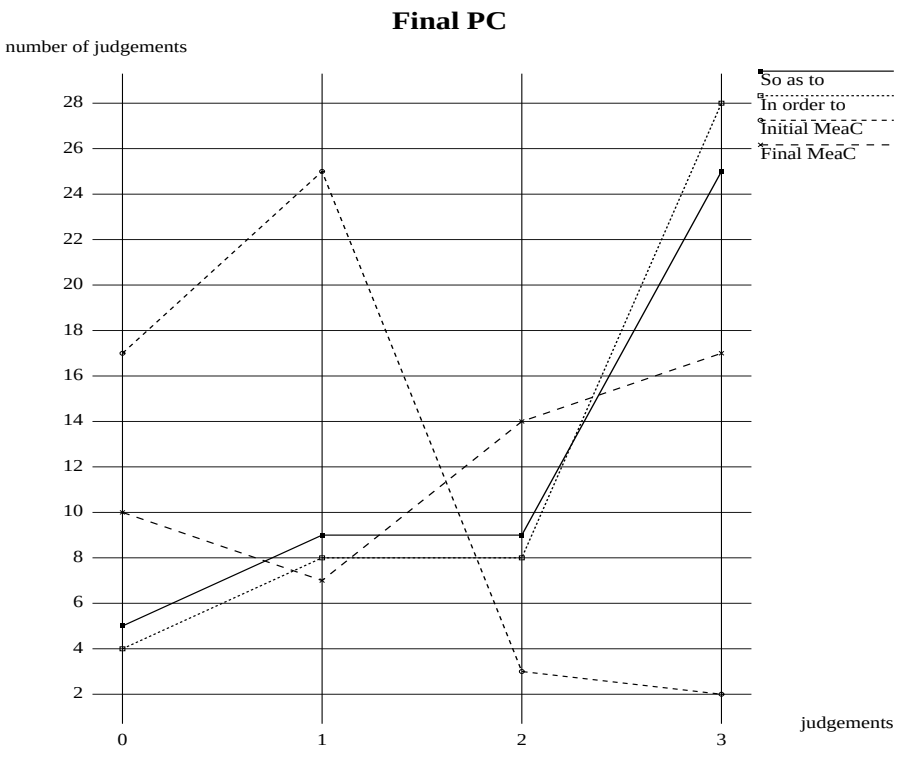


Figure 4.3: Distribution of *By*



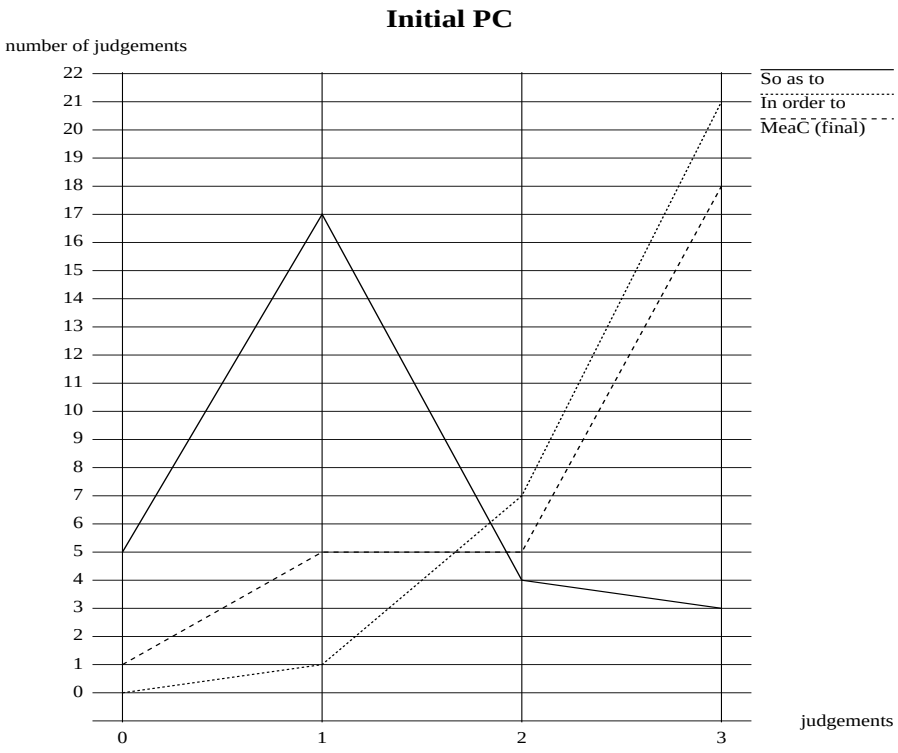


Figure 4.4: Distribution of Final and Initial PCs

So as to. Consider Fig. 4.1. The distributions of judgements for paraphrases with *so as to* in the case of FPCs and IPCs are considerably different. I split the judgements into two categories, summing up on the one hand 0's and 1's, and on the other 2's and 3's: in the case of FPC a paraphrase with *so as to* is considered acceptable in 70% of the cases, but in the case of IPC only in 24% of the cases. This difference is significant, as it was confirmed by running the χ^2 test, which yield $p < .001$.

In order to. The hypothesis that PCs can always be paraphrased with *in order to* is basically borne out. In particular, a paraphrase with *in order to* is considered acceptable in 75% of the cases for FPCs, and in 97% of the cases for IPCs. The difference between these two distributions is significant, $p < 0.02$. Below I will discuss the cases in which a paraphrase of an FPC with *in order to* is not considered as good.

By. In FPCs, the acceptability for a paraphrase in which the MeaC is in initial position is very low, being only 10%. It is much higher for the paraphrase in which the MeaC is in final position: 64%. The statistical significance is very high, $p < .001$.

In the case of an initial PC, the acceptability is 79%.

I'll turn now to the distributions of the different connectives separately for FPCs and IPCs. In my opinion, the two graphs in Fig. 4.4 are quite striking.

FPCs The judgement distributions for *so as to* and *in order to* are very similar, and in fact the difference between these two distributions is not statistically significant. The judgement distribution for a MeaC in final position doesn't seem to suggest anything on whether this kind of paraphrase is generally acceptable or not; however, there is a significant difference with the two distributions for *so as to* and *in order to* — $p < .001$.

Finally, the judgement distribution for MeaC in initial position shows that this is not an acceptable paraphrase. We already saw that the difference between such distribution and the distribution for MeaC in final position is statistically significant; such difference is also significant with respect to the distributions for *in order to* and *so as to* in this graph, $p < .001$.

IPCs It is *so as to* that becomes unacceptable, while *in order to* and MeaCs have very similar distributions. However, the difference between the latter two is still statistically significant, $p < .05$. The difference with the distribution of *so as to* is vastly greater, $p < .001$.

Thus, the answers to the three questions I set up at the beginning appear to be:

1. It is true that PCs can be paraphrased (almost) always with *in order to*.

2. There is no significant difference between paraphrases with *so as to* and *in order to* for FPCs; however, *so as to* becomes unacceptable in the case of IPCs. This latter effect may actually be due to other factors, such that *so as to* may require a discourse context to link into — in Ex. (4.20) *so as to* seems more acceptable:

   (4.20) *There are a few problems with washing this fabric: so as to avoid the color running, wash it in cold water.*

3. As far as paraphrases with MeaCs are concerned, those with the MeaC in initial position are not acceptable. In general it appears that for both IPCs and FPCs the paraphrase with a MeaC in final position is acceptable, although it is not judged as good as the paraphrases with infinitival connectives.

   A last remark on a paraphrase with a MeaC in the case of an IPC: often the scope of an IPC will encompass more than just the following main clause, namely, the pattern will be: *To do* β, *do* α_1*; do* α_2*; ... do* α_n. The question is then, what should the paraphrase with a MeaC be, *Do* β *by doing* α_1*; do* α_2, *... , do* α_n or *Do* β *by doing* α_1, α_2, *... ,* α_n? In my experiment, I had one such example, and I proposed both alternatives to my informants, although the graphs in Figs. 4.3 and 4.4 only include the former. In any case, the judgements were not enlightening, as informants judged the two paraphrases exactly in the same way, with 4 considering both unacceptable and 3 considering both acceptable — see page 193 in Appendix B.

Clearly these results ask for an explanation in terms of the semantics/ pragmatics functions of the connectives and the actions described — I will try such an analysis in Sec. 5.1.4.2, after I discuss relations between actions.

Let me conclude by pointing out the two cases of FPC in which the paraphrase with *in order to* was considered less acceptable than in other cases. The first one is

(4.21a) *Work* **to** *finish half a room at a time.*

(4.21b) *Work* **so as to** *finish half a room at a time.*

(4.21c) *Work* **in order to** *finish half a room at a time.*

(4.21d) **By working** *finish half a room at a time.*

(4.21e) *Finish half a room at a time* **by working**.

I think (4.21a) is an anomalous case, in that 7 informants out of 8 judged the paraphrase with *in order to* unacceptable, and 8 out of 8 the one with the final MeaC unacceptable. Instead, *so as to* was judged acceptable by 5 informants out of 8. I think the infinitival *to* here really conveys a sense of performing α *in a certain way*, rather than to achieve β.

The other problematic example is

(4.22a) *Go into the kitchen* **to** *get me the coffee urn.*

(4.22b) *Go into the kitchen* **so as to** *get me the coffee urn.*

(4.22c) *Go into the kitchen* **in order to** *get me the coffee urn.*

(4.22d) **By going** *into the kitchen get me the coffee urn.*

(4.22e) *Get me the coffee urn* **by going** *into the kitchen.*

Also in this case basically every paraphrase was poorly judged: 6 unacceptable for *so as to*, 4 unacceptable for *in order to*, 6 unacceptable for a final MeaC. However, here I think the results are strongly linked to the relation between actions that is expressed by the purpose clause, as I will explain in Sec. 5.1.4.2.

4.2 Negative Imperatives

The reader may recollect that negative imperatives, by telling the agent what he should *not* do, are one explicit way of identifying and pruning the choices an agent is faced with while acting. In Sec. 2.2, I put forward the hypotheses that a negative imperative is produced

1. When S expects H to overlook a choice point. The choice point is sometimes identified through a side effect that the wrong choice will cause.
2. When S expects H to choose the wrong alternative among many, possibly infinite, ones. Namely, S expects H to be aware of that choice point, or to become aware of it while executing the instructions.

In this chapter, I will show how such hypotheses are borne out by showing the correlation between two particular surface forms and the two hypotheses above.

4.2.1 Corpus

I collected data from the same two "how-to-do" books I used for purpose clauses, plus a few from detergent and toiletry containers. I collected 70 instances of negative imperatives, that I have divided into two classes: negative imperatives proper, characterized either by the negative auxiliary *don't* [15], or by negative polarity items, such as *never* and *nothing*; the other class is formed by verbs such as *take care, be sure* and the like – *TC verbs* for short – followed by a negative infinitival complement, such as *"not to ..."*, or *"to ... negative polarity item ..."*. I will call the former class of imperatives the *DONT* type and the latter the *neg-TC* type.

The 70 instances are distributed as follows — Table 4.2 summarizes the distribution:

[15] I am not distinguishing between *don't* and *do not*. See [Horn, 1989] for an analysis of the differences between the semantics of the contracted and non contracted forms.

- 43 instances of the *DONT* type, of which 39 with the auxiliary *don't*, and 4 with *never*:[16]

  (4.23) *Your sheet vinyl floor may be vinyl asbestos, which is no longer on the market.* **Don't sand it or tear it up**[17] *because this will put dangerous asbestos fibers into the air. It must be covered over.*

  (4.24) **Never mix cleaners containing acid or ammonia with chlorine bleach.** *The chemical reaction releases the chlorine as a poisonous gas.*

- 27 examples of the *neg-TC* type:

  - 4 with *take care*

    (4.25) *To book the strip, fold the bottom third or more of the strip over the middle of the panel, pasted sides together,* **taking care not to crease the wallpaper sharply at the fold.**

  - 3 with *be sure* or *make sure*:[18]

    (4.26) *To wash, load clothes into tub,* **making sure not to overfill it**.

  - 20 with *be careful*

    (4.27) *If your plans call for replacing the wood base molding with vinyl cove molding,* **be careful not to damage the walls** *as you remove the wood base.*

DONT		Neg-TC		
don't	never	take care	make sure	be careful
39	4	4	3	20
43		27		

Table 4.2: Distribution of Negative Imperatives

4.2.2 Negative Imperatives in the literature

Negation has been the subject of intense study in the semantics and pragmatics literature, as the volume by Horn [1989] shows; for some more recent work on the topic, see [Moser, 1992]. However, negative imperatives have not received much attention, neither in semantics and pragmatics, nor in computational linguistics, for reasons that are in a sense complementary.

[16] There are 3 further instances of DONT imperative that negate the verb *forget*: I haven't included them in my analysis, as a verb such as *forget* is used to tell H to adopt an intention to do α, as in *Don't forget to bed these tiles in*, rather than to prevent H from doing α.

[17] Five of the DONT instances are actually disjunctions; I counted them as one instance rather than two.

[18] I have a further example in which the argument to *be sure* describes a state, *Be sure that no adhesive has seeped through the joint.* I won't consider it in the following.

In semantics and pragmatics, negation has been studied a lot, but imperatives haven't. As [Merin, 1991] points out, in linguistics the only current monograph on English imperatives is [Davies, 1986], which in fact includes a chapter on negative imperatives; other work on imperatives is [Schmerling, 1982]. In philosophy, an extensive study of imperatives is [Hamblin, 1987].

On the contrary, in the computational linguistics literature, positive imperatives have received lots of attention — [Alterman *et al.*, 1991], [Chapman, 1991], [Cohen and Levesque, 1990], [Vere and Bickmore, 1990], [Winograd, 1972] — but not so negative imperatives, with the exception of [Vere and Bickmore, 1990]. Besides, to my knowledge not much work has been done on interpreting negation per se: an exception is [Iwanska, 1992], that represents negation as a complement operator in a Boolean algebra. However, her model only deals with declarative sentences and it is not clear how it would apply to imperatives.

In the following, I will review [Hamblin, 1987; Davies, 1986]; [Vere and Bickmore, 1990] was described in Sec. 3.3.3, where I also briefly discussed the way negative commands are dealt with: they are considered as global constraints on the goals the system may adopt.

4.2.2.1 Hamblin

Hamblin includes under *imperatives* a variety of linguistic constructions, that in some way convey imperative content, including indirect speech acts [Searle, 1975] of the kind

(4.28) *Jones, I want you to take charge of Division 4.*

The semantics of imperatives is given in terms of possible worlds:[19]

> [Hamblin] adopts a 'possible-world' semantics for propositional content, sorted into states and state changes (ordered pairs of states), the latter again partitioned into acts causally attributable to agents ('deeds') and mere happenings. All of these are ultimately defined as subsets of possible worlds. To distinguish proper — that is, intended viz. 'intensional' — satisfaction of an imperative from merely 'extensional', possibly accidental satisfaction, Hamblin introduces the notion of strategy: a function allocating a deed to each time/context pair. The interpretation of an imperative — its 'addressee action reduction' — is given by the weakest partial strategy (the disjunction of all partial strategies) which keeps the addressee within the set of possible worlds designated by the propositional content. [Merin, 1991, 674-675]

As far as negation is concerned, Hamblin claims that 'five different and independent negations' can apply to imperatives. Without examining all of them, I would just like to point out that Hamblin draws a difference between negation applied to a stative predicate, such as *Be here at lunch*, whose negation specifies the complementary end-state; and negation

[19] I will actually rely on [Merin, 1991] to discuss [Hamblin, 1987].

applied to action-specifying predicates, which changes an instruction such as *Run* from a request to carry out that action into one to refrain from doing it, *Don't run*.

Notice the difference between the two: the stative *be here at lunch* is understood to mean roughly *Take steps to be here at lunch*, whereas *Don't be here at lunch* does not mean (roughly) *Don't take steps to be here at lunch*.

One could thus conclude that *Don't be here at lunch* is interpreted as *Take steps not to be here at lunch*, namely, that negation takes narrow scope: the differences in meaning determined by the different possible scopes of negation have been an important base for all the work on the semantics on negation [Horn, 1989], and they will become relevant again in Davies's analysis described in the following.

4.2.2.2 Davies

In her monograph on imperatives [1986], Davies first of all notices, as Hamblin also does, that imperatives are not only used to convey commands or requests, but also pleas, exhortation, invitations, offers, advice, warnings, and instructions. She notices that instructions seem closer to advice in that they too are not aimed at getting H to do something, but rather, they tell him how to do something. According to Davies, instructions differ from advice in suggesting a recognized means to an end, one which will guarantee success.

She also observes that scholars such as [Schmerling, 1982] suggest that

> all uses of imperatives have in common that they constitute an attempt to bring about a state of affairs in which the proposition expressed by the imperative is true. However, definitions like this seem to make reference, not so much to the imperative's illocutionary force, but rather to its perlocutionary effect. They fail to draw the important distinction between the actual intentions of the speaker and the intentions of what the imperative utterance constitutes an expression; it is after all possible that a speaker, in using an imperative to order someone to do something, is in fact attempting to get him to do quite the opposite, knowing that he is somewhat contrary by nature. [p.39]

Davies suggests that the contrast is that while a declarative sentence asserts a proposition, an imperative sentence merely presents one. According to Davies, there is a convention governing the utterance of imperatives, similar to the convention of truthfulness underlying the use of declaratives. While the speaker who utters a declarative which asserts a proposition p is conventionally assumed to accept that p is true, the speaker who utters an imperative which presents a proposition p is conventionally assumed to accept p's being made true.

In this context her account of negative imperatives goes as follows. First of all, she notices that many scholars find negative imperatives somewhat peculiar, as, while an imperative puts forward some sort of action for the addressee to perform, the presence of negation is somehow incompatible with this function.

She notices that other researchers, while recognizing the difference between internal and external negation, don't think that such distinction applies to negative imperatives; in particular [Lyons, 1977] questions whether there are two interpretations for negative imperatives, paraphrasable respectively as

(4.29a) *I (hereby) impose upon you the obligation* **not to make it so that p holds**;

(4.29b) *I (hereby) impose upon you the obligation* **to make it so that not-p holds**.

She reports that Lyons argues that the second of these interpretations is not typical of negative imperatives, which function as prohibitions, and are used 'not normally ... as instructions to carry out, but to refrain from carrying out, some course of action'. Lyons rejects the latter interpretation on the grounds that 'The speaker does not want the addressee to bring about a state-of-affairs of which not-p is true; for this state-of-affairs already exists'. Davies notices that while this is certainly true of the situation Lyons envisages, in which S wants to forbid H to do something he apparently intends to do, a prohibition may equally well serve to tell someone to stop doing something he is already engaged in: *Don't be so silly* may serve the same purpose as *Stop being so silly*. In such cases, S's intention seems to be just that rejected by Lyons, namely that H bring about a state of affairs where not-p is true, since it is not true at the moment of utterance.

Moreover, it seems to me that the interpretation in (4.29b) is the intended reading for the interpretation of the stative *Don't be here at lunch*, as Hamblin suggests.

Davies then suggests that the difference is rather between cases where S is rejecting as undesirable a type of behaviour which she thinks H may exhibit, and cases where she is prescribing a case of behaviour which she considers appropriate for H to exhibit. Davies notices that negation may be associated either with the presentation constituted by the imperative, or with the particular proposition which is presented; she concludes that the difference between the two possible interpretations lies, not in the type of action implied, whether doing or not-doing, but in S's attitude towards the possible action presented. When negation is associated with the presentation, its utterance will constitute a rejection of the possibility rather than an acceptance of it. On the other hand, in case negation is understood as associated with the proposition that is presented, the presentation is not affected by the presence of the negation, and therefore its utterance will still constitute an expression of the acceptance of a possibility; it is simply that the possibility presented as acceptable will be one of something's not being done.

While the most common interpretation of imperatives is the former, namely, negation attached to the presentation, the latter explains usages such as the following:

(4.30a) *A: I don't think I'll go to the party.*

(4.30b) *B: All right, don't go then, if you don't want to.*

In uttering the imperative in (4.30b), the speaker is clearly expressing, not a rejection of the possibility of the addressee's going, but rather her acceptance of his not going.

4.2.3 Different uses for different Negative Imperatives

The semantic distinctions described above don't seem to be particularly relevant to my data, where what is negated is always an action.

Therefore, the intended meaning always seems to be (4.29a), *I (hereby) impose upon you the obligation not to make it so that p holds* (in Lyons's terms), or *a rejection of the possibility that the proposition becomes true*, in Davies's terms. Moreover, it is clear that a *DONT* imperative could be used when a *neg-TC* one is used: an expression like *take care not to do* α entails *don't do* α.[20] In fact, in terse instructions only *DONT* imperatives are found.

However, in the instructional text I have examined, there are pragmatic distinctions between the usages of *DONT* and *neg-TC* imperative: *DONT* imperatives are used when S expects H to choose a wrong alternative, while *neg-TC* imperatives are used when S expects H to overlook a choice point. The distinction between different scopes of negation could be relevant to these different usages, if the scope of negation is considered with respect to some kind of INTEND operator used to represent the intentions that H adopts — see Sec. 4.2.3.2.

That there are indeed pragmatic factors affecting the choice of a surface form versus the other is shown by the infelicitous usage of a – really occurring! – *DONT* imperative:

(4.31) # *If you must replace a tile, first cut around the edges with a circular saw. Set the blade to the depth of the tile and* **don't damage adjoining tiles**.

The previous example is not felicitous because, in the context of an assembly task, *damage* is not a choice that an agent has at his disposal, but rather, it is a side effect that may derive from certain choices in executing *cut edges with a saw*: therefore *neg-TC* would be more appropriate than *DONT*.

4.2.3.1 Speaker's expectations and Hearer's choices

DONT imperatives. *Don't do* α appears to be used when S thinks that H is likely to come to a choice point, and intentionally choose one course of action over another. Some situations in which this can happen are:

- When S provides H with general goals, or with rules of behavior to be always adopted in certain circumstances, as in (4.23) above; another example is

  (4.32) *You can put parquet down over a variety of surfaces, whether old or new, if they are firm, clean, smooth, and dry.* **Don't put parquet down on a surface that is below ground level because of the moisture problem.**

[20] It is not clear what *entails* really means here, given that instructions are difficult to model in the usual model-theoretic semantics. I am using this term in an intuitive sense.

- Another circumstance in which S may use a *DONT* imperative is when α is an undesirable alternative to a β that S tells H to do, as in

  (4.33) *Caring for the floor. A good paste wax - not a water-based wax - will give added protection to the wood. Buff about twice a year; wax about once a year. Excessive waxing can cause wax to build up, detracting from the floor appearance. Dust-mop or vacuum your parquet floor as you would carpeting.* **Do not scrub or wet-mop the parquet.**

  Clearly, S thinks that H, after adopting the intention of cleaning the parquet, may choose to do so in a wrong way. Using a *neg-TC* verb in this case would be infelicitous, as it would seem to imply that H could unintentionally choose to perform either scrub or wet-mop. Notice that in this example the general goal that H has to adopt, namely, *cleaning the parquet*, is essential to interpret the negative imperative (and for that matter the previous sentence too): in fact H has first to understand that *dust-mop, vacuum, scrub, wet-mop*, all achieve *cleaning the parquet*; and that, although the last two, *scrub* and *wet-mop*, may in general be performed to achieve a goal such as *cleaning floor*, they are not a viable alternative in this case.

- *Don't do* α can be used to tell H that an action β should not be followed by α, in particular if α affects the outcome of β. Consider:

  (4.34) *Use a vinyl-to-vinyl paste for any type of border you plan to hang over wallpaper. To paste the border for hanging, cover the entire back with paste and book the strip;* **don't crease the folds**.

  Actually, in this case it is less clear whether *creasing the folds* can be seen in terms of an intentional choice point. Rather, the usage of a *DONT* imperative seems to imply that the two actions of *booking the strip* and *creasing the folds* are independent: as we will see in (4.39b), the latter can also be seen as a culmination of the former.

Neg-TC imperatives. In general, *neg-TC* imperatives are used when S expects H to overlook a certain choice point; such choice point may be identified through a possible side effect. Moreover, a *neg-TC* is used when S relates the negated action α to another action β in the discourse. A form like *"Do β. Take care not to do α"* appears to be used when

- α is an undesirable way of performing β. The description of β is always underspecified, and therefore H has many degrees of freedom in executing it. Consider

  (4.35) *To make a piercing cut, first drill a hole in the waste stock on the interior of the pattern. The diameter of the hole must be larger than the width of the blade. If you want to save the waste stock for later use, drill the hole near a corner in the pattern.* **Be careful not to drill through the pattern line.**

  β is *drill a hole near a corner in the pattern.* The interpretation of *near* still leaves H some choices as regards the exact position where to drill: S constrains them by saying *Be careful not to drill through the pattern line.*

- α is an undesirable effect of β, which may be under H's control, or under the control of external laws, as in (2.13), repeated here for convenience:

  (4.36) *To hang the border, begin at the least conspicuous corner. The work will go much faster if you have someone hold the folded section while you apply the border to the wall.* **Take care not to drip paste onto the wall**.

  There are infinite ways of executing *hanging the border*; some of them will have *dripping paste onto the wall* as a side effect. S tells H to take active steps in order to avoid such a side effect.

4.2.3.2 Intentionality

So far, I have claimed that S's expectations on H's choice points affect the surface form of a negative imperative. S's expectations are reflected in whether S sees the actor, namely H, as an intentional one: which brings me to briefly discuss whether the verb expressing the negated action encodes intentionality in the agent role.

In general, verbs are ambiguous in this respect; consider the well-known ambiguity of *roll*:

(4.37) *Bill rolled down the hill*

where Bill may or may not be performing the action volitionally.

> It seems that any Actor, if animate, is subject to this ambiguity, unless the verb specifically selects for a volitional Agent, as do, for instance, *buy* and *look*. [Jackendoff, 1990, p. 128]

With respect to negative imperatives, consider how *roll* behaves: if *roll* is taken in its non volitional reading, negative imperatives pattern as follows:

(4.38a) *# Don't roll down the hill.*

(4.38b) { *Take care / make sure / be sure / be careful* } *not to roll down the hill.*

However, if we take *roll* in its volitional interpretation, the judgements seem to be reversed. From this we could infer that *DONT* selects for intentional readings of verbs, *TC* for unintentional ones.

This is another account for the infelicity of (4.31) above. In the context of assembly instructions, *damage* is an action that an agent doesn't perform intentionally — therefore, there is a conflict between what *don't* selects for and the unintentional reading of *damage*.

Tying this back to S's expectations on H's choices, notice that an action description can always be further constrained; therefore, even if a verb selects for a volitional agent, its modifiers can bring in aspects of choice that the agent may not be aware of. Consider

(4.35) again: while an agent will drill intentionally, what he can do unintentionally is to overlook the exact location of the drilling site. Contrast (4.35) with (4.33): in the latter, using a form such as *take care not to scrub or wet-mop* would be infelicitous, as it seems to imply that the agent could choose either scrubbing or wet-mopping by accident. Therefore, the usage of a *neg-TC* imperative draws H's attention to the possibly unintentional aspects of the action to be performed.

The effect of intentionality on the choice of surface form suggests that the two different negative imperatives can be modelled as exhibiting different scopes of negation with respect to some INTEND operator [21] that represents H's intentions: a *DONT* imperative would be interpreted as *NOT (INTEND (H p))*, a *neg-TC* one as *INTEND(H, NOT(p))*.[22]

This representation makes clear that, in the case of a DONT imperative, H adopts the intention to do p independently from the negative imperative itself, either because it has been inferred from the previous text, or because it is a choice that comes up while executing the instructions; in a sense, the intention is given, already mutually known to both S and H.

In the case of a *neg-TC* imperative, instead, the intention of doing p has not arisen yet, and S communicates to H that in fact a new intention should be adopted, of doing / achieving *NOT(p)*.

Clearly this idea is very preliminary. Also the idea that the lexical semantics of verbs affects the usage of *don't* or a negated *TC* verb requires further investigation, as in certain cases, the negated verb seems to be at odds with the choice of the negative element: for example, a verb like *forget* is by default read in its unintentional meaning. Nevertheless, the unmarked form is *don't forget*, while *take care not to forget* is marked, and means *take active steps — e.g. notes — not to forget.*

4.2.3.3 Action relatedness

Another factor that seems to affect the choice of *DONT* over *neg-TC* is whether S sees the negated action α as closely related to another action β in the discourse. Consider the contrast between

(4.39a) *To paste the border for hanging, cover the entire back with paste and book the strip;* **don't crease the folds**.

(4.39b) *To book the strip, fold the bottom third or more of the strip over the middle of the panel, pasted sides together,* **taking care not to crease the wallpaper sharply at the fold.**

In (4.39a), *creasing* is seen as an independent action from *booking*, with the former following the latter; in (4.39b), *crease* is seen as one possible culmination of *fold*, and therefore, much more closely related to it.

[21] No commitment to any formalism here!
[22] Christine Nakatani, p.c.

Action relatedness clearly affects the structure of the discourse: in Ex. (4.39a), *crease the folds* is presented as at "the same level" that *book the strip* is, therefore, in a tree representation of discourse, [23] the nodes representing the two actions would be sisters; instead, in Ex. (4.39b), *crease the folds* is presented as subordinate to *folding*, and therefore the node corresponding to the latter would be in a dominance relation to the node corresponding to the former.

4.3 Summary

In this chapter, I have presented a pragmatic analysis of purpose clauses and negative imperatives. I would like to conclude by pointing out some tighter connections between the two than just the theoretical ones that they are both related to the choices that the agent adopts, and that they both shed some light on the claim that goals direct the accommodation process.

The tighter connection between purpose clauses and negative imperatives comes from the interaction of negation, imperatives and purpose clauses.

1. First of all, we may consider Negative PCs, namely *Do* α **not** *to do* β. I have no examples of a negative purpose clauses per se; such examples would seem to require a semantic analysis like in (4.29b), *I (hereby) impose upon you the obligation to make it so that not-p holds*. However, as I mentioned in Sec. 4.1.3, a PC may express negation, in a sense, by describing a goal of *preventing* an event from happening. On the semantics of *prevention* see for example [Ortiz, 1993].

2. We could also have a negative main clause associated with a PC, namely, **Don't** *do* α *to do* β, as in

   (4.40) *Don't use chemicals to clean your parquet.*

   The previous example is made up, as I have no natural occurrences of such pattern; however is seems perfectly plausible. Such example raises various possible questions, including

   - The presence of negation seems to strongly suggest that the purpose clause is not used to express a goal, but rather, the circumstances in which a certain cleaner should not be used. Negation seems to transform the instruction from a command to do something into a general *policy* to adopt in certain circumstances. An issue worth exploring is how this affects the interpretation of the PC.
   - Can a *neg-TC* imperative appear as the main clause associated to a Purpose Clause? Intuitively it would seem impossible, as the purpose clause expresses a goal that should become object of H's intentions, while a *neg-TC* points to

[23] On the tree representation of discourse see among others [Grosz and Sidner, 1986; Polanyi, 1988; Mann and Thompson, 1988].

unintentional aspects of an action. However, analysis of a much bigger corpus is necessary to check whether this intuition is plausible.

As I already mentioned, my analysis of Purpose Clauses is much more developed than the one of Negative Imperatives. Therefore, for the rest of this document I will concentrate on computational issues arising from PCs.

Chapter 5

Purpose Clauses: computational consequences

In Chapter 2 I emphasized the role that *goals* play in the *accommodation* process as applied to instruction interpretation. I also claimed that such inferences are exemplified in a particularly clear way when the goal is explicitly expressed, as in the case of *purpose clauses*. In this chapter, I will use my analysis of purpose clauses to follow up on these claims, and on the others that I put forward in the introduction. More specifically, through my analysis of purpose clauses:

- by showing that β constrains α, I give support to the claim that there is no *direct mapping* between NL action descriptions and stored knowledge;
- by shedding light on some necessary accommodation inferences, I support the claim that the mapping between surface form and stored knowledge must be *computed*, and that *goals* guide this computation;
- finally, I suggest specific characteristics for the action representation formalism. In fact, purpose clauses appear to express generation or enablement, supporting the proposal, made by [Allen, 1984], [Pollack, 1986], [Grosz and Sidner, 1990], [Balkanski, 1993], that these two relations are necessary to model actions.

The main aim of this chapter is to describe the relations between actions that purpose clauses express, and the inference processes that their interpretation requires. I will conclude by describing the consequences of these data for my computational approach.

5.1 Relations between actions

So far, I have mentioned that α *contributes* to achieving the goal β. The notion of *contribution* can be made more specific by examining naturally occurring purpose clauses. They mainly express *generation*, in some cases *enablement*, and in few remaining cases other

relations that still need to be properly defined — see Sec. 5.1.4.4. Grosz and Sidner in [1990, p.431] have already observed that

> *Contribute* is a place holder for any relation ... that can hold between actions when one can be said to contribute (for example, by generating or enabling) to the performance of the other.

However, Grosz and Sidner don't provide any evidence in terms of naturally occurring data to support the statement that *contribute* can be instantiated as *generation* or *enablement*.

Analogously, Balkanski [1990; 1992a; 1993] does use her formalization of generation and enablement to model rationale and means clauses, but she doesn't present very detailed evidence from naturally occurring data to justify such choices.

5.1.1 Abstraction

As I mentioned in Sec. 4.1.3, purpose clauses relate actions at different levels of abstraction. One would then expect to find examples in which α and β are simply related by an abstraction relation, that could be defined in intuitive terms as

$$\forall \alpha, \beta \; [\alpha \xrightarrow{ISA} \beta] \Longrightarrow [\text{OCCURS}(\alpha) \longrightarrow \text{OCCURS}(\beta)]$$

In Ex. (4.12) — *Vacuum or dust-mop your parquet to clean it* — *vacuum parquet* cannot occur without *cleaning parquet* occurring too. However, no example in my corpus simply expresses abstraction — (4.12) is constructed. It seems that abstraction would rather be expressed with a MeaC:

(5.1) *Clean your parquet by vacuuming or dust-mopping it.*

Also, an IPC would be more felicitous:

(5.2) *To clean your parquet, vacuum or dust-mop it.*

It seems that the notions of *given* and *new* could be relevant to explaining the fact that PCs, or at least FPCs, don't generally express abstraction. In fact, notice that if

$$\alpha \xrightarrow{ISA} \beta$$

then the information provided by β is already provided by α: in Ex. (4.12), the purpose clause *to clean it* does not add any new information with respect to the main clause *vacuum or dust-mop your parquet*. Instead, in Exs. (5.1) and (5.2), the MeaC and the postposed main clause respectively describe which of the possible specializations of *cleaning* should be performed.

5.1.2 Generation

Generation is a relation between actions, where by actions here I mean action occurrences. Generation has been extensively studied, first in philosophy by Goldman [1970], and then in discourse analysis by Allen [1984], Pollack [1986], Grosz and Sidner [1986; 1990], Balkanski [1990; 1993].
According to Goldman, intuitively generation is the relation between actions conveyed by the preposition *by* in English – *turning on the light* **by** *flipping the switch.* [1]

Goldman distinguishes among four kinds of generation relations. Subsequent work has been mainly influenced by *conditional* generation. In particular, Pollack [1986] and later Balkanski [1990] formalized the notion of *conditional generation*, specifying that it has to hold for *act-types*. They then used it to define the notion of *generation* between action occurrences.

(5.3) α conditionally generates β iff:

1. α and β are simultaneous;
2. α is not part of doing β;
3. when α occurs, a set of *generation-enabling* conditions $\mathcal{C}$ hold, such that the joint occurrence of α and $\mathcal{C}$ imply the occurrence of β.

As regards the generation-enabling conditions $\mathcal{C}$, in the case of the generation relation between *flipping the switch* and *turning on the light*, $\mathcal{C}$ will include that the wire, the switch and the bulb are working; in the case of the generation relation between *uttering the words "do you know the hospital's phone number"* and *asking one's officemate what the hospital's phone number is*, $\mathcal{C}$ will include, among other conditions, that the officemate can hear the question.[2]

As Balkanski notes, it may be argued that generator and generatee are not exactly simultaneous. For example, the light does not go on until a few milliseconds after the switch has been flipped. Goldman, however, points out that even if there is a time gap between flipping the switch and turning on the light, it would still be incorrect to say that an agent flipped the switch "and then" turned on the light.

Balkanski's version of generation is [1992a, p. 270]:[3]

(5.4) α generates β if and only if

1. α and β are actions performed by the same agent at the same time
2. there is a set of conditions, $\mathcal{C}$, such that
   (a) these conditions hold during the performance time of α and
   (b) the act-type of α conditionally generates the act-type of β under $\mathcal{C}$.

As Goldman observes in [1970], generation doesn't hold between α and β if α is part of a sequence of actions $\mathcal{A}$ to do β: however, generation may hold between the whole

[1] We will see in Sec. 5.1.4.2 that, although rarely, a *by* construction can express other relations.
[2] This latter example is from [Pollack, 1986].
[3] I have changed her notation to make it consistent with mine.

sequence $\mathcal{A}$ and β. [4] Therefore, in the following I will introduce the terms *direct* and *indirect generation*: a *direct* generation relation holds between α and β if α generates β; an *indirect* generation relation holds between α and β if α belongs to a sequence of actions $\mathcal{A}$ which generates β. Notice that by *sequence* I mean a temporally partially, not totally, ordered set of actions.

Generation is a pervasive relation between action descriptions in naturally occurring data. However, it appears from my corpus that *by* clauses are used less frequently than purpose clauses to express it:[5] quite uncontroversially, 84 of the 164 PCs express direct generation — this is about 50% of the data— while in the same corpus there are only 27 *by* clauses. (2.2b) and (2.1b) are examples of purpose clauses that express direct generation; another representative example, picked at random, is

(5.6) *Press firmly* **to bed tile in the bond coat**.

Moreover, another 28 examples seem to express indirect generation — see below on the problem of distinguishing indirect generation from enablement.

Therefore, it does look like generation in instructional text is mainly expressed by means of purpose clauses. This is consistent with Balkanski's findings regarding the speaker's beliefs about the actor(s)'s beliefs and intentions expressed respectively by a purpose clause and by a means clause. In the former case, the speaker believes: that the actions and relations between actions are intended on the part of the actor; furthermore, that the actor believes that the actions are about to occur, and that α will contribute to β. Not necessarily do these beliefs hold in the case of a means clause — see Sec. 4.1.2.2 above. These observations on the difference in beliefs and intentions in purpose and means clauses make sense in the context of instructional text, in which the instructor wants the agent to adopt certain intentions, and certain beliefs about actions and relations between actions: therefore, it is plausible that S uses purpose clauses rather than means clauses to convey such beliefs and intentions.

5.1.3 Enablement

Following first Pollack [1986] and then Balkanski [1990; 1992b; 1993], *enablement* holds between two actions α and β if and only if an occurrence of α brings about a set of conditions that are necessary (but not necessarily sufficient) for the subsequent performance of β.

[4] [Balkanski, 1990, p.9] formalizes this by means of a complex act-type constructor called **sequence**.

[5] Generation can also be expressed with a simple free adjunct. In (5.5) "removing the left finger" generates "transferring loop to needle":

(5.5) *Take needle in right hand and insert into loop from underneath. Remove left index finger,* **transferring loop to needle**.

However, this use of free adjuncts is not very common – see [Webber and Di Eugenio, 1990].

Balkanski in [1993, p.38] distinguishes two types of enablement, *gen-enable* and *exec-enable*, distinguished because in the former case, the performance of the enabling action brings about a required generation-enabling condition, while in the latter, it brings about a required executability condition. Balkanski illustrates the difference by means of the following two examples:

(5.7a) *Mary's inserting a diskette "enabled" her to back up her file.*

(5.7b) *John's looking up Mary's phone number "enabled" him to call her.*

According to Balkanski, in (5.7a) inserting a diskette brings about a condition, namely that the diskette is inserted, that is necessary for the backing up action to be *successful*. If the diskette is not inserted, then Mary can still try to back up a file, e.g., by typing "copy file", but the intended result will not be achieved. On the other hand, in (5.7b) looking up a number brings about a condition, namely that the telephone number is known, that is necessary for the calling action to be *executable*: if the number is not known, then John cannot even try to call Mary. Thus, in (5.7a), the enabling action α — *Mary's inserting a diskette* — brings about a condition $\mathcal{C}_i$ — *in(diskette)* — belonging to the set $\mathcal{C}$ of generation-enabling conditions on a generation relation between γ — *Mary's typing "copy file"* — and β — *Mary's backing up her file*; in the case of (5.7b), instead, there is no such intervening generation relation.[6]

Balkanski's definition of enablement is as follows:[7]

(5.8) α **enables** β if and only if

1. the time of α is prior to the time of β,
2. there are a set of conditions, $\mathcal{C}$, such that one of the conditions in $\mathcal{C}$, $\mathcal{C}_i$, holds as a result of the performance of α, and either
   (a) there is a third action γ, and γ conditionally generates β under $\mathcal{C}$; or
   (b) $\mathcal{C}$ are the executability conditions on β.

Schematically, these two kinds of enablement are schematized as in Fig. 5.1— the "/" indicates executability conditions when it is associated to an action, and generation-enabling conditions when it is associated to a conditional generation relation.[8]

Given this definition of enablement, it would seem that only about 7% of my data, namely, 12 examples, express it — I am saying *would seem* because sometimes it is not clear whether α should be seen as enabling β, or as belonging to a sequence of actions $\mathcal{A}$ generating β — see the discussion below. Among these 12 examples, a couple of the most representative ones are

(5.9) *Unscrew the protective plate* **to expose the box.**

Unscrew the protective plate enables *taking the plate off* which generates *exposing the box*.

[6] I will come back to this example below.

[7] This is my paraphrase of her enablement definition E0 plus enablement axiom 2 [Balkanski, 1993, p.42].

[8] This is my rendering of Balkanski's schemas [1993, p.41].

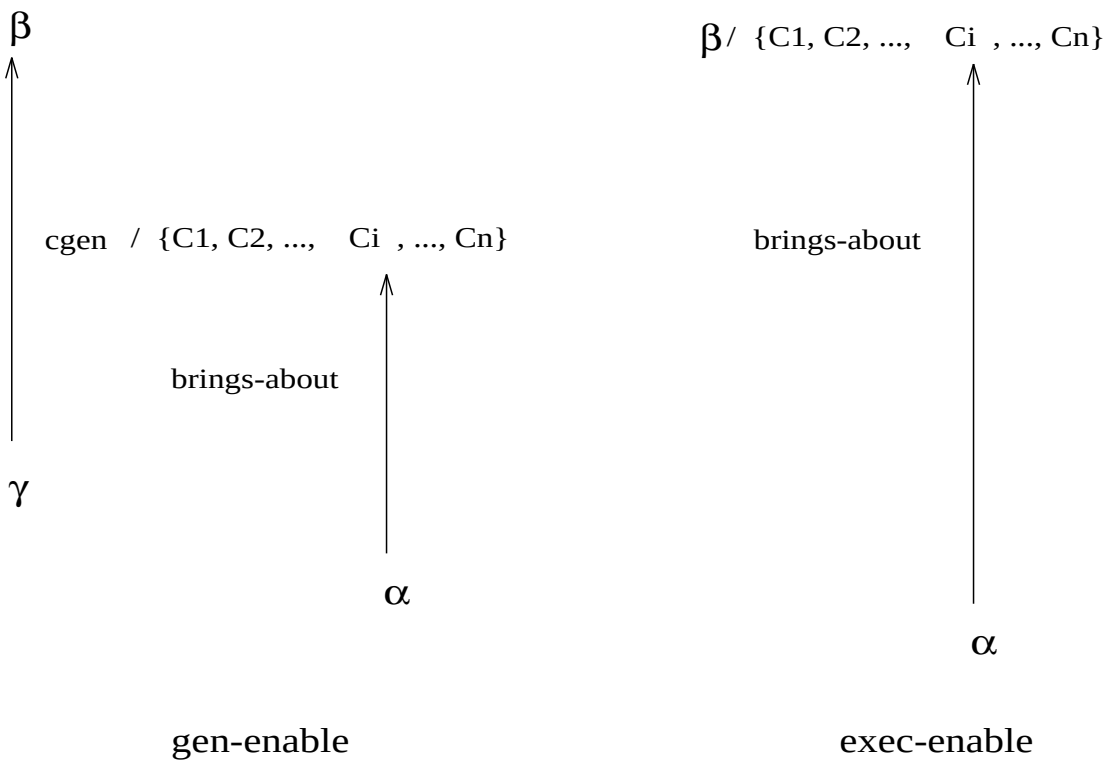


Figure 5.1: Gen-enable and Exec-enable

The other example is

(5.10) **To fuse appliques**, *slip the shaped piece of fusible webbing between the garment and the applique; press the layers together using a warm iron.*

Slip the shaped piece of fusible webbing between the garment and the applique brings about a necessary condition for *pressing the layers together using a warm iron* to generate *fuse appliques*.

To conclude, I would like to point out that I see one major problem with Def. (5.8), and another problem with Balkanski's analysis of Ex. (5.7b).

The problem with Def. (5.8), that Balkanski herself acknowledges, is as follows: given her action formalism, condition 1 amounts to requiring that the execution of α is over before β begins. This is too restrictive, in that it may be necessary that α continues for the whole execution of β. Consider a (constructed) example such as:

(5.11) *Hold the cup under the spigot* **to fill it with coffee.**

In (5.11), *hold* has to continue for the whole duration of *fill*: still, I would contend it is an example of *gen-enable*, in that *holding the cup under the spigot* brings about a condition —that the cup is in the proper position— necessary for an action such as *depressing the [coffee urn] lever* to generate *fill the cup with coffee*.

I also would like to point out that her analysis of Ex. (5.7b) is not the most natural one, in that it can be analyzed in quite the same way that Ex. (5.7a) is. In fact, it does seem to me that there is a third action γ = *John's dialing Mary's phone number*, and a generation relation holding between γ and β = *John's calling Mary* under the condition *John knows Mary's phone number*. The analysis that Balkanski gives of (5.7b) seems to rest on the assumption that the action *G's calling X on the phone* is basic; by the same token, then, *G's backing up one's file* in Ex. (5.7a) could be considered basic as well.

Balkanski herself notices that her notion of *exec-enable* may raise some criticism [1993, p.44], in that the exec-enable relation may be deemed not necessary. One could in fact claim that it is always possible to reason at lower and lower levels of detail, appealing to further generation relations, until one gets "all the way down" to (unconditional) basic actions. Given that basic actions cannot have conditions attached to their performance, all enablement relations at this lowest possible level of description are instances of gen-enable. Balkanski's response to such criticism is that, although it is theoretically possible to describe all actions in terms of basic actions, such descriptions are often unintuitive. The generation relation between saying "hello" and moving one's vocal cords in a particular way, for example, rests on an awkward description. Rather, she notices that we can decide on some set of actions that we assume are "basic" in the domain of interest, and not consider how actions in that set are actually performed. Given such a set of actions, Balkasnki offers two options: such actions can either be taken as basic in Goldman's sense, i.e. executable at will (as Pollack does [1986]), in which case the exec-enable relation is not necessary; or they can be taken to be conditionally executable, in which case both the exec-enable and the gen-enable relations are necessary.

I agree with her observations on basic actions, and on the fact that the set of basic actions cannot be predefined, but that it depends on the domain[9]. Moreover, I think that her distinction actually boils down to one's knowledge representation choices: I think the same action can be seen as either *conditionally executable*, as she considers *G's calling X on the phone* in (5.7b), or as the generated action in a conditional generation pair, as I have shown *G's calling X on the phone* can also be considered. Therefore, I suspect that almost every example can be explained by *appealing to further generation relations*, but without necessarily getting down to basic actions. I will come back to my formalization of enablement in the next chapter.

5.1.4 Issues in action relations

5.1.4.1 Why generation and enablement?

Pollack in her thesis [1986] is particularly vocal in advocating these two relations as the basis for action representation. Her motivations stem from the need for a perspicuous representation for an agent's plans, one that can support her view of plans as *mental phenomena*, as I discussed in Sec. 3.2.1.2. She notices that in the plan inference literature plans are generally derived by composing basic planning operators, which are represented as follows:

```
Header:  β
Preconditions:  P
Body:  α
```

She criticizes such a representation because the relation between α and β is not precisely defined: researchers have assumed any of *causes, is-a-precondition-of, is-a-way-to*. She then prefers to use a representation based on *generation* and *enablement*, which she can precisely define.

A further motivation for using *generation* and *enablement* in modeling actions is that they allow us to draw conclusions about action execution as well – a particularly useful consequence given the discussion in Sec. 2.3 on the necessity of grounding action in performance, which is in turn partially motivated by the requirement of animating instructions in AnimNL.

As has already been observed by other researchers, if α generates β, two actions are described, but only α, the generator, needs to be performed. In Ex. 2.2, there is no *creating* action per se that has to be executed: the physical action to be performed is *cutting*, constrained by the goal as explained above.

In contrast to generation, if α enables β, after executing α, β still needs to be executed. α has to temporally precede β, in the sense that α has to begin, but not necessarily end, before β.

[9]This observation had been already made by Pollack in her thesis [1986, p.59].

Notice that, in the same way that the generatee affects the execution of the generator, so the enabled action affects the execution of the enabling action. In (5.11) above, the goal *fill [the cup] with coffee* constrains the position in which the agent has to *hold* the cup, namely, upright, with the opening facing upwards, and centered under the spigot.

All these observations on generation and enablement are summarized in Table 5.1.

	Generation	**Enablement**
Can β constrain α?	Y	Y
Temporal constraints	α and β simultaneous	start(α) must precede start(β)
Execute	only α (generator)	α (enabling) β (enabled)

Table 5.1: Generation and enablement

5.1.4.2 Generation, enablement and purpose connectives

I would now like to go back to the results in Sec. 4.1.4, and try to correlate them to generation and enablement. It seems to me that the results in the experiment point to the following facts:

1. *so as to* and *by* are felicitous when the relation between actions is generation;

2. *in order to* is always acceptable, more so when the relation expressed is indirect generation or enablement;

3. the difference between pre- and postposed purpose clauses discussed by Thompson can be expressed in terms of indirect generation / enablement on the one hand, generation on the other. In fact, IPCs, whose scope very often encompasses more than one clause, tend to express indirect generation or enablement, while FPCs tend to express direct generation. This would explain why *so as to* is not acceptable as a paraphrase of an IPC.

These observations for example explain the distribution of judgements for Ex. (4.22): the paraphrase with *in order to* is the one most preferred, while *so as to* and MeaC are judged unacceptable by all informants. However, it remains unclear why the judgements on the acceptability of the paraphrase with *in order to* are split exactly in the middle.

However, the previous observations are not sufficient to explain all the facts. In particular, they don't explain why

1. In an example of FPC that clearly expresses *enablement*, the paraphrase with *so as to* is considered more acceptable — eight *totally acceptable* judgements — than the one with *in order to* — six "3", one "2", one "1":

   (5.12a) *Slip paper towels between iron and fabric* **to** *catch stray wisps of the heated fusible webbing.*

   (5.12b) *Slip paper towels between iron and fabric* **so as to** *catch stray wisps of the heated fusible webbing.*

   (5.12c) *Slip paper towels between iron and fabric* **in order to** *catch stray wisps of the heated fusible webbing.*

   It is possible that there is a further component to the meaning of *so as to*, relating to the *manner* of the action; this component of the meaning of *so as to* does not necessarily coincide with the relation between the two actions being *generation*, although there is a strong correlation between the two.

2. A *by* paraphrase is sometimes acceptable for an IPC. After all, if both *so as to* and *by* mainly express generation, they should both be infelicitous in the case of IPCs: however, *by* paraphrases are rated as much more acceptable than *so as to* paraphrases. This issue deserves further attention.

5.1.4.3 Generation versus enablement

The distinction between *generation* and *enablement* appears to be perfectly clear when only two actions are involved. In fact, mistaking one for the other can indeed have unpleasant consequences, as Fig. 5.2 shows. In Fig. 5.2, it is clear that Effie interprets the instruction *When you want to turn, either put on your blinker or stick your arm out of the window* as expressing a generation relation between *put on your blinker* and *turn*, while the intended relation is enablement, in that e.g. *putting on your blinker* brings about executability conditions to achieve a (*proper*) turn. To Effie's excuse, one should notice that the first two instructions — *When you want to go, you step on the gas* and *When you want to stop, you step on the brake* — are indeed interpreted as expressing generation; as a side remark, notice that a *when* clause is another type of adjunct clause that can express generation.

However, one issue that has to be addressed is the fact that, in the presence of a sequence of actions $\mathcal{A} < \alpha_1, \alpha_2, ..., \alpha_n >$ that generate another action β, it is not clear what relation is holding between each single α_i and β. Consider the following example:

(5.13) **To install,** *use nails or adhesive to fasten gypsum wallboard to existing walls or studs. Tape corners and cover the nail heads.*

In this example, we have three actions, α_1 = *fasten gypsum wallboard*, α_2 = *tape corners*, α_3 = *cover the nail heads*, whose sequence [10] generates β = *install*. What is the

[10] Actually α_2 and α_3 can be executed in either order or even interleaved. In fact, remember that by sequence I mean a temporally partially ordered set.

Figure 5.2: Generation or enablement?

individual relation between say α_1 and β? Intuitively it doesn't seem that it should be analyzed as an *enablement* relation. However, also (5.10), that was analyzed as expressing enablement, could be analyzed in this way: we could say that the sequence $<$ *slip applique, press firmly* $>$ generates *fusing appliques*.

Schematically, this situation is represented in Fig. 5.3.

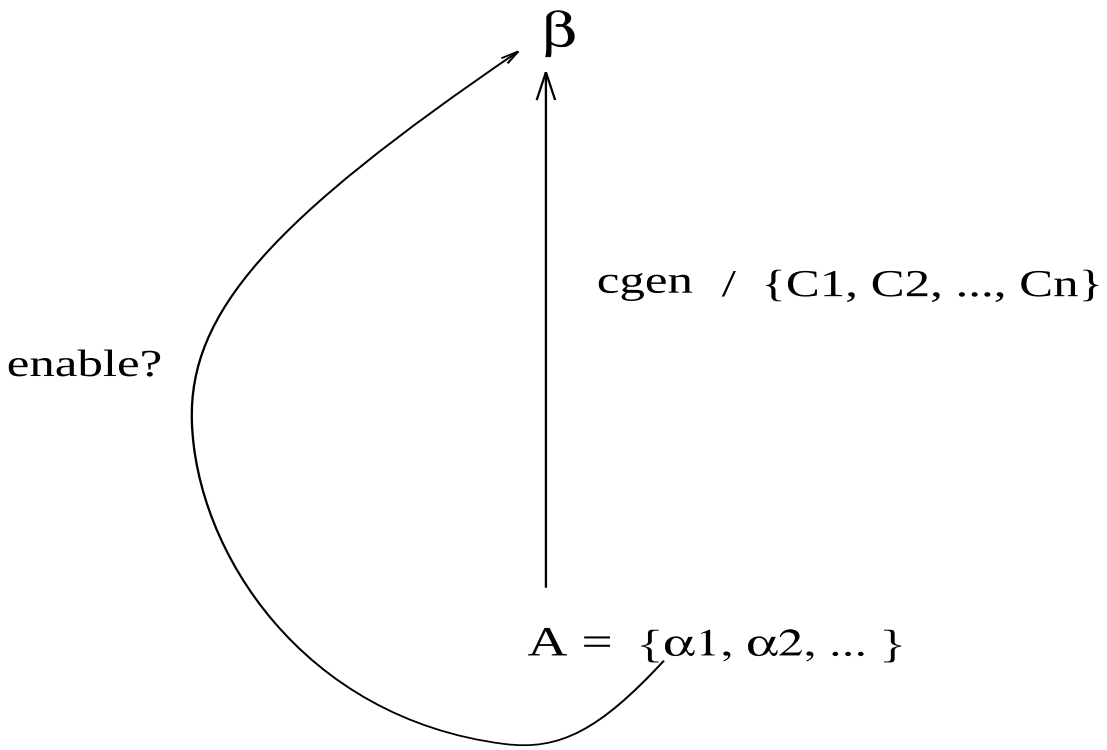


Figure 5.3: Indirect generation or enablement?

The reader may wonder why the question is relevant after all: its relevance comes from the fact that I am using these relations to both analyze naturally occurring examples, and to express the knowledge necessary to analyze them. Therefore we may have knowledge such that $\mathcal{A} = < \alpha_1, \alpha_2, ..., \alpha_n >$ generates β, but the surface PC is of the form α_1 *to* β: intuitively we could consider it as α_1 enabling β, while according to the stored knowledge α_1 enables some α_j belonging to $\mathcal{A}$, and the whole sequence generates β. Notice that I found no example in which the surface form was *Do* α_i *to do* β with $i > 1$: namely, a case in which there is a sequence generating β, but there has been no mention of $\alpha_1, ...\alpha_{i-1}$. However, this may be parasitic on my stored knowledge, rather than on the naturally occurring data.

In conclusion, I would like to point out that basically I will use the term *enablement* in two ways, one descriptive, the other formal. I have used it descriptively so far in my data analysis, which has been done independently from the formal representation of the

knowledge necessary to interpret them. The term *enablement* will instead be used more precisely in the next two chapters: in cases in which the algorithm discovers that α_i belongs to a sequence $\mathcal{A}$ generating an action β, where the actions in the sequence may be related to each other by enablement relations, I will reserve the term *enablement* only to the relation holding between any pair $< \alpha_i, \alpha_k >$ belonging to $\mathcal{A}$; the relation between each α_i and β will be termed instead *indirect generation*, as mentioned at the end of Sec. 5.1.2.

5.1.4.4 Other relations

The definitions of generation and enablement given above — (5.4) and (5.8) respectively — don't seem to adequately account for all examples.

Reconsider the definition of enablement in (5.8), and in particular, the temporal constraint that the starting time of α is before the starting time of β. In general, (5.8) is interpreted as meaning that β is executed as soon as all executability conditions become true; however, the execution times of α and β could be actually separated by a considerable amount of time, even years, as shown by examples such as the following

(5.14) *Save wallpaper and leftover adhesive* **to repair areas that may be damaged**.

While (5.8) would indeed account for (5.14), it is an issue left for further research whether more precise relations between the perfomance time intervals of α and β should be explicitly included in the representation of an enablement relation between α and β.

In other cases, the relation between α and β does not seem to fit into the definition of generation or enablement at all. Consider [11]

(5.15) *Build this simple but elegant corner shelf* **to display all the other decorative pieces found in this issue.**

It is clear that *building this simple but elegant corner shelf* is not necessary[12] *to display all the other decorative pieces found in this issue.* I think various factors are playing a role in examples such as (5.14) and (5.15).

- First of all, so far I have presented instructions as commands according to which H adopts certain intentions and acts. However, instructional text also provides general rules of behavior, or advice on how to act in particular situations etc: in these cases, a PC provides, not so much a goal to achieve, but a description of the situation in which a certain behavior is appropriate, where by *appropriate* I mean that the behavior is effective, but not necessary. This seems to be the intended interpretations for both (5.14) and (5.15): for the former, we have *Save wallpaper and leftover adhesive* **in case you want** *to repair areas that may be damaged*; for the latter, *Build this simple*

[11] (5.15) could also be interpreted as a PC in syntactic terms.

[12] The definition of enablement (5.8) doesn't explicitly enforce that the condition $\mathcal{C}_i$ brought about by α is necessary; its necessity derives from the fact that $\mathcal{C}$ belongs either to the executability conditions on β, or to the generation enabling conditions between γ and β.

but elegant corner shelf **in case you want** *to display all the other decorative pieces found in this issue*. With respect to Balkanski's formalization of the Speaker's beliefs about the agent's beliefs and intentions, it seems that her finding that in utterances with PCs S believes that β will be intended on the part of the agent needs some refinement.

- In (5.14), the temporal effect also depends on the fact that α describes a *maintenance* action, namely, an action that brings about a state of the world that has to hold for a certain time τ. Notice the difference with *achievement* actions, whose importance is in achieving the change in the state of the world, not in the time that this new state has to hold for. As usual, the difference between the two types of action is not so clear-cut, but it makes intuitive sense. A crisper distinction could probably be obtained by looking at examples of maintenance actions — this is an issue left for future work.

- The other factor coming into play is whether α brings about **necessary** executability or generation-enabling conditions. In fact, the condition that α brings about may not be necessary, but simply facilitate or improve execution.

  Balkanski in [1990, p.27] mentions the *facilitate* relation between actions:

  > The data[13] also included instances of one action facilitating another action. For example, the utterance "*you can tell by the fact that it's a little easier if we turn the whole thing upside down*" is describing a situation where one agent believes that turning the structure upside down will facilitate the subsequent task of inserting screws. An activity Γ_1 *facilitates* another activity Γ_2 when the occurrence of Γ_1 brings about conditions that make the subsequent performance of Γ_2 "easier". The difficulty here is in formalizing a notion of "easier" that is relevant to such situations; in particular, there seems to be the need for a way of measuring the relative difficulty of a task.

  Again, the boundary between enablement and facilitation is not so clear-cut. First of all, facilitation could just be seen as α enabling an underspecified β: *Turning upside down facilitates inserting screws* because *Turning upside down enables inserting screws without having to hold them while screwing*.[14] Second, it is not clear whether executability conditions are always necessary: it depends on one's perspective, which in turn depends on one's *utilities*:

  > The central idea of mathematical decision theory is that a numerical utility function can be used to evaluate decisions ... A single numerical value is used to summarize the advantages of a set of actions. A typical utility function would be the profits realized from a particular investment outcome. [Feldman and Sproull, 1977, p.159]

[13] Her data is a videotape of two people building a piece of furniture (a swing glider).
[14] Charlie Ortiz, p.c.

The advantages of a certain action may also include that the resulting state makes it easier to do something else. Consider for example

(5.16) **To tile bathroom walls,** *remove wall-mounted basins or toilets.*

Is this enablement or facilitation? in a sense, it depends on H's perspective and on the situation at hand: in fact, it may be that in one's particular bathroom it is possible to tile the walls even without removing fixtures. Analogous considerations seem to hold for

(5.17) **To cool,** *place on rack.*[15]

It may be the case that it is not necessary to place the object in question on a rack to have it cool, but that doing so simply accelerates the cooling process.

In my opinion, a richer notion of actions and their relations, and of plans, e.g. including utility functions, is needed to address these issues.

5.2 Inference Processes

In Chapter 2, I talked about two kinds of inferences that I am interested in: one concerns how adjuncts constrain the interpretation of the action to be performed, as in (2.1b) and (2.2b); the other regards computing expectations about the location of β —the action described in the PC— and as a consequence, about the location of certain objects that β manipulates.

In the following, I will go into more detail as regards such inferences. A disclaimer before doing so: as I mentioned in Chapter 1, the focus of my work is not on learning. The problem I am addressing is understanding how the input logical form, that does not necessarily match the stored knowledge, is related to such knowledge, rather than discovering as yet unknown methods to achieve a certain goal.

Notice that all inferences stem from the same process: H tries to find the connection between α and β, by exploiting the fact that β describes the goal to be achieved. In computational terms, this amounts to using β as an index into the Knowledge Base, finding a collection of methods $\mathcal{M}_i$ that achieve β, and trying to relate α to an action that appears in one or more $\mathcal{M}_i$. During this process, we may found out that α is more or less specific than the stored action $\gamma_{i,j}$ that appears in a $\mathcal{M}_i$, or even incompatible with it; or that the connection between α and β holds, but only under certain expectations.

This is very much in the spirit of Allen's plan inference algorithm presented in Fig. 3.2: what is innovative in my work is the way the match is computed, and the inferences that are carried out in the process.

[15] This example is a variation of one in [Thompson, 1985].

5.2.1 Computing structural compatibility

The inferences I will discuss here are in a sense structural: namely, they are based on the fact that the action descriptions dealt with can be compared one to the other, and ordered according to some partial order, that in the next chapter I will identify with subsumption. Moreover, when a more flexible match between two action descriptions than simply one being an instance of the other is allowed, there must be some evidence that the match is tenable: so far, I have been talking about *goals* guiding the accommodation process, and therefore supporting the match between α and $\gamma_{i,j}$. This is the type of inference that I have studied in depth and that I will describe in the next section. However, one could foresee another kind of accommodation, in which the roles of α and β are in a sense switched: a goal β may not be specific enough to identify methods $\mathcal{M}_i$ that achieve it. However, some methods may be selected if they achieve a δ_i which is a subtype of β: α can then be used to select the correct $\mathcal{M}_i$. I will discuss this in Sec. 5.2.1.2.

5.2.1.1 Compatibility between α and $\gamma_{i,j}$, under β

Suppose H has knowledge of the form [16]

(5.18)

$$\mathcal{M}_{37} = \texttt{GENERATES}(\texttt{[cut(agent,square,in-half,along-diagonal)]}_{\gamma_{37}}, \texttt{[create(agent,two-triangles)]}_{\delta_{37}})$$

and he gets input instructions such as:

(5.19a) *[Cut the square in half along the diagonal with scissors]*$_{\alpha_1}$ *[to create two triangles]*$_{\beta}$.

(5.19b) *[Cut the square in half]*$_{\alpha_2}$ *[to create two triangles]*$_{\beta}$.

(5.19c) *[Cut the square in half with scissors]*$_{\alpha_3}$ *[to create two triangles]*$_{\beta}$.

(5.19d) *[Cut the square in half along a perpendicular axis]*$_{\alpha_4}$ *[to create two triangles]*$_{\beta}$.

In each example, the same relevant knowledge, namely (5.18), is retrieved by means of the goal β, which exactly matches δ_{37}. After (5.18) has been retrieved, the algorithm compares α_i with γ_{37}.[17]

1. In (5.19a), α_1 is more specific than γ_{37}. The inference is that α_1 is in fact the action to be executed, which is possible because the added modifier *with scissors* is compatible with γ_{37}.

2. In (5.19b), α_2 is less specific than γ_{37}: in this case, the inference to be computed is that the action to be performed is γ_{37} — this inference is illustrated in Fig. 5.4.

[16] In the next chapter, I will lay down all the details of the representation.

[17] γ_{37} should actually be referred to as $\gamma_{37,1}$. I will keep the simpler notation.

3. (5.19c) has the characteristics of both (5.19a) and (5.19b), given that α_3, when compared with γ_{37}, lacks the modifier regarding the position where to cut, but adds the instrument modifier *with scissors*; in this case, the inference to be computed is that the action to be performed is γ_{37} augmented with the added modifier *with scissors*.

4. Finally, in (5.19d), α_4 is incompatible with γ_{37}. At the least, the agent needs to be able to recognize such incompatibility.

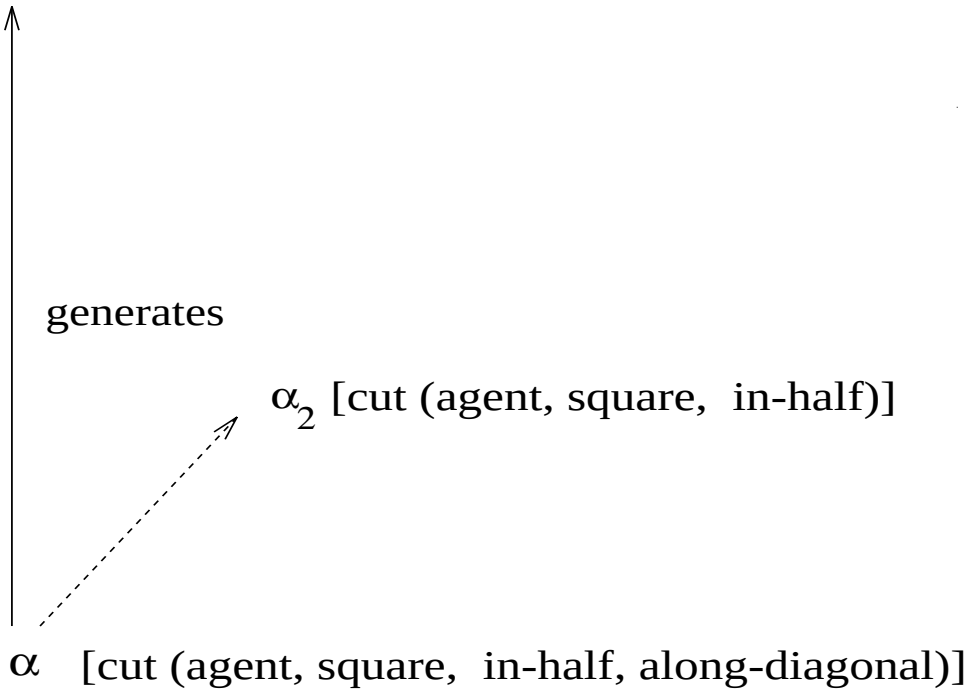


Figure 5.4: Computing more specific action descriptions

Analogously, in (5.11) we have to infer a richer description of α = *holding the cup* in order for α to enable *fill cup with coffee*. However, in this case the inference is more complicated because the augmented description of *hold* is not already available as part of the stored knowledge, as for *cut square in half along the diagonal*, but derives from some of the conditions on the generation relation between *depress the lever* and *fill cup with coffee*. Schematically it could be represented as shown in Fig. 5.5. I won't implement this inference explicitly, but extending my algorithm to deal with it would be possible.

Notice that all these inferences require that we are able to understand the relation between different linguistic descriptions: to do so, in Ch. 6 I propose an action representation formalism based on a taxonomy of action descriptions, which crucially exploits the classification algorithm provided by hybrid systems in the KRYPTON tradition [Brachman *et al.*,

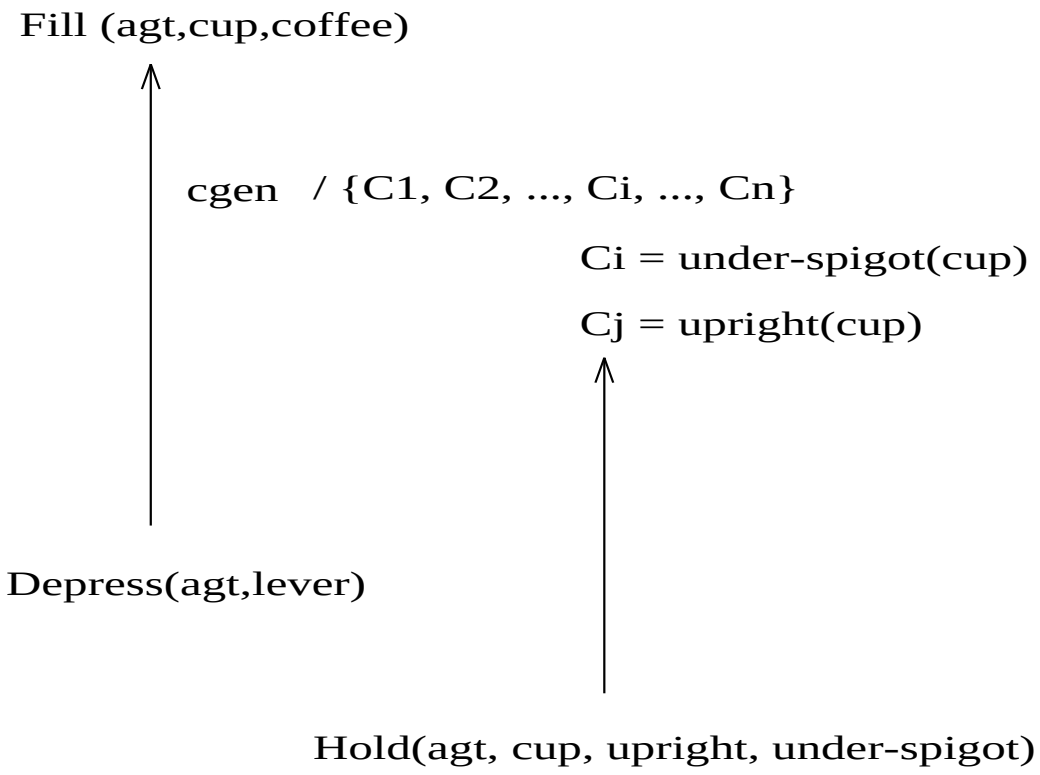


Figure 5.5: Inferring a more complete description for *hold*

1983b].

5.2.1.2 Compatibility between β and δ_i, under α

As mentioned above, one could consider a different kind of accommodation, where the match between input instruction and knowledge is supported by α rather than by the goal β: that is, one could allow β to retrieve the $\mathcal{M}_i$'s that achieve subtypes δ_i of β, and exploit α to select the correct $\mathcal{M}_i$('s).

Suppose that H, in addition to the knowledge about obtaining triangles from squares in (5.18), has some further knowledge that cutting squares in half, this time along a perpendicular axis, results in creating two rectangles:

(5.20)

$$\mathcal{M}_{38} = \texttt{GENERATES(cut(agent,square,in-half,along-perp-axis)]}_{\gamma_{38}}, \texttt{[create(agent,two-rectangles)]}_{\delta_{38}})$$

Consider now the possible input instructions in (5.21), instructions that parallel the ones in (5.19):

(5.21a) *[Cut the square in half along the diagonal]*$_{\alpha_1}$ *[to create two polygons]*$_{\beta}$.

(5.21b) *[Cut the square in half]*$_{\alpha_2}$ *[to create two polygons]*$_{\beta}$.

(5.21c) *[Cut the square in half with scissors]*$_{\alpha_3}$ *[to create two polygons]*$_{\beta}$.

(5.21d) *[Cut the square in half along a perpendicular axis]*$_{\alpha_4}$ *[to create two polygons]*$_{\beta}$.

For all the cases in (5.21), β would retrieve both (5.18) and (5.20), as δ_{37} and δ_{38} are both subtypes of β. Then

- As regards (5.21a) and (5.21d), both α_1 and α_4 are specific enough to select the right method, as each is consistent only with γ_{37} and γ_{38} respectively. This has the side effect of further specifying the goal, by identifying the two polygons mentioned in β as triangles and rectangles respectively.

- On the other hand, in (5.21b) and (5.21c), neither α_2 nor α_3 is specific enough to discriminate between the two methods.

I will get back to possible implementations of these inferences in Sec. 7.2.1.3.

5.2.2 Computing expectations about object locations

As is reported in a paper about AnimNL [Webber *et al.*, 1993, p.2],

> The *expectations* that instructions lead agents to form serve as an influence on behavior — what they do and what they look for — over and beyond their current perceptions. As such, expectations from instructions complement information from the world in guiding an agent's behavior.

Some expectations that arise from instructions may regard duration of processes, or consequences of actions, or locations of objects — this latter type is the one I will discuss.

Expectations regarding locations of objects are quite common in scenarios where an action changes the *perceptual space* an agent has access to, clearly under the (reasonable) assumption that agents only have *limited perception*: they cannot have up-to-date knowledge of any part of their environment which is outside their direct perception. In the instruction

(5.22) *Open the safe and hand over the money.*

one expects the money to be found inside the safe. Analogously in

(5.23) *Remove access cover A. Raise handle on cartridge and turn to unlock position.*

the action of removing the cover opens up a new space, inside which the agent will find the cartridge.

Further, notice that the inferences about object locations are in a sense secondary, the primary inference being as follows: if H executes an action α that changes H's perceptual space and/or the space that is accessible to H, and α is executed for the purpose of doing β, then expect the new perceptual space μ to be the site either of β, or, if α belongs to a sequence $\mathcal{A}$ generating β, of the rest of the actions belonging to $\mathcal{A}$. In particular, when α results in the agent going to a place μ with the purpose of doing β, one can infer μ to be the site of β.

This can be seen by reconsidering (2.3a) and (2.3b), repeated here for convenience:

(5.24a) *Go into the kitchen to get me the coffee urn.*

(5.24b) *Go into the kitchen to wash the coffee urn.*

In both cases, H goes to the kitchen, which is then expected to be the location of the (rest) of the actions belonging to the plan to achieve β. As I mentioned in Ch. 2, in (5.24a) the expectation $\mathcal{E}_1$ that the referent of the coffee urn is in the kitchen is developed, while this does not happen in (5.24b).

What I will argue is that it is the relation $\mathcal{R}$ between α and β, combined with the specific knowledge about both α and β, that gives rise to the expectations about the location of objects involved in an action. Notice in fact that such expectations don't derive simply

from knowledge about the action to be performed. For example, neither (5.25a) nor (5.25b) convey much information about the locations of the coffee urn or of the washing materials.

(5.25a) *Get me the coffee urn.*

(5.25b) *Wash the coffee urn.*

Moreover, such expectations can't simply be attributed to *world knowledge*. For example, one explanation of the oddness of (5.26) is that one doesn't normally find washing equipment in the living room. As such, (5.26) indeed appears to convey the expectation $\mathcal{E}_2$ that *the washing materials are in the living room*, which goes against world knowledge.

(5.26) *Go into the living room to wash the coffee urn.*

The difference between the two may be explained by appealing to a notion deriving from the planning literature and called, alternatively, *qualifiers, applicability conditions* [Schoppers, 1988], *constraints* [Litman and Allen, 1990], or *usewhen conditions* [Tate, 1987]: these are conditions that must hold for an action to be relevant, and are not meant to be achieved. If β has among its applicability conditions that an argument be at μ for β to even be relevant, then a locational expectation develops as in (5.24a). If not, a weaker expectation arises, as in (5.24b).[18]

What usage does an agent make of such expectations? They can be used in the process of *reference grounding*, namely, of binding the referring expressions in the instructions to objects in the world. Expectations may have a pruning effect on possible referents: in (5.24a), even if there is a coffee urn right in front of the agent, he won't assume that that is the urn he is supposed to get. Expectations can also guide the agent's behavior: if the agent steps into the kitchen and there is no visible coffee urn, he will start searching for one, for example opening cabinet doors etc.

A different behavior will result if the agent can interpret some features of the kitchen environment as evidence against $\mathcal{E}_1$. For example, there may be a note laying around *the urn is not here but in the living room*, or a person who looks like they're in charge of things.

Schematically, one could represent this kind of inference as in Fig. 5.6 – β is the goal, α the instruction to accommodate, A_k the actions belonging to the plan to achieve β, C the expectations computed while deriving the relation $\mathcal{R}$ between α and β. [19]

[18] Notice that this occurs even within a single clause: *Get me the coffee urn in the kitchen* leads to the same expectation as (5.24a), while *Wash the coffee urn in the kitchen* doesn't.

[19] Balkanski in [1993, p.101] has the following comment on this kind of inference: *"assumptions" would no longer be needed in addition to conditions on generation relations because the difference between these two types of conditions appear to rest on a difference between the beliefs of the speaker and that of the actor.*

I think she is mistaken in interpreting my claim as that there are different kinds of conditions involved. Expectations (which I used to call assumptions in previous work) are not necessarily further conditions, as they may regard one or more pre-defined generation-enabling conditions. Rather, an expectation singles out one of these conditions as particularly relevant to the current situation, for example because it gives rise to a particular kind of behavior — in Ex. (5.24a), if the agent goes into the kitchen and doesn't see a coffee urn, he will start looking for one. Besides, expectations don't stem from *a difference between the beliefs of the speaker and that of the actor.* In uttering (5.24a), S has exactly the same belief that H will derive from it, namely, that *the coffee urn is in the kitchen*; it is during the process of H's understanding the relation $\mathcal{R}$ between α and β that expectations arises.

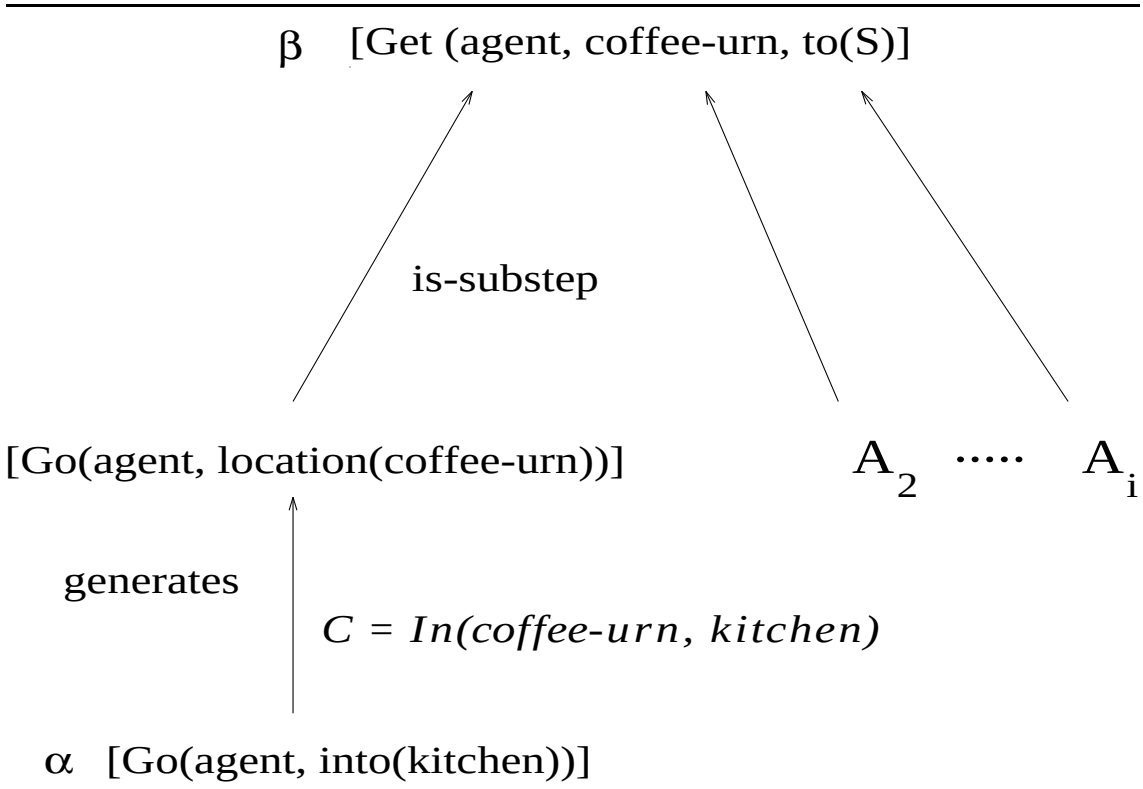


Figure 5.6: Computing expectations

5.3 Summary

In this chapter, I have analyzed Purpose Clauses from a computational point of view. First, I described which relations between actions are expressed in an instruction like *Do* α *to do* β: I focussed on generation and enablement, and on some problems related to using them. Then I discussed some accommodation inferences that are necessary to deal with purpose clauses.

The inferences necessary to understand PCs impose not only processing constraints, but also requirements on the action representation system: more specifically, it is necessary to represent actions descriptions both from a linguistic point of view, which among others requires representing them at different levels of abstractions, and from a planning point of view. In the next chapter, I will describe the representation formalism in full details.

In Sec. 5.2 I sketched two kinds of accommodation inferences that directly derive from purpose clauses, namely, computing the relation between the input and the stored action descriptions; and inferring the expectations under which a certain instruction makes sense. Both kinds of inferences are derived by exploiting the constraints deriving from the goal. The algorithm that implements them is described in Ch. 7.

Chapter 6

The Action Representation Formalism

The formalism I will present addresses the different demands posed by the need to represent both linguistic and planning knowledge about actions.

I will present a hybrid system whose taxonomy is expressed by means of Conceptual Structure primitives [Jackendoff, 1990]; the taxonomy is augmented by means of an action library, containing commonsense plans about actions.

I will start by summarizing the observations I made in previous chapters on the features that an action representation formalism should include to be adequate to model NL descriptions of actions – these observations mainly rest on the analysis of purpose clauses that I presented in the previous two chapters, on the analysis of negative imperatives presented in Ch. 4, and on a less detailed analysis of other constructions, such as *free adjuncts* and *check constructions*, on which I report elsewhere [Webber and Di Eugenio, 1990]. Notice that in the whole discussion by *action* I mean *action type*, and by *action description*, *action type description*.

1. **Linguistic** knowledge about action descriptions.

   (a) It has probably become obvious by now, but let me reiterate that, like individuals, sets of individuals, propositions [Quantz and Kindermann, 1990] etc. actions should be part of the underlying ontology of the representation formalism. This has also been advocated by Jackendoff, who justifies it through our ability to refer to an action – *I did* **it**, or to ask questions about them – **What** *did you do?* [Jackendoff, 1983].

(b) Action descriptions can always be further specified. Consider:

(6.1a) *Apply paste to the wall.*

(6.1b) *Using a paint roller or brush, apply paste to the wall.*

(6.1c) *Using a paint roller or brush, apply paste to the wall, starting at the ceiling line and pasting down a few feet and covering an area a few inches wider than the width of the fabric.*

Therefore the formalism must be able to represent action description at various levels of specificity, and to represent not just the usual participants in an action such as agent or patient, but also means, manner, direction, extent etc.

(c) The formalism must be able to support computations that involve actions that do not exactly correspond to its stored knowledge; in fact, as I showed in Sec. 5.2.1.1, it must be able to understand the relation between the input action descriptions and the stored ones.

2. **Planning knowledge** about actions.

(a) The formalism must be able to represent various relations among actions, such as at least temporal relations, generation and enablement.

(b) To reason about complex instructions, the formalism must include notions of action decomposition into substeps, of *applicability* conditions, of *effects* expected to occur when an action of a given type is performed: all these notions are typical of knowledge used by planning and plan inference systems — see for example [Fikes and Nilsson, 1971; Schoppers, 1988; Litman and Allen, 1990].

To address these issues, I propose a formalism composed of two KBs. The first one stores linguistic knowledge about actions, implemented by means of the T-Box of a hybrid system: the semantic primitives of the T-Box representation are those defined by Jackendoff in his work on Conceptual Structures [1983; 1990].

The terms defined in the ontology of the hybrid system are used in the *action library*, which contains *commonsense* planning knowledge about actions, expressed by means of *recipes* or *plans*: these could include simple information such as *to loosen a screw, you have to turn it counterclockwise*, up to more complex procedures, such as how to assemble a piece of furniture.

The value of a hybrid KR system to this enterprise is in associating different inference mechanisms with different kinds of knowledge. Consider Ex. (5.19d), repeated here for convenience:

(6.2) *Cut the square along a perpendicular axis to create two triangles.*

As mentioned in the previous chapter, the relation between *cut the square along a perpendicular axis* and *create two triangles* is at least odd, if not ill-formed, because cutting a square along an axis won't create two triangles — it is inconsistent with the stored knowledge. However, the two action descriptions by themselves are perfectly well-formed. In

order to track down the ill-formedness above, a homogeneous representation system that does not support distinguishing between different kinds of knowledge could e.g. try to prove that *cut the square along a perpendicular axis* is not a well-formed description, a clearly undesirable behavior.

In the following, I will first describe hybrid systems, and the reasons why the classification algorithm they are endowed with is necessary to perform the inferences described in Sec. 5.2.1.1. I will then describe Jackendoff's Conceptual Structures (CS for short), and show how the CS primitives can be used to express knowledge about actions in the context of a T-Box. I will then turn to describing the *recipes* in the action library.

6.1 The action taxonomy

6.1.1 Hybrid Systems

Hybrid Knowledge Representation systems, such as KRYPTON [Brachman *et al.*, 1983b], KL-TWO [Vilain, 1985], LOOM [Mac Gregor, 1988], BACK [Quantz and Kindermann, 1990] and CLASSIC [Brachman *et al.*, 1991], stemmed from the KL-ONE formalism [Brachman and Schmolze, 1985].

According to [Mac Gregor, 1991, p.386]

> KL-ONE and its descendants share a commitment to the following architectural features:
>
> - They are logic-based (i.e., they appeal to first-order logic for their semantics).
> - They draw a distinction between *terminological* and *assertional* knowledge, and each system implements its own specialized term-forming language.
> - They include a *classifier* that organizes terms (concepts) into a taxonomy, based on subsumption relationships between terms.

Hybrid systems are composed of two parts: a terminological part, or T-Box, that is used to define terms, and an assertional part, or A-Box, used to assert facts or beliefs.

The terminological language is designed to facilitate the construction of expressions that describe classes of individuals. The formal constructs in a terminological language are called terms or *concepts*.

The purpose of an assertion language is to state constraints or facts that apply to a particular domain or world. In general, the A-Box language employs a somewhat restricted first-order logic, for example function-free, in order to make theorem proving computationally tractable.

All T-Box information is definitional in nature. The T-box language has two main categories, *concepts* and *roles*, roughly corresponding to frames and slots. Concepts are organized in a hierarchy of structured terms. The relation between terms in the hierarchy

is *subsumption*. In the following, I will take advantage of the very clear introduction to subsumption found in [Woods, 1991]. To start with, Woods notices that in virtually every semantic network formalism, there is at least one link that relates more specific concepts to more general concepts (from which generic information can be inherited). Such links have been variously named "is a", "kind of", "subset of", "member of", "subconcept of", "subkind of", "superconcept", "ako", etc. Such links are used to organize concepts into a hierarchy or some other partial ordering. Woods calls this structure a "taxonomy". The taxonomy is used for storing information at appropriate levels of generality and automatically making it available to more specific concepts by means of a mechanism of inheritance. More general concepts in such a partial order are said to *subsume* more specific concepts, and a more specific concept is said to *inherit* information from its subsumers. Classification is the operation of assimilating a new description in a taxonomy of existing concepts by automatically linking it directly to its most specific subsumers and to the most general concepts that it in turn subsumes. Most other semantic networks, as well as frame-based representations and object-oriented programming systems, provide such inheritance only for concepts that are directly placed by their designers (or by specific programs) at the correct position in the taxonomy to inherit what is intended.

Later in the same article Woods gives a more detailed definition of *taxonomic structure* [p.79-81]. He notes that the structure that results from organizing conceptual descriptions on the basis of subsumption is a partial ordering, and that, depending on the concept-forming operators involved, the structure may be a formal lattice with respect to subsumption (that is, a partial ordering in which each pair of concepts has a unique least upper bound in the form of a most specific joint subsumer and a unique greatest lower bound in the form of a single most general common subsumee). For any collection of relations and taxonomic concepts, there is a rather large and combinatoric space of all possible composite descriptions that can be formed from them. This space of all descriptions, organized by the collection of all such subsumption relations, has been referred to as the "virtual taxonomy", because, although its structure is important, one never wants to make it explicit in the memory of a computer (or a person).

Woods continues by observing that, whenever a reasoning agent constructs an explicit collection of concept nodes and computes and records the subsumption relationships among those concepts, the result is a subgraph of the virtual taxonomy. All of the operations of classification and deduction operate on the explicit taxonomy, and every description that is added to it is in some sense already in the virtual taxonomy (except when adding a new primitive concept or relation). Thus, classifying a new concept is, in a sense, merely making explicit a concept that already exists in the virtual taxonomy.

Finally, Woods defines *classification* as follows [p.82]:

> Classification is ... the process by which new concepts are added to an existing taxonomy and linked in at the appropriate position or by which a collection of concepts linked by some recorded subsumption relationships is expanded to include all direct subsumption relationships that can be inferred by means of the structural subsumption rule. Note that this operation is an operation on a system of concepts, unlike subsumption, which is an operation on a pair of

concepts.

To sum up:

T-Box. It has its own *term-definition language*, which provides primitives to define *concepts* and *roles* — the latter are used to express relations between concepts. It is a virtual lattice determined by the *subsumption* relation between concepts. It uses a *concept classifier*, which takes a new concept and determines the subsumption relations between it and all the other concepts in a given Knowledge Base. By means of the classifier, a new concept can be automatically assimilated into the taxonomy by "placing it in the right place", i.e. linking it to its most specific subsumers and its most general subsumees.

A-Box. As [Mac Gregor, 1991, p.389] notices, while there is considerable agreement within the hybrid systems community on what terminological capabilities should be provided, there is no comparable consensus on what the assertional component should look like. However, there are two features, each of which appears in the assertional component of the more recent hybrid systems, that may eventually become established characteristics of classification-based technology. The first feature is a general capability for specifying the attachment of constraints to concepts, e.g. by specifying *rules* attached to a concept, and the second is an object-centered approach to the assertional reasoning component.

Finally, the concept of *recognizer* will be relevant to the inferences I will discuss:

> The set of concepts that an individual in a knowledge base belongs to is called the *type* of that individual. ... Each of the above three systems[1] implements an assertional reasoner called a *recognizer* [2] whose function is to maintain current values for the types of all individuals in the knowledge base. [Mac Gregor, 1991, p.389]

I will assume from now on that my ontology in the T-Box includes action descriptions besides the usual hierarchies about "entities". I would like now to again pose the question: why use a hybrid KR system, with the added complexity that classification involves, instead of using a simpler representation augmented with an inheritance mechanism? The answer lies in two crucial characteristics of a T-Box: the facts that the taxonomy is a virtual lattice, and that it can be extended by means of the classifier.

In Sec. 5.2.1.1 I showed that it is necessary to understand the relation between the various possible logical forms α_i and the stored α. I have found subsumption to be both powerful enough and restrictive enough to capture all relations of interest. For example, in (5.19d) the classifier is crucial: the action descriptions *cut the square in half along the diagonal* and *cut the square in half along a perpendicular axis* are structurally exactly the same, but the incompatibility is detected when examining the fillers of the role *along* on the two

[1] BACK [Quantz and Kindermann, 1990], LOOM [Mac Gregor, 1988], CLASSIC [Brachman *et al.*, 1991].
[2] Called *realizer* elsewhere, e.g. in [Nebel, 1990].

different concepts.[3] Notice that computing subsumption doesn't per se yield incompatibilities between two concepts. However, all hybrid systems augment the basic mechanism so that various queries can be posed to the KB: one of these is whether two concepts are disjoint, which is answered by *and*ing the two concepts and checking whether the resulting concept is inconsistent.

Moreover, notice the importance of having a *virtual* lattice. Suppose we have a certain description of an action, such as the stored γ_{37} in (5.18): even keeping the names of predicate, arguments and modifiers fixed, it is always possible to envision a surface string that does not exactly correspond to γ_{37}; therefore it is important to be able to understand the relation between stored knowledge and input description as the need arises.

Many researchers have mentioned that taxonomies of actions are necessary, especially for plan inference. However, they have either focussed on their use in representing only planning, and not also linguistic knowledge — [Devanbu and Litman, 1991], [Wellman, 1988], described in Sec. 6.2.2, [Kautz, 1990], [Tenenberg, 1989], described in Sec. 7.3; or, as in [Balkanski, 1993], just noted that action descriptions form a lattice, but not exploited the power of the virtual lattice, as I have here, to capture differences between the input and the stored action descriptions.

Before describing the features of the hybrid system CLASSIC [Brachman *et al.*, 1991], I will summarize how I use it.

1. First of all, actions are full fledged entities in my formalism; namely, a subtaxonomy of the general taxonomy will comprise action description concepts. In practice, verb phrases will be mapped to concepts in the T-Box, decomposed according to Jackendoff's Conceptual Structures — see Sec. 6.1.3.

2. The A-Box as usual contains assertions about individuals; however, the individuals in this case will include both individuals in the traditional sense, and *action description* instances — notice that individual action descriptions are not action tokens. In practice, this corresponds to reifying action descriptions, just as propositional descriptions were reified in PSI-KLONE [Brachman *et al.*, 1979].

3. The parser produces a logical form which relates one or more action descriptions according to the connectives contained in the input utterance. The recognizer with which the A-Box is equipped will therefore compute the set of types of which such logical forms are instances.

4. Finally, those individuals that play the role of goals will be used as indexes to retrieve relevant recipes from the action library; if they are not goals, they will be matched with the substeps of the recipe(s) found in the retrieval step.

The interplay between the various parts will become clearer in Secs. 6.1.3 and 6.2.2, and in the next chapter.

[3] There is actually no real role *along*: this is just an approximation, see Sec. 6.1.3 for the full details of the formalism.

6.1.1.1 The CLASSIC system

Here I will describe the CLASSIC system that I use in my implementation [Borgida *et al.*, 1989; Brachman *et al.*, 1991]. [Woods and Schmolze, 1991, p.38] gives the following introduction to CLASSIC:

> Here [in CLASSIC], we find that certain traits of KRYPTON have remained: CLASSIC was designed to have tractable subsumption but with greater expressibility. We also find that some traits are gone: there is no physical separation of the T and A boxes. In CLASSIC, the language for describing terms is the same as the language for describing individuals. The definition versus assertion distinction remains, but it is less clear than in KRYPTON given the lack of separate T and A boxes. The classifier only pays attention to definitional operators. ... Overall, the goals of CLASSIC appear to be more pragmatic than those of KRYPTON. CLASSIC's designers have taken advantage of their experience regarding user needs, implementational methods, complexity measures, and the importance of certain distinctions. The result is a tractable and reasonably expressive KR system.

CLASSIC's language is described as follows in [Brachman *et al.*, 1991, p.407-409]:

Concept Forming Operators.

- **PRIMITIVE, DISJOINT-PRIMITIVE.** They allow a user to form concepts that cannot be fully specified by necessary and sufficient conditions. A **DISJOINT-PRIMITIVE** concept is just like a **PRIMITIVE** concept, except that any concepts within the same "disjoint grouping" are known to be disjoint from each other.
- **AND.** The **AND** operator creates a new concept that is the conjunction of the concepts given as arguments. For example, if `WHITE-WINE` and `FULL-BODIED-WINE` are two concepts that have been previously defined, their conjunction is defined as

  (**AND** `WHITE-WINE` `FULL-BODIED-WINE`)

- **ONE-OF.** A **ONE-OF** concept (or enumerated concept) enumerates a set of individuals, which are the only instances of the concept. For example, a wine whose body could be either full or medium would have the restriction

  (**ALL** `body` (**ONE-OF** `Full Medium`))

Role Restrictions. The five operators **ALL, AT-LEAST, AT-MOST, FILLS,** and **SAME-AS** form expressions known as *role restrictions*.

A universal value restriction, or **ALL** restriction, specifies that all the fillers of a particular role must be individuals described by a particular concept expression. For example, a `CALIFORNIA-WINE` might be defined as a wine whose region is a California region, where the California regions are Napa Valley, Sonoma Valley, etc. The `region` role restriction would be written

(**ALL** `region CALIFORNIA-REGION`)

AT-LEAST and **AT-MOST** restrictions restrict the minimum and maximum number of fillers allowed for a given role on a concept or individual. For example, part of the definition of a wine might be that it is made from at least one kind of grape, which would be written

(**AT-LEAST** `1 grape`)

where `grape` is a role.

The **FILLS** operator specifies that a role is filled by some specified individuals (although the role may have additional fillers). For example, we might define the concept `CHARDONNAY-WINE` as a wine whose grapes include chardonnay; the restriction would be written as

(**FILLS** `grape Chardonnay`)

A **SAME-AS** restriction requires that the individual found by following one attribute path is the same individual as that found by following a second attribute path.[4] For example, suppose that there is a food and a drink associated with each course at a meal. Then the concept `REGIONAL-COURSE` might be defined as a course where the food's region is the same as the drink's region. This would be written as (**AND** `MEAL-COURSE` (**SAME-AS** `(food region) (drink region)`))

Moreover, it is possible to use procedures in specifying concepts, by means of the **TEST-C / TEST-H** operators. Finally, one can express rules of the form A $\Rightarrow$ B where A and B are concepts. Rules are not considered by the classifier and are thus considered assertional.

[Brachman *et al.*, 1991, p.405] describes the deductive inferences that CLASSIC provides:

- *Completion:* Logical consequences of assertions about individuals and descriptions of concepts are computed; there are a number of "completion" inferences CLASSIC can make:
  - *Inheritance:* Restrictions that apply to instances of a concept must also apply to instances of specializations of that concept. In a sense, then, properties are "inherited" by more specific concepts from those that they specialize.
  - *Combination:* Restrictions on concepts and individuals can be logically combined to make narrower restrictions.
  - *Propagation:* When an assertion is made about an individual, it may hold logical consequences for some other, related individuals. CLASSIC "propagates" this information forward when an assertion is made.[5]

[4] Attributes express 1:1 relations.

[5] For example suppose we know that Sue drinks Chateau d'Yquem Sauterne, and we tell CLASSIC that Sue drinks only dry wines. The information is then propagated that the individual `Chateau-d-Yquem-Sauterne` must be a dry wine. [Brachman *et al.*, 1991, p.411].

- *Contradiction detection:* It is possible to accidentally assert two facts about an individual that are logically impossible to conjoin. CLASSIC detects this kind of contradiction.
- *Incoherent concept detection:* It is possible to accidentally give a concept some restrictions that combine to make a logical impossibility, thereby not allowing any instances of the concept to be possible. CLASSIC detects this kind of inconsistent description.

- *Classification and subsumption:*
  - *Concept classification:* All concepts more general than a concept and all concepts more specific than a concept are found.[6]
  - *Individual classification:* All concepts that an individual satisfies are determined.[7]
  - *Subsumption:* Questions about whether or not one concept is more general than another concept are answered.

- *Rule application:* Simple forward-chaining rules have concepts as antecedents and consequences. When an individual is determined to satisfy the antecedent of a rule, it is asserted to satisfy the consequent as well.

6.1.2 Conceptual Structure representation

One of my criticisms of action representation formalisms for plan inference in NL is that they too often neglect the linguistic side of the input description. The issue is then to find a representation that is both linguistically motivated, and that is also useful with respect to the representation of planning knowledge. I think that work done in lexical semantics, and in particular in its subfield of lexical decomposition, answers these two requirements.

In [Shapiro, 1992, p.812-813], Pustejovski characterizes lexical semantics as follows:

> The study of natural language semantics is the investigation of how and what linguistic utterances denote. The subdiscipline of lexical semantics, then, can be seen as the study of how and what the words in a language denote. We can partition the concerns of lexical semantics into the two issues of *meaning* (the *how*) and *reference* (the *what*). Some lexical semantics theories actually have little if anything to say about the problem of reference, and concentrate on the *structural* properties of word meaning,[8] while those providing for modes of reference often have nothing to contribute to the structural aspects of word meaning.

According to [Levin and Pinker, 1991, p.2-3]

[6] Note that object-oriented programming languages usually have inheritance, but not classification.
[7] This was called *recognition* or *realization* in Sec. 6.1.1.
[8] For example [Jackendoff, 1990].

> The assumption underlying much of this current linguistic research — that syntactic properties of phrases reflect, in large part, the meanings of the words that head them — also provides a powerful new methodology for studying word meaning. ... When the technique of searching for syntax-relevant distinctions is applied to many words and many constructions, a small set of semantic elements tends to recur. ... Jackendoff's paper introduces the notion of "conceptual semantics" — a characterization of the conceptual elements by which a person understands words and sentences, to be distinguished from much of formal linguistic semantics which characterize the abstract relation between words and sentences and the external world.

Within lexical semantics, work on *lexical decomposition* is the closest to my concern of providing a principled representation for action verbs. Fass and Pustejovsky in [Shapiro, 1992, p.806] note that lexical decomposition is that branch of lexical semantics concerned with the internal semantic structure of lexical and conceptual items within a lexicon. They observe that the focus of lexical decomposition is on how the lexical items are semantically similar and distinct by virtue of shared knowledge structures containing semantic primitives, while its goal is to provide the necessary and sufficient conditions for the meaning of every lexical item in a subject domain or language. If primitives and structures are taken as an exhaustive set on top of which all expressions in the language are expressed, then the meaning of any lexical item in the language must be derived from these terms.

Jackendoff's Conceptual Structures [1990] seems to provide a good foundation for lexical decomposition. In particular, there are two reasons that make it appropriate for my purposes, and that I will illustrate in the following: first, the primitives of his decompositional theory capture important generalizations about action descriptions and their relationships to one another, and second, they reveal where information may be missing from an utterance and may have to be provided by inference. Given that one of my concerns, and the concerns in AnimNL, is that of interpreting a NL instruction from the surface down to its (simulated) execution, it is clear that this latter characteristics is particularly important.

A comment before introducing the notation and some minor modifications to the theory as presented in [White, 1992], and applying it to the representation of the clause *Go into the kitchen*. According to Fass and Pustejovski in [Shapiro, 1992, p.808-809], Jackendoff believes in the cognitive primacy of the primitives used within his system, and the role of these primitives in performing inferences. They point out that it is important to realize that Jackendoff sees semantic representation in general as a subset of conceptual structure, where *conceptual structure* is a term for the language of mental representation. Finally, they observe that much of Jackendoff's approach begins with an analysis of spatial relations and how they are realized in the language.

Coming now to Jackendoff's theory proper, an entity may be of ontological type *Thing, Place, Path, Event, State, Manner* or *Property*. The conceptual structure for a KITCHEN

is shown in (6.3a) below:

(6.3a) $[_{\text{Thing}}\ \textsc{kitchen}]$

(6.3b) $[_{\text{Thing}}\ \textsc{room}]$

Square brackets indicate an entity of type Thing meeting the enclosed featural description. Small caps indicate atoms in conceptual structure, which serve as links to other systems of representation; for example, the conceptual structure for a room (6.3b) differs from that of a kitchen only in its choice of constant. Jackendoff leaves the determination of their similarities and differences to a system of representation better suited to the task, able to address perceptual distinctions: for example, in AnimNL this would correspond to the graphics simulation system. Moreover, some of these distinctions can be captured by means of a taxonomy — see Sec. 6.1.3.

To distinguish instances of a type, [Zwarts and Verkuyl, 1993] augment Conceptual Structures with an index, as in (6.4):

(6.4) $[_{\text{Thing}}\ \textsc{room}]_1$

Conceptual structures may also contain complex features generated by conceptual functions over other conceptual structures. For example, the conceptual function IN: Thing → Place may be used to represent the location *in the kitchen* as shown in (6.5a) below. Likewise, the function TO: Place → Path describes a path that ends in the specified place, as shown in (6.5b):

(6.5a) $[_{\text{Place}}\ \text{IN}([_{\text{Thing}}\ \textsc{kitchen}]_k)]_l$

(6.5b) $[_{\text{Path}}\ \text{TO}([_{\text{Place}}\ \text{IN}([_{\text{Thing}}\ \textsc{kitchen}]_k)]_l)]_m$

(6.5c) $[_{\text{Path}}\ \text{TO}(l)]_m$

(6.5c) is an equivalent representation of (6.5b), where the index l stands for the entire Place constituent. This move considerably lessens the complexity of representing large conceptual structures; to further lessen this complexity, indices and ontological types will often be left out.

To complete the representation of *Go into the kitchen*, it remains only to add the conceptual function GO: Thing × Path → Event:

(6.6) $[_{\text{Event}}\ \text{GO}([\textsc{you}]_i, m)]$
$\quad [_{\text{Path}}\ \text{TO}([\text{IN}([\textsc{kitchen}])])]_m$

To distinguish *Walk into the other room* from (6.6), [White, 1992] adds an indication of manner to the basic representation:[9]

(6.7) $\begin{bmatrix} \text{GO}(i, m) \\ [_{\text{Manner}}\ \textsc{walking}] \end{bmatrix}$

White's final addition to Jackendoff's theory is a new semantic field. Semantic fields, such

[9]Though this is clearly intended, Jackendoff never explicitly represents such a distinction.

as Spatial and Possessional, are intended to capture the similarities between sentences like *Jack went into the kitchen* and *The gift went to Bill*, as shown in (6.8) below:

(6.8a) $[\text{GO}_{\text{Sp}}([\text{JACK}], [\text{TO}([\text{IN}([\text{KITCHEN}])])])]$

(6.8b) $[\text{GO}_{\text{Poss}}([\text{GIFT}], [\text{TO}([\text{AT}([\text{BILL}])])])]$

In fact, in Fass and Pustejovsky's words [Shapiro, 1992, p.809]

> Jackendoff ... explores the idea that the predicates within the spatial domain may be used to analyze concepts within other semantic domains. The basic idea here is that for different semantic fields such as possession, identification, circumstantial, and temporal, a verb from the spatial field can acquire a new meaning using the same primitives because it is being evaluated relative to a new field.

Verbs like *go* leave the semantic field underspecified, whereas verbs like *donate* specify a particular field. White's new semantic field is called Control. It is intended to represent the functional notion of *having control over* some object. For example, in sports, the meanings of *having the ball*, *keeping the ball*, *getting the ball*, and *losing the ball* embody this notion, and are clearly quite distinct from their Spatial and Possessional counterparts: Control captures these additional similarities in an analogous way. The similarity between *Jack has the money* and *Jack has the ball*, for instance, is shown in (6.9):

(6.9a) $[\text{BE}_{\text{Poss}}([\text{MONEY}], [\text{AT}([\text{JACK}])])]$

(6.9b) $[\text{BE}_{\text{Ctrl}}([\text{BALL}], [\text{AT}([\text{JACK}])])]$

Notice that the notion of Control is very relevant to AnimNL's domain, given that any action involving direct physical manipulation requires that the agent have the object to be manipulated under his control.

An important CS primitive that will play a role in the following is CAUSE — [Jackendoff, 1990, p.44]:

> An Event [can be elaborated] as the Event-function CAUSE plus two arguments. The first argument, if a Thing, is the agent; if an Event, is Cause. The second argument, an Event, is the Effect.

Schematically

(6.10) $[_{\text{Event}}\ \text{CAUSE}([_{\text{Thing/Event}}]_i, [_{\text{Event}}\ ]_j)]$

Later, Jackendoff notes a confusion between two tiers into which conceptual roles fall: a *thematic tier* dealing with motion and location, and the related notions of *Goal* and *Theme*; and an *action tier* dealing with *Actor* and *Patient* relations. For example, in (6.11) *Bill* is both Theme and Actor, but *the room* is only the Goal, not also the Patient: in fact, in (6.11) there is no Patient.

(6.11) *Bill entered the room.*

According to [Jackendoff, 1990, p.128]

> The action tier enables us to dissect the traditional notion of Agent into a number of independent parts ... One sense of Agent, "extrinsic instigator of action", is captured by the role "first argument of CAUSE", an element in the thematic tier. However, a second sense is "volitional actor". This appears, for instance, in the well-known ambiguity of *Bill rolled down the hill*, where Bill may or may not be performing the action willfully. Generally, it seems that any Actor, if animate, is subject to this ambiguity, unless the verb specifically selects for a volitional Agent, as do, for instance, *buy* and *look*.

To sum up, the Event functions described so far, and in particular CAUSE, capture only the notions of Theme, Source, and Goal. To capture the notions of Actor and Patient, Jackendoff adds a new dimension to all such functions, which is

(6.12) $\mathrm{AFF}([_{\mathrm{Thing}}]_i, [_{\mathrm{Thing}}]_j)$

Without going into all the details that Jackendoff examines, (6.13b), which represents (6.13a), illustrates the usage of the two tiers:

(6.13a) *Harry gave Sam a book*

(6.13b)

$$\left[\begin{array}{l} \mathrm{CAUSE}\,([\textsc{harry}],\,[\mathrm{GO}_{\mathrm{Poss}}\,([\textsc{book}],\,\left[\begin{array}{l}\mathrm{FROM}\,([\textsc{harry}])\\ \mathrm{TO}\,([\textsc{sam}])\end{array}\right])] \\ \mathrm{AFF}\,([\textsc{harry}],[\textsc{sam}]) \end{array}\right]$$

As I mentioned above, it makes sense to use Jackendoff's Conceptual Structures in this work for two reasons:

1. the primitives of the theory capture important generalizations about action descriptions and their relationships to one another;
2. a logical form expressed in terms of CS highlights where information may be missing from an utterance and needs to be provided by inference. For example, the input utterance

   (6.14) *Remove the fuel cap to fill the tank.*

   is mapped into the following representation:

$$\left[\begin{array}{l}\text{CAUSE}([\textsc{you}],\ [\text{INCH}([\text{BE}([\textsc{fuel-cap}],\ k)])]) \\ \text{AFF}([\textsc{you}],[\textsc{fuel-cap}]) \\ \text{FOR}(\beta)\end{array}\right]_{\alpha}$$

$$\left[\ \text{AT}-\text{END}-\text{OF}([\text{FROM}([\text{AT}([\textsc{fuel-cap}])])])\ \right]_{k}$$

$$\left[\begin{array}{l}\text{CAUSE}([\textsc{you}],\ \left[\begin{array}{l}\text{INCH}([\text{BE}([\ ]_j,\ [\text{IN}([\textsc{tank}])])])\\ \text{FILL}\end{array}\right] \\ \text{AFF}([\textsc{you}],[\textsc{tank}])\end{array}\right]_{\beta}$$

This can be glossed as the agent causes the fuel cap to be away from where it is now, for the purpose of the agent causing something to be in the tank. Notice the purpose clause in the example: the function FOR is used in [Jackendoff, 1990] to encode the subordinating function for purpose clauses. Another CS primitive that I haven't mentioned so far is INCH, which is used to encode inchoative readings of verb phrases. Jackendoff notes that apparently INCH introduces a redundancy, as (6.15a) and (6.15b) appear to encode the same event.

(6.15a) [GO ([X], [$_{\text{Path}}$ TO ([$_{\text{Place}}$ Y])])]

(6.15b) [INCH([BE ([X], [$_{\text{Place}}$ Y])])]

I refer the reader to [Jackendoff, 1990, p.93-95] as regards the reasons why this is not the case.

The conceptual atom FILL, introduced by [White, 1992], serves as a pointer to semantic information not captured by the decomposition.[10] Finally, AT-END-OF is another Place function introduced in [Jackendoff, 1990].

This representation makes evident features that are not explicit in the given sentence, such as where the fuel cap has to be removed from. The representation finesses this as AT([FUEL-CAP]), namely, where the fuel cap is now. Similarly, there is no explicit mention of the liquid to be put into the tank, so that its place in the representation is left unspecified. (If the utterance were *to fill the tank with fuel*, then *fuel* would occupy that position.)

6.1.3 Integrating Hybrid Systems and Conceptual Structures

Both the formalisms I have discussed so far respond to some of the desiderata I listed earlier: in particular, a hybrid system provides the flexibility required in dealing with action descriptions that don't exactly match the stored knowledge; a semantic representation such as Jackendoff's is linguistically motivated and manages to capture generalizations, such as that *carry* is a *move* action augmented with a specific physical means of moving the object.

[10] Such atoms will be the fillers for the role `lex-item` discussed in Sec. 7.2.1.

Given that the two formalisms both show promise, the next natural step is to integrate them. Both will benefit from this integration.

Defining terms in the T-Box by means of linguistically sound primitives will transform the T-Box into a real lexicon, or at least, into a Knowledge Base onto which the logical form produced by the parser can be easily mapped to — see [White, 1992] on the reasons why CS is particularly suitable as the logical form produced by a parser based on the Combinatory Categorial Grammar formalism [Steedman, 1990; Steedman, 1991].

On the other hand, a KL-ONE style representation will make it possible to use Conceptual Structures in a computational framework, by endowing it with a hierarchical organization and with the possibility of extending the lexicon. A flavor of hierarchical organization is present in [Jackendoff, 1990] itself. Consider (6.3), repeated here for convenience:

(6.16a) [$_{\text{Thing}}$ KITCHEN]

(6.16b) [$_{\text{Thing}}$ ROOM]

ROOM and KITCHEN are atoms in Conceptual Structures, and so are many other entities, such as LIQUID and WINE. Jackendoff postulates a type Thing which is a collection of all atoms, with no further structure imposed. Although he mentions that e.g. there are rules to determine that WINE satisfies the feature LIQUID, he never goes into the algorithm for performing such computation. This can of course be done by means of a taxonomy rooted in the concept *Thing*. *Liquid* will be defined as a subconcept of *Thing*, and *wine* as a subconcept of *liquid*, e.g. by adding to *liquid* the restrictions *made from grapes* and *fermented*.

Jackendoff defines all these concepts as atoms because they cannot be defined at the level of Conceptual Structures: this is readily captured in terminological systems by defining them as primitive concepts, namely, concepts for which necessary but not sufficient conditions can be stipulated – apart maybe from *wine*, as *made from grapes* can be considered as a defining condition.

In Figs. 6.1, 6.2 and 6.3, I present part of the T-Box that I have implemented in CLASSIC [Brachman *et al.*, 1991] and whose complete CLASSIC version the reader can find in Appendix C.

The graphical conventions are as follows (adapted from [Woods and Schmolze, 1991, p.9]. Concepts are represented by ellipses and roles are represented by small circles containing inscribed squares. Attached to roles are value restrictions and other role "facets" (e.g., names and number restrictions), and attached to concepts are roles and structural conditions, i.e. SAME-AS constraints. Value restrictions on roles are indicated by directed arrows pointing from the role to a concept that constrains possible fillers. Structural conditions are indicated by diamond shaped lozenges enclosing an equal sign. Furthermore, in my figures thick lines correspond to subsumption relations, and thick dashed lines to the transitive closure of the subsumption relation; thin dashed lines connect the relevant roles to the SAME-AS map; shaded ellipses indicate individuals.

There are six subtaxonomies of the topmost concept *classic-thing* which is not shown in the figure. They are rooted respectively in *entity* — equivalent to Jackendoff's Thing —

place, path, event, property and *state*. Fig. 6.1 shows parts of the *entity*, *place* and *path* hierarchies, while Figs. 6.2 and 6.3 focus on the action hierarchy, in particular the former on the *go* subhierarchy, the latter on the representation of *cause* and *act-on* — *move-sth-swh* is an acronym for *move something somewhere*.

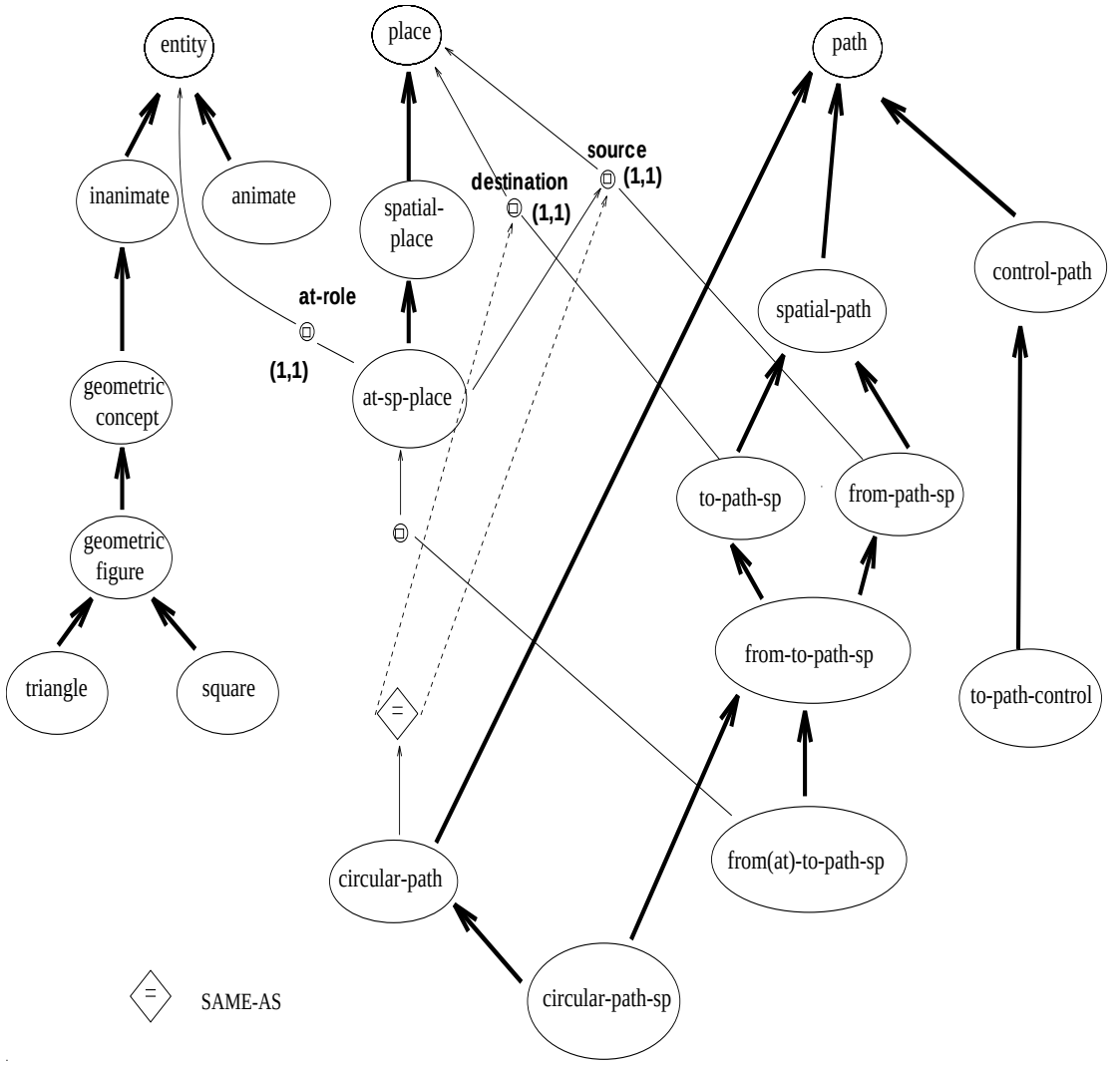


Figure 6.1: The *path* hierarchy

Entity. The taxonomy rooted in *entity* is very straightforward; here only the subconcepts *animate* and a small part of the subhierarchy rooted in *inanimate* are shown.

Place. The concepts belonging to this hierarchy correspond to conceptual functions of the form F: Thing → Place. Some such functions are AT, IN, ON. In Fig. 6.1, I show only the concept *spatial-place*, and its subordinate *at-sp-place*, corresponding to the

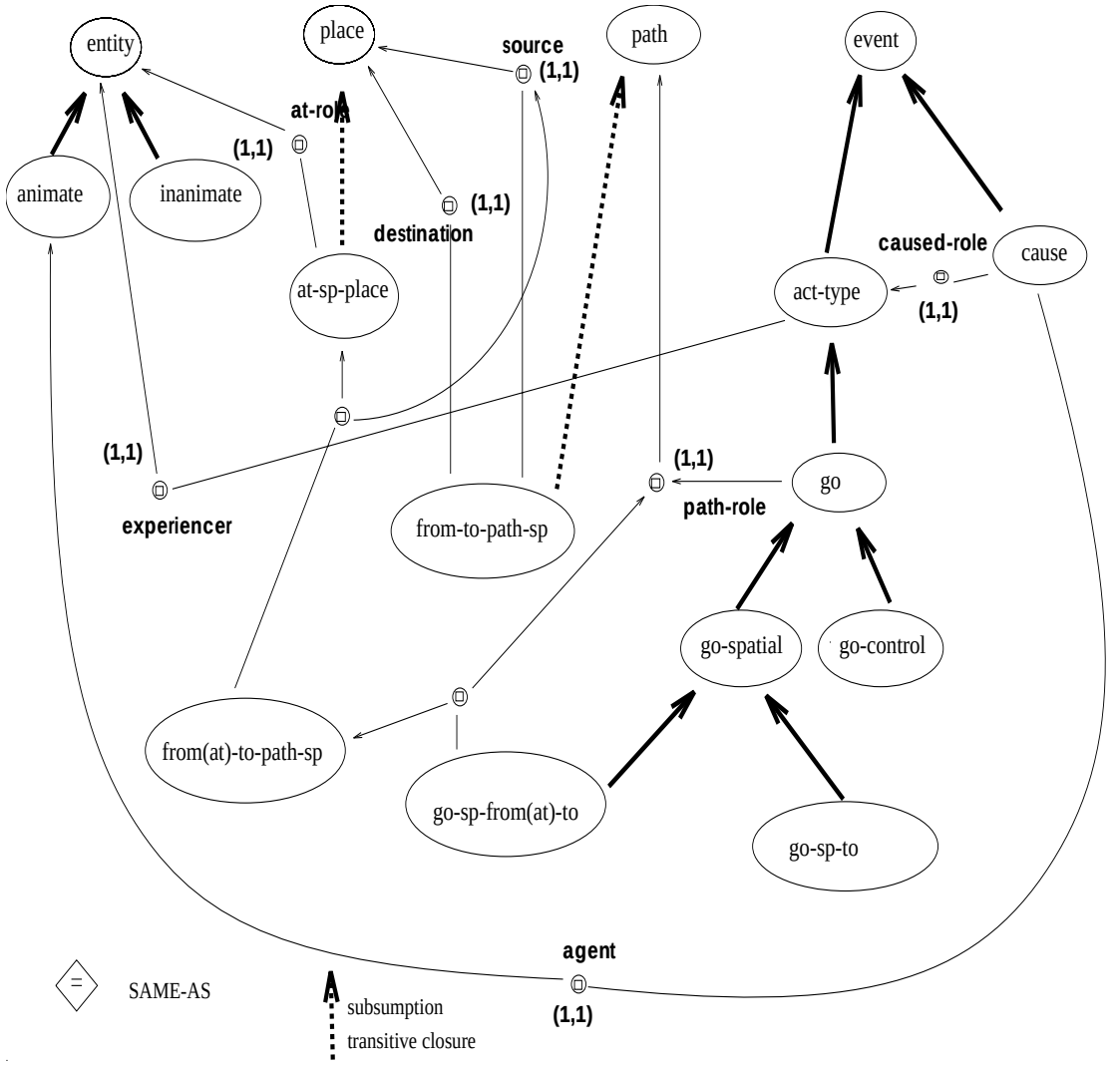


Figure 6.2: The *go* hierarchy

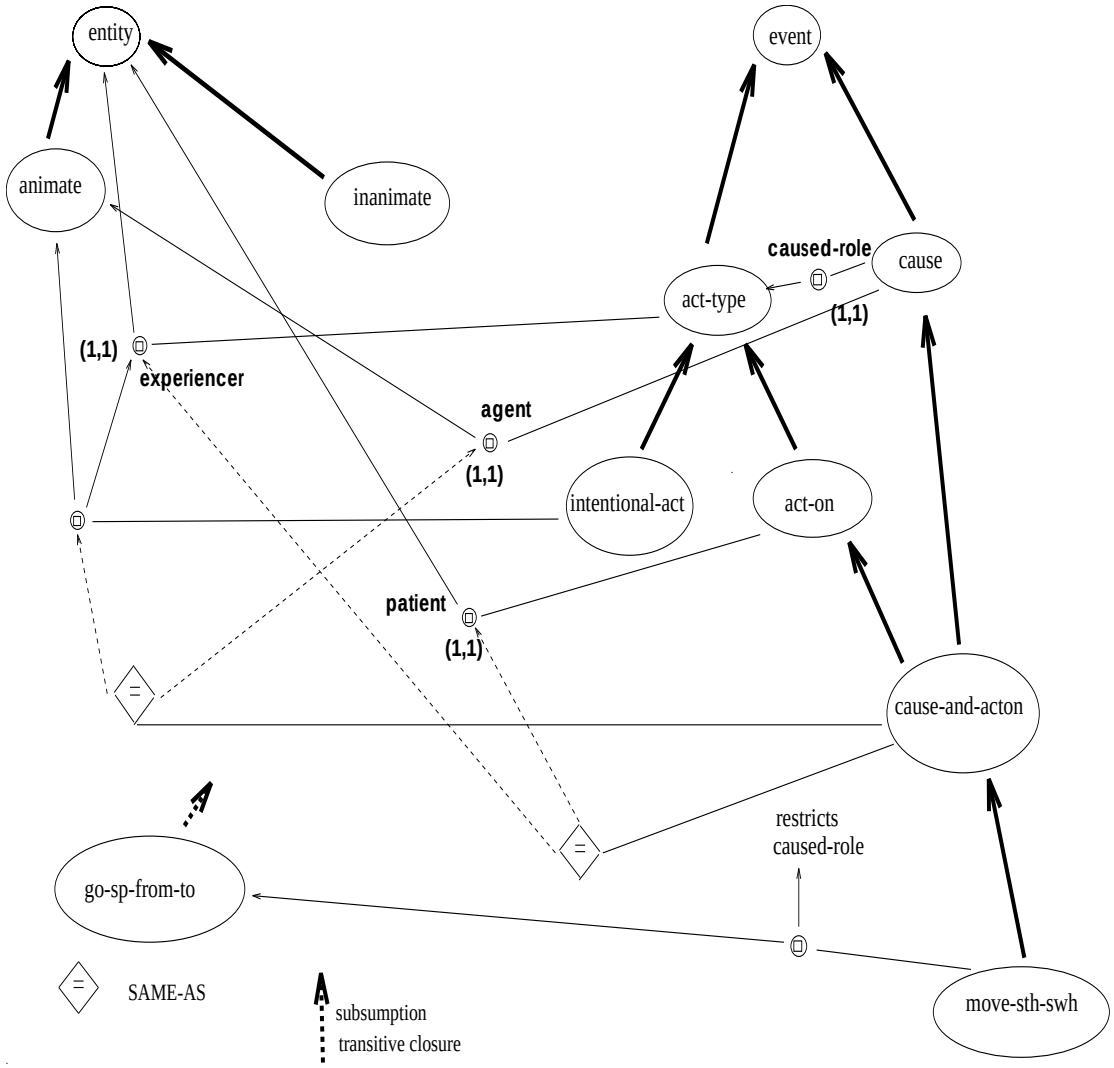


Figure 6.3: The *action* hierarchy

AT conceptual function. *at-sp-place* has a single role `at-role` with exactly one filler, of type *Entity*.

Path. The concepts belonging to this hierarchy represent functions yielding *Path*s. There are different kinds of paths, corresponding to the different semantic fields: in the figure only two of them, *spatial-path* and *control-path*, are represented. In Appendix C the reader can see that also *comp-path*, corresponding to the semantic field *composition*, and *ident-path*, corresponding to the semantic field *identificational*, have been defined. The same distinctions apply to *Place*s. In CLASSIC the semantic field is represented by defining a role `semfield-role` — not shown in the figures — whose value restriction is the concept *semfield* defined by enumeration:

```
(cl-define-concept 'semfield
     '(one-of spatial control ident contact composition))
```

Fig. 6.1 shows the subhierarchy rooted in the *spatial-path* concept. Note in particular:

1. *from-to-path-sp* is defined by means of multiple inheritance, being a daughter of both *to-path-sp* and *from-path-sp*; so is *circular-path-sp*, daughter of *circular-path* and *from-to-path-sp*.
2. The *from-to-path-sp* concept has two roles, `source` and `destination`, each of which has a filler *place*. *from-to-path-sp* corresponds to the complex Conceptual Structure:

$$\begin{bmatrix} \text{FROM}([\text{Place}]) \\ \text{TO}([\text{Place}]) \end{bmatrix}$$

   The concept *from(at)-to-path-sp* restricts the role *source* inherited from *from-to-path-sp* to be filled by *at-place*. It therefore corresponds to

$$\begin{bmatrix} \text{FROM}([\text{AT}([\text{Thing}])]) \\ \text{TO}([\text{Place}]) \end{bmatrix}$$

3. The concept *circular-path* imposes on its two roles *destination* and *source* the structural constraint that their fillers be equal — expressed by the construct **SAME-AS** in CLASSIC.

Event. Subtypes of *event* are *cause* and *act-type*, as shown in both Figs. 6.2 and 6.3.

The reader may wonder why *cause* is not defined as an *act-type*: this is to maintain the distinction between the thematic and action tiers that Jackendoff argues for and that I discussed in Sec. 6.1.2.

cause has two roles, the `agent`, restricted to be *animate*, and the `caused-role`, restricted to be an act-type. This introduces three further restrictions with respect to Jackendoff's definition of CAUSE in (6.10): (6.10) allows the `agent`[11] to be a

[11] In Jackendoff's terms, `agent` would be "source" or "theme", and the `caused-role` "theme" or "goal".

Thing, therefore also an inanimate entity, or an Event; the `caused-role` an Event, here instead restricted to be an act-type. I don't think these restrictions are too severe, as they reflect the characteristics of my application domain, and they could easily be removed.

Fig. 6.2 shows part of the hierarchy rooted in *go*, which corresponds to the conceptual function GO: Thing × Path → Event. The concept *go* has as roles `experiencer`, inherited from *act-type*, and `path-role`. Actually there is a third role, `semfield-role`, which is not shown in the figure. The requirement is that the *path* filler on the role `path-role` on the various more specialized *go*'s — *go-spatial*, *go-control* etc — has the same semantic field as *go*. The concept *go-sp-from(at)-to* restricts the role `path-role` of *go* to be a *from(at)-to-path-sp*.

Fig. 6.3 shows the part of the hierarchy more related to actions in the intuitive sense of the term, namely, to *intentional act*s that may affect other entities, as in *act-on* — notice that *go* in Fig. 6.2, *intentional-act* and *act-on* are siblings.[12] The two *thematic* and *action* tiers discussed on Pag. 103 are kept distinct, and included in the definition of actions by defining *cause-and-acton* as daughter of both *cause* and *act-on*. Notice the following two requirements on *cause-and-acton*:

1. The filler of the `experiencer` role inherited from *act-on*, and of the `agent` role inherited from *cause* must be the same. In CLASSIC, this is expressed by means of the following **SAME-AS** restriction:

   (6.17) (**SAME-AS** `(experiencer)` `(agent)`)

   It should be read as: the filler of the role `experiencer` inherited from *act-on* is the **SAME-AS** the filler of the role `agent` inherited from *cause*. As mentioned in Sec. 6.1.1.1, **SAME-AS** is a very restricted form of *structural condition*, as it requires all the roles belonging to the two paths to be attributes, and the relation between the two fillers at the end of the paths to be equality.

2. The filler of the `patient` role inherited from *act-on*, and the filler of the `experiencer` role of the *act-type* which is in turn the filler of `caused-role` inherited from *cause* must be the same. The relevant **SAME-AS** restriction is:

   (6.18) (**SAME-AS** `(patient)` `(caused-role experiencer)`)

   Again, this definition of *cause-and-acton* is not as general as it should be. For example, the verb *give*, as presented in (6.13), does not respect this restriction, and therefore could not be a descendant of *cause-and-acton*: in fact, the filler of `caused-role` is a *go-poss*, whose `experiencer` is the book; however, the filler of `patient` is *Sam*. This definition of *cause-and-acton* is in any case sufficient for my purposes, and could be easily modified to accommodate other verbs.

Finally, the concept *move-sth-swh* — for *move something somewhere* — is defined as a subconcept of *cause-and-acton* by imposing the restriction that the filler of the

[12] Of course the definition of *intentional-act* as an *act-type* whose `experiencer` is *animate* is an oversimplification, especially in light of the discussion in Sec. 4.2.3.2. However, it wouldn't be difficult to finesse the representation and to give a more accurate account of volitionality, along the lines of [Jackendoff, 1990, p. 128-129].

`caused-role` be *go-sp-from-to*. Notice that this definition of *move-sth-swh* exactly captures its Conceptual Structure definition shown in the header of the action recipe in Fig. 6.4, which will be discussed in Sec. 6.2:

$$\left[\begin{array}{l} [\text{CAUSE}([\text{AGENT}]_i, [\text{GO}_{\text{Sp}}(j,k)])] \\ \quad \left[\begin{array}{l} \text{FROM}(m) \\ \text{TO}(l) \end{array}\right]_k \end{array}\right]$$

To conclude, I would like to draw the reader's attention to the fact that the hierarchies in Figs. 6.1, 6.2 and 6.3 only show the subsumption links that are explicitly expressed in the initial definition of the TBox. When such definitions are processed, additional subsumption links are computed by the classifier: for example, *cause-and-acton* is recognized as a descendant of *intentional-act*. In fact, *cause-and-acton* inherits the restriction (**ALL** `agent animate`) from *cause*; given the **SAME-AS** requirement in (6.17), this restriction is propagated to the role `experiencer` inherited from *act-type* through *act-on*: however, an *act-type* with an animate agent is an *intentional-act*, therefore *act-on* is recognized as a descendant of *intentional-act* as well.

6.2 The action library

The action library contains simple commonsense plans about actions. Following [Pollack, 1986; Balkanski, 1993] I will call such plans *recipes*.

In the abstract, the syntax of recipes could be described as follows in a BNF form:

RECIPE BNF

RECIPE	→	BASIC-RECIPE | NON-BASIC-RECIPE
BASIC-RECIPE	→	BASIC-HEADER QUALIFIER* EFFECT+
NON-BASIC-RECIPE	→	HEADER BODY QUALIFIER* EFFECT+
BASIC-HEADER	→	basic-act-type
HEADER	→	act-type
BODY	→	act-type+ ANNOTATION*
ANNOTATION	→	act-type$_1$ enab act-type$_2$ |
		act-type$_1$ TEMP-REL act-type$_2$
QUALIFIER	→	state
EFFECT	→	state
TEMP-REL	→	before | meets | overlaps ...

One instance of a recipe, which illustrates a method for *moving-sth-swh*, is shown in Fig. 6.4.

The distinction between BASIC-RECIPE and NON-BASIC-RECIPE is due to the fact that *basic-act-types* are considered as primitives at this level of representation, and they don't have hierarchical decomposition — I take here advantage of the *AnimNL* system, that will provide the necessary decomposition in lower level actions [Webber *et al.*, 1992; Levison, 1993].

It is a notorious difficult problem to define what a *basic act-type* is, as I already mentioned in Sec. 5.1.3. From [Pollack, 1986, p.59]

> Basic actions are, essentially, actions that can be performed at will; typical examples in the literature are an agent's raising his arm, or turning his head. But there is a second important condition on basic actions: they are not performed by doing another action. ... Intuitively, when we model some domain of action and consider executability in that domain, we do not reason "all the way" to the level of bodily actions. Instead we decide, perhaps arbitrarily, on some set of actions that we assume executable at will, and do not consider how actions in that set are actually performed. So, for instance, within the domain of computer mail, we might assume that all actions that consist in typing a command are executable at will.

For the purpose of this thesis, I will assume that CS Event functions that don't have another event as argument are basic: this implies that all the act-types which are descendants of GO are basic. Also other action types, that, as Jackendoff says, just serve as links to other systems of representation better suited to determine their properties, are basic — cf. PHYSICAL-WASH in Fig. 7.10.

Notice that both the *act-type* and the *state* appearing in the definition are the corresponding T-Box concepts — a small portion of the *state* hierarchy is shown in Fig. 6.6. However, in Fig. 6.4 the act-types and the states are expressed in the original CS formalism, not to clutter the figure too much, especially given the high number of **SAME-AS** restrictions necessary to express coreference. Also notice that in certain recipes, as in Fig. 7.10, the operator ACHIEVE is used; ACHIEVE is defined as a function from states to actions [Pollack, 1986]. The CLASSIC definition for ACHIEVE can be found in Appendix C.

As we can see from the BNF, individual action entries have a *header*, *body*, *qualifiers* and *effects*. The terminology, especially *header* and *body*, is reminiscent of STRIPS [Fikes and Nilsson, 1971]; however the relations between these components are expressed in terms of *enablement* and *generation*—for example the body *generates* its header.

The representation does not employ preconditions, because it is very difficult to draw the line between what is a precondition and what is part of the body of the action. One could say that the body of a *move-sth-swh* simply consists of a *transfer* of an object from one place to another; and that a precondition for a *move-sth-swh* is having control over that object. However, consider a heavy object: 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 of the action.

The choice of not having preconditions has been motivated elsewhere in AnimNL with more extensive arguments — [Geib, 1992]. Summarizing, Geib's main argument is that

Header

$[\mathrm{CAUSE}([\textsc{agent}]_i, [\mathrm{GO}_{\mathrm{Sp}}(j,k)])]$

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

Body

- $[\mathrm{GO}_{\mathrm{Sp}}([i, [\mathrm{TO}(m)])]_{\gamma_1}$

- $[\mathrm{CAUSE}(i, [\mathrm{GO}_{\mathrm{Ctrl}}(j, [\mathrm{TO}([\mathrm{AT}(i)])])])]_{\gamma_2}$

- $\left[\begin{array}{l}\mathrm{GO}_{\mathrm{Sp}}(i,k)\\ [\mathrm{WITH}(j)]\end{array}\right]_{\gamma_3}$

Annotations

- γ_1 *enables* γ_2
- γ_2 *enables* γ_3

Qualifiers

- $[\mathrm{BE}_{\mathrm{Sp}}(j, m)]$

Effects

- $[\mathrm{BE}_{\mathrm{Sp}}(j, l)]$

Figure 6.4: A *Move Something Somewhere* action

> Finally ... an agent might be using an action to accomplish more than one intention. For example, suppose an agent has three goals: clearing a block, having the robot's hand at table level, and having the robot's hand empty. While a successful unstack operation will accomplish all three of these tasks, it is not necessary that each of the unstack subactions be successful in order to achieve all of these goals: the block might slip out of the robot's hand and break. However, since this is unrelated to the system's goals, preconditions that would prevent it would not need to be enforced before taking the action.
>
> In summary, the preconditions of an action depend crucially on the environment in which the action is undertaken and the goals it is invoked to achieve.

Therefore, the view that is embodied in the action recipes is to express what is traditionally expressed by means of preconditions by means of actions, which may be substeps in executing another action — namely, they may belong to a sequence that generates the header. As in previous chapters, the term *sequence* should not be interpreted as meaning that there is a total temporal order holding between the actions belonging to the sequence: the order may be partial, and actions in a body may have other relations holding between them, such as enablement in Fig. 6.4. The *annotations* on the body specify the relations between the subactions.

From the planning tradition I retain the notions of *qualifiers* and *effects*. Qualifiers, alternatively called *applicability conditions* [Schoppers, 1988], *constraints* [Litman and Allen, 1990], or *usewhen conditions* [Tate, 1987], express conditions that must hold for an action to be relevant, and are not meant to be achieved. For example, *unplug* x is relevant only if x is plugged in. In Fig. 6.4, there is one qualifier, that requires j to be at m.[13]

Effects are what must be true after an action has been executed.

The CS representation is also useful for computing qualifiers and effects in a systematic way: some of them can be precompiled from the representation itself. For example, for every action including a component δ of j moving from where it is to l in its header, i.e.

$$\left[\mathrm{GO}_{\mathrm{Sp}}\left(j, \begin{bmatrix} \mathrm{FROM}(m) \\ \mathrm{TO}(l) \end{bmatrix}\right)\right]_{\delta}$$

that after δ, j must be at l, therefore we can include this in the effects of the action. Given the further restriction that j cannot be in two different places at once,[14] we may infer that j cannot be at l now, and thus precompute the qualifier that j should be at m — where $l \neq m$.

A final disclaimer. First of all, notice that clearly the recipe for *move-sth-swh* in Fig. 6.4 is not the only one possible: for example an object could be moved by means of an electronic device, or by asking someone else to perform the action, etc. This would simply imply

[13] Notice that the qualifiers in a sense apply to the subactions, in that the one in the example makes γ_1 relevant.

[14] Note that there are semantic flieds, such as *Poss*, to which this restriction does not apply.

adding another recipe to the Action Library. Given the hierarchical organization for the Action Library, described in Sec. 6.2.2, and the way the algorithm retrieves recipes and matches them to the input description, this is not a problem.

A more important concern is that the recipe for *move-sth-swh* is not complete. More specifically, neither the qualifier nor the effect list is exhaustive: they both merely list some necessary conditions. The problem of listing all necessary and sufficient conditions plagues AI systems: for example, well-known are the *qualification* and the *frame* problems. The first arises with regard to universally quantified sentences, such as "All birds fly". [Genesereth and Nilsson, 1987, p.117] notes that we might express this as
$\forall$`x Bird(x)` $\Rightarrow$ `Flies(x)`. This sentence might be useful for certain limited purposes, but we would soon be confronted by the fact that ostriches, which are birds, do not fly. The axiom could be changed as follows:

$$\forall \texttt{x Bird(x)} \wedge \neg \texttt{Ostrich(x)} \Rightarrow \texttt{Flies(x)}$$

Even this sentence does not accurately capture the *real* world, however, because several other kinds of birds do not fly: baby birds, dead birds, wingless birds, and so on. The list of such *qualifications* is very long if not endless: most universally quantified sentences would have to include an infinite number of qualifications if they were to be interpreted as accurate statements about the world. Work on default reasoning addresses this problem.

On the other hand, the *frame problem* is the problem of characterizing the aspects of a state that are not changed by an action, and it arises for example in the context of defining planning operators [Genesereth and Nilsson, 1987, p.271-2]. These operator descriptions are not complete, as they describe the facts that become true as a result of executing instances of each operator, and indirectly they describe the facts that become false. However, they do not specify anything about the facts that were true beforehand and that remain true afterward or about the facts that were false beforehand and that remain false afterward. The frame problem is to write *frame axioms* that indicate the properties that remain unchanged after each action. Typically, the number of frame axioms is proportional to the product of the number of relations and the number of operators in our conceptualization: what makes the frame problem a problem is that, in worlds of realistic complexity, there are many more actions and relations, and so many more axioms are necessary.

The connection between the fact I don't list all the qualifiers / effects and the qualification / frame problem is clear: how is it possible to list all relevant conditions? Consider for example any action of physical manipulation. At a minimum it requires that the agent can physically perform it: which may span an infinite number of conditions, for example that the agent is not paralyzed, or that he is strong enough to lift the object to be manipulated. Some of these conditions appear to be qualifiers — if the object is too heavy for the agent to lift, and the agent knows it, there is no point in trying to perform the action; on the other hand, if the agent is tied up, getting untied may accomplish an executability condition, namely, that the agent is free to perform the action. This problem is often bypassed in action representation formalisms — e.g. in [Goldman, 1970; Pollack, 1986; Balkanski, 1993] — by requiring that the agent be in *standard conditions* with respect to the action to be performed.

I am mentioning all this in order to justify my representation: the qualifiers and effects I list are merely a part of the required list of conditions, and I make no claim about the exhaustiveness of the condition lists I provide.

6.2.1 Relations between actions

Generation. I adopt the definition of *generation* provided by Balkanski in her dissertation [1993], and modify it to include the case of a *sequence* $\mathcal{A}$ of actions generating another action — cf. Sec. 5.1.2:

(6.19) $\mathcal{A}$ **generates** β if and only if

1. $\mathcal{A}$ is a sequence of actions, and β is an action performed by the same agent at the same time;[15]
2. there is a set of conditions, $\mathcal{C}$, such that
   (a) these conditions hold during the performance time of $\mathcal{A}$ and
   (b) $\mathcal{A}$ [16] conditionally generates β under $\mathcal{C}$.

Enablement. Again, the definition of enablement is very similar to Balkanski's; with respect to (5.8) I modify the first clause of the definition so as to include cases in which α only has to start, but not necessarily to finish, before β; and I also modify clause 2(b) to reflect the fact that I don't have preconditions in my recipes.

(6.20) α **enables** β if and only if

1. the *start* time of α is prior to the *start* time of β,
2. there is a third action γ, such that
   (a) either γ conditionally generates β under $\mathcal{C}$, and one of the conditions in $\mathcal{C}$, $\mathcal{C}_i$, holds as a result of the performance of α; or
   (b) there is a recipe for β in which γ appears as a substep, and α conditionally generates γ.

Consider once more the question asked at the end of Sec. 5.1.4.3: what relation is holding between each single α_i and β, in the presence of a sequence of actions $\mathcal{A}$ $< \alpha_1, \alpha_2, ..., \alpha_n >$ that generate another action β. I had answered that question by deciding to reserve the term *enablement* only to the relation holding between pairs $< \alpha_i, \alpha_k >$ belonging to $\mathcal{A}$; the relation between each α_i and β will be termed instead *indirect generation*, as mentioned at the end of Sec. 5.1.2. In the context of recipes then, this is equivalent to allow enablement relations within the body of an action, and to implicitly assume that the relation between body and header is in fact generation, even if the relative conditions are not explicitly spelled out — after all, certain conditions, such as the *standard* ones, are implicitly always present.

[15] This should be interpreted as meaning that the temporal extents of all the actions belonging to $\mathcal{A}$ and of β coincide.

[16] Meaning, the sequence composed of the act-types α_1, α_2, ..., α_n belonging to $\mathcal{A}$. Once again, I remind the reader that *sequence* does not mean *total* temporal order.

Temporal Relations. I assume the temporal relations between time intervals defined in [Allen, 1984]: there are thirteen of them, including

- BEFORE(t_1, t_2) — time interval t_1 is before time interval t_2, and they do not overlap in any way;
- MEETS(t_1, t_2) — t_1 is before t_2, and t_1 ends where t_2 starts.

Allen's work has had an enormous impact on research on temporal issues in Artificial Intelligence and Computational Linguistics; for example the predicates HOLDS and OCCURS used in [Pollack, 1986] and [Balkanski, 1993] derive from Allen's formalization. Given that temporal relations are not the focus of my work, I won't have anything else to contribute about them — in fact they won't play any particular role in the examples I will present in the next chapter. With a somewhat sloppy notation, I will write BEFORE(α, β), with the understanding that the actions are coerced to their corresponding time intervals.

6.2.2 Hybrid Systems and Recipes

Given that I implement action concepts in the terminological language of a hybrid system, the next step is to also implement the Action Library by means of the same language. In fact, this allows me to use the same or similar mechanisms of classification to keep an organized KB of action recipes. The idea of having plan abstraction hierarchies is not new in plan inference / planning work, as it is present in different forms in various systems, for example [Kautz, 1990; Tenenberg, 1989], that will both be described in Sec. 7.3. However, there are only few systems known to me that exploit subsumption to perform reasoning about plans: SUDO-PLANNER [Wellman, 1988], CLASP [Devanbu and Litman, 1991] and T-REX [Weida and Litman, 1992]. [17]

In SUDO-PLANNER actions — those events controllable by the planner — are defined by means of the T-Box of NIKL [Moser *et al.*, 1983]. Actions are (simplified descriptions of) medical procedures, such as (from [Wellman, 1988, p. 47])

> A **surgery** is an **action**
> with one **route** which is an **invasive-path-into-body**.

Plans are specifications for a course of action, and are composed of three different kinds of constraints: *action constraints*, that specify that an action of the specific type belongs to the plan; *action policy constraints*, that specify relations between actions and events; *conditional constraints*, which are action or policy constraints that have effect only under some observed conditions. Wellman's representation of plan doesn't include the notion of substep, and also effects are dealt with separately.

Plan classification extends normal classification by computing subsumption among the different kinds of constraints. Wellman's definitions of action and plan are quite distant

[17] A project addressing similar issues is also under way at the German Institute for Artificial Intelligence (DFKI), in Saarbrücken: however, as far as I know, no publications are available.

not only from linguistic considerations, but from the more usual STRIPS definition of operator: therefore, I will assume this short description suffices, and I will turn to CLASP, which deals with issues closer to my interests.

CLASP extends the notion of subsumption and classification from frame-based languages to plans. The CLASP representation language provides description-forming operators that specify temporal and conditional relationships between actions represented in CLASSIC. Subsumption in CLASP builds on term subsumption in CLASSIC and illustrates how term subsumption can be exploited to serve special needs. In particular, the CLASP algorithms for plan subsumption integrate work in automata theory with work in term subsumption.

Plans are built out of *action*s, whose definition reproduces the classic STRIPS planning operator definition, as it includes the roles `precondition, add-list, delete-list, goal`, all restricted to be *state* — [Devanbu and Litman, 1991, p.132]:

```
(DEFINE-CONCEPT
   Action
   (PRIMITIVE
       (AND Classic-Thing
            (AT-LEAST 1 ACTOR)
            (ALL ACTOR Agent)
            (ALL PRECONDITION State)
            (ALL ADD-LIST State)
            (ALL DELETE-LIST State)
            (ALL GOAL State))))
```

As far as plans are concerned,

> plan concept expressions are compositionally defined from action and state concepts using the plan description forming operators SEQUENCE, LOOP, REPEAT, TEST, OR, and SUBPLAN. [Devanbu and Litman, 1991, p.132]

The ordinary subsumption algorithm provided by CLASSIC and called *o-subsumption*, where *o* stays for *object*, is extended into *p-subsumption* to deal with *plan* and *scenario classification*, where *scenario*s are individual plans. The details of *p-subsumption* can be found in [Devanbu and Litman, 1991, p.134-135].

Finally, T-REX [Weida and Litman, 1992] is similar in spirit to CLASP, but is able to capture finer temporal relations, and integrates terminological reasoning with constraint network reasoning to classify plans.

CLASP (and T-REX) therefore takes the first step toward building and managing a hierarchy of plans — however, CLASP doesn't go beyond a representation of actions based on STRIPS operators. As I have shown, a much richer representation of actions is necessary, and here I will show how these richer descriptions of actions will be used to build recipes. Notice however that I haven't dealt with extending subsumption. As the reader will see, this part of my representation language leaves many questions open, due to the fact

that the CLASSIC language is fairly constrained, and that therefore many representation choices are affected by its limitations. Such limitations could be overcome by extending the CLASSIC language; however, studying these extensions was not the main focus of my thesis, and therefore it is left for future work.

Before describing the action library, I would like to spend a few words on why the CLASSIC language is expressively limited. In the literature on hybrid systems, there is a constant debate on the trade-off between expressive power of the terminological and / or assertional languages and the computational tractability of the relevant algorithms. [Woods and Schmolze, 1991, p.34-35] notes that [Brachman *et al.*, 1983a] made initial arguments, and [Brachman and Levesque, 1984] later made stronger arguments that the answers given by a KR system should be fully correct with respect to a formal semantics and that the operations should be performed in a predictable and reasonable amount of time. A common interpretation of these arguments was that these algorithms should be sound, complete and tractable.

However, Woods and Schmolze point out that these requirements can have very severe consequences for the system's inferential algorithms: the expressive power of each component of a KR system should be limited, since otherwise a system cannot be simultaneously sound, complete and tractable for classification and other inferential algorithms. To meet these requirements, the expressive power of KRYPTON's T-Box language was specifically chosen such that determining subsumption would be tractable.

Finally, Woods and Schmolze observe that, while some researchers followed the KRYPTON strategy of developing KR systems with limited expressive power and limited inferential services, others chose instead to explore systems with wide expressive power and expanded referential services. The researchers decided it was better to have greater expressive and inferential power, even though it would preclude having both completeness and tractability. As it turns out, complete and tractable inferential algorithms are nearly impossible to attain for any but the weakest of languages.[18]

As far as CLASSIC computational complexity is concerned, it turns out that *subsumption between descriptions in CLASSIC 1 is intractable in the regular semantics because of the presence of sets (one-of).* [19] CLASSIC 1 is CLASSIC version 1; CLASSIC 2 is now available, which introduces a primitive role hierarchy, which is another source of intractability. Proof of the first result can be found in [Borgida and Patel-Schneider, 1993].

6.2.2.1 Recipes in CLASSIC

The hierarchies that represent recipes are shown in Figs. 6.5 and 6.6. In particular, Fig. 6.5 shows part of the recipe hierarchy, while Fig. 6.6 shows some auxiliary concept definitions, such as *annotation-type*, and a small portion of the *state* hierarchy. No **SAME-AS** relation is shown. The corresponding CLASSIC definitions can be found in Appendix C.

[18] [Woods, 1991, Sec. 5.4, p.45-47] presents more details on complexity results: particularly interesting is Table 2, which summarizes worst case complexity results for a variety of terminological languages.

[19] Peter Patel-Schneider, p.c.

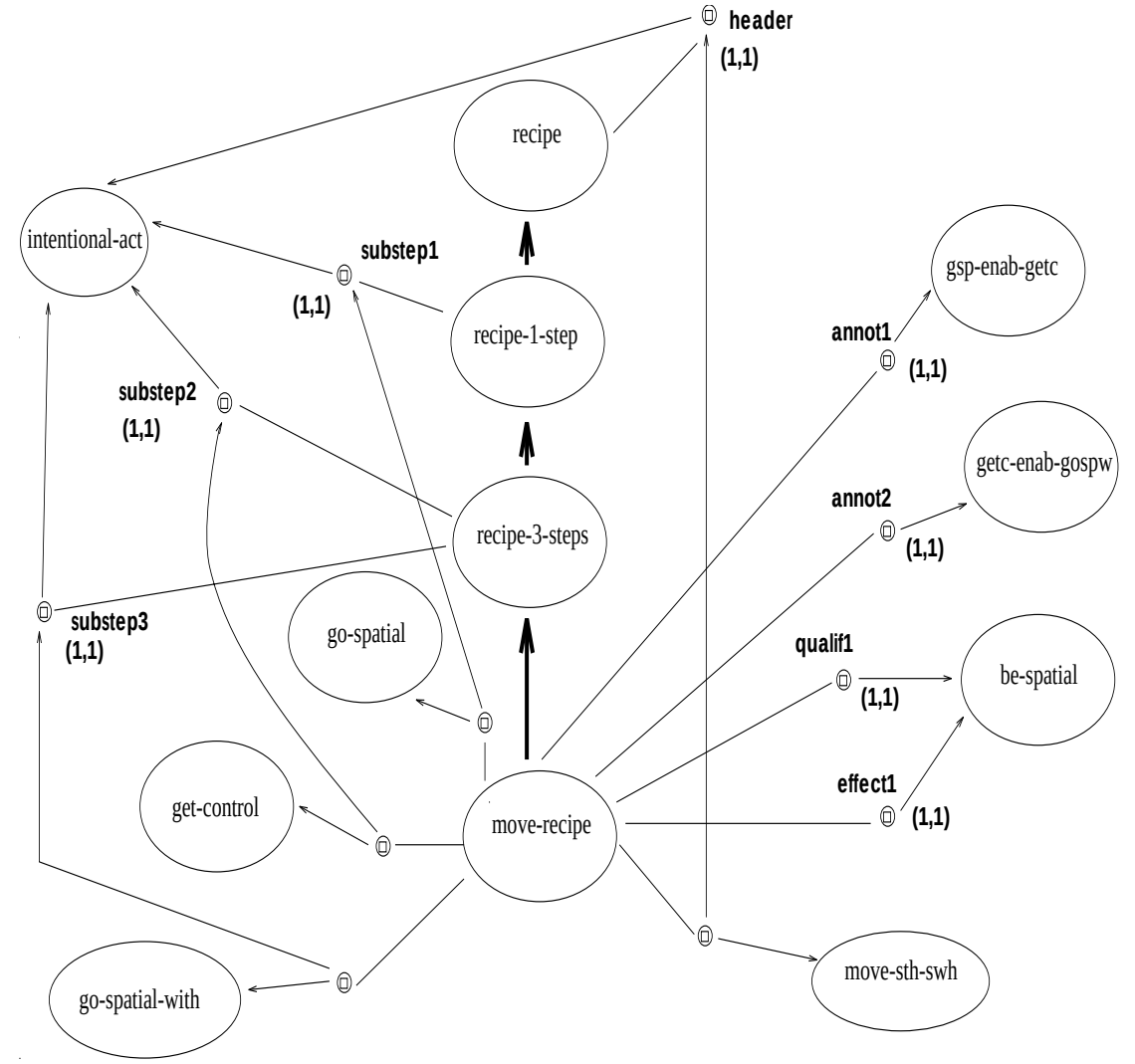


Figure 6.5: The *recipe* hierarchy

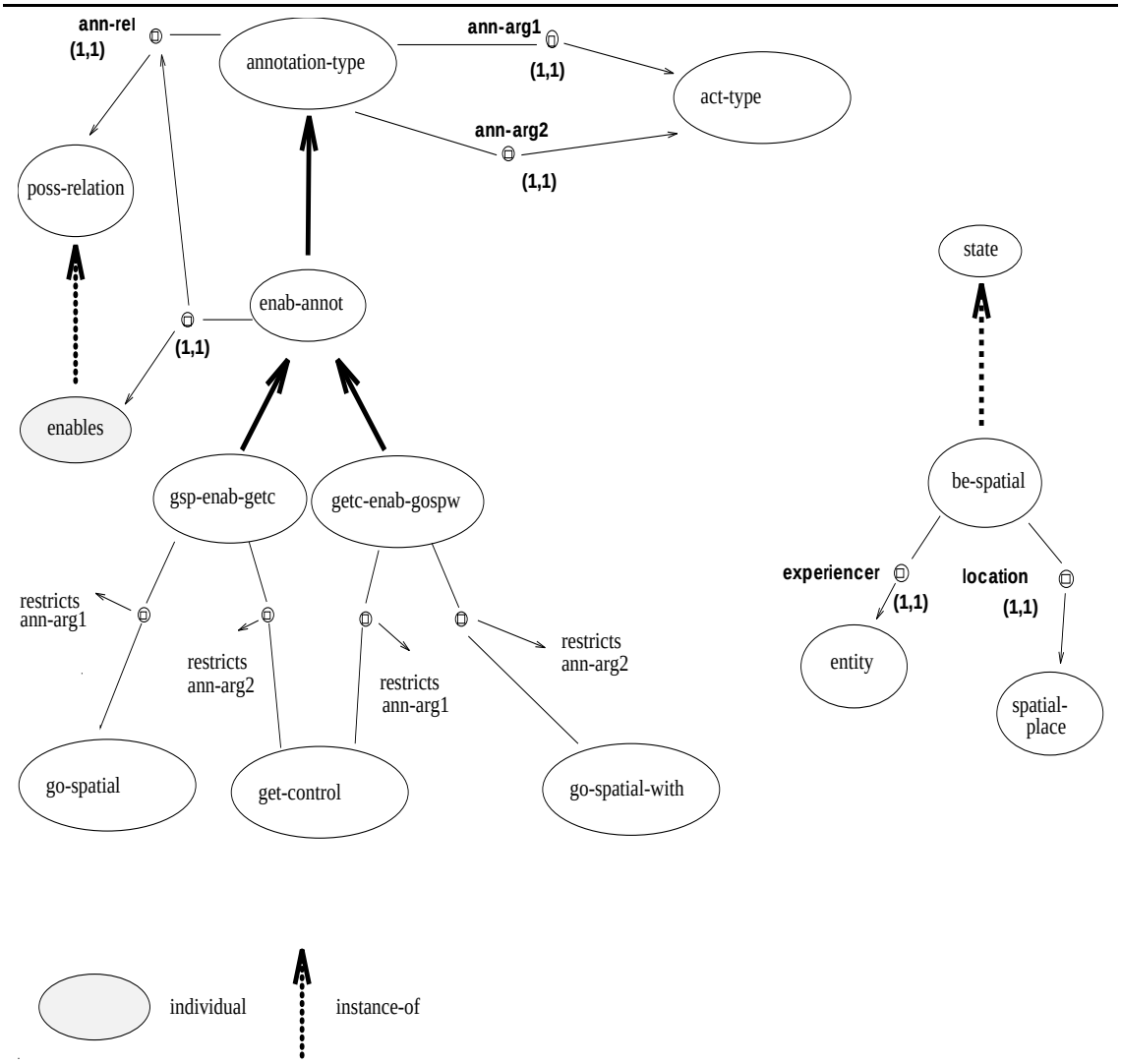


Figure 6.6: The *annotation* hierarchy

As a preliminary remark, notice that the most natural way of translating the BNF for *recipe* into CLASSIC would be by defining a topmost concept *recipe*, with roles `header, substeps, annotations, qualifiers and effects`, properly restricted. However, as I will show below, this is impossible because of the lack of role differentiation in CLASSIC. I have however defined a simple interface, so that a user can input a recipe in a way corresponding to the BNF; the input definition is then translated into the CLASSIC definition — see Pag. 124.

The issues I want to draw the reader's attention to are:

1. The *recipe* concept is simply defined as having a *header*, while the substeps appear only on the subconcepts of *recipe*, namely, *recipe-1-step*, with its descendant *create-triangles*, not shown here, and *recipe-3-steps*, with its descendant *move-recipe*. This is the first choice the reader may find puzzling. In fact, a more natural way of defining *recipe* would be to attach to it a role `substeps` with the following restrictions:
   (**ALL** `substeps intentional-act`) (**AT-LEAST** `1 substeps`)
   Unfortunately CLASSIC doesn't provide the *differentiation* operator on roles: therefore, it is impossible to express that e.g. *move-recipe* has 3 substeps, each of which has a different value restriction, *go-spatial*, *get-control* and *go-spatial-with* respectively. Clearly, this way of defining substeps, by introducing a role each, implies that if we have a recipe with n substeps, then the corresponding concept *recipe-n-steps* needs to be defined.

   Notice that in the CLASP definition of *Action* all the required roles, such as `precondition`, can be describe in a compact way because the filler is a single state, e.g.

   (**AND** `Off-Hook-State Idle-State`)

   This is clearly impossible to do with `substep`, which describes actions whose descriptions are mutually exclusive, e.g. *go-spatial* and *get-control*.

2. A concept *annotation-type* is defined with three roles. The first one is `ann-rel`, which is restricted to the concept *poss-relation* defined by enumeration as *enables, precedes, during ...* — only the individual *enables* is shown in Fig. 6.6. The other two roles on *annotation-type* are `ann-arg1` and `ann-arg2`, that specify the two actions to be related: both are restricted to be *act-type*s; the actual fillers will have to corefer with the substeps of the recipe — as usual, structural conditions are not shown in the figures, but the corresponding **SAME-AS** restrictions can be found in Appendix C. As in the case of substeps, each annotation is defined as a role on the corresponding recipe — *move-recipe* has two, expressed by means of the two roles `annot1` and `annot2`.

3. *move-recipe* has one qualifier and one effect, both expressed by means of a role, restricted to be *be-spatial* in both cases. Part of the *state* hierarchy is shown in Fig. 6.6.

A last observation: as I said, I use the action library as a repository of information on recipes, rather than to exploit the classification mechanism. Nevertheless, classification does come into play when more specific recipes are defined: for example, a *move-recipe2* concept with further qualifiers, or with a more specific *substep1*, would be correctly classified as a descendant of *move-recipe*. However, no reasoning of the kind described in CLASP is performed. For example, the *move-recipe* in Fig. 6.5 imposes a total order on its substeps through the annotations. Suppose I were to define *move-recipe2* in which the temporal order of the substeps were BEFORE(γ_1, γ_3) and BEFORE(γ_2, γ_3), without any explicit ordering given on γ_1 and γ_2. *move-recipe2* should be recognized as subsuming *move-recipe*, and in fact this is one of the subsumption inferences CLASP performs. Such extensions are left for future work.

6.2.2.1.1 Recipe interface. As I mentioned above, the CLASSIC definition of a recipe does not directly correspond to its BNF definition; therefore, I have defined a very simple interface that allows the user to input a recipe in a more natural way, without having to worry about expressing it in a precise CLASSIC format, e.g. about specifying value restrictions by means of the operator `all`.

In the interface, the input format for a recipe is as a LISP structure, as follows:

```
(defstruct recipe
  (name nil :type symbol)
  (header nil :type symbol)
  (body nil :type list)
  (same-as-restrs nil :type list)
  (annotations nil :type list)
  (qualifiers nil :type list)
  (effects nil :type list)
)
```

The user will give the following command:

```
(make-recipe :name 'move-recipe :header 'move-sth-swh
             :body '((substep1 go-spatial)
                     (substep2 get-control)
                     (substep3 go-spatial-with))
             :annotations '((annot1 gsp-enab-getc)
                            (annot2 getc-enab-gospw))
             :qualifiers '((qualif1 be-spatial))
             :effects '((effect1 be-spatial))
             :same-as-restrs '(((header agent)(substep1 agent))
                               ((header agent)(substep2 agent))
                               ((header agent)(substep3 agent))
                               ((header patient)(substep2 patient))
                               ((header patient)(substep3 with-role))
```

```
((header caused-role path-role destination)
 (substep3 path-role destination))
((header caused-role path-role source)
 (substep1 path-role destination))
(substep1 (annot1 ann-arg1))
(substep2 (annot1 ann-arg2))
(substep2 (annot2 ann-arg1))
(substep3 (annot2 ann-arg2))
((qualif1 experiencer)
 (header patient))
((effect1 experiencer)
 (header patient))
((qualif1 location)
 (substep1 path-role destination))
((effect1 location)
 (substep3 path-role destination))))
```

The function `parse-recipe` will first build a CLASSIC `concept-expression` corresponding to this input, for example by inserting the necessary CLASSIC operators such as `all` and `same-as`; then it will call `(cl-define-concept name conc-expr)`: in this example `name` is `move-recipe`, and the `move-recipe` found in Appendix C is added to the Action Library.

6.3 Summary

In this chapter, I have presented the action representation formalism I have devised. I have discussed its three main components, the underlying hybrid system, the semantic primitives derived from Jackendoff's Conceptual Structure, and the action library.

The TBox stores the linguistic knowledge about action descriptions, expressed by means of CS primitives; the terms defined in the TBox, which represent action types, are then used as building blocks in the recipes stored in the Action Library. Also the Action Library is expressed by means of the CLASSIC language: however, the CLASSIC language is too weak to properly represent recipes, and should be suitably extended. This is left for future work.

In the next chapter, I will describe the algorithm that uses the described Knowledge Bases and that implements the inferences schematically described in Sec. 5.2.

Chapter 7

The algorithm

In this chapter, I will describe the algorithm that takes the logical form produced by the parser, and builds an initial *plan graph*, representing the structure of the beliefs, expectations and intentions the agent adopts on the basis of the instruction alone. The algorithm is embedded in the *AnimNL* system described in Appendix D [Webber *et al.*, 1991; Webber *et al.*, 1992; Webber *et al.*, 1993], and assumes the existence of separate *AnimNL* modules for

1. parsing the input and providing a logical form expressed in terms of Conceptual Structures primitives [White, 1992];
2. managing the discourse model and solving anaphora. [1]

The *plan graph*, whose data structure I will describe in Sec. 7.1.7, should be thought as representing the structure of H's beliefs and intentions. I am fully aware that, given the master-slave assumption underlying my work, what I am really representing is S's beliefs on H's beliefs and intentions — a reminder to the reader that S stands for Speaker and H for Hearer. I am also fully aware that such representation is not sufficient to account for miscommunication, as Pollack showed in her thesis [1986]; moreover, that a fuller representation could benefit from Balkanski's findings about beliefs and intentions in PCs and in Means Clauses [Balkanski, 1993]. However, first of all, the representation I adopt is sufficient to support my claims; second, representing differences in Speaker's and Hearer's beliefs was not the focus of my thesis: I do plan to address this issue in future work though, as it also bears on the interpretation of negative imperatives, as discussed in Sec. 4.2.

Notice that the algorithm produces only the initial plan graph, namely, it tries to model what intentions the agent adopts simply based on the input instruction and the stored knowledge: the current situation is not taken into account. As an example, consider the instruction *Go into the kitchen to get me the coffee urn*. The algorithm will build the plan graph shown in Fig. 7.9; however, this plan may need to be expanded with a step *open the door* if the door to the kitchen is closed — these expansion steps are performed by other *AnimNL* modules, in particular by the box **plan expansion** in Fig. D.1.

[1] While the parser is indeed working, the other modules are either being implemented or being designed.

Fig. 7.1 gives a top level description of the algorithm. Comparing it to Allen's algorithm in Fig. 3.2, my algorithm addresses in more detail the steps of matching the new action into the *E-plan*, which corresponds to one of the `Recipe`s that are retrieved by means of the goal — notice that Allen's algorithm takes the *E-plan* for granted, while my algorithm finds the relevant `Recipe`s. What my algorithm lacks is the recursive step of matching an action with the decomposition of the substeps of the `Recipe`, if no direct match between the action and any of the substeps is found. This recursive match would be useful for dealing with a greater variety of input instructions, however it would not add anything either to computing the *match* between the input and the stored action description, or to computing expectations.

In the following, I will discuss the various steps of the algorithm one at a time, and after that, I will illustrate how it implements the inferences presented schematically in Figs. 5.4 and 5.6.

7.1 The algorithm steps

7.1.1 Input / Output

1. The Logical Form of the input sentence is produced by the parser described in [White, 1992], and expressed in terms of Conceptual Structures. The content of each clause, which I take to be an action description, is reified; therefore there is one "major" CS component for each clause expressed in the sentence, plus the necessary connectives. Ex. (5.24a), *Go into the kitchen to get me the coffee urn*, is mapped into

   (7.1)

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

   $$[\mathrm{CAUSE}(r, [\mathrm{GO}_{\mathrm{Sp}}([\textsc{urn-of-coffee}]_s, w)])]_{\beta}$$

   $$\left[\begin{array}{l} \mathrm{FROM}(t) \\ \mathrm{TO}([\mathrm{AT}([\textsc{me}])]) \end{array}\right]_{w}$$

   The FOR-function is derived from the *to*-phrase and encodes the purpose relation holding between the *go*-action α and the *get*-action β.[2] In the algorithm in fig. 7.1 I refer to these major subcomponents, α and β, as α_i.

2. In the context of understanding a complex instructional text, it is clear that the algorithm should take into account the `PlanGraph` built so far. This introduces problems of both understanding how the different sentences relate to one another, and of building the discourse structure. Both problems have already been tackled in the literature, in particular in work on theories of discourse structure [Grosz and

[2] The FOR-function is introduced by Jackendoff to represent purpose clauses, as I noted in Sec. 4.1.2.

INPUT:

1. Logical form of input sentence, composed of one action description α_i per clause, plus connectives.
2. The preexisting `PlanGraph` and the list of active nodes.

OUTPUT: Updated `PlanGraph`.

1. **(PREPROCESSING)**
   (a) Add each α_i to the A-Box.
   (b) Choose `Goal` $\in \{\alpha_i\}$ according to the surface structure.
2. **(RETRIEVAL)**
   Retrieve the list $\{\texttt{Recipe}_l\}$ indexed by `Goal`, i.e. those recipes whose header is instantiated by `Goal`.
3. FOR each $\texttt{Recipe}_l$ DO
   (a) FOR each $\alpha_i \in \{\alpha_i\}$, $\alpha_i \neq$ `Goal`, DO
   FOR each $\gamma_{l,j} \in \texttt{Body(Recipe}_l\texttt{)}$ DO
      i. **(COMPUTING COMPATIBILITY)**
      Check the compatibility of α_i and $\gamma_{l,j}$.
      ii. **(COMPUTING EXPECTATIONS)**
      IF α_i and $\gamma_{l,j}$ are compatible
      THEN compute set of expectations $\{\mathcal{E}\}$.
      iii. IF there is **no** $\gamma_{l,j}$ such that α_i and $\gamma_{l,j}$ are compatible
      THEN $\texttt{Failure}_l$.
4. IF $\forall\, l\ \texttt{Failure}_l$
   THEN Signal user *Can't process instruction.* **(SIGNAL)**
   ELSE Choose best interpretation. Update PG. **(UPDATE)**

Figure 7.1: The algorithm top level

Sidner, 1986; Grosz and Sidner, 1990; Webber, 1991] and of plan inference applied to discourse [Litman and Allen, 1990; Lambert and Carberry, 1992].

What I propose here is to keep track of *active nodes*, namely, those nodes that are open for expansion: they will minimally include the goal currently in focus and the nodes just added to the tree. I suspect building this list of active nodes would obey the same constraint Webber introduces to account for *discourse deixis* [1991, p.124]:

> Integration of the discourse meaning of the next clause only takes place at the *right frontier* of the discourse structure.

Note that I make the same assumption made in AnimNL, namely, that instructional text is processed in *steps* [Webber *et al.*, 1992, p.5]:

> Instructions are given to AnimNL in *steps* consisting of one or more utterances. ... While there are no firm guidelines as to what a single instruction step should encompass, often steps are organized around small coherent substasks (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.

For the moment, I haven't implemented any active node management schema, and the algorithm assumes that there is no preexisting `PlanGraph`. Devising the management of the active node list is left for future work.

7.1.2 Preprocessing

1. The various α_i are mapped into CLASSIC definitions, according to the knowledge encoded in the TBox, and then added to the ABox as individuals. As an example, the CS representation of α in (7.1) is mapped into the following representation (assuming that `you` and `kitchen` have already been created as individuals):

(7.2)
```
(cl-create-ind 'ink '(and in-sp-place
                          (fills in-role kitchen)))

(cl-create-ind 'tok '(and to-path-sp
                          (fills destination ink)))

(cl-create-ind 'go-kitchen '(and go-spatial
                                 (fills experiencer you)
                                 (fills path-role tok)))
```

`cl-create-ind` is the CLASSIC construct that creates an individual with the required specifications. The creation of an individual activates all the relevant inference mechanisms — cf. Sec. 6.1.1.1.

Actually, the current state of the implementation is such that the parser is detached from the algorithm, which receives directly in input expressions such as those shown in (7.2). However, it wouldn't be difficult to write a simple interface that translates the output of the parser into CLASSIC definitions, given that the format of the parser output is

```
[[you,o9],[kitchen,o10],[definite_ref,o10],[to,s10,s9],[in,o10,s10],
 [act,o9,e11],[go,o9,s9,e11],[spatial_field,e11],
 [imperative_mood,e11], [for,e12,e11],
 [urn,o11],[container_for,o12,o11],[coffee,o12],[definite_ref,o11],
 [me,o13],[act_on,o9,o11,e12],[cause,o9,e13,e12],[go,o11,s11,e13],
 [spatial_field,e13],[from,s12,s11],[to,s13,s11],[at,o13,s13]
]]
```

2. One of the α_i's is designated as `Goal`, and used as the *anchor* to retrieve recipes from the Action Library. The choice of `Goal` depends on the surface form: clearly, if the input instruction only includes a main clause, `Goal` will be the action description corresponding to the main clause; for *Do* α *to do* β, `Goal` is β; for *Do* α *by doing* β, α. Other cases, e.g. *when* clauses, still require a pragmatic analysis to understand whether one of the action descriptions can always be designated as `Goal`; it seems clear that for cases such as conjunctions, that can express various relations between actions, it may be impossible to establish this a priori.

7.1.3 Retrieval

`Goal` is used to index into the Action Library. In Sec. 5.2.1.1 I mentioned that the goal β should exactly match the header of the recipe; the algorithm actually allows the match to succeed as long as the goal is an instance of the header of the recipe. Clearly if `Goal` is an instance of a header δ, then it will be an instance of all concepts δ^{sup}, where δ^{sup} subsumes δ. Therefore any recipe whose header is one of the δ^{sup}'s will be retrieved. So far, I have adopted a simple filtering mechanism that filters the resulting set of recipes to keep only the bottom-most ones among them.

7.1.4 Computing compatibility

This is the step where the classification mechanism is mainly used. I will describe it fully in Sec. 7.2.1.

It may happen that α_i matches more than one $\gamma_{l,j}$: if α_i matches more than one, but not every $\gamma_{l,j} \in$ `Body(Recipe`$_l$`)`, then the algorithm chooses the first matched $\gamma_{l,j}$ — "first" according to the order imposed by the relations expressed in the annotations on `Recipe`$_l$. This is consistent with the pragmatic analysis of PCs, that showed that, if in *Do* α *to do* β α belongs to a sequence $\mathcal{A}$ that generates β, then α corresponds to the first $\alpha_i \in \mathcal{A}$.

If by any chance α_i matches every substep in `Recipe`$_l$, then the hypothesis is that the action description α_i is so general that it can match the whole recipe, as in an input

instruction like *Do something to do* β — therefore the resulting `PlanGraph` will simply result in the expansion of the chosen $\texttt{Recipe}_l$, without α_i being added to it, as it doesn't add any further information.

7.1.5 Computing expectations

Again, I refer the reader to the full description in Sec. 7.2.2 below.

7.1.6 Signal

The user is notified of the fact that the instruction can't be processed, because no recipe has been found to be compatible with the input instruction. Notice that `Failure` can be caused by different factors, some of which don't actually amount to real failures. Such factors include at least

1. There is no `Recipe` whose header is instantiated by `Goal` — this can be due to the fact that the Action Library is not complete.

2. `Goal` has selected at least one $\texttt{Recipe}_l$, but there is no $\gamma_{l,j}$ such that α_i and $\gamma_{l,j}$ match. This can be due to an incoherent instruction, or to the fact that the search mechanism should be extended. Consider

   (7.3) *Go into the kitchen to get me the suitcase from the car.*

   In this case, my algorithm would signal failure, as it would compute that *Go into the kitchen* contributes to *get me the suitcase from the car* under the expectation that *the car is in the kitchen*, which is untenable — see Sec. 7.2.2. However, it is certainly possible to think of scenarios where (7.3) makes sense, for example, the car keys are in the kitchen.

   This requires having more sophisticated search mechanisms, in particular recursion, and taking knowledge about the current situation into account. For example, in Ex. 7.3, the recursive step could find that α — *Go into the kitchen* — matches a substep of the decomposition of γ_2 — *get control over the object to be moved* — from the body of the *move*-action in Fig. 6.4.

3. The input instruction is incompatible with the stored knowledge. This can be a real failure, such as in the case of *Cut the square in half along the perpendicular axis to create two triangles*, or possibly a failure of a default assumption, such as *Turn the screw clockwise to loosen* — it is possible that the screw is left-handed, so that the instruction is in fact correct.

Some of these "failures" should trigger corrective behavior on the part of the instructor, others reactive and / or learning behavior on the part of the agent. How the system should behave in case of failure is a whole research issue in itself, that I haven't addressed in my thesis at all, and that is in general not central to my main research interests.

7.1.7 Update

If in the end there is more than one $\texttt{Recipe}_l$ which successfully matches, then heuristics should be applied to check whether one of them has the "best fit" with the input instruction: presumably, this would be the `Recipe` whose header and body have the best match with respectively `Goal` and $\{\alpha_i\}$. Clearly this evaluation function should take into account various factors, including the discourse context, represented by the `PlanGraph` built so far. Notice also that my hypothesis is that the whole expansion of $\texttt{Recipe}_l$ is added to the `PlanGraph`.

The update step creates the nodes corresponding to the `Goal`, to the α_i, $\alpha_i \neq$ `Goal`, and to the $\gamma_{l,j}$; and the edges either deriving from the `Body` and `Annotations` on the `Recipe`, or computed during steps 3(a)i and 3(a)ii in Fig. 7.1. All these structures are then added to `PlanGraph`. It may happen that the algorithm finds out that some α_i and $\gamma_{l,j}$ can be collapsed, in this case only one node, corresponding to the action description `(and` α_i $\gamma_{l,j}$`)`, is created.

Now is the time to give a few more details on the `PlanGraph` data structure: it is composed of nodes that contain descriptions of individual actions, and edges that denote relations between these actions.

The *plan graph* is a labelled directed graph - i.e. a triple (N, E, L) with $E \subseteq N \times N \times L$ and $L \subseteq \text{RelNames} \times \mathcal{P}(\mathcal{E})$, where N are the nodes, E the edges, RelNames the names of the possible relations between the actions represented in the nodes, and $\mathcal{P}(\mathcal{E})$ the power set of the possible expectations — the set $\mathcal{E}$ corresponds to all the possible individuals of type *state*, according to the definitions in the TBox. I am using the power set of $\mathcal{E}$ because the set of possible expectations associated with an edge can have any cardinality, including zero.

Nodes. A node contains an individual action description, i.e. a CLASSIC individual expressed in the CLASSIC language and using CS primitives, as described in Sec. 6.1.3. Besides, the node is annotated with the effects of the action, which are described in the relevant recipe, and possibly with the expectations deriving from the qualifiers in the recipe — see below.

Edges. The edges represent various relations between actions. The label on an edge minimally includes the name of the relation holding between the actions described in the nodes the edge links, and possibly some expectations. The possible relations between actions are:

1. *Temporal*, such as *before* and *meets*, discussed in Sec. 6.2.1. Also, in future extensions to the plan graph in which states are represented, an edge to link an action and a state will have to be introduced: its meaning is that the action has to start when a given state comes to hold — this would model instructions such as *turn the gas off when the tofu has turned golden.*
2. *Generation* and *is-substep*. An edge *is-substep* links two nodes containing action descriptions α and β if β corresponds to the header of a `Recipe` and α to one

of `Recipe`'s substeps — namely, as discussed in Ch. 5, if α indirectly generates β. If the body is composed by a single action, *is-substep* reduces to *generation*.

3. *Enablement*.

4. *And, or*. These edges are not included in the implementation, but may be needed to represent actions belonging to a certain set: *and* will link two actions that both have to be performed, but that are totally unrelated to one another, such as *Prepare a dessert and do laundry* [3]; two actions are linked by an *or* edge if they are in alternative, such as *vacuum or dust-mop the parquet*.

5. *Avoid.* Some of my future work will concern how to integrate in this framework the analysis of negative imperatives proposed in Sec. 4.2 . A different kind of edge may be needed to indicate that an action should be avoided in the context of executing another action.

Expectations. Some of the expectations associated with the plan graph derive from the accommodation process; in this case, an expectation is associated to a relation between two actions, and therefore, to the corresponding edge in the plan graph: BE_{sp}([URN], [IN([KITCHEN])]) will be associated to the *is-substep* edge relating *go into the kitchen* and *get the coffee urn* — see Fig. 7.9.

Other expectations directly derive from the qualifiers associated with an action, and are associated with the node describing that action. For example, in continuing Ex. (5.24a) the instructor could say *Before you pick up the urn, be sure it is unplugged*. A qualifier associated to the recipe for *plug* is that the object to be unplugged is plugged in — the corresponding expectation BE_{ident}([URN], plugged-in) will be associated to the node corresponding to UNPLUG(URN).

7.2 Interpreting *Do α to do β*

I would like now to go back to the form of data that the algorithm is mainly intended to handle, namely, *Do α to do β*. With respect to the algorithm, this implies that in step 1 in Fig. 7.1, `Goal` is chosen to be β. The only α_i left is α. In the following I will describe steps 3(a)i and 3(a)ii: `Goal`, α_i and $\gamma_{l,j}$ from the algorithm are respectively renamed β, α and γ.

[3] The fact that they are unrelated may hold only at the understanding level; when execution time comes, the agent will possibly have to make choices, for example which action to execute first, or even which subactions to interleave.

7.2.1 Checking compatibility between α and γ

The issue at this point is to infer the structural relation between α and γ, namely, to check whether one of the two action descriptions subsumes the other, or in case this doesn't hold, make sure that the two descriptions are consistent with each other. At an abstract level, the matching step can be concisely described as checking the characteristics of the concept `(and` α γ`)`, and in particular as posing the following queries — the next query is posed only if the answer to the previous one is negative:

(7.4a) `(and` α γ`)` $\stackrel{?}{=} \alpha$, i.e., does γ subsume α?

(7.4b) `(and` α γ`)` $\stackrel{?}{=} \gamma$, i.e., does α subsume γ?

(7.4c) `(coherent (and` α γ`))`?

Various issues deserve more consideration:

1. The first thing to notice with respect to the queries in (7.4) is that α is not a concept, but an individual; therefore the concept `(and` α γ`)` can't be directly built. I will discuss this issue in the next section.

2. A second problem is that some concepts represent types of spatial locations, for which this notion of structural compatibility is too restrictive. For example, while neither of the μ_i's in (7.5) subsumes the other, and `(and` μ_1 μ_2`)` is incoherent, they can be considered compatible if TABLE$_1$ is in KITCHEN$_1$:

   (7.5a) $[_{\text{Place}}\ \text{IN}([\text{KITCHEN}_1])]_{\mu_1}$

   (7.5b) $[_{\text{Place}}\ \text{AT}([\text{TABLE}_1])]_{\mu_2}$

   To really understand such compatibilities, a geometric reasoner would be necessary: we are planning to develop one for *AnimNL*, but for the moment I have included only some simple notions of compatibility in the algorithm. One such notion is that concepts that are both of type *place* are compatible with each other, barring any knowledge to the contrary, as in the case we know TABLE$_1$ is in DINING-ROOM$_2$. As we will see, this notion of compatibility of locations will play a role when computing expectations, discussed in Sec. 7.2.2.

   Also *paths* are considered compatible, unless various conditions obtain. One such condition is that it is not acceptable to have a path whose start and end points are the same, unless it is an instance of `circular-path`; therefore the algorithm would find (7.6a) and (7.6b) to be incompatible:

   (7.6a) $[_{\text{Path}}\ \text{TO}(s)]$

   (7.6b) $[_{\text{Path}}\ \text{FROM}(s)]$

3. The mechanisms that compute the compatibility between α and the various γ_i are always active, therefore they are used also to recognize whether `Goal` is an instance of a certain header: in this case *compatibility* is restricted to *instance*, and no other relations between `Goal` and header are accepted. However in Sec. 7.2.1.3 I will discuss possible relaxations of this requirement.

4. Finally, as we will see when discussing Ex. (7.10b), sometimes the descriptions to be matched are not of actions, but of states — effects of an action or arguments to an ACHIEVE operator. The same mechanisms discussed here apply to state descriptions as well.

7.2.1.1 Implementation of the queries

As mentioned above, α is not a concept, but an individual; therefore the concept `(and` α γ`)` can't be directly built. Before discussing how the queries are implemented, some definitions are necessary.

1. If ρ is an individual,

   ρ^{conc} `= (and (cl-ind-parents` ρ`)`
   `(further-restrictions` ρ `(cl-ind-parents` ρ`)))`

   namely, ρ^{conc} is the most specific, possibly virtual concept of which ρ is an instance.[4] `(further-restrictions` ρ `(cl-ind-parents` ρ`))` checks whether ρ has further roles that are not inherited from its parents. For example, refer to Fig. 7.2 and consider a ρ like

   ```
   (and cause-and-acton
          (fills agent you)
          (fills patient sq1)
          (fills lex-item cut)
          (fills caused-role go-comp-inhalf1)
          (fills location along-diagonal1)
          (fills instrument scissors1))
   ```

   This will be recognized as an instance of the concept
   γ = `cut-sq-inhalf-along-diagonal`; however, ρ also adds to γ the role `instrument`. This is possible in CLASSIC, as roles, that have to be "declared" before being used, can possibly be on any concept or individual. Therefore
   `(further-restrictions` ρ `(cl-ind-parents` ρ`))` checks whether ρ has roles $\mathtt{R}_i$ that are not inherited from its parents, and computes a virtual concept that includes the $\mathtt{R}_i$'s too. To do this, `further-restrictions` *and*'s ρ's parents with restrictions

[4] For those who know CLASSIC: some of these definitions are slightly simplified with respect to the implementation, that has to take into account the specific input / output requirements of the CLASSIC functions.

such as
(all R_i (and (cl-ind-parents (cl-ind-role-filler ρ))))
R_i is a role that either does not appear on any of the parents; or that appears on at least one parent, but whose filler on ρ is of a more specific type than the value restriction on R_i on the parent.

2. If ρ is a concept, ρ^{ind} is an instance of ρ.

3. ρ_{perf} is the action instance to be performed.

Now let's consider the queries in (7.4).
(7.4a), (and α γ) $\stackrel{?}{=} \alpha$, can be directly implemented by means of the CLASSIC predicate cl-instance?, namely as (cl-instance? α γ). In this case $\rho_{perf} = \alpha$.

The other two queries in (7.4b) and (7.4c) — (and α γ) $\stackrel{?}{=} \gamma$; (coherent(and α γ)) — are answered by (check-real-comp α γ), described below. Given the more flexible notion of compatibility necessary to account for spatial concepts it is not possible to simply use CLASSIC queries, such as (cl-disjoint? γ α^{conc}) to answer (coherent(and α γ)), as for example μ_1 and μ_2 in (7.5) above would be found to be incoherent.

The two queries are collapsed because in both cases ρ_{perf} = (and α^{conc} γ)^{ind}: the role fillers on ρ_{perf} are computed by (check-real-comp α γ). However, not to lose the conceptual distinction between the two cases in (7.4b) and (7.4c), the algorithm also checks whether (member α^{conc} (cl-concept-ancestors γ)): a positive answer means that α is an instance of an ancestor of γ, namely, that (and α^{conc} γ) = γ. In this latter case, a simplified version of check-real-comp is used.

check-real-comp does the following:[5]

1. for each role R_i common to α and γ:

   (a) if R_i is filled both on α and β, the filler must be the same, modulo spatial concepts like *Path, Place*, for which the primitive geometric reasoner discussed above takes over;

   (b) if R_i is filled on α but not on γ, the filler on α must be compatible with the restrictions on the role on γ;

   (c) if R_i is filled only on γ, the filler on γ must be compatible with the restrictions that α inherits from its parents: notice that this happens when α inherits R_i from α^{conc} without filling it, e.g. α may be defined as an an instance of cut-location without specifying the filler for the location role;

   (d) if R_i is filled on neither, then the respective value restrictions must be compatible.

2. If a role R_i appears only on one of either α or γ, I make the simplifying assumption that it doesn't affect compatibility. This clearly does not hold in general.

[5]Notice that also SAME-AS restrictions have to checked. This is done by check-same-as, mentioned in footnote 8.

If one or both of the fillers of R_i is in turn a complex concept, compatibility is recursively checked. Notice that a simplified verson of `check-real-comp` is used when α^{conc} is an ancestor of γ, as in this case roles are either common to α and γ, or they appear only on γ.

It is clear that this procedure partly mimics classification. However, classification could not be used directly as: first, α is an instance and not a concept; second, as mentioned above with regard to Ex. (7.5), classification would consider inconsistent concepts that are compatible on geometric grounds.

7.2.1.2 Variations on "Cut the square in half"

Let's go back to my main example about cutting squares in half, discussed in Sec. 5.2.1.1, and let's consider again Exs. (5.19a) through (5.19d), repeated here for convenience:

(7.7a) *[Cut the square in half along the diagonal with scissors]*$_{\alpha_1}$ *[to create two triangles]*$_{\beta}$.

(7.7b) *[Cut the square in half]*$_{\alpha_2}$ *[to create two triangles]*$_{\beta}$.

(7.7c) *[Cut the square in half with scissors]*$_{\alpha_3}$ *[to create two triangles]*$_{\beta}$.

(7.7d) *[Cut the square in half along a perpendicular axis]*$_{\alpha_4}$ *[to create two triangles]*$_{\beta}$.

Part of the knowledge necessary to deal with these examples is represented by the CLASSIC definitions shown in Figs. 7.2 and 7.3; a graphical representation of those definitions is shown in Fig. 7.4 — as usual, to keep the network readable, some roles are not shown. `Comp` is the `Composition` semantic field. The `lex-item` role is meant to store the actual verb used in the input instruction — as mentioned in Sec. 6.1.2, the CS decomposition does not capture all the semantic information associated with a lexical item; the role `lex-item` is a crude solution to the problem of retrieving this missing semantic information, as it keeps track of what was actually said in input.

Before discussing how the examples (7.7a) through (7.7d) are dealt with, notice that

- The definition of the recipe for creating two triangles from a square is [6]

```
(cl-define-concept 'create-two-triangles
     '(and recipe-1-step
           (all header create-parts-from-whole)
           (all substep1 cut-sq-inhalf-alongaxis)
           (all effect1 be-comp-sq-tr)))
```

- γ from the recipe is `cut-sq-inhalf-alongdiag`, defined as follows in Fig. 7.2

[6] With respect to the recipe in App. C, `same-as` restrictions are missing. Moreover, the problem of how to represent the cardinality of the resulting set of triangles is left unsolved.

```
(cl-define-concept 'diagonal
      '(disj-prim geometric-concept type diagonal))

(cl-define-concept 'perp-axis
      '(disj-prim geometric-concept type perp-axis))

(cl-define-concept 'along-sp-place
      '(and spatial-place (all along-role entity)))

(cl-define-concept 'along-diagonal
      '(and along-sp-place (all along-role diagonal)))

(cl-define-concept 'along-perp-axis
      '(and along-sp-place (all along-role perp-axis)))

(cl-define-concept 'go-comp
      '(and go
           (fills semfield-role composition)
           (all path-role comp-path)))

(cl-define-concept 'go-comp-plus
      '(and go-comp (fills polarity yes)))

(cl-define-concept 'go-comp-minus
      '(and go-comp (fills polarity no)))

(cl-define-concept 'go-comp-inhalf
      '(and go-comp-plus
            (all path-role to-path-comp-inhalf)))
```

Figure 7.2: The *cut* hierarchy in CLASSIC — Part A

```
(cl-define-concept 'reduce-to-pieces
     '(and cause-and-acton
           (all caused-role go-comp-plus
           (all lex-item (one-of cut break))))

(cl-define-concept 'cut-location
     '(and reduce-to-pieces
           (fills lex-item cut)
           (all location spatial-place)))

(cl-define-concept 'cut-sq-inhalf-alongdiag
     '(and cut-location
           (all patient square)
           (all caused-role go-comp-inhalf)
           (all location along-diagonal)))

(cl-define-concept 'create-act
     '(and cause-and-acton
           (all caused-role go-comp)
           (all lex-item (one-of create form))))

(cl-define-concept 'create-whole-from-parts
     '(and create-act
           (all caused-role go-comp-plus)))

(cl-define-concept 'create-parts-from-whole
     '(and create-act
           (all caused-role go-comp-minus)))
```

Figure 7.3: The *cut* hierarchy in CLASSIC — Part B

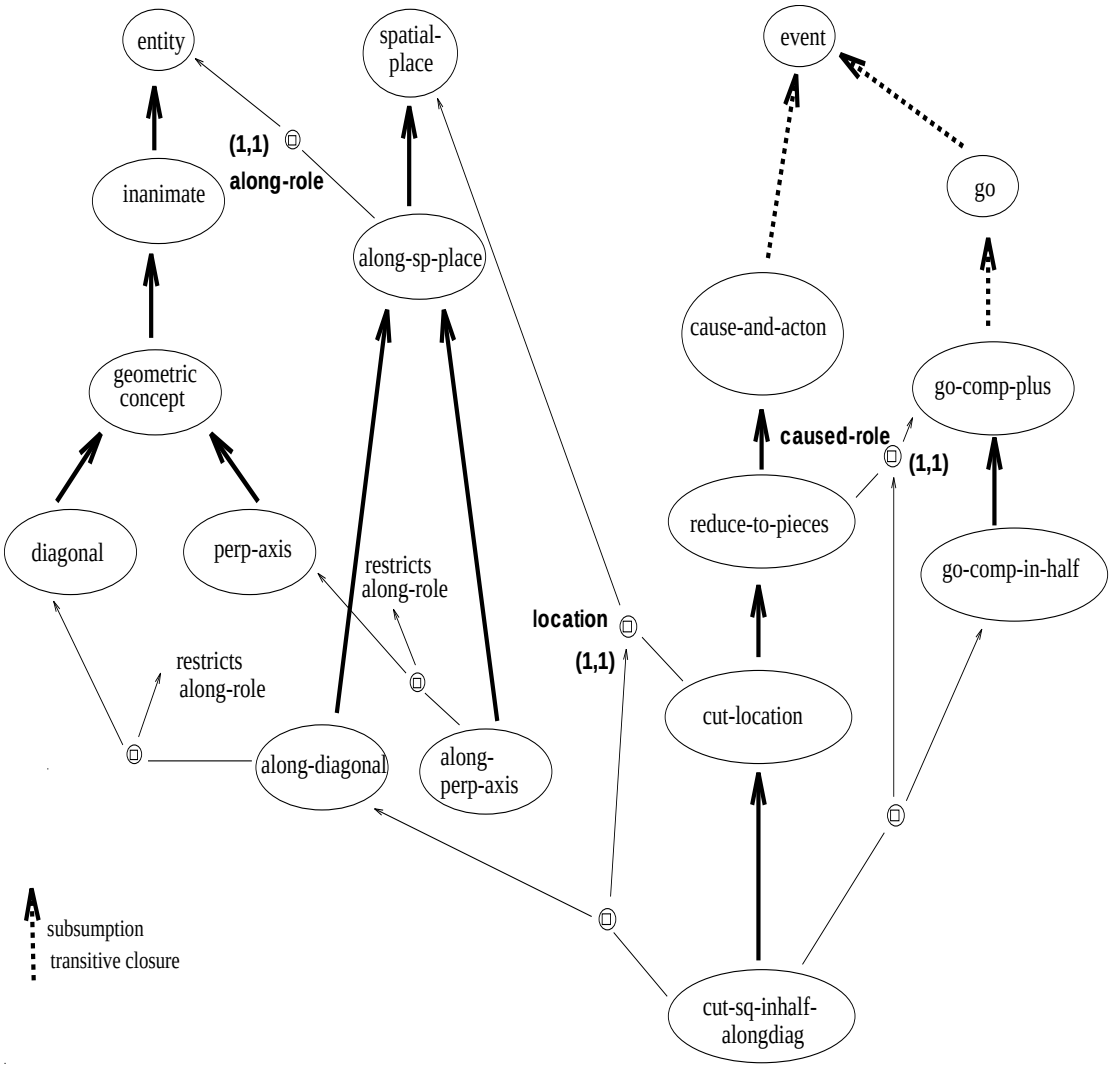


Figure 7.4: The *cut* hierarchy in network form

```
(cl-define-concept 'cut-sq-inhalf-alongdiag
                   '(and cut-location
                         (all patient square)
                         (all caused-role go-comp-inhalf)
                         (all location along-diagonal)))
```

- Remember that the logical form for α_i is added to the ABox as an *individual*. Therefore all the arguments that appear below in `fills` restrictions — `you, sq1, go-comp-inhalf1` — have already been created as individuals of the obvious types by means of `(cl-ind-add-concept <individual> <concept>)`.

Let's now derive the examples:

1. In (7.7a), α_1 is

   ```
   (and cause-and-acton
          (fills agent you)
          (fills patient sq1)
          (fills lex-item cut)
          (fills caused-role go-comp-inhalf1)
          (fills location along-diagonal1)
          (fills instrument scissors1))
   ```

   α_1 is recognized as an instance of γ `cut-sq-inhalf-along-diagonal` through `(cl-instance? `α_1 γ`)` — see Fig. 7.5. Notice that α_1 is actually an instance of the virtual α_1^{conc}:

   ```
   (and cut-sq-inhalf-alongdiag
        (all instrument entity))
   ```

   Given that no further restrictions are found by the COMPUTE EXPECTATIONS step, only the node for α_1, and no node for γ, is added to the `PlanGraph` in the UPDATE step.

2. In (7.7b), α_2 is

   ```
   (and cause-and-acton
          (fills agent you)
          (fills patient sq2)
          (fills lex-item cut)
          (fills caused-role go-comp-inhalf2))
   ```

   therefore, an instance of the virtual concept

   ```
   (and reduce-to-pieces
        (all patient square)
        (fills lex-item cut)
        (all caused-role go-comp-inhalf))
   ```

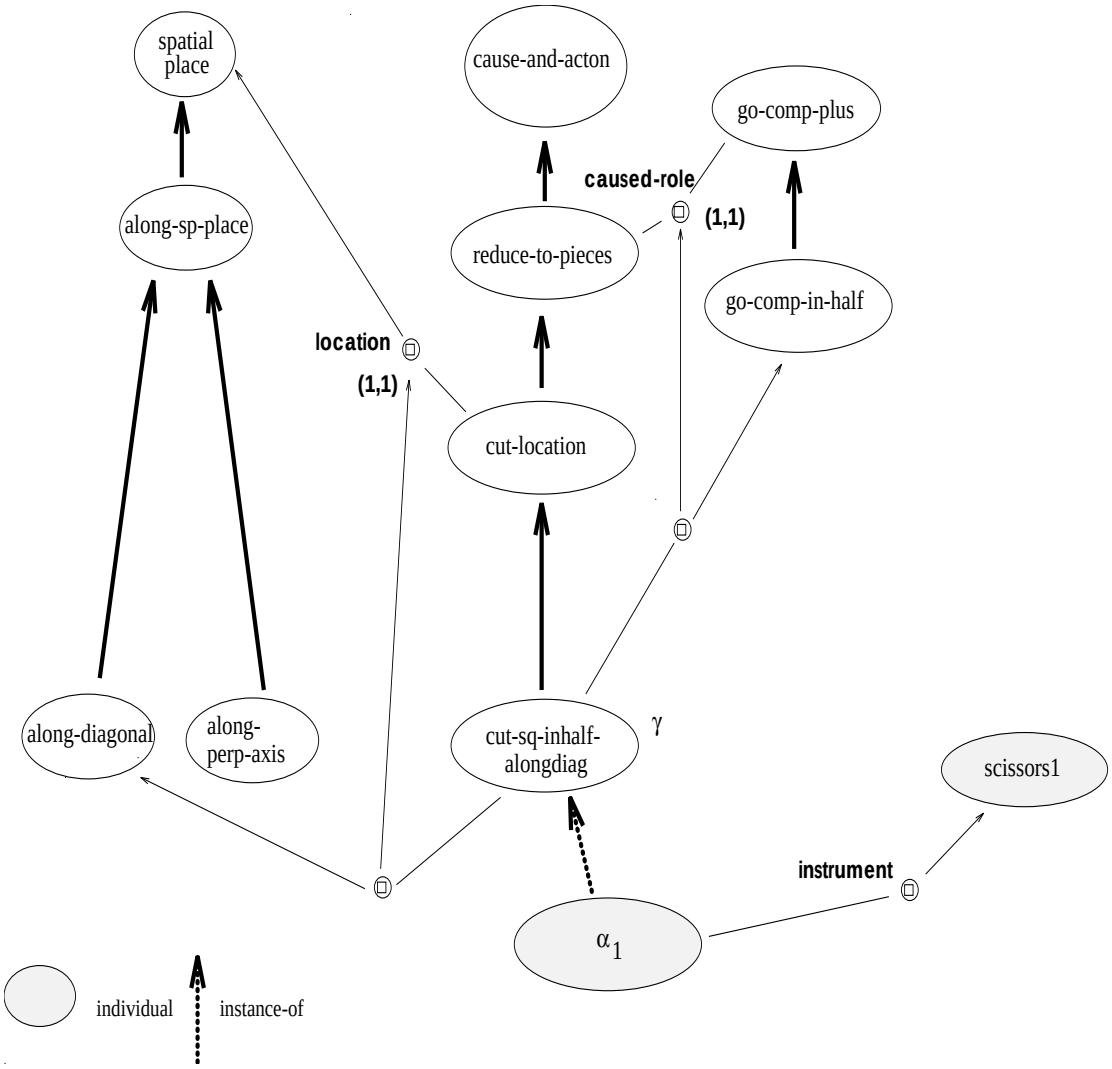


Figure 7.5: A more specific *cut*

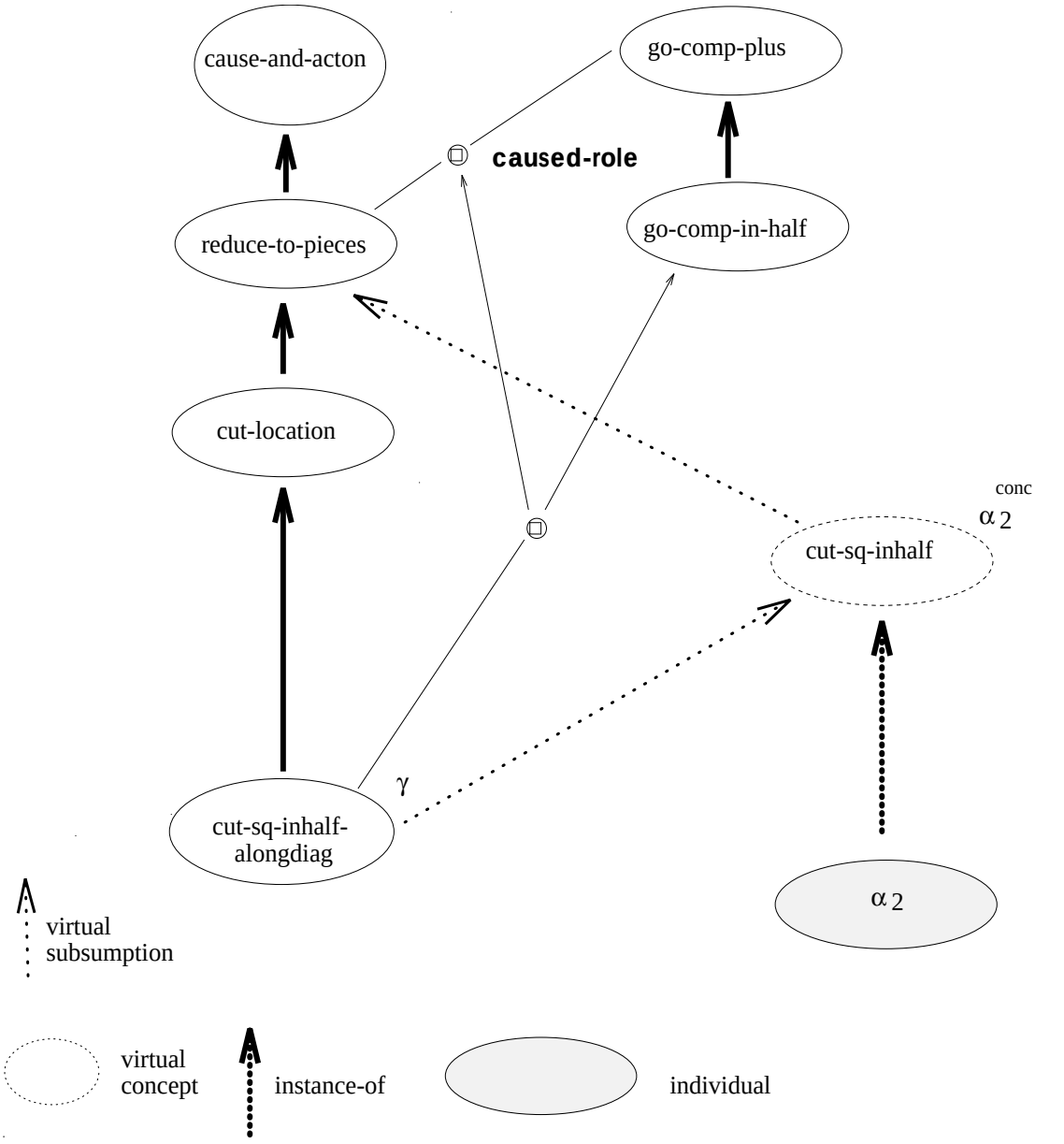


Figure 7.6: A less specific *cut*

As shown in Fig. 7.6, α_2^{conc} is an ancestor of γ. Only one node will be added to the `PlanGraph`, containing $\rho_{perf} = \gamma^{ind}$, computed as described above, namely, with the common roles filled with the fillers from α_2.

3. In (7.7c), α_3 is an instance of the virtual concept α_3^{conc}

```
(and reduce-to-pieces
     (all patient square)
     (fills lex-item cut)
     (all caused-role go-comp-inhalf)
     (all instrument entity))
```

   Therefore neither one of α_3^{conc} or γ subsumes the other, but they have a common descendant `(and` α_3^{conc} γ`)`, namely

```
(and reduce-to-pieces
     (fills lex-item cut)
     (all caused-role go-comp-inhalf)
     (all location along-diagonal)
     (all instrument entity))
```

   The action to be performed is taken to be `(and` α_3^{conc} $\gamma)^{ind}$. See Fig. 7.7.

4. Ex. (7.7d) is recognized as inconsistent with the stored knowledge — see Fig. 7.8. α_4^{conc} is

```
(and reduce-to-pieces
     (fills lex-item cut)
     (all caused-role go-comp-inhalf)
     (all location along-perp-axis))
```

   `diagonal` and `perp-axis` have been defined as disjoint concepts (cf. definitions in Fig. 7.2); therefore, although `along-perp-axis` and `along-diagonal` are both subtypes of the concept `along-place`, they are recognized as incompatible. Equivalently, the concept `(and` α_4^{conc} γ`)` would be found to be incoherent by the classifier. A `failure` is signalled.

Another example that I have modelled, where the same kind of reasoning described above obtains, is

(7.8) *Turn screw to loosen.*

Given the knowledge that *turning screw counterclockwise* generates *loosening screw*, Ex. (7.8) is dealt with as Ex. (7.7b). *Turn screw* is recognized as subsuming *Turn screw counterclockwise*: given the surface form it is the latter action that should in fact be executed — the reader can find the relevant action and recipe definitions in Appendix C. As I already mentioned, this conclusion may be wrong if the screw is in fact one of those that must be turned clockwise to be loosened. However, if the system only knows of one type of screw, the conclusion is correct from its point of view.

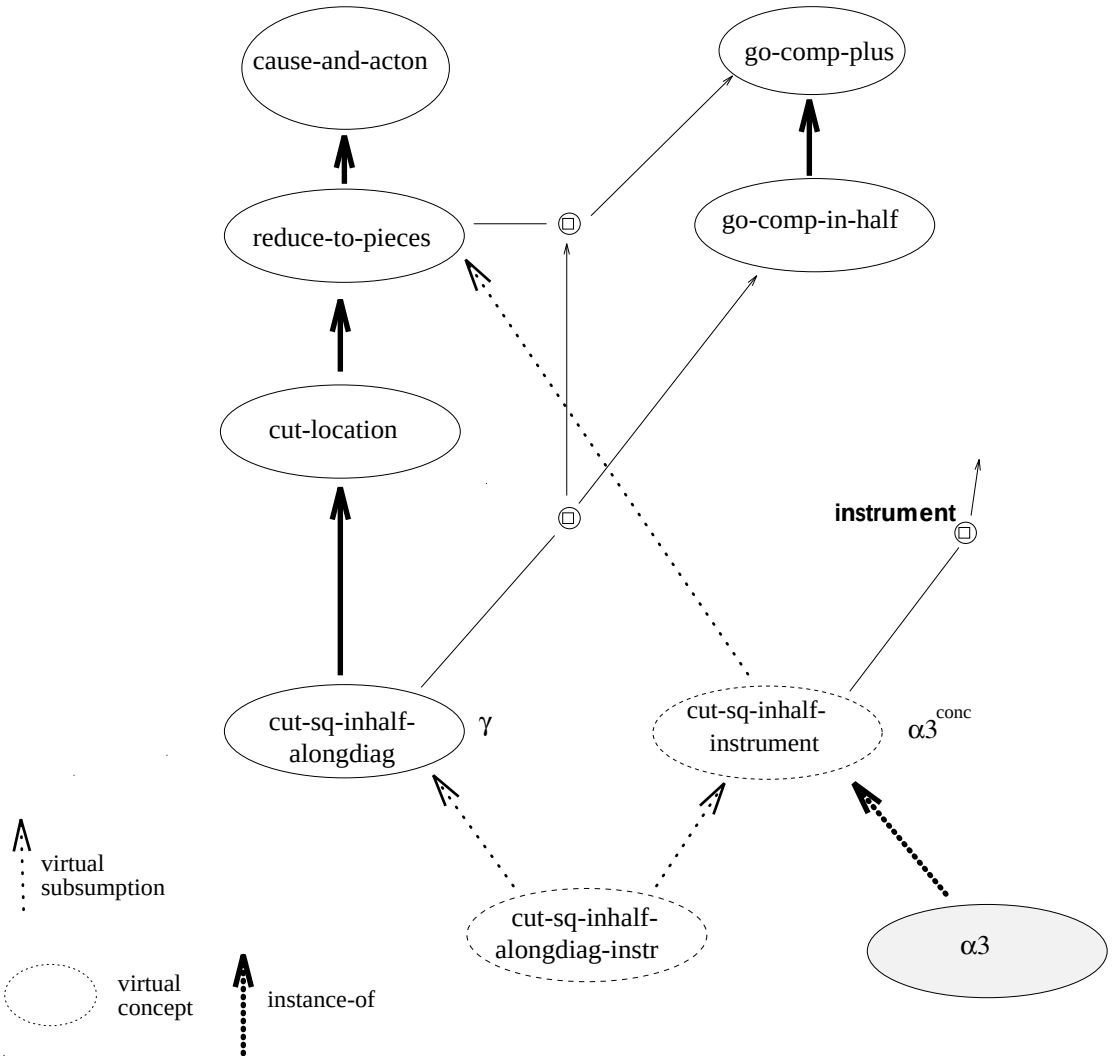


Figure 7.7: A more and less specific *cut*

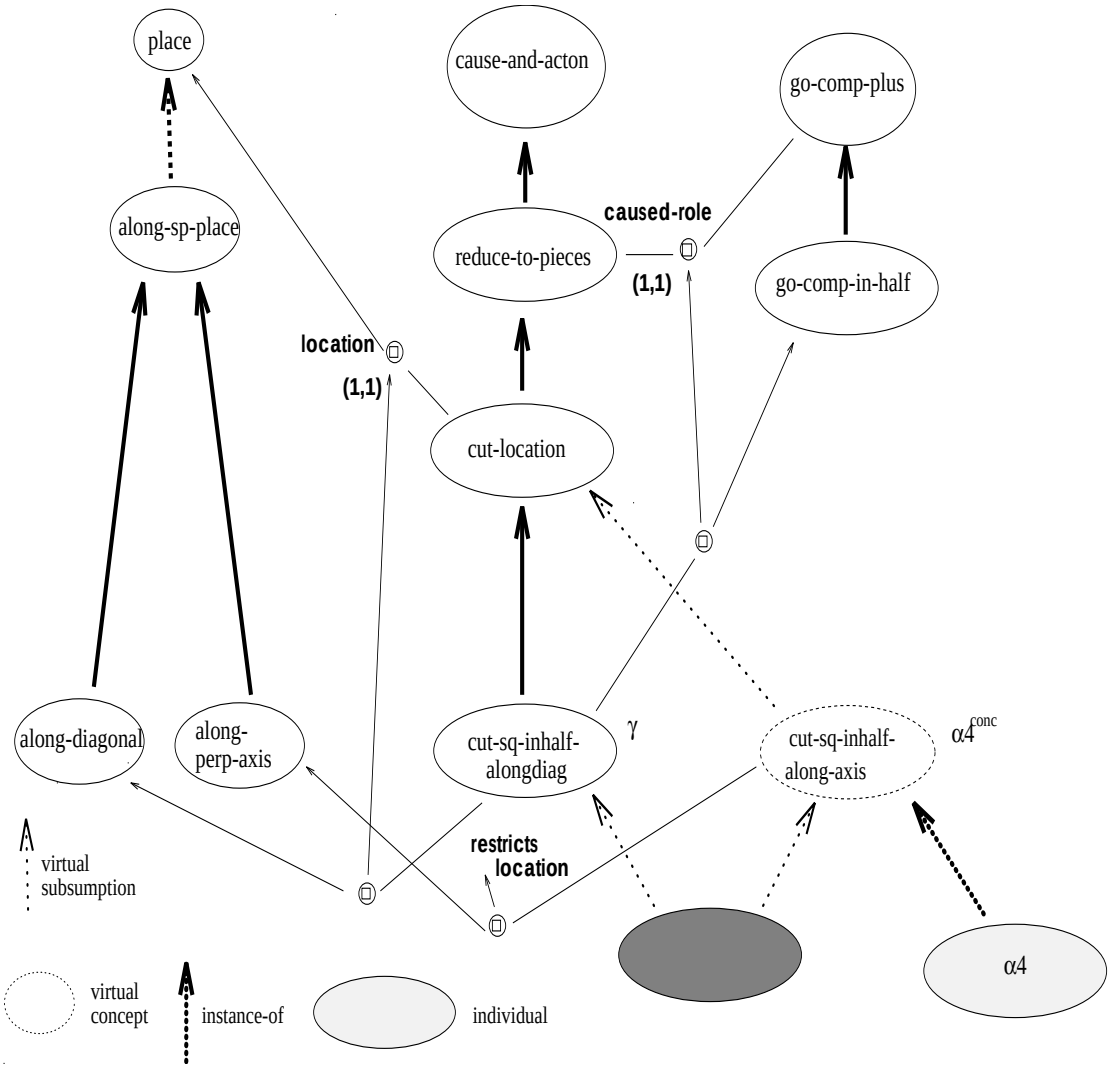


Figure 7.8: An inconsistent *cut*

7.2.1.3 Variations on the goal

As I discussed in Sec. 5.2.1.2, a different kind of accommodation process could allow β to vary with respect to the stored knowledge: namely, β could be allowed to retrieve recipes whose headers are subconcepts of β, and α could be used to select among those recipes, if possible. Let's go back to the examples in (5.21), repeated here for convenience:

(7.9a) *[Cut the square in half along the diagonal]*$_{\alpha_1}$ *[to create two polygons]*$_{\beta}$.

(7.9b) *[Cut the square in half]*$_{\alpha_2}$ *[to create two polygons]*$_{\beta}$.

(7.9c) *[Cut the square in half with scissors]*$_{\alpha_3}$ *[to create two polygons]*$_{\beta}$.

(7.9d) *[Cut the square in half along a perpendicular axis]*$_{\alpha_4}$ *[to create two polygons]*$_{\beta}$.

Also, suppose we have a recipe about cutting squares in half along an axis:[7]

```
(cl-define-concept 'create-two-rectangles
     '(and recipe-1-step
           (all header create-parts-from-whole)
           (all substep1 cut-sq-inhalf-alongaxis)
           (all effect1 be-comp-sq-rect)))
```

Now β from (7.9) would select both the previous recipe and the one about creating two triangles, as both triangles and rectangles are polygons; the various α_i would then be matched against the unique substep from each of the recipes, and the correct recipe would then be selected, if α_i is specific enough — see Sec. 5.2.1.2.

It is clear that, the more abstract the goal is with respect to the stored knowledge, the less selective it is, up to the extreme of retrieving every recipe; however α can still help select the right one(s) if it is specific enough. Clearly if both α and β are more abstract than their counterparts from a certain recipe no reasonable inference seems to be possible.

7.2.2 Computing expectations

As I discussed in Sec. 5.2.2, expectations regarding locations of objects are quite common in scenarios where an action changes the *perceptual space* an agent has access to, under the assumption that agents only have *limited perception*: they cannot have up-to-date knowledge of any part of their environment which is outside their direct perception. As mentioned in Sec. 5.2.2, the agent uses such expectations in the process of *reference grounding*, namely, of binding the referring expressions in the instructions to objects in the world. In *AnimNL*, limited perception is modeled by what is called a *portal assumption*:

[7] With respect to the recipe in App. C, `same-as` restrictions are missing.

> Portals like doors connect opaquely-bounded three-dimensional subspaces, 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. [Webber *et al.*, 1992]

Further, notice that the inferences about object locations are in a sense secondary, the primary inference being as follows: if one executes an action α that changes H's perceptual space, and / or the space that is accessible to H, and α either enables β, or is part of a sequence $\mathcal{A}$ generating β; then expect the new perceptual space μ to be the site either of β, or of (some of) the remaining actions belonging to $\mathcal{A}$. In particular, when α results in the agent going to a place μ with the purpose of doing β, one will infer by this that μ is the site of β.

The two examples that I will use to illustrate computing expectations are (5.24a) and (5.24b), repeated here for convenience:

(7.10a) *Go into the kitchen to get me the coffee urn.*

(7.10b) *Go into the kitchen to wash the coffee urn.*

7.2.2.1 Expectations about object locations

I will now show how the algorithm computes $\mathcal{E}_1$, *coffee urn in kitchen*, in the context of (7.10a). In this section, I will use the CS notation to describe actions and recipes — as usual, all such descriptions expressed in the CLASSIC language are contained in Appendix C.

The steps of the algorithm are as follows.

Step 1 and 2. The logical form produced by the parser was shown above in (7.1). `Goal` is set to β, which is found to be an instance of the header of the general *move*-action recipe shown in Fig. 6.4, and of no other recipe. Therefore, the `Recipe` *move*-action is used for further processing.
Step 3. α and the three substeps γ_1, γ_2, γ_3 are checked for compatibility.

α and γ_1 are as follows:

(7.11a) $[\mathrm{GO}_{\mathrm{Sp}}([\textsc{you}]_r, [\mathrm{TO}([\mathrm{IN}([\textsc{kitchen}])])])]_\alpha$

(7.11b) $[\mathrm{GO}_{\mathrm{Sp}}(i, [\mathrm{TO}(m)])]_{\gamma_1}$

They are compatible, since α is an instance of γ_1, with i bound to YOU and m to [IN([KITCHEN])]. The algorithm still has to verify whether there is any other information on i or m in the `Recipe` that makes i's or m's bindings untenable, such as, for example, $m = [\textsc{outdoors}]$.[8] The only other constraint on these parameters comes from the qualifier $[\mathrm{BE}_{\mathrm{Sp}}(j, m)]$. Thus we can assume that (7.10a) is really acceptable, and conclude:

(7.12) $\mathcal{E}_1 = [\mathrm{BE}_{\mathrm{Sp}}([\textsc{coffee-urn}], [\mathrm{IN}([\textsc{kitchen}])])]$

Given that α and γ_1 are compatible, the conclusion is that α *generates* γ_1 under $\mathcal{E}_1$. I characterize the relation between the two as *generation*, because if α is executed, γ is executed as well. Notice that even if α is an instance of β, I consider the relation between them to be generation, and not simply abstraction, because of $\mathcal{E}_1$. $\mathcal{E}_1$ can be considered as a condition on the associated conditional generation relation, a condition that the surface form has singled out. Therefore the two nodes cannot be collapsed: if $\mathcal{E}_1$ fails — for example because there is a note in the kitchen saying *The urn is in the basement* — another relation between α and β may need to be found, and as a consequence the `PlanGraph` may have to be modified.

The algorithm still checks whether α may be compatible with other substeps of the *move*-recipe. α and γ_2 are incompatible, as α is a simple GO_{sp}, while γ_2 involves CAUSE:
$[\mathrm{CAUSE}(i, [\mathrm{GO}_{\mathrm{Ctrl}}(j, [\mathrm{TO}([\mathrm{AT}(i)])])])]_{\gamma_2}$

γ_3 is:

(7.13) $$\left[\begin{array}{l} \mathrm{GO}_{\mathrm{Sp}}(i, \left[\begin{array}{l}\mathrm{FROM}(m)\\ \mathrm{TO}([\mathrm{AT}([\textsc{me}])])\end{array}\right]) \\ \\ [\mathrm{WITH}(j)] \end{array}\right]_{\gamma_3}$$

The same inferences that yield that α_1 is compatible with γ_1 under $\mathcal{E}_1$, could conclude that α_1 is compatible with γ_3 under

(7.14) $\mathcal{E}_2 = [\mathrm{BE}_{\mathrm{Sp}}([\mathrm{AT}([\textsc{me}])], [\mathrm{IN}([\textsc{kitchen}])])]$

However, α and γ_1 match "better" than α and γ_3. In fact, while α is an instance of γ_1, it is more abstract than γ_3: γ_3 restricts the `path-role` to be *from-to-path*, while the corresponding restriction on α is *to-path*; moreover γ_3 has a role *with-role*, which α doesn't

[8] These verifications are performed by `check-same-as`, that checks that the `SAME-AS` restrictions on the recipe hold with respect to α and β from the input. As `SAME-AS` restrictions are the way coreference is expressed in CLASSIC, `check-same-as` will find any other constraints on i or m through the `SAME-AS` restrictions they appear in.

have. Furthermore, as mentioned in Ch. 5, in naturally occurring examples in general α corresponds to the first action of a sequence generating β. Therefore, the match between α and γ_1 is preferred.

Also notice that the lexical items used in the surface form affect which match is preferred: if (7.10a) were *Come into the kitchen to get me the coffee urn*, then in fact the match between α and γ_3 would be preferred. As mentioned above, the CS decomposition doesn't capture all the semantic information associated to a lexical item, and it is to take into account these other factors that `lex-item` can be used.

Step 4. Finally, the various nodes and edges corresponding to the actions and the relations between them — α, γ_1, γ_2, γ_3, the edge labeled *generation* between α and γ_1 — are created and added to the preexisting `PlanGraph`. The *is-substep* and *enables* edges deriving respectively from the body and the annotations of the *move*-action recipe are also created and included in the updated `PlanGraph`. Given that the previous `PlanGraph` is empty, we obtain the structure shown in Fig. 7.9, where the labels on the nodes are clearly only suggestive of their contents. Notice that building the nodes corresponding to primitive actions such as $\mathrm{GO}_{sp}(\mathrm{TO}(\mathrm{AT}(\mathrm{urn})))$ implies matching them with the corresponding recipe in the action KB. Being a primitive action, the recipe only lists qualifiers and effects. For GO_{sp}, the qualifier is that the agent is not at the end of the path yet, and the effect that he is. The body is not specified, as it has to be mapped into the lower level primitives which are provided by *AnimNL*.

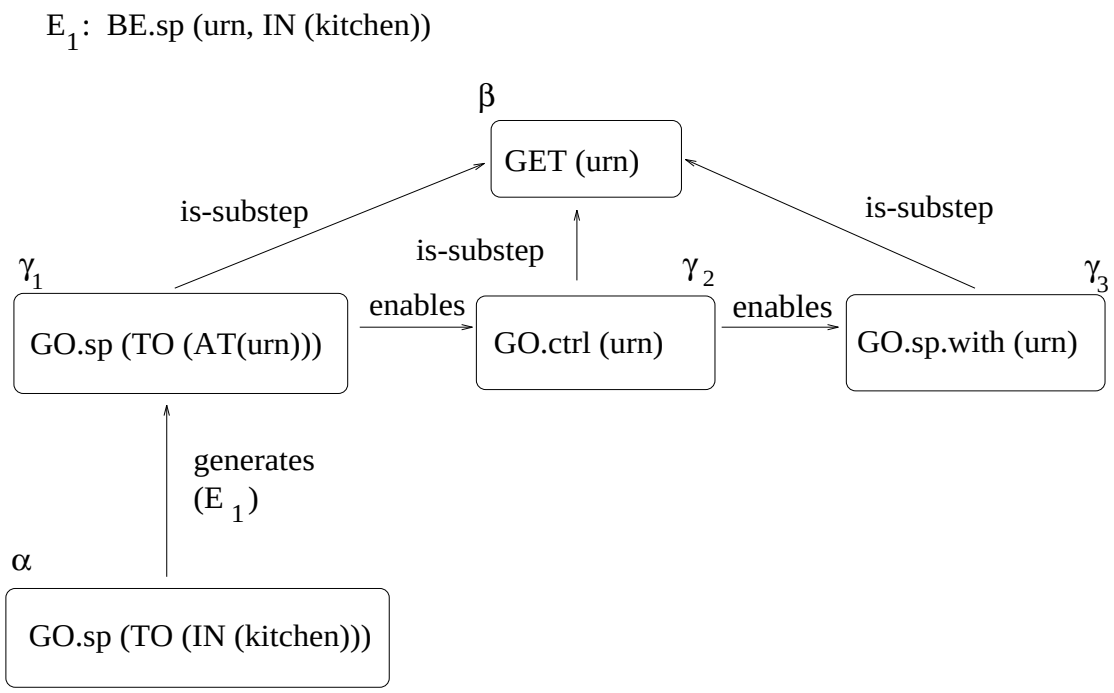


Figure 7.9: PlanGraph for *Go into the kitchen to get me the coffee urn*

7.2.2.2 Expectations about the action site

It remains to be shown how in (7.10b), the expectation that is built is that it is the washing site, not the urn, that is in the kitchen.

Fig. 7.10 shows a recipe for *wash*. Some things to notice about it are:

1. The header can be paraphrased as *the agent causes the patient to "go" along the path whose end point is the property* `clean`. The function GO in this case has the semantic field `Identi`ficational.

2. γ_1 and γ_2 are both expressed in terms of the ACHIEVE operator discussed in Sec. 6.2. It seems proper to express the first two substeps of the *wash*-action in these terms, rather than by means of a more specific action, such as $\mathrm{GO_{sp}}$.

   Notice that there is a difference between (7.15a) and (7.15b)

   (7.15a) $[\mathrm{GO_{sp}}(i, [\mathrm{TO}([\mathrm{AT}(j)])])]$

   (7.15b) $[\mathrm{ACHIEVE}(i, \mathrm{BE_{sp}}(i, [\mathrm{AT}(j)]))]$

   For (7.15a) we want the agent to perform an action of $\mathrm{GO_{sp}}$; while for (7.15b), we are only interested in the end result, that the agent is at j.

   Given this, expressing γ_1 in the *move*-action as (7.15a), while γ_1 in the *wash*-action is expressed as (7.15b), may seem arbitrary. However, the *move*-action basically encodes one of the senses of *fetch*, namely, *to go or come after and return with* (The American Heritage Dictionary): therefore, what is encoded is that the agent must physically go to the location of the object; while in the case of *wash* it is only necessary that the two conditions of the agent and the patient being at the WASHING SITE be both true before γ_3 starts.

3. γ_1 and γ_2 are left unordered. This enables subsequent planning processes to decide the right order in which to perform them: this order depends on the knowledge the agent has on the state of the world, for example where the agent believes the patient is.

4. γ_3 is simply denoted as PHYSICAL-WASH: once more, it is left to other processes to specify a situationally appropriate sense of physical washing.

Referring back to the algorithm in Fig. 7.1, steps 1 and 2 are the same as for (7.10a), apart that `Goal` matches the header of the *wash*-action. For step 3, we try to match α with a substep γ_i of the *wash*-action. To deal with the ACHIEVE operators, the algorithm first retrieves the effects of $\mathrm{GO_{Sp}}$, that are then matched to the state argument to ACHIEVE.

Header

$[\text{CAUSE}([\text{AGENT}]_i, [\text{GO}_{\text{ident}}(j, k)])]$
$\quad [\ \text{TO}([\text{AT}([\text{CLEAN}])])\ ]_k$

Body

- $[\text{ACHIEVE}(i, \text{BE}_{\text{Sp}}(i, [\text{AT}([\text{WASHING-SITE}])]))]_{\gamma_1}$
- $[\text{ACHIEVE}(i, \text{BE}_{\text{Sp}}(j, [\text{AT}([\text{WASHING-SITE}])]))]_{\gamma_2}$
- $[\text{PHYSICAL-WASH}(i, j, [\text{AT}([\text{WASHING-SITE}])])]_{\gamma_3}$

- Annotations -

- γ_1 *enables* γ_3
- γ_2 *enables* γ_3

Qualifiers

- $[\text{BE}_{\text{Ident}}(j, \text{AT}([\text{DIRTY}]))]$

Effects

- $[\text{BE}_{\text{ident}}(j, \text{AT}([\text{CLEAN}]))]$

Figure 7.10: A *Wash* action

α and γ_1 are respectively:

(7.16a) $[\text{GO}_{\text{Sp}}([\text{YOU}]_r, [\text{TO}([\text{IN}([\text{KITCHEN}])])])]_\alpha$

(7.16b) $[\text{ACHIEVE}(i, [\text{BE}_{\text{Sp}}\ (i, [\text{AT}([\text{WASHING-SITE}])])])]_{\gamma_1}$

The effects of (7.16a) are

(7.17) $[\text{BE}_{\text{Sp}}([\text{YOU}]_r, [\text{IN}([\text{KITCHEN}])])]_\alpha$

Therefore, α and γ_1 are found to be compatible under the hypothesis

(7.18) $\mathcal{E}_1 = [\text{BE}_{\text{Sp}}([\text{AT}([\text{WASHING-SITE}])],[\text{IN}([\text{KITCHEN}])])]$

Given that the only unbound parameter is i, which is no further constrained, and that IN and AT are both *place* functions, and therefore compatible with each other, we can assume that (7.18) is compatible with the rest of the knowledge we have. As mentioned above, some more detailed geometric reasoning is needed to understand that the place $[\text{AT}([\text{WASHING-SITE}])]$[9] is contained in the place $[\text{IN}([\text{KITCHEN}])]$. This same more detailed geometric reasoner should reject the analogous expectation $\mathcal{E}_2$ deriving from Ex. (7.3), *Go into the kitchen to get me the suitcase from the car*:

(7.19) $\mathcal{E}_2 = [\text{BE}_{\text{Sp}}(\text{CAR}, [\text{IN}([\text{KITCHEN}])])]$

Notice that α could not directly match γ_2, that requires the `experiencer` parameter of BE_{sp} to be the patient; however, α could match a substep of the action chosen to expand γ_2, such as the *move*-action in Fig. 6.4. However, the analysis of naturally occurring purpose clauses in Ch. 5 shows that in general α is just part of a top level sequence that achieves β, rather than of a sequence of actions that achieves a substep of β. Also, in general my strategy is to look for the match that requires the fewer assumptions, and that postulates the "simplest" relation between the two actions: intuitively, abstraction is "simpler" than generation, given that it doesn't involve any conditions; and in turn generation is "simpler" than enablement, given that enablement can involve other actions.

Given that a match between α and γ_1 is found, and no match with γ_2 and γ_3, the algorithm stops. Finally, the `PlanGraph` is updated similarly to what discussed above, as shown in Fig. 7.11.

7.3 Structural compatibility: related work

The algorithm that I described and implemented checks for compatibility between the input action description and the stored one by testing whether the two corresponding complex

[9] This representation of WASHING-SITE is slightly different from the implemented one, in which `washing-site` is defined directly as an `at-sp-place`. I retain this representation here for readibility.

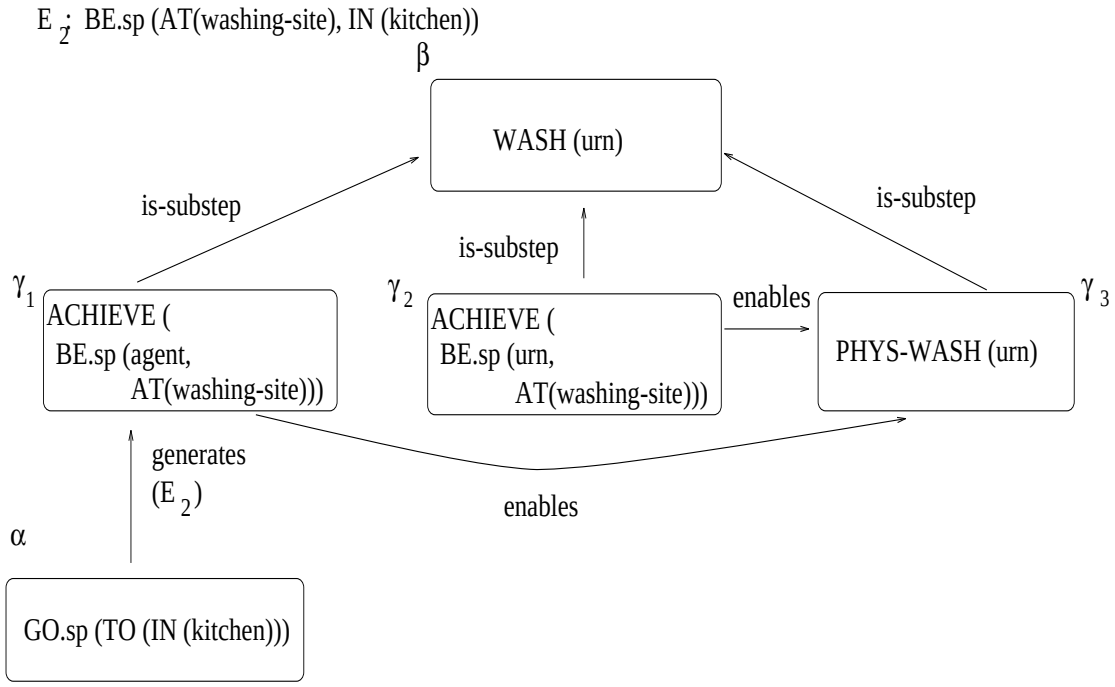


Figure 7.11: PlanGraph for *Go into the kitchen to wash the coffee urn*

objects match, or can be taken as matching. The problem of structural matching is obviously not new, and many problems in computer science, from matching actual and formal parameters in a procedure, to the computation of the most general unifier of two terms in the unification algorithm, could be viewed as performing structure matching. To restrict the field somewhat, in the following I will concern myself only with matching of complex structures; moreover, the focus will be on matching that has to occur in the absence of complete information. I will report on work whose motivations stem either from Natural Language processing or from plan inference and planning. Obviously, the *classification* algorithm itself performs structure matching. In Sec. 6.2.2, I already discussed two systems that extend subsumption to actions and plans, [Wellman, 1988] and CLASP [Devanbu and Litman, 1991].

7.3.1 Rule systems and pattern matching

It is clear that the problem of structure matching arises for example in rule-based systems. I will report here on some work by Joshi and Rosenschein [1976; 1978], which addresses the problem of performing matches with only partial information. Joshi and Rosenschein's system consists of RULES, a small active set of rules (a subset of potentially large set of rules, LRULES), partially ordered by specificity, and FACTS, a small active set of facts (a subset of potentially large set of data base facts, LFACTS). The critical feature of their inference method is that only a partial match of the antecedent of a rule is needed.

Rules are partially ordered by *specificity*. [10] Rule R_i is equal to or less than rule R_j if and only if the antecedent of R_i (represented as a set of forms) is a subset of the antecedent of R_j (represented as a set of forms).[11] Thus R_1: (P Q R) $\rightarrow E_1$ is less than R_2: (P Q R S) $\rightarrow E_2$, (R_1 is the minimal of the two), but it is not comparable to R_3: (P Q U V) $\rightarrow E_3$, since the antecedent of R_1 is not a subset of the antecedent of R_3 and vice versa.

Furthermore, if R_i' is like R_i, except that some unbound variable in R_i is bound in R_i', then R_i' is less than R_i: (P ?X B/?Y) is less than (P ?X ?Y).

The partial order on rules is then exploited to perform partial matches. The notion of a subset S $\subseteq$ FACTS *identifying* a rule R is used: this means that S is a subset of the antecedent of R. S is *maximal* if S identifies one or more rule in RULES and for all S‘ $\subseteq$ FACTS and S‘ $\supset$ S, S' does not identify any rule in RULES. If FACTS is the only maximal subset then the minimal rule that covers FACTS is chosen. Otherwise, when no single rule is able to cover FACTS, for each maximal subset the minimal rule that covers the subset is selected.

As regards the result of the application of a rule (or rules), that will be (the union of) the instantiated consequent(s) of the rule(s) corresponding to the maximal subset(s). The set of these instantiated consequent(s) become the new FACTS. The critical feature of this inference system is that it is not necessary to match the left hand side of the rewrite rule in its entirety. With suitable variable bindings, the antecedent covers as much of the FACTS as possible, while covering any item explicitly marked as necessary and containing no more unneeded items than any other such cover that is comparable to it (in terms of the partial ordering on RULES).

For example, let RULES be
R_1: (P Q R S M) $\rightarrow$ (P Q R S M U)
R_2: (P Q R S T G) $\rightarrow$ (P Q R S T G V)
R_3: (P Q R) $\rightarrow$ (P Q R W)

If FACTS = (P Q) then the antecedents of R_1, R_2, and R_3 all cover FACTS. (P Q) is the only maximal set. R_3 is minimal because it has the fewest unmatched items as compared to the other covers that are comparable to it, in terms of the partial order on RULES; this is so because (P Q R) is a subset of both (P Q R S M) and (P Q R S T G). Hence, R_3 is chosen and its consequent is instantiated, so now FACTS = (P Q R W). Note that R was not matched, but since R_3 has been selected, R has been also instantiated, namely, it has been inferred. W has been inferred also.

Let RULES be the same as before. If FACTS =(P Q M G) then there are two maximal sets, (P Q M) and (P Q G). R_1 covers (P Q M) and R_2 covers (P Q G). Hence, the result is the union of the instantiated consequents of R_1 and R_2, i.e.,

(P Q R S T G V M U)

[10] Joshi and Rosenschein's use of the term *specificity* is not very felicitous, as it roughly corresponds to ordering the rules according to the number of items, whether predicates or variables, that have to be matched in their antecedents, rather than to their antecedents really being one more specific than the other, which would be the usual understanding of the word *specificity*.

[11] Forms include constants, variables, and predicate constants that may be applied to constants, variables and other predicates.

Again, R, S, T have been instantiated, although not matched in the antecedent of R_1 and R_2.

Notice that therefore the partiality of the match is with respect to RULES, not to FACTS: namely, all the FACTS have to be covered, but each rule may cover only part of FACTS, and the antecedent of a certain rule may be more specific than the subset of FACTS it covers. Some of the inferences I perform can be seen as related to this approach to partial match: we could take α_{stored} as the antecedent of a rule, and α_{input} as the FACTS to be covered. The assumption that α_{stored} is the action to be performed when α_{input} is less specific than α_{stored}, is comparable to firing a rule even if its consequent is more specific than FACTS. However, in Joshi's approach all the FACTS must be covered, possibly by taking the union of more than one rule; therefore, it would seem not to cover the cases in which α_{input} is more specific than α_{stored}, and the ones in which neither subsumes the other, but `(and` α_{input} α_{stored}`)` is coherent: such cases would correspond to situations in which there are FACTS not covered by RULES. Clearly the first case is rather trivial if one has classification / realization at one's disposal, but the definition of *specificity* in Joshi's system is rather simpler than subsumption.

7.3.2 Unification based matching

As mentioned above, the unification algorithm itself can be seen as performing some kind of structural matching. In [1988], Charniak extends unification with nonmonotonic equality. The work presented in the paper is motivated by the fact that Charniak notes that a major component of story comprehension is understanding the character's actions in terms of the character's *motivations*: this problem can be characterized as *abduction*, as one wants to explain the observed actions in terms of an inferred motivation. Charniak divides motivation analysis into four parts:

1. finding possible motivations;
2. seeing if a suggested motivation is consistent with the action in question;
3. determining what additional information is needed to integrate the new line with the suggested motivation; and
4. if there is more than one possible motivation, selecting the most plausible.

After noticing that the first and fourth parts are the most important, but also the most difficult and least understood, Charniak concentrates on the second one — is the proposed motivation consistent with the given action? — and tries to give it a *reasonably formal basis*. He notices that his algorithm will also handle the third — determining the extra information needed.

Charniak views the check for consistency as a kind of pattern matching, where the incoming action is matched against actions we would expect given certain motivations. A problem arises when matching roles in the hypothesized explanation (e.g., "the instrument of the action") against objects in the story (e.g., `telescope-22`): in a first order representation,

this is equivalent to matching a Skolem function against a constant. Charniak proposes to handle this matching by an extension of unification which allows two terms to unify if they can be proven nonmonotonically equal, namely, if their equality is consistent with everything we know.

Charniak's starting point is that *motivations* are generally encoded by means of "frames", "scripts", etc. He notices that when such representations are translated into predicate calculus, role names and step names in the frame version are translated to existential variables in the predicate calculus version.

> If there was an instance of cigarette lighting, then there must have been an agent, a fire source, an event of setting fire to the fire source, etc. ... The next step is to translate from existentially quantified variables to Skolem functions. Since the existential variables are within the scope of the universally quantified variable `?l:light-cigarette`,[12] the Skolem functions will be functions of this variable. [p.277-278]

Charniak explains why the need to match Skolem functions against objects from the story arises: in looking for motivations an action A is given, and what is looked for is a supertask (or goal) G, such that A is a substep of G. To find such actions one would request,

```
(retrieve '(sub-step ?goal A)) ;A is a substep of what plan?
```

For example, to find a motivation for `light-1`, the action of Norm lighting a match, the following request would be made:

```
(retrieve '(sub-step ?goal light-1))
    ;light-1 is a substep of what plan?
```

Presumably one of the statements initially found would be:

```
(sub-step ?l:light-cigarette (light-step ?l))
;Light-steps of cigarette lighting are substeps of same.
```

We are then faced with unifying the following statements:

```
(sub-step ?goal light-1))
(sub-step ?l:light-cigarette (light-step ?l))
```

The only problem in unifying these two is allowing `light-1` to match `(light-step ?l)`, which is a problem of matching a constant against a Skolem function.

Charniak then turns to discussing how the problem can be solved. First of all he notices that unification has to be extended with equality. However, this is not sufficient, as unification with equality allows matching Skolem functions against constants, but only in the

[12] Charniak's notation is as follows: `?e:elephant` indicates that `?e` is universally quantified over elephants.

context of a suitable proof of equality. However in motivation analysis typically cannot be proved the necessary equality relation. In

```
(sub-step ?goal light-1))
(sub-step ?l:light-cigarette (light-step ?l))
```

> we cannot prove that `light-1` is equal to the Skolem function. *Indeed, deciding this is what our abduction procedure is all about*, since if they were equal we would know why Norm lighted the match—in order to light a cigarette.[p.280]

Therefore, the problem of abductive unification is reduced to that of proving the consistency of an equality statement between a Skolem function and an instance of an object. To do so, Charniak resorts to *the equality of indiscernibles*: if two objects have all the same properties (size, shape, location, etc.), then they are equal. Charniak notes that in general this rule is not useful in AI systems, because it requires complete information about the two objects, and it is computationally intractable for objects about we know a lot. However, in this case these two problems are not so bad. First, one does not need perfect information to prove consistent equality. Second, the knowledge about a specific Skolem function is not too large, as the function is defined only within the boundaries of the clause in which it is introduced. Charniak concedes that in the case of plans this clause will be rather large, containing a conjunction of statements which describe all the steps of the plan, all of the objects used in the plan etc. However, it seems unlikely that more than ten or twenty relevant facts will be known. Therefore Charniak concludes that to use the non-monotonic version of equality of indiscernibles all that is required is to find out everything we know about the Skolem function, substitute in the instance, and then prove the result consistent.

Given the knowledge encoded in the plan for `light-cigarette`, we would find out the following about `light-step`: that it is of type `set-fire-to`, that the agent of `light-cigarette` sets fire to a patient which is the `fire-source` of the `touch-step`, that sets fire to the cigarette. The constants from the story — `light-1, match-22` — would then be substituted into such constraints, and they would be checked for consistency. This could recursively generate other consistency statements.[13] For the details of the algorithm, I refer the reader to [Charniak, 1988].

My own research is clearly closely related to Charniak's. Going back to his four ingredients for motivation analysis, my work could be characterized as an instance of *(3) determining what additional information is needed to integrate the new line with the suggested motivation*. In my work, step (1), finding possible motivations, is made easier by the fact that I exploit the explicit goal mentioned in the purpose clause; I have concentrated on (3), and, as I've shown in Sec. 7.2.1, I have considered cases where the additional information comes from the input description, or from the stored knowledge; moreover, I have shown how sometimes checking for structural matching helps detect incompatibility.

A very close connection between my algorithm and Charniak's is the following. The latter, while checking consistency, for every variable `?x:TYPE` finds the set of objects whose type

[13] Constraints are imposed on formulas to ensure termination of the algorithm.

is consistent with TYPE. This includes

1. all objects OB of type TYPE;
2. all objects OB of TYPE', with TYPE' subset of TYPE;
3. all objects OB of TYPE'', where TYPE is a subset of TYPE'', and (consistent (inst OB TYPE)).[14]

The way I check compatibility between the input and stored action descriptions follows this schema; however, in my case types are complex structures, while in Charniak's paper they are primitive terms.

7.3.3 Plan inference

In Sec. 3.2 I already extensively described the motivations behind plan inference work, and various approaches to the problem. I will now go back to some of those systems and focus on whether and how their algorithms perform some sort of structural matching. In general in all these approaches action descriptions are primitive terms, while in my case they are structured.

7.3.3.1 Kautz

The problem that [Kautz, 1990] sets out to solve is to infer the plan(s) underlying a sequence of events. Kautz's representation is based on *event hierarchies*. An event hierarchy is a collection of first-order axioms expressing abstraction and decomposition relations among events. The decomposition axioms include both information about components of actions and equality and temporal constraints, preconditions, and effects — Fig. 7.12 shows a decomposition axiom for the event *MakePastaDish*.

By using such event hierarchies, Kautz's algorithm is able to infer what plans a certain event can be part of; at each iteration, the algorithm tries to find an abstract event that covers all the events considered so far and the current one. Suppose the initial event description is ϵ: it may be inferred that ϵ can be part of the plans to achieve G_1 or G_2. A second event description ω is in turn part of plans to achieve G_1 or G_3. Finally, by merging the two subgraphs, Kautz obtains a description of what plans both ϵ and ω may be part of - see Fig. 7.13.

In general, the difference between my algorithm and Kautz's is that I don't assume an exact description of the action to be performed, and in fact I use a known goal to compute the more refined actions needed to satisfy the goal. Kautz goes the other way, as he exploits known and complete descriptions of events to infer higher and unknown goals.

Furthermore, Kautz's event hierarchies only encode planning, rather than linguistic, knowledge: they correspond to my recipes. It is true that Kautz does provide role functions on

[14] (consistent (inst OB TYPE)) means "it is consistent to assume that OB is an instance of TYPE.

$\forall$x Make.Pasta.Dish(x) $\Rightarrow$

Components	MakeNoodles(s1(x)) $\wedge$
	MakeNoodles(s2(x)) $\wedge$
	Boil(s3(x)) $\wedge$
Equality	agent(s1(x)) = agent(x) $\wedge$
constraints	result(s1(x)) = input(s3(x)) $\wedge$
Temporal	During(time(s1(x)), time(x)) $\wedge$
constraints	Before(time(s1(x)), time(s3(x))) $\wedge$
	Meets(time(x), postTime(x)) $\wedge$
Preconditions	Dexterous(agent(x)) $\wedge$
Effects	Holds(readyToEat(result(x)),postTime(x))

Figure 7.12: A plan in Kautz's formalism

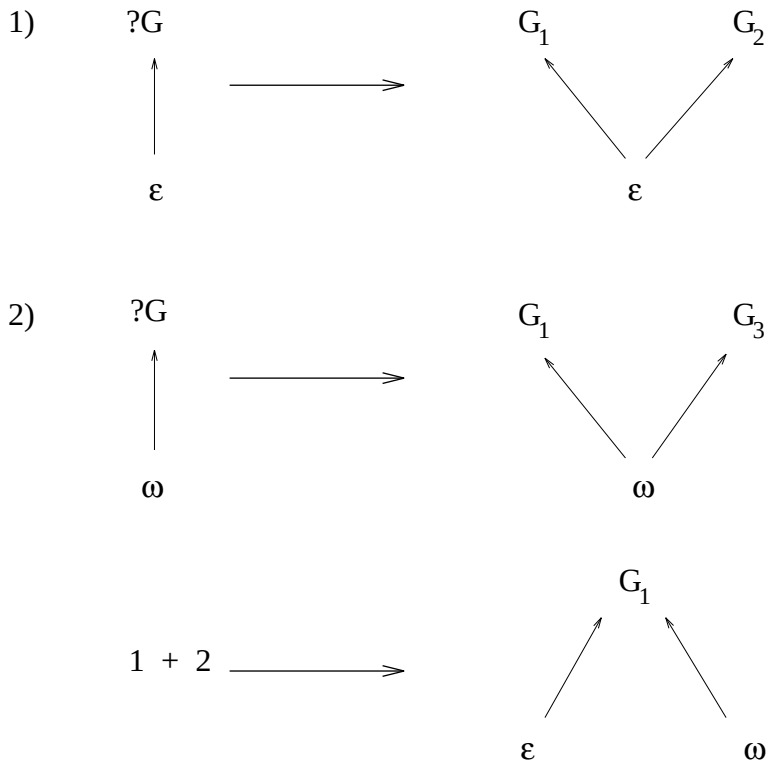


Figure 7.13: Kautz's inferences

event tokens, that yield the parameters of the event; however, the parameters filling the roles are not structured, they are basic terms, and therefore no structural matching is performed on them. In general, events are terms rather than structures, and terminological inferences are not of concern.

7.3.3.2 Pollack

I already discussed the main contributions of [Pollack, 1986; Pollack, 1990] in Sec. 3.2.1.2; here I would like to briefly discuss the algorithm she implements. She is concerned with finding the relation between a known goal G and a known action α, relation that for her is embodied in an *explanatory plan*, or *E-plan*, which is built out of beliefs and intentions as explained in Sec. 3.2.1.2. Pollack's algorithm finds the proper E-plan relating G and α by searching in parallel for E-plans of which α is part, and for E-plans which achieve G. The important point is that certain conditions have to hold for an E-plan to achieve G. The conditions arise from generation relations that relate pairs of actions which appear in the E-plan, and form a totally ordered sequence that links α to G. If these conditions don't hold, then the algorithm will search for another E-plan, possibly independent from α, that can achieve G, and will suggest it to the user – see Fig. 7.14.

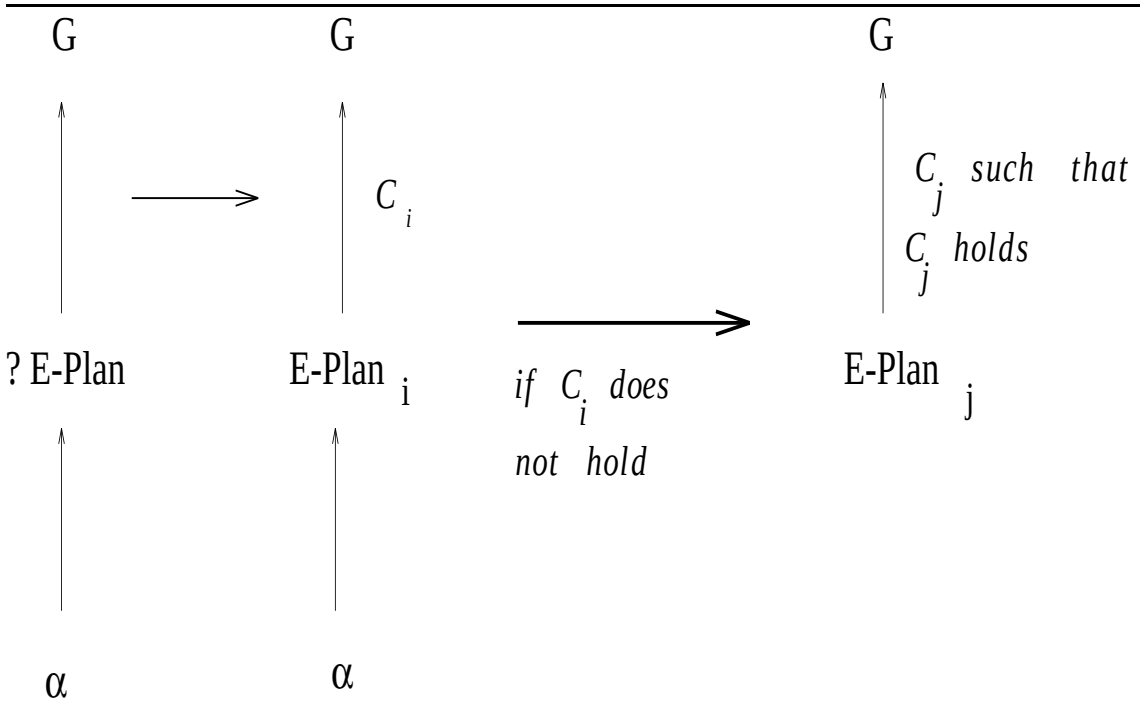


Figure 7.14: Pollack's inferences

Therefore, both my and Pollack's algorithm heavily rely on already knowing the goal. However, if α doesn't achieve the goal, my approach is to compute constraints on it, so that α can be understood as α', that does achieve G; instead, Pollack finds another E-plan

that does achieve G, where α and the new E-plan are not necessarily related. Moreover, Pollack, although her general motivations arise from modelling NL dialogues, doesn't take the linguistic structure of action descriptions into account. She represents actions by means of predicates, and her *recipes-for-action* are limited to conditional generation relations between two action types. So once more, in her work no structural matching is performed on the inner structure of action terms, which are considered as primitives.

7.3.3.3 Lochbaum

As described in Sec. 3.2.1.3, Lochbaum's work [1991a; 1991b] focuses on how, in the context of collaborative activity, the agents build a SharedPlan to achieve a certain known goal **A**. If the two agents are indicated by G_i and G_j, Lochbaum's algorithm models G_j's attempt to relate γ to **A**, where γ is the type of the *Act* G_i has just mentioned. The algorithm examines all the known *recipes* for **A** that G_j knows, to see whether γ is contained in such recipe. If yes, it computes possible constraints that may have to hold for the particular relation $\mathcal{R}$ between γ and **A** to hold. The recipes are represented by means of the action formalism devised by Balkanski [1990].

Lochbaum's algorithm builds *augmented recipe graphs* which are composed of two parts, a *recipe graph* or *rgraph*, representing activities and relations between them, and a set of constraints, representing conditions on the agents and times of these activities. The structure of an rgraph mirrors the structure of the recipe to which it corresponds: each activity and activity relation in an rgraph is derived from the corresponding act-type and act-type relation in its associated recipe. The set of constraints is derived because the constructors and relations used in specifying recipes may impose agency and timing constraints on the successful performance of act-types.

Lochbaum's work can be seen as an improvement on both Pollack's and Kautz's. With respect to both, Lochbaum uses a much more complex representation of actions.

Pollack's algorithm doesn't look for parameter bindings that may support the relation between the E-plan and the goal G: parameters are automatically bound, and the only test performed is whether the ground conditions hold with respect to the agent's beliefs. On the other hand, Lochbaum's algorithm computes the possible variable bindings under which such conditions may hold.

Kautz's algorithm is limited to reasoning about actions performed by the same agent, and also deals only with ground events, while Lochbaum allows more flexibility by reasoning about actions whose agents and time are not specified.

With respect to my algorithm, again, Lochbaum's suffers from not taking into account a complex action hierarchy. Although Balkanski's representation calls for the usage of a lattice of act-types, the usage of such lattice is never specified; and in fact Lochbaum remarks:

> further research is required to determine how to represent and use this type of information [act-type lattice] in the augmented rgraph representation. [1991b, p.22]

As regards the constraints that Lochbaum computes, they are of limited scope, as they concern either the agent(s) or the time of an activity. My algorithm instead is able to compute more complex constraints, such as those relative to the location of an object that an action manipulates. On the other hand, Lochbaum deals with multiple agents, while I don't.

7.3.4 Planning and abstraction

Making use of abstract operators as well as their more specific specializations has been a trend in planning since at least Sacerdoti's work on NOAH [1977]. However, abstraction in planning has almost always been meant either as operator decomposition, or as having more specific operators that inherit preconditions and effects from their ancestors — one example is Kautz's representation described above. To my knowledge, the only one that deals with really abstract operators, namely, operators that can solve a planning problem on an abstract representation of the domain is [Tenenberg, 1989]. Tenenberg claims that the novelty of his approach emerges from two main sources: first, the use of inheritance orthogonally throughout a planning system, from inheritance of object types, to relations between object types, to actions that effect object types; second, the precise specification of the relations between levels, which support the relations holding between solutions derived at these different levels.

> Abstract representations typically differ from lower level representations by distinguishing between those aspects of a domain which can be considered details, and those of primary importance. ... For example, `Bottles` and `Cups` can be considered abstractly as `Containers`. Object classes are used as the basis for an inheritance structuring on relations between objects, and actions applied to objects. Therefore, abstract actions effect relations between elements of abstract object classes. For example, one might abstract the predicate `InBottle(WineX,BottleY)` to `InContainer(LiquidX,ContainerY)`, and abstract the operator `pourBottle(BottleY)` to `pourContainer(ContainerY)`. Abstract plans are specialized by choosing, for each abstract step, a concrete step that achieves the desired effect over a smaller class of objects. [p.475]

Tenenberg gives a complete specification of how to abstract first-order theories, and building on this formalization, of how STRIPS systems can be abstracted by abstracting their operators; abstract STRIPS systems of this type have the *downward-solution property*, namely, each plan at an abstract level is specializable by an isomorphic concrete plan, and for every abstract level inference, there exists a set of isomorphic images that defines the space of specializations.[15]

Tenenberg's contribution is an important one; however, it should be clear that once again the terms of the representation are atoms, and not structured objects. The results he

[15] Although the solutions in [Tenenberg, 1989] hold for an arbitrary number of levels of abstraction, he refers to two levels in particular, abstract and concrete.

obtains with first order theories could probably be obtained as well exploiting subsumption, as was done in CLASP [Devanbu and Litman, 1991]: in fact, CLASP subsumption between plans seems to be more flexible than Tenenberg's model. In the latter, plans must always be structurally isomorphic across levels, while in the former, plan classification allows to define plan subclasses by adding substeps to the superclass.

7.4 Summary

In this chapter I have presented the algorithm that builds the first pass of the plan graph from the input instruction. I have shown how the inferences discussed in Sec. 5.2.1.1 are implemented: in particular, how *compatibility* of two action descriptions is computed by exploiting the classification mechanism of the hybrid system; and how certain expectations about the location of the objects manipulated by the actions described in the input and about the site of some of these actions are computed.

Many issues can be pursued for future work, among them

- A representation of the Hearer's intentions that distinguishes between Speaker's and Hearer's beliefs. As mentioned in Sec. 4.2, this would be necessary to model negative imperatives; and it would be crucial to address issues of miscommunication and repair, that may arise when the algorithm fails to find the connection between α and γ. If `failure` were to be addressed, then one further issue would be what recovery strategies to adopt.
- In Secs. 7.2.1 and 7.2.2 I mentioned that a geometric reasoner would be needed to both compute compatibility, and to compute expectations: in particular this is necessary in the latter case, given that the expectations I compute regard locations of objects and of activities.
- One of the assumptions behind the (implemented) algorithm is that the previous `PlanGraph` is empty. Clearly this is an unattainable simplification, as the ultimate goal of my work and of the *AnimNL* project is to process a sequence of instructions. Therefore, the issue of integrating the local structure into a preexisting plan graph has to be addressed: this brings in further issues of discourse modeling and so forth.
- While discussing Ex. 7.8, I mentioned that some kind of default reasoning should be brought in, as for example only the most common type of screw, but not all of them, is loosened by turning it counterclockwise.
- It is necessary to devise more complex search mechanisms, especially a recursive step to look at the expansions of the substeps of a recipe, if no match between the input action and any of the substeps has been found.
- The algorithm is not able to deal with actions that may be construed as contributing to achieving more than one substep of β. Consider

  (7.20) *Take the coffee urn to the kitchen to wash it.*

  The main action α *take the coffee urn to the kitchen* achieves that both the agent

and the object be at the WASHING SITE.

- The algorithm may return more than one `Recipe`, that matches the input, in which case it is necessary to infer which one "fits best". Intuitively this would be the recipe that best fits both α and β, but it is not clear how "best fit" should be evaluated.

Chapter 8

Conclusions

8.1 Summary

Human agents are extremely flexible in dealing with Natural Language (NL) instructions. More specifically, in this thesis I have argued that

1. Most instructions don't exactly mirror the agent's knowledge, but are understood by *accommodating* them in the context of the general plan the agent is considering; the agent's accommodation process is guided by the *goal(s)* that s/he is trying to achieve.

2. The concept of *goal* itself is pervasive in NL instructions: a NL system which interprets instructions must be able to recognize and/or hypothesize goals; it must make use of a flexible knowledge representation system, that facilitates computing the description of the action to be performed; and it may need to execute some specialized inferences to perform such computation.

To support my claims

1. I defined *accommodation* in the context of understanding instructions, building upon Lewis's original definition [1979]. I showed that the assumption made in the literature both on plan inference and on instruction understanding, that there is a direct map between the logical form of the NL instruction and the system's knowledge about actions, is untenable when dealing with naturally occurring data.

2. I presented a pragmatic analysis of Purpose Clauses (PCs), infinitival "to" constructions as in *Do* α *to do* β, and of Negative Imperatives.

3. I analyzed some of the computational issues that interpreting PCs gives rise to, namely, the two questions of which relations between actions PCs express, and of which inferences an agent has to perform to understand PCs. My answers to these questions affect both the knowledge about actions to be represented, and the algorithm that processes such instructions. In particular my analysis, by showing that

purpose clauses express *generation* or *enablement* between α and β, lends support to the proposal, made in [Pollack, 1986; Balkanski, 1993], that these two relations should be used in modelling actions.

4. I proposed an action representation formalism that provides the required flexibility. Its two main components are the TBox which encodes *linguistic* knowledge about actions, and the Action Library, which encodes *planning* knowledge about actions.

   The TBox is expressed in the CLASSIC language [Brachman et al. 91]. Hybrid systems provide classification algorithms that compute subsumption between terms: the classifier proves crucial to understanding the relation between the input and the stored action descriptions. To guarantee that the primitives of the representation are linguistically motivated, I derived them from Jackendoff's work on Conceptual Structures [1990].

   The Action Library contains *commonsense plans* about actions called *recipes*, and employs notions derived from the planning literature. The action terms used in the plans are those defined in the TBox.

5. I presented an algorithm that implements inferences necessary to understand *Do* α *to do* β, and supported by the formalism I propose. The goal β is used as the anchor to retrieve plans from the action library; the classifier is used to infer whether α can be considered as matching (one of) the substeps in the recipe for β. The notion of *match* is much broader than simply *being an instance of*: for example, a match may hold under expectations concerning locations of objects. In Ch. 7, I showed both how the *match* between two action descriptions, and how expectations about object locations are computed. The implementation of the algorithm is used in the *AnimNL* system to build the initial plan graph.

8.2 Future Directions

In the course of the dissertation I have mentioned many open problems and directions for future research: I will comment here on those that seem more promising to me.

Purpose Expressions

As I mentioned in Ch. 5, one issue on which I have done some preliminary work is the pragmatic difference between different connectives expressing *purpose*, for example *to*, *in order to*, *so as to*. Apart from infinitival constructions, other ways of encoding purpose include *so that*; certain usages of *and*, as in *Open the box and hand me the red block*; and Means Clauses, although the inferences about S's beliefs are different for Purpose and Means clauses. Data needs to be collected as regards these other constructions.

After analyzing this extended corpus, I think one can answer some of the questions I posed in Sec. 4.1.4: in particular, why *so as to* is generally considered unacceptable if the adjunct clause precedes the main clause, and whether there is a correlation between the relations

between the two actions α and β and the felicitousness of the various connectives. Other factors are likely to affect the pragmatic acceptability of the different purpose expressions: for example, the *given / new* status of the information respectively expressed in the main and in the adjunct clause, and S's beliefs about H's beliefs and intentions that such purpose expressions convey.

Modelling the discourse

The algorithm I proposed in Ch. 7 builds a representation that I see lacking in at least three respects with regard to providing a real model of the discourse:

- The discourse model, meant as the repository of information about the entities talked about, is totally absent. Instructions, in particular cooking recipes, very often describe changing properties of objects: after executing *Cut the square in half to create two triangles* the referent in the world to which the description *the square* used to apply does not exist any more, having been transformed into two triangles — on the topic of how the discourse model has to change, and of how referring expressions are affected by the different status of the entities they refer to, see [Webber and Baldwin, 1992].

  While it could be argued that the absence of the discourse model doesn't directly affect the structural inferences I proposed, the discourse model would play some role in the case of computing object locations. Let's consider (7.10a), *Go into the kitchen to get me the coffee urn*, again. Notice that while definite descriptions are normally either *textually evoked* or *inferable*, *the coffee urn* here is *brand-new anchored*, to use Prince's taxonomy [1981]: namely, the hearer has to create a new referent for it, but the new entity is *anchored*, as it is linked to another entity already in the discourse model, *the kitchen*.[1]

  Therefore an instruction such as (7.10a) may affect the discourse model not just by introducing a new referent: the computed expectation should be part of the properties ascribed to it. However, this is just an expectation: if it doesn't hold against the world, as in the case in which the agent goes into the kitchen and finds a note "the urn is in the living room", it has to be retracted.

  Notice also that the presence of instructions such as (7.10a) has interesting consequences for the interaction between different modules in the *AnimNL* system. Suppose there is another coffee urn in the discourse model: to avoid taking the new description as referring to it, the discourse model should not be accessed before the expectations are computed.

- The discourse history is entirely lacking. Clearly, this would be necessary to process sequences of instructions, or *steps*, discussed on Pag. 129. However, the presence of the discourse history could affect my algorithm more directly than in this general way: in fact, it could be used to help select the right recipe. Recall from Ch. 7 that

[1] Actually Prince defines an Anchor as an NP properly contained in the NP of interest; I am here stretching that definition.

it is possible for the goal β to select more than one recipe, and that α is then used to select the correct one. This may happen in particular when the goal is more abstract than the header of any recipes it may retrieve — cf. Sec. 7.2.1.3. In these cases, the discourse history may help.

Moreover, to interpret PCs that describe events to be prevented, the discourse history is necessary: in such cases, β is often expected to happen in the context of a γ mentioned earlier. Consider:

(8.1) *Line the tub with cardboard* **to protect the finish and to prevent debris from clogging the drain.**

β_1, *protecting the finish* and β_2, *preventing debris from clogging the drain*, can be interpreted only in the context of the general goal γ the agent is pursuing, namely, tiling the wall above the bathtub.

- The third issue I want to raise is that the `PlanGraph` is actually a representation of the Speaker's beliefs about the Hearer's beliefs and intentions. As I mentioned in Ch. 7, such representation is sufficient for supporting my claims, but wouldn't be sufficient, for example, to model miscommunication between S and H, and to account for some failures of the algorithm itself — as in the case H only knows about left-handed screws, and therefore finds an instruction such as *Turn the screw clockwise to loosen* incorrect. A well developed Speaker / Hearer Model would build on top of Balkanski's findings about beliefs and intentions in PCs and in Means Clauses [Balkanski, 1993], discussed in Sec. 4.1.2.2, to include also cases in which instructions describe general rules of behavior, or advice on how to act in particular situations: these are cases in which a PC provides, not so much a goal to achieve, but a description of the situation in which a certain behavior is appropriate, as for example in the case of a negative main clause, *Don't use chemicals to clean your parquet floor*. Another component that should be taken into account while building the Speaker / Hearer Model is the *choices* that S believes H has with respect to acting.

Negative Imperatives

Recall that in Sec. 4.2 I showed how S can use different types of negative imperatives to identify and prune H's possible choices:

- A *DONT* imperative is used when S believes H to be aware of a certain choice point, but expects him to choose the wrong alternative among many — possibly infinite — ones.
- A *neg-TC* imperative is used when S expects H to overlook a choice point. The choice point is sometimes identified through a side effect that the wrong choice will cause.

Various issues need to be addressed to give a fuller account of negative imperatives:

1. How different factors affect the choice of one form of negative imperative, as discussed in Sec. 4.2 — S's beliefs on H's possible choices, intentionality in actions, action "relatedness" in discourse.

2. The whole issue of the computational consequences of negative imperatives needs to be addressed: I believe that the action formalism I presented is adequate for dealing with negative imperatives too, provided intentional acts are represented in a more perspicuous way. However, other kind of inferences may be necessary to process negative imperatives. Moreover, different kind of edges may have to be added to the `PlanGraph`: for example an edge such as *Avoid* could be used to relate two actions, one of which should be avoided in the context of executing another action.

3. Another issue worth considering is the interaction between negation and purpose clauses, as I discussed in Sec. 4.3.

Action Representation

In Ch. 6 I mentioned various aspects of my action representation formalism that need improvement. The main ones are

1. *Maintenance actions*. In Ch. 5 I discussed examples such as (5.11), *Hold the cup under the spigot to fill it with coffee*, and (5.14), *Save wallpaper and leftover adhesive to repair areas that may be damaged*, where α describes a *maintenance* action, namely, an action that brings about a state of the world that has to hold for a certain time τ. Intuitively, it is clear that there is a difference between *maintenance* and *achievement* actions, whose importance is in achieving the change in the state of the world, not in the time that this new state has to hold for. This difference may simply amount to the grain size of the time interval that the state achieved after executing a certain action has to hold for: it may be years for a *maintenance* action, while it can sometimes be considered almost an instant for an *achievement* action. A crisper distinction could probably be obtained by looking at examples of maintenance actions, and would have to be included in the formalism. In fact, the difference between *maintenance* and *achievement* actions may affect the agent's future behavior, and therefore, in the case of *AnimNL*, it may affect its planning processes: for example, in the case of a *maintenance* action α, the agent should avoid other actions that may change the state brought about by α.

2. *Enablement*. It is necessary to be clearer on the representation of *executability conditions* on α, which in turn affects the representation and interpretation of enablement: I represent some of such conditions indirectly as substeps in a recipe for α, but I don't think this is sufficient.

3. *Hierarchical representation of plans.* In Sec. 6.2.2, I mentioned two major shortcomings of the representation in the action library:

(a) The fact that some roles, such as `substeps`, `annotation`, `qualifier`, cannot be described in a compact way. A definition as in (8.2) for the role `substeps` is untenable:

(8.2) (**ALL** `substeps intentional-act`)(**AT-LEAST** `1 substeps`)

In fact, more specific recipes would inherit `substeps` from the concept *recipe*; however, because of the lack of the `DIFFerentiation` operator on roles in CLASSIC, it would not be possible to specify that each `substeps` has a different value restriction. Therefore, in the current implementation, roles such as `substeps` have to be explicitly enumerated. In order to express recipes more perspicuously, a possible solution is to extend the underlying CLASSIC language to include the `DIFFerentiation` operator.

(b) I didn't modify subsumption to take into account more complex notion of hierarchical relations between recipes, for example different temporal orders imposed on annotations belonging to recipes in some kind of hierarchical relation. A simple example of this is: the *move-recipe* in Fig. 6.5 imposes a total order on its substeps through the annotations. Suppose I were to define *move-recipe2* in which the temporal order of the substeps were BEFORE(γ_1, γ_3) and BEFORE(γ_2, γ_3), without any explicit ordering given on γ_1 and γ_2. *move-recipe2* should be recognized as subsuming *move-recipe*.

This type of reasoning is performed in CLASP [Devanbu and Litman, 1991], which is built on top of CLASSIC: therefore a first extension to my formalism would be to integrate the more complex action descriptions I use with the subsumption mechanisms proposed in CLASP.

The algorithm

In Sec. 7.4 I discussed numerous ways in which the algorithm I present in Ch. 7 could be extended. I think the major issues in that respect are:

1. The lack of a *recursive step* to look at the expansions of the substeps of a recipe, if no match between the input action and any of the substeps has been found.

2. The need for more complex inferences, as the one necessary to infer *hold the cup under the spigot* **upright** in (5.11), *Hold the cup under the spigot* **to fill it with coffee.** The inference is more complex because the augmented description of *hold* is not already available as part of the stored knowledge, as for *cut square in half along the diagonal*, but derives from some of the conditions on the generation relation between *depress the lever* and *fill cup with coffee* — see Fig. 5.5.

3. The fact that the algorithm can't deal with actions whose execution achieves more than one of the substeps of a certain `Recipe`.

4. If in the end there is more than one `Recipe` which successfully matches, then heuristics should be applied to check whether one of them has the "best fit" with the input instruction: presumably, this would be the `Recipe` whose header and body have the best match with respectively β and α.

5. The lack of the discourse context, represented by the `PlanGraph` built so far. This issue relates to work on discourse structure as mentioned above, and to the management of the list of active nodes, namely, of those nodes open to further expansion. The presence of the discourse context would also affect the evaluation function that has to decide which is the "best fit" `Recipe`.

Appendix A

Corpus

I include here all the purpose clauses and negative imperatives I collected, subdivided into the subtypes I discussed in Secs. 4.1.1 and 4.2. For each example, I note the source. Note the following: I use boldface to indicate the clause(s) of interest; while emphasis and capital letters come from the source itself. Some examples may include more than one purpose clause: however, given that they are subdivided into types, the example will be repeated in a different section if the two PCs are not of the same type, e.g. one PCs describes a change in H's knowledge, and a second one introduces a new referent. Only the highlighted clause is counted obviously.

A.1 Purpose Clauses

A.1.1 Bring event about

1. Transfer painting pattern to piece and finish according to personal preference. Be sure to leave base piece and foot of cutout unfinished **to allow wood glue to adhere to finished pieces**. [CWP, 1989]

2. Start in one corner of the room and measure floor-to-ceiling height for the first panel. Subtract 1/2" **to allow clearance top and bottom** to maneuver the panel into place. [CWP, 1989, 9]

3. Shut off the power, unscrew the protective plate **to expose the box.** [CWP, 1989, 9]

4. On last sealer or primer coat, sand flat surfaces lightly **to level and remove all brush or wipe marks**, with very fine grade sandpaper. [CWP, 1989, 19]

5. **To cut sharp exterior corners,** cut the first line and keep on going past the corner. Loop around in the waste portion of the stock, and come back to cut the second line. [CWP, 1989, 20]

6. **To cut sharp interior corners,** cut the first line up to the corner, then back the black out of the stock and cut the second line. [CWP, 1989, 20]
   `Comment: The magazine says` *black*`, but it's probably` *blade*.

7. If you wish to enlarge or reduce the pattern, divide a sheet of paper into larger or smaller squares than those found on the patterns. **To transfer the pattern to your grid,** note where each line on the vertical drawing crosses either a vertical or horizontal line on the pattern in this section. Using a soft lead pencil locate the same point on your grid and place a dot there. Continue this procedure until all intersecting points have been located, then join the dots. [CWP, 1989, 38]

8. Insert dowel **to temporarily secure cutout with baseboard.** [CWP, 1989, 64]

9. **To cut parts F** measure 1-3/4" from the back and mark. Measure from that point 7-1/2" and mark. Measure up from the bottom 1-3/4" and mark. Draw a line from the top mark to the bottom mark to create the slant for the sides. Cut along this edge. [CWP, 1989, 71]

10. Build this simple but elegant corner shelf **to display all the other decorative pieces found in this issue.** [CWP, 1989, 74]

11. **5. To assemble bed,** lay headboard posts parallel on a flat surface. Use the guide to align the frame and headboard piece. If using dowels carefully drill both the frame and headboard piece and post holes together for proper alignment. This is a difficult step, and if you are a beginner use flat head brass tacks to assemble the pieces.
    6. Put a small amount of wood glue on the post and assemble headboard pieces. Align footboard pieces and repeat the procedure. Allow glue to dry before assembling side pieces. [CWP, 1989, 76]

12. **To finish,** cut muslin and fleece to 12 1/2 x 18 1/2 inches. Pin muslin to mat, right sides facing, atop fleece. Sew edges together, leaving an opening.
    Turn, sew closed and quilt. [CC, 1988, 7]

13. **To finish the dolls with an antiqued look,** lightly brush on a thin coat of burnt umber oil-base paint. Rub paint into the wood with a turpentine-saturated rag; then wipe away excess with a clean, soft rag. [CC, 1988, 48]

14. **To fuse appliques,** slip the shaped pieces of fusible webbing between the garment and the applique'; press the layers together using a warm iron. Note: Slip paper towels between iron and fabric **to catch stray wisps of the heated fusible webbing.** [CC, 1988, 52]

15. To make a similar quilt from new fabrics, follow the instructions below. Refer to the photograph for color and placement suggestions. Use 1/4-inch seams; preshrink fabrics.
    TO CUT THE BLOCKS: Make a 5x5-inch cardboard template. (measurement includes 1/4-inch seam allowance). Cut 56 squares from dark pindotted fabric.
    Cut a 4 1/2-inch-square template in half diagonally for edging triangles; add seam allowances to one triangle; set the other triangle aside. Cut 30 triangles from dark pindotted fabric (cut long side of triangle with the straight grain).
    For corner triangles, cut remaining triangle template in half; add seam allowances. Cut four triangles from dark pindotted fabric (cut short sides of triangle with the straight grain). Set pieces aside.
    Each Nine-Patch block has four dark-colored and five light-colored squares. Cut a 2x2-inch template. (measurement includes seam allowance). Cut 288 squares from dark fabrics, 360 from light fabrics.
    TO PIECE BLOCKS: Sew a strip of three squares (light, dark, light). Sew another strip of three (dark, light, dark). Sew a third strip the same as the first. Piece the three strips together to make a checkerboard pattern. Repeat for 72 blocks.
    TO ASSEMBLE: Working diagonally from one corner, piece squares into strips following the diagram *below.* Join strips into rows. For inner border, cut two 2 1/2x54-inch and two 2 1/2x64 1/2-inch strips; piece if necessary. Join the short strips to the top and bottom of

the pieced top. Join long strips to sides. For outer border, cut two 8 1/2 x 58 inch and two 6 1/2x80 1/2-inch strips; piece if necessary. Join to edges in same manner above. Piece the backing to the quilt-top size. Layer the backing, batting, and top into a quilt sandwich. Baste together, then quilt as desired. Bind the edges with bias tape or 2-inch-wide fabric strips. [CC, 1988, 63]

16. **To use in a dried bouquet,** fasten floral wire to each stem with floral tape. [CC, 1988, 75]

17. **To promote vigorous blooming,** spray your flowers with a foliar fertilizer once every two weeks. [CC, 1988, 77]

18. **To finish the pillows**, cover the piping with the bias stripping and sew to the perimeter of the needlepoint design. Add ruffle (optional) by cutting the yardage in 7-inch-wide strips. Sew the strips end to end to make a 4 1/2-yard loop; fold in half lengthwise and gather to fit pillow perimeter. Sew to pillow front over piping line. Folding ruffles in, sew pillow back to right side of pillow front (leave opening for turning). Trim seams, clip corners, and turn pillow right side out. Stuff with fiberfil and sew closed. [CC, 1988, 77]

19. **To start a strip,** slip the hook through the top of the backing and pull the end of a wool strip through the top so it extends about 1/2 inch (trim end later.) [CC, 1988, 82]

20. **To mix light colors,** place white in the dish first and add color slowly until desired color is reached. [CC, 1988, 83]

21. **To prepare acrylic plastic,** cut two 4x4 pieces. Mark and score the acrylic plastic along the lines with a utility knife against a straightedge about 20 times. Bend until it breaks over the edge of the table. [CC, 1988, 88]

22. **To open the fibers in the fabric,** soak the fabric bundle in the solution for 15 minutes. [CC, 1988, 88]

23. Heat on stove **to simmer.** [CC, 1988, 88]

24. Whipstitch the feet to the legs **to secure.** [CC, 1988, 90]

25. **To shape the mouth**, fold the face piece on the mouth line; sew a 1/8-inch seam. [CC, 1988, 91]

26. **To attach arms,** use doubled white carpet thread and push needle in one side of the body and out the other side at points indicated by dots on the pattern. Thread needle through arm at dot, then through one of the holes in the tiny button. Pass needle back through second hole in button, through arm and body to opposite side. Pull thread tightly **to pull arm snugly against body.** [CC, 1988, 98]

27. **To print the checkerboard,** paint the end of the 3/4-inch wood scrap with rust paint; press the end onto the corner of the center square. Skip a 3/4-inch space; print the next check. Repeat until the checkerboard is filled, using previous line of checks as a guide. Decorate the borders around the checks with gray-painted 1/4-inch scrap; print ends with rust hearts printed over gray hearts. Varnish. [CC, 1988, 98]

28. **To prepare the working area,** clear away rugs, furniture, and anything else that might get in the way. Cover adjacent finished areas with paper or plastic sheeting. [Hallowell, 1988a, 42]

29. Wash the wall with a good household cleaner, such as trisodium phosphate (TSP, available in hardware stores) **to remove grease film, which can impair the adhesive.** [Hallowell, 1988a, 42]

30. **To tile bathroom walls,** remove wall-mounted basins or toilets. [Hallowell, 1988a, 42]

31. Leave a slight gap, about 1/32", between adjacent panels **to permit expansion.** [Hallowell, 1988a, 43]

32. **To install**, use nails or adhesive to fasten gypsum wallboard to existing walls or studs. Tape corners and cover the nail heads. Use cement-based or mastic adhesive for ceramic tile on gypsum wallboard. [Hallowell, 1988a, 44]

33. In pouring a new slab for a tile base, be sure to place a plastic vapor barrier between the slab and the ground, and reinforce the slab thoroughly to prevent any cracking. Finish the slab with a light brooming **to give it a slightly rough texture,** which will make the adhesive bond very well to the slab. [Hallowell, 1988a, 44]

34. Waterproof membrane, which comes in a kit, consists of fiber glass mesh and light tar. **To apply,** first cut the fabric **to fit the wall in a vertical section.** Roll on a thin layer of tar and then roll the piece of mesh into the tar. Cut a second piece of mesh **to fit at a right angle over the first piece**[1] and roll it into the tar. Apply a second layer and let it dry. [Hallowell, 1988a, 44]

35. If you must place new tile over old, clean it thoroughly **to remove scum, mineral buildup, coatings, wax, and dirt.** [Hallowell, 1988a, 44]

36. **To tile to the ceiling**, install new backing above the old. Shim the backing if necessary **to make it flush with the old tile.** [Hallowell, 1988a, 45]

37. **To cut the hole,** see the drawings below. [Hallowell, 1988a, 47]

38. The best time to check a slab for dampness is just after a rainstorm, when the ground is saturated with water. **To make the test,** place several 1' squares of plastic (bread wrappers will do) on the slab. Tape all the edges down **to trap any moisture**.

    After 24 hours, check the squares. If the underside are fogged with moisture, the slab is too wet to lay tile on. [Hallowell, 1988a, 47]

39. Marking the cuts. If your tiles have ridged backs, make cuts parallel to the ridges. **To mark for straight cuts,** place the tile, finished surface up, exactly on top of the last full tile you set. Then place another tile on top of this one, with one edge butted against the wall. [Hallowell, 1988a, 47]

40. **To fit a tile to an irregular contour,** match it with a contour gauge and transfer the outline to the tile with a felt-tip pen. You can also cut a pattern from cardboard and then transfer the shape to the tile. [Hallowell, 1988a, 47]

41. **To cut a hole in tile,** first drill through it with a masonry bit. If the tile is held in a vise, protect it with a piece of scrap wood on each side.
    After the hole is drilledd, pass the rod saw (a cutting cable available at your dealer) through the hole and hook it to your hacksaw. If you don't have a vise, clamp the tile to the edge of a table, protecting the tile with pieces of wood. [Hallowell, 1988a, 47]

42. Nail wood battens along the working lines **to help keep the tiles straight.** [Hallowell, 1988a, 48]

43. **To keep tiles from breaking,** keep all but essential traffic off the floor until you grout the joint. [Hallowell, 1988a, 49]

[1] Both this and the immediately preceding infinitival are ambiguous between a rationale clause and a PC in a syntactic sense.

44. (caption to figure) Straightedge marked with tile spacing will help you maintain straight course and even spacings. Check with square often **to assure professional look.** [Hallowell, 1988a, 50]

45. **To install a grab bar in a tub or shower enclosure,** locate and mark the center of the appropriate wall studs with pencil marks on the ceiling before you tile. After the tile is up, locate the studs by following down the wall from the ceiling marks with a level. Screw the bars directly to the studs through holes drilled in the tile. [Hallowell, 1988a, 51]

46. On outside corner, set one column of tile with bullnose tiles **to cover the unfinished edges of the tiles on the adjoining wall.**[Hallowell, 1988a, 53]

47. Press firmly **to bed the tile in the bond coat.** [Hallowell, 1988a, 56]

48. **To insert tiles in a wall of gypsum board,** cut out openings the same size as the tiles. Apply a beard of glue along the edges of a few pieces of wallboard. Slip them at an angle through the opening and fasten against the back side of the wall until dry, closing the opening as much as possible. [Hallowell, 1988a, 61]

49. These are the steps involved in applying grout. [Hallowell, 1988a, 63]

    (a) Spread about a cup of grout **to familiarize yourself with the process.** ... Work the trowel back and forth at different angles to the grout **to force it into the joints.** ...

    (b) After this first area, perhaps 5 square feet, has been grouted, scrape off the trowel and go over it again **to pick up the excess.** This time hold the trowel at about a 50 degree angle and work at a diagonal to the joints to minimize disturbing the grout there. Clean your trowel repeatedly in a bucket of water as you work.

50. On new tile installation, wait at least two weeks before applying the sealer **to give the grout a chance to cure completely.** [Hallowell, 1988a, 63]

51. Tiles in a panel are held firmly in position by the grout, but those in a mosaic sheet are not, despite the mesh backing. You may have to move them slightly right after laying them in the adhesive **to keep the grout lines straight.** [Hallowell, 1988a, 63]

52. After washing the tile, rinse it thoroughly **to remove detergent film;** then wipe dry with a soft, dry cloth. [Hallowell, 1988a, 64]

53. If you have hard water spots and deposits may build up on tiles that are in shower enclosures, on sink counters, and in other wet areas. You can help prevent such buildups by keeping the tile surfaces dry. Use a sponge to wipe water off the tiles after showering or doing dishes. **To remove water deposits and soap film from the tile,** apply household ammonia or a one-to-one mixture of vinegar and water, rinse thoroughly, and dry. [Hallowell, 1988a, 64]

54. Hold different tiles to the light **to compare the quality of the color and the depth it appears to have.** [Hallowell, 1988a, 64]

55. **To remove resilient sheet flooring,** cut it into strips about 6" wide with a utility knife, being careful not to damage the subfloor. The flooring will come up rather easily as it separates from the backing, which is still stuck to the floor. Soak this backing in water – without flooding the floor – and then use a floor scraper to remove it. [Hallowell, 1988a, 69]

56. **To remove old resilient tiles,** use a floor scraper as shown in the drawing below. If tiles don't come up easily, warm them **to soften the adhesive.** You can do this with an old iron or a propane torch. Once you break the tiles loose you'll find the floor still covered with the rough adhesive. Paint on adhesive remover, allow to stand, then scrape up the softened adhesive. If you can't remove the tiles, cover with an underlayment sheet. [Hallowell, 1988a, 69]

57. **To remove paint or sealers,** sand the floor to bare concrete with a floor sender and n.4 or n.5 open-cut-sandpaper. Remove high spots by rubbing with a coarse abrasive stone. Finish the preparation by scouring the floor with a stiff bristle brush and vacuuming up all loose material. [Hallowell, 1988a, 70]

58. Snap chalk lines across these points **to get diagonal working lines.** [Hallowell, 1988a, 70]

59. If the tile has a pattern or grain running in one direction, cut either right or left half tiles **to match the pattern or grain direction of the floor.**[Hallowell, 1988a, 71]

60. **To cut tiles,** score along the mark with a utility knife and snap the tile along the line. For intricate cuts, use a pair of heavy scissors. The tiles will cut more easily in this case if warmed in sunlight or over a furnace vent. Don't overheat or the tiles may scorch or melt. [Hallowell, 1988a, 71]

61. **To mark and cut border tiles,** position a loose tile exactly over one of the tiles in the last row closest to the wall (A) making sure that the grain or pattern is running in the right direction. Place another loose tile on top of the first, butting it against the wall (B). Using this tile as a guide, mark tile A with a pencil or score it with a utility knife. When cut, tile A will fit exactly in the border. [Hallowell, 1988a, 72]

62. The finished wall will look better if you avoid narrow border tiles at the corners of the wall. **To arrange this,** lay a row of loose tiles on the floor from the midpoint to the corner. If the space from the last full tile to the corner is less than one half tile, move the vertical working line over a distance of one half tile. [Hallowell, 1988a, 73]

63. **To replace a damaged tile,** first warm it and the adhesive below with a propane torch or an old iron. Pry up the tile using a putty knife or cold chisel. Remove any excess backing or adhesive from the floor until the surface is smooth and deep enough to install the new tile. Next apply the proper adhesive, keeping it back about 1/4" from the perimeter. Be careful not to get any on the surrounding tiles. Drop the new tile into place and press down firmly. Weights along the edges will ensure that the tile does not pop up before the adhesive sets. [Hallowell, 1988a, 73]

64. You can also put the can of adhesive in hot water **to soften it.** [Hallowell, 1988a, 77]

65. Work the adhesive from several different directions **to spread it evenly.** [Hallowell, 1988a, 77]

66. (caption of a drawing) Tap tiles with rubber mallet as you work **to bed them** and **to lower any high corners.** [Hallowell, 1988a, 78]

67. Work **to finish half a room at a time,** but keep your eye on the clock. Within 4 hours of the time the tile was laid, you must roll it **to set the tile fully** and **to level any pieces that have popped up.** [Hallowell, 1988a, 78]

68. **To allow for pattern repeat,** divide the height (in inches) of the wall by the number of inches between the pattern repeat; if you have a fractional remainder, round it off to the next highest number. For example, a 96-inch wall height divided by an 18-inch pattern repeat gives you 5.33 repeats, rounded off to 6.
    Then multiply the repeat measurement by the number of repeats you'll need to determine the working height figure you must use. In the case above, multiplying the 18-inch pattern repeat by 6 repeats tells you that you must calculate your wallpaper needs according to the requirements of the 108-inch wall, rather than the actual 96-inch wall. [Hallowell, 1988b, 38]

69. **To remove the wallpaper,** you can use either a steamer, available for rent from your dealer, or a garden sprayer. Sometimes, you can add a liquid to the water **to hasten the paste-dissolving process,** but ask your dealer before adding anything to the water.
A steamer converts water to steam that runs through a hose to a pan with a trigger. You simply move the pan along the wall, allowing steam to penetrate the covering. **To use a garden sprayer,** you just fill it with very hot water and spray the water onto the wallpaper. [Hallowell, 1988b, 38]

70. Don't hang the wallpaper over an existing wallpaper if the ink from the existing covering comes off; it could bleed through the new covering. **To test,** moisten a small piece of the covering with a clean sponge. If the ink comes off, seal the old covering first with an undercoat that seals stains. [Hallowell, 1988b, 40]

71. **To repair holes and other surface damage**, see page 70. [Hallowell, 1988b, 41]

72. **To paste and book the strips,** follow these steps:
1) Place the strips for one wall in the center of your work table, pattern side down. Align the first strip to be hung with the edge of the table.
2) Using a roller or brush, apply paste evenly to the back of the first strip (excess paste that laps over the edges will be caught by the back of the next strip.) Using an outward stroke, work from the center toward the edge. Be sure to apply enough paste to the edges.
3) **To book the strip,** fold the bottom third or more of the strip over the middle of the panel, pasted sides together, taking care not to crease the wallpaper sharply at the fold. Be sure the edges are aligned.
4) Fold over the remaining portion of the pasted strip until it meets the first cut end. Again, make sure the edges are aligned. Loosely roll booked strip and store it off the pasting table until you're ready to hang the strip.
5) If necessary, trim the selvage. **To do this**, place the pasted and booked strip (its edges aligned precisely) on the table. On one side, align a straightedge with the trim marks. Using a razor knife, cut through both layers. turn the strip around and follow the same procedure for the opposite side. [Hallowell, 1988b, 44]

73. In most situations, the butt seam is the best way to join two strips of wallpaper, since it's the least noticeable seam. To make one, tightly butt the edge of the strip you're hanging to the edge of the previously hung strip. Be very careful not to stretch the wallpaper. Roll the seam **to flatten it** and prevent the edges from curling. [Hallowell, 1988b, 45]

74. When a wall irregularity will cause the edge of the strip you're hanging to overlap the previously hung strip, make a double cut seam. **To do this**, complete the hanging of the second strip, disregarding the overlap. Immediately place a straightedge at the center of the overlap and, using a razor knife, cut through both layers of wallpaper. Remove the top cut-off section of the overlap. Carefully peel back the edge of the top strip until you can remove the bottom cut-off section. Smooth down the top strip; the seams should butt tightly together. [Hallowell, 1988b, 45]

75. **To hang a strip around an outside corner,** butt the new strip to the previous strip and smooth as much as you can around the corner without buckling the wallpaper at the top and bottom corners. [Hallowell, 1988b, 48]

76. **To paper around recessed windows,** see the illustrations below. [Hallowell, 1988b, 49]

77. After you've pasted and booked a strip, allow it to sit long enough to become limp. Then use a soft, natural bristle brush for pressing and smoothing it onto the wall; avoid overbrushing. **To lay the nap in the same direction,** finish with upward strokes. [Hallowell, 1988b, 50]

78. **To hang the wallpaper,** follow the steps on page 47. [Hallowell, 1988b, 51]

79. **To roll seams flat,** glue a scrap of flocked wallpaper over the roller head; press lightly on the seams **to flatten them** and prevent curling edges. [Hallowell, 1988b, 51]

80. Before you can hang the first strip, you'll need to establish plumb. **To do this**, measure from the ceiling corner 1/2 inch less than the width of the wallpaper. Mark the ceiling at that point. Move to the far end of the wall and measure in the same distance from the corner; mark the ceiling at this point also. Tack a string rubbed with chalk to the ceiling at both of these marks and snap a chalk line between them. [Hallowell, 1988b, 51]

81. Use a vinyl-to-vinyl paste for any type of border you plan to hang over wallpaper. **To paste the border for hanging,** cover the entire back with paste and book the strip; don't crease the folds.
To hang the border, begin at the least conspicuous corner. The work will go much faster if you have someone hold the folded section while you apply the border to the wall. Take care not to drip paste onto the wall and be sure to remove excess paste.
Use your fingers to press the border into corners; avoid crease marks. Because the border is narrow, it's not necessary to be concerned about how straight the corners are. Just work the border into the corner and continue around the room. [Hallowell, 1988b, 52]

82. After you've finished cleaning, wipe the surface with a clean rag **to remove any dough.** [Hallowell, 1988b, 52]

83. **To remove dirt, grease, and stains before they can penetrate the wallpaper,** thoroughly wash the soiled area with a mild soap and cold water solution. [Hallowell, 1988b, 52]

84. You can apply a protective coating to a nonvinyl wallpaper **to make it easier** to clean and prolong its life.[Hallowell, 1988b, 52]

85. Save wallpaper scraps and leftover adhesive **to repair areas** that may be damaged or stained in the future. [Hallowell, 1988b, 52]

86. Take the compost bag from the pot, open it and add enough water **to make the compost thoroughly wet.**
`Source: SimpleGrow`

87. **To unlock the knob from the underside of the lid**, turn it firmly towards you as illustrated here.
`Source: instructions for assembling a pot cover.`

88. Put a block of wood in its mouth **to hold it open.**
`Source: recipe for Roast Suckling pig, from [Prince, 1981, 234].`

89. **To carve,** place head to left of carver. Remove forelegs and hams. Divide meat down center of back. Separate the ribs. Serve a section of crackling skin to each person.
`Source: recipe for Roast Suckling pig, from [Prince, 1981, 235].`

Create new object

1. Piece the three strips together **to make a checkerboard pattern.** [CC, 1988, 63]

2. **To make a similar quilt from new fabrics,** follow the instructions below. Refer to the photograph for color and placement suggestions. [CC, 1988, 63]

3. Stuff stocking **to form a body 8 inches in circumference.** [CC, 1988, 87]

4. Join the short ends of the hat band **to form a circle.** [CC, 1988, 93]

5. Make three pieced-square blocks and one solid-square block. Join the blocks **to form a square.** [CC, 1988, 98]
6. **To make a piercing cut,** first drill a hole in the waste stock on the interior of the pattern. The diameter of the hole must be larger than the width of the blade. [CWP, 1989, 21]
7. Draw a line from the top mark to the bottom mark **to create the slant for the sides.**[CWP, 1989, 71]
8. In most situations, the butt seam is the best way to join two strips of wallpaper, since it's the least noticeable seam. **To make one,** tightly butt the edge of the strip you're hanging to the edge of the previously hung strip. Be very careful not to stretch the wallpaper. Roll the seam to flatten it and prevent the edges from curling. [Hallowell, 1988b, 45]

A.1.2 Prevent event

1. If you're pad sewing a stack of veneer, put a sheet of poster board on the bottom and the top of the stack **to prevent small pieces from breaking off.** [CWP, 1989, 21]
2. **To prevent weeds from getting a foothold in the garden,** mulch your seedlings when they are several inches high. Use a thick layer of straw, shredded bask, ground corn cobs, cocoa bean hulls, leaves, or newspaper. The mulch will smother weeds and increase soil moisture. [CC, 1988, 74]
3. Bind the edges of the canvas with masking tape **to prevent raveling.** [CC, 1988, 77]
4. Tape raw edges of fabric **to prevent threads from raveling as you work.** [CC, 1988, 77]
5. Reinsert roughed-out bottle in lathe; move the steady rest in as closely as possible **to minimize the tendency of the irregular wood to grab the tool as it rotates.** [CC, 1988, 86]
6. Remove any doors that open into the room by tapping out the hinge pins (don't unscrew the hinges) and cut with a fine-toothed saw. **To minimize splintering,** cover the cutline on both sides with masking tape. [Hallowell, 1988a, 42]
7. Line the tub with cardboard to protect the finish and **to prevent debris from clogging the drain.** [Hallowell, 1988a, 42]
8. In pouring a new slab for a tile base, be sure to place a plastic vapor barrier between the slab and the ground, and reinforce the slab thoroughly **to prevent any cracking**. Finish the slab with a light brooming to give it a slightly rough texture, which will make the adhesive bond very well to the slab. [Hallowell, 1988a, 44]
9. Starting at the wall. Many traditional tile setters use this method. Its roots are in the traditional way of setting tiles, in which tile setters worked from one end **to avoid disturbing the carefully leveled mortar bed.** [Hallowell, 1988a, 48]
10. If you start tiling against a wall that's out of square, the last course of tile along the opposite wall may have to be cut progressively wider or narrower. **To avoid this,** correct the room's layout and squareness using the following method. ... [Hallowell, 1988a, 52]
11. The vertical lines. Measure to find the midpoint of a wall and mark it on the horizontal line. Starting at this midpoint, measure with the tile stick or set a row of loose tiles on the batten to determine the size of the tiles at each end of the wall. Don't forget the spacers. If the end tiles will be less than half a tile, move your midpoint mark one-half tile **to avoid narrow pieces at each end.** Then use the level and straightedge to mark a vertical line on the wall through the mark.[Hallowell, 1988a, 55]

12. First, cover the drain and line your tub with cardboard **to prevent damage.**[Hallowell, 1988a, 57]

13. Butter (i.e. apply adhesive to) the last row of tiles individually **to prevent adhesive showing above the tiled area.** [Hallowell, 1988a, 60]

14. After this first area, perhaps 5 square feet, has been grouted, scrape off the trowel and go over it again to pick up the excess. This time hold the trowel at about a 50 degree angle and work at a diagonal to the joints **to minimize disturbing the grout there.** Clean your trowel repeatedly in a bucket of water as you work. [Hallowell, 1988a, 63]

15. **To prevent stains,** wipe up spills as soon as they happen. [Hallowell, 1988a, 64]

16. **To protect the floor from indentation,** remove any small metal domes or glides from furniture legs, replacing them with wide glides or furniture cups like those shown in the drawing below. [Hallowell, 1988a, 73]

17. Work carefully **to avoid nicking or chipping the wall's surface.** [Hallowell, 1988b, 38]

18. Some paper-backed vinyls have a tendency to curl at the edges. **To prevent this,** make sure to book each strip before hanging. [Hallowell, 1988b, 46]

19. **To avoid scratches, creases, and folds (even slight creases or scratches can greatly impair the appearance of foil),** roll up the strip, pattern side in. [Hallowell, 1988b, 50]

20. To roll seams flat, glue a scrap of flocked wallpaper over the roller head; press lightly on the seams to flatten them and **prevent curling edges.** [Hallowell, 1988b, 51]

21. Use premixed vinyl adhesive to install fabrics laminated to paper backing. Trim all selvages before hanging. Then spread paste on the fabric's backing and hang, butting edges. **To prevent staining,** keep adhesive off the fabric surface. [Hallowell, 1988b, 51]

A.1.3 Augment agent's knowledge

1. Transfer pattern to heavy paper or cardboard. Measure dimensions of box from pattern **to determine the amount of material you will need.** [CWP, 1989, 36]

2. New tile on the floor may mean you'll have to trim the doors to allow for the added height. **To determine where the door must be cut,** place a tile on the floor against the door and mark. Allow an additional 1/8" clearance for inside doors. On exterior doors allow for weather-stripping on the bottom edge of the door. [Hallowell, 1988a, 42]

3. **To check a floor for dips**, run a long straightedge over the floor and look at it against the light for gaps under the board. [Hallowell, 1988a, 43]

4. To lay ceramic tile over a vinyl or linoleum floor, use epoxy adhesive or a good mastic. Be sure that the floor is 1 1/4" thick, drilling a small hole **to check** if necessary. [Hallowell, 1988a, 44]

5. Before you start to mark your working lines, make a dry run – laying the tile out on the floor – **to help determine the best layout** and minimize the number of cut tiles. [Hallowell, 1988a, 49]

6. From time to time, check with a square and straightedge **to make sure the courses are straight.** [Hallowell, 1988a, 50]

7. The vertical lines. Measure **to find the midpoint of a wall** and mark it on the horizontal line. Starting at this midpoint, measure with the tile stick or set a row of loose tiles on the batten **to determine the size of the tiles at each end of the wall.** Don't forget the spacers. If the end tiles will be less than half a tile, move your midpoint mark one-half tile to avoid narrow pieces at each end. Then use the level and straightedge to mark a vertical line on the wall through the mark. Repeat the process with the other walls. [Hallowell, 1988a, 51]

8. The most pleasing effect is to center the tiles so that those at each end are of equal width. To do this, measure and mark the midpoint of the wall on the horizontal working line. Starting with the edge of one tile on the center mark, stand a row of loose tiles along the back of the tub **to determine the size of the end tiles.** Don't forget the spacers. If the end tiles are larger than half a tile, work from a centered vertical working line. With level and straightedge, mark the vertical working line on the backing. [Hallowell, 1988a, 54]

9. The illustration below shows typical positions of the vertical working line on an end wall. Choose the one that best suits your situation; then make a dry run with loose tiles **to check the layout for inconvenient cuts.** [Hallowell, 1988a, 55]

10. **To find the amount of tile you need,** first find the area of the floor by multiplying the overall length of the room by its width, both in feet. Deduct the area of any protrusions into the room, such as a kitchen counter or a bathtub. For a room with an odd shape, such an L-shaped room, divide the floor area into rectangles. Then measure each and add the areas together. Once you have found the area, add 5 percent so you will have extra tiles for cutting, waste, and later repairs. [Hallowell, 1988a, 65]

11. Resilient tile comes in boxes that contain enough to cover 45 sq. feet. **To find the number of boxes you need,** divide the overall floor area (including the 5 percent extra) by 45. [Hallowell, 1988a, 68]

12. Refer to the section on preparing the surface, on the facing page, **to determine if you'll need additional tools and materials for preliminary work.** [Hallowell, 1988a, 68]

13. Marking the working lines. After you have prepared the wall surface and removed the base molding, check the floor along the wall to see if it is level. If it isn't, find and mark the lowest spot. Then measure up to a point 1/4" less than the width of the base molding, as shown in the drawing below. From this new point, measure up the wall a distance equal to four tiles **to establish the height of your horizontal working line.** Establish a level line across the wall through this point. [Hallowell, 1988a, 72]

14. Concrete makes a good subfloor if it is dry. **To make sure,** place 1' squares of plastic in half a dozen spots around the floor, carefully taping down all edges. This is best done during wet months. After 24 hours take up the plastic. If there is condensation underneath, moisture is coming through the concrete and it is unsuitable for the parquet. [Hallowell, 1988a, 77]

15. Begin [it's continuation of previous example] by measuring **to find the exact center of two facing walls, A and B**. Snap a chalk line (AA) between the two center points. Find the center of walls C and D and again snap the chalk line (BB). **To check that these lines cross at a 90 degrees angle,** use the 3-4-5 method: From the center measure 3' down line AA and 4' down line BB. The diagonal distance between these two points will be exactly 5' if the lines are square. [Hallowell, 1988a, 77]

16. **To decide how much wallpaper you need**, first measure the wall or room with a steel tape. [Hallowell, 1988b, 37]

17. Measure the height and width of each wall (including openings); then multiply the two figures **to determine the total area of the wall in square feet**. Add the square footage of all the walls to be covered **to determine the total area.** [Hallowell, 1988b, 37]

18. **To figure the total number of single rolls you need,** divide the total square footage of wall space by 30 (25 for European rolls) square feet. If you're left with a fractional remainder of square feet, buy an additional roll. [Hallowell, 1988b, 37]

19. Wallpaper dealers stock adhesives for every type of installation. **To find an adhesive suitable for your material,** check the manufacturer's instructions, or ask your dealer. [Hallowell, 1988b, 38]

20. You may want to hang a coordinating border around the room at the top of the walls. **To determine the amount of border,** measure the width (in feet) of all walls to be covered and divide by three. Since borders are sold by the yard, this will give you the number of yards needed. [Hallowell, 1988b, 38]

21. To allow for pattern repeat, divide the height (in inches) of the wall by the number of inches between the pattern repeat; if you have a fractional remainder, round it off to the next highest number. For example, a 96-inch wall height divided by an 18-inch pattern repeat gives you 5.33 repeats, rounded off to 6.
    Then multiply the repeat measurement by the number of repeats you'll need **to determine the working height figure you must use.** In the case above, multiplying the 18-inch pattern repeat by 6 repeats tells you that you must calculate your wallpaper needs according to the requirements of the 108-inch wall, rather than the actual 96-inch wall. [Hallowell, 1988b, 38]

22. Before you cut, study the pattern **to determine how you want it to appear on the wall.** [Hallowell, 1988b, 43]

23. **To find the square footage of the ceiling,** multiply the length of the room by the width. [Hallowell, 1988b, 51]

A.2 Negative Imperatives

A.2.1 DONT imperatives

1. Soak garment for 3 minutes. Gently squeeze suds through. Rinse thoroughly in cool or cold water. Roll in towel to remove excess water. **Do not wring or twist.** Dry flat, away from sun or heat.
   `SOURCE:Woolite, cold water wash`

2. Pour Neet into palm of hand and spread thickly over hair to be removed. **Do not rub in.** ... If hair removes easily, rinse off Neet with wet cloth or in shower with lukewarm water. **Do not use soap.** ...
   Caution: **Do not use on irritated, inflamed or broken skin.** ... **Do not use Neet around the eyes, inside nose or ears.**
   `SOURCE:Neet, hair remover`

3. **Do not apply to broken or irritated skin.**
   `SOURCE:Shower to Shower, body powder`

4. Caution: **Do not puncture or incinerate container. Do not expose to heat or store at temperatures above 120 F.**
   `SOURCE:Old English, furniture Polish`

5. To use: Shake well. Turn can upside down and dispense into hand. Work or comb through towel-dried hair and style as desired. **Do not rinse.** Apply to roots for maximum lift and volume.
   `SOURCE:Finesse, Hair Mousse`

6. To use the Citronella 'Liquid Light' cartridge, remove the plastic wick cover. **Do not attempt to adjust wick.** Discard when empty. **Do not attempt to refill.**
   `SOURCE:Lamplight Farms' Citronella "Liquid Light" Insect Repellant.`

7. Remove any doors that open into the room by tapping out the hinge pins **(don't unscrew the hinges)** and cut with a fine-toothed saw. [Hallowell, 1988a, 42]

8. **Don't try to install tile over a springy surface.** [Hallowell, 1988a, 43]

9. Your sheet vinyl floor may be vinyl asbestos, which is no longer on the market. **Don't sand it or tear it up** because this will put dangerous asbestos fibers into the air. It must be covered over. [Hallowell, 1988a, 44]

10. Cutting tile. Shown below are the common methods of cutting tile. When using tile nippers, **don't try to cut the first time on the line;** cut short of it and then carefully nibble up. Too big a bite the first time usually results in chipping beyond the line. [Hallowell, 1988a, 47]

11. (caption of drawing) If backing is damaged, fill and smooth with patching plaster; **don't overfill.** When dry, paint with latex primer. [Hallowell, 1988a, 52]

12. For jack-on-jack bond, set the first tile on the batten with one side aligned exactly with the vertical line. Press the tile firmly into the adhesive. **Don't slide it;** this will push adhesive into the joints. [Hallowell, 1988a, 53]

13. (drawing caption) Wall tiles should be set in step, or pyramid, pattern. Place each with a slight twist; **do not slide**. Use 6-penny finishing nails to space tiles with no lugs. [Hallowell, 1988a, 53]

14. The positions of any flush-mounted or recessed accessories, such as soap dishes and towel bars, should be marked on the wall before the adhesive is applied. **Don't cover the marks with adhesive.** [Hallowell, 1988a, 55]

15. The positions of any flush-mounted or recessed accessories, such as soap dishes and towel bars, should be marked on the wall before the adhesive is applied. **Don't cover the marks with adhesive.** [Hallowell, 1988a, 55]

16. Setting the first tile. If you nailed a batten support to the wall, set the first tile at the intersection of the vertical guideline and the batten. Set the tile firmly but **don't slide it.** [Hallowell, 1988a, 56]

17. After washing the tile, rinse it thoroughly to remove detergent film; then wipe with a soft, dry cloth. For stubborn dirt, scrub tiles with a white, cleansing powder. **Don't use a cleaner containing bleach;** this can pull the color out of the colored grouts. [Hallowell, 1988a, 64]

18. This material [vinyl-asbestos] is no longet on the market. If you think it might be in your house, **do not tear it up or sand it.** Simply cover with another flooring material. [Hallowell, 1988a, 67]

19. (All the following in boldface). Caution: **Do not sand or tear up the floor** if it is vinyl-asbestos. This material contains fibers that can damage your lungs if inhaled. If in doubt, just cover the floor with 1/4" plywood. [Hallowell, 1988a, 69]

20. To cut tiles, score along the mark with a utility knife and snap the tile along the line. For intricate cuts, use a pair of heavy scissors. The tiles will cut more easily in this case if warmed in sunlight or over a furnace vent. **Don't overheat** or the tiles may scorch or melt. [Hallowell, 1988a, 71]

21. You can put parquet down over a wide variety of surfaces, whether old or new, if they are firm, clean, smooth, and dry. **Don't put parquet down on a surface that is below ground level because of the moisture problem.** [Hallowell, 1988a, 76]

22. Resilient tile surface. Parquet can be put over linoleum or vinyl tile floors if they are smooth and in good shape. If the vinyl tile is several years old, it may be vinyl-asbestos. In this case **do not disturb it** because the asbestos particle are a health hazard. [Hallowell, 1988a, 77]

23. Placing the parquet. ... **Do not slide except for the small amount necessary with tongue-and-groove pieces.** [Hallowell, 1988a, 77]

24. (caption of a drawing) Spread adhesive with notched trowel. **Do not cover working lines.** [Hallowell, 1988a, 78]

25. Spread adhesive with notched trowel. **Do not cover working lines.** [Hallowell, 1988a, 78]

26. Let the floor dry overnight and then sand the floor. Practice on some old plywood first if necessary; **don't let the sander remain in one place** or you'll damage the floor. [Hallowell, 1988a, 79]

27. If you must replace a tile, first cut around the edges with a circular saw. Set the blade to the depth of the tile and **don't damage adjoining tiles.** [Hallowell, 1988a, 79]

28. Dust-mop or vacuum your parquet floor as you would carpeting. **Do not scrub or wet-mop the parquet.** [Hallowell, 1988a, 79]

29. If you're using a lightweight or porous wallpaper, profit from these "don't":

    (a) **Don't hang the wallpaper over dark painted walls;** it could show through. Lighten the walls with a flat, oil-base enamel undercoat before hanging the wallpaper.

    (b) **Don't paper over a dark colored or patterned wall covering.** First, lighten the background with paint or lining paper.

    (c) **Don't hang the wallpaper over an existing wallpaper** if the ink from the existing covering comes off; it could bleed through the new covering. To test, moisten a small piece of the covering with a clean sponge. If the ink comes off, seal the old covering first with an undercoat that seals stains.

    [Hallowell, 1988b, 40]

30. Cut a strip 1/2 inch wider than the widest measurement (**don't disregard the leftover piece of wallpaper**; you'll use it to cover the adjacent side of the corner.) [Hallowell, 1988b, 46]

31. Use a vinyl-to-vinyl paste for any type of border you plan to hang over wallpaper. To paste the border for hanging, cover the entire back with paste and book the strip; **don't crease the folds.** [Hallowell, 1988b, 52]

32. Step 4. Lift brush straight up, letting excess paint drip back into pail. Gently slap both sides of brush against inside of pail two or three times. **Don't wipe brush across lip of pail**, or bristles may separate into clumps, leaving less paint on brush. [Hallowell, 1988b, 40]

Never

1. Place the first tile with a gentle rocking motion into the corner formed by the two battens. **Never slide tile on the adhesive;** it pushes the adhesive up between the joints. [Hallowell, 1988a, 49]

2. **Never mix cleaners containing acid or ammonia with chlorine bleach.** The chemical reaction releases the chlorine as a poisonous gas. [Hallowell, 1988a, 64]

3. **Never slide tile into position** or the adhesive will come up through the joints. [Hallowell, 1988a, 71]

4. If you haven't used a floor sander, practice on a full sheet of old plywood first because it takes a little practice to learn how to control one. **Never let the sander remain in one spot while in use,** or it will immediately sand a depression in your floor. [Hallowell, 1988a, 76]

A.2.2 neg-TC imperatives

Take care

1. Snip at all the triangular marks on the pattern to ease the seam, **taking care not to cut the stitching.**
   `Source: opportunistic collection.`

2. Attach washboard to the base assembly with finishing nails driven carefully through back sides of washboard. **Take care not to split wood frame**, drill very small pilot holes for the nails or blunt nail tips to avoid splitting the wood. [CWP, 1989]

3. To book the strip, fold the bottom third or more of the strip over the middle of the panel, pasted sides together, **taking care not to crease the wallpaper sharply at the fold.** [Hallowell, 1988b, 44]

4. To hang the border, begin at the least conspicuous corner. The work will go much faster if you have someone hold the folded section while you apply the border to the wall. **Take care not to drip paste onto the wall** and be sure to remove excess paste. [Hallowell, 1988b, 52]

Be sure, make sure

1. If the subflooring is accessible from underneath, as it is from the unfinished ceiling of a basement, it is best to work from the underside. In this case, drill a pilot hole at the squeaky spot. **Be sure not to drill it deeper than 1 1/4 inches** or you will penetrate the top surface. [CWP, 1989]

2. Border fabric can be pasted on top of fabric walls after the wall is completely dry. Use the same procedure as for pasting walls, but **make sure not to drip paste onto the fabric.** [Hallowell, 1988b, 63]

3. To wash, load clothes into tub, **making sure not to overfill it**.
   `Source: opportunistic collection.`

Be careful

1. Center and mark the placement of the two crossbraces on the table's bottom surface. Drill pilot holes to secure the crossbraces; **be careful not to penetrate the table top surface.** [FH, 1990]

2. Cut under eaves of cabin with chisel, **being careful not to chip roof**. [HC, 1988, 81]

3. Make a small 1/4-inch slit (cut on the bias to avoid ravels) in the center of each area to be stuffed. **Be careful not to snip the surface fabric.** [HC, 1988, 83]

4. When glue is dry finish assembly with finishing nails. **Be careful not to split wood.** [CWP, 1989]

5. To make a piercing cut, first drill a hole in the waste stock on the interior of the pattern. The diameter of the hole must be larger than the width of the blade. If you want to save the waste stock for later use, drill the hole near a corner in the pattern. **Be careful not to drill through the pattern line.** [CWP, 1989]

6. If the damage has occurred in a smaller area, you can cut out that portion of the wall with a compass or saber saw, **being careful not to pierce the inside wall**, and make a patch out of wallboard. [CWP, 1989]

7. Whichever pattern you choose, begin each new row from the same side of the room as the first one. Measure and cut the last tile as required. After you have laid a section, the tiles should be bedded in. Do this with a rubber mallet or by hammering lightly on a block of padded wood large enough to cover several tiles at once. This process sets the tile firmly in the adhesive and levels each with the others. In moving the padded block about, **be careful not to push a tile out of line.** [Hallowell, 1988a, 50]

8. Punch hole through center of damaged tile with hammer and nail set or large nail. **Be careful not to damage the surface behind**. [Hallowell, 1988a, 52]

9. If your plans call for replacing the wood base molding with vinyl cove molding, **be careful not to damage the walls as you remove the wood base.**[Hallowell, 1988a, 68]

10. To remove resilient sheet flooring, cut it into strips about 6" wide with a utility knife or linoleum knife, **being careful not to damage the subfloor.** [Hallowell, 1988a, 69]

11. Following application instructions on the label, begin spreading the adhesive near where the working lines intersect. Spread the adhesive up, but not over, the working lines that will guide your tile placement. **Be careful not to cover too large an area at one time**, or the adhesive may set up before you can cover it; start with an area about 3' square. [Hallowell, 1988a, 71]

12. To replace a damaged tile, first warm it and the adhesive below with a propane torch or an old iron. Pry up the tile using a putty knife or cold chisel. Remove any excess backing or adhesive from the floor until the surface is smooth and deep enough to install the new tile. Next, apply the proper adhesive, keeping it back about 1/4" from the perimeter. **Be careful not to get any on the surrounding tiles.** Weights along the edges will ensure that the tile does not pop up before the adhesive sets. [Hallowell, 1988a, 73]

13. Spread the adhesive with notched trowel held at about a 30 degrees angle. Work the adhesive from several directions to spread it evenly. If you can't see the working lines through the adhesive, **be careful not to cover them.** [Hallowell, 1988a, 77]

14. In most situations, the butt seam is the best way to join two strips of wallpaper, since it's the least noticeable seam. To make one, tightly butt the edge of the strip you're hanging to the edge of the previously hung strip. **Be very careful not to stretch the wallpaper.** [Hallowell, 1988b, 45]

15. To combat mildew and to provide strong adhesion, use a vinyl adhesive, never a wheat paste. Apply paste to the wall (unless the wallpaper manufacturer specifies differently), **being careful not to overpaste.** [Hallowell, 1988b, 50]

16. With your hands, lightly smooth the fabric into place. Work from the center out to the sides, **being careful not to stretch the fabric.** [Hallowell, 1988b, 63]

17. You can lay the door across sawhorses. Working on a horizontal surface allows you to apply a good coat of paint without worrying about the paint collecting in the lower corners of the panels and eventually dripping down. But, **you do have to be careful not to apply too thick a coat** or the paint may "puddle". [Hallowell, 1988b, 76]

18. A second way to remove molding is to locate the nails and, using a nailset, drive each one all the way through the molding. After the molding is lifted off, pull out the nails. For hardwood moldings, this method is the best, but **be careful not to crack the molding.** [Hallowell, 1988b, 86]

19. Remove the block and press the panel firmly into place. To force the adhesive into tight contact, knock on the panel with a rubber mallet or hammer against a padded block. **Be careful not to mar the surface.** [Hallowell, 1988b, 92]

20. When nailing the panels, **be careful not to mar the surfaces.** [Hallowell, 1988b, 92]

Appendix B

Paraphrase Experiment

I report here the data of the experiment I ran to collect judgements on paraphrases of purpose clauses. I remind the reader that I ran an informal experiment in which I asked 11 native speakers to judge the pragmatic felicity of various possible paraphrases on 10 of my examples; the judgements were on the scale [0-3], with 0 totally unacceptable, and 3 totally felicitous. 8 informants answered, although one of them didn't answer all the questions. For each example, I list the tested paraphrases, and for each paraphrase I give a list of four pairs; the first element in each pair is the judgement, the second is how many informants gave that judgement.

Final Purpose Clause

1. Place a plank between two ladders to create a simple scaffold.
   (a) Place a plank between two ladders so as to create a simple scaffold.
   [0-], [1-1], [2-2], [3-5]
   (b) Place a plank between two ladders in order to create a simple scaffold.
   [0-], [1-], [2-2], [3-6]
   (c) By placing a plank between two ladders create a simple scaffold.
   [0-], [1-3], [2-3], [3-2]
   (d) Create a simple scaffold by placing a plank between two ladders.
   [0-], [1-], [2-2], [3-6]
2. Line the tub with cardboard to prevent debris from clogging the drain.
   (a) Line the tub with cardboard so as to prevent debris from clogging the drain.
   [0-], [1-], [2-2], [3-6]
   (b) Line the tub with cardboard in order to prevent debris from clogging the drain.
   [0-], [1-], [2-], [3-8]

(c) By lining the tub with cardboard prevent debris from clogging the drain.
[0-], [1-8], [2-], [3-]

(d) Prevent debris from clogging the drain by lining the tub with cardboard.
[0-], [1-1], [2-3], [3-4]

3. Slip paper towels between iron and fabric to catch stray wisps of the heated fusible webbing.

(a) Slip paper towels between iron and fabric so as to catch stray wisps of the heated fusible webbing.
[0-], [1-], [2-], [3-8]

(b) Slip paper towels between iron and fabric in order to catch stray wisps of the heated fusible webbing.
[0-], [1-1], [2-1], [3-6]
`NB: one 2.5, counted as 2`

(c) By slipping paper towels between iron and fabric catch stray wisps of the heated fusible webbing.
[0-1], [1-6], [2-], [3-]

(d) Catch stray wisps of the heated fusible webbing by slipping paper towels between iron and fabric.
[0-], [1-2], [2-4], [3-2]

4. Go into the kitchen to get me the coffee urn.[1]

(a) Go into the kitchen so as to get me the coffee urn.
[0-3], [1-3], [2-2], [3-]

(b) Go into the kitchen in order to get me the coffee urn.
[0-1], [1-3], [2-1], [3-3]

(c) By going into the kitchen get me the coffee urn.
[0-5], [1-3], [2-], [3-]

(d) Get me the coffee urn by going into the kitchen.
[0-4], [1-2], [2-2], [3-]

5. Work to finish half a room at a time.

(a) Work so as to finish half a room at a time.
[0-1], [1-2], [2-1], [3-4]

(b) Work in order to finish half a room at a time.
[0-3], [1-4], [2-1], [3-]

(c) By working finish half a room at a time.
[0-6], [1-2], [2-], [3-]

[1] I remind the reader that this example is constructed.

(d) Finish half a room at a time by working.
[0-6], [1-2], [2-], [3-]

6. Place on rack to cool.[2]

(a) Place on rack so as to cool.
[0-1], [1-3], [2-2], [3-2]

(b) Place on rack in order to cool.
[0-], [1-], [2-3], [3-5]
`NB: one 2.4 counted as 2`

(c) By placing on rack cool.
[0-5], [1-3], [2-], [3-]

(d) Cool by placing on rack.
[0-], [1-], [2-3], [3-5]

Initial PC

1. To determine the amount of border, measure the width of all walls to be covered and divide by three.

(a) So as to determine the amount of border, measure the width of all walls to be covered and divide by three.
[0-1], [1-5], [2-1], [3-1]

(b) In order to determine the amount of border, measure the width of all walls to be covered and divide by three.
[0-], [1-], [2-], [3-8]

(c) Determine the amount of border by measuring the width of all walls to be covered and dividing by three.
[0-], [1-], [2-2], [3-6]
`NB: one 2.5, counted as 2`

2. To finish the dolls with an antiqued look, lightly brush on a thin coat of burnt umber oil-base paint.

(a) So as to finish the dolls with an antiqued look, lightly brush on a thin coat of burnt umber oil-base paint.
[0-], [1-3], [2-3], [3-1]

(b) In order to finish the dolls with an antiqued look, lightly brush on a thin coat of burnt umber oil-base paint.
[0-], [1-], [2-3], [3-4]
`NB: one 2.5 counted as 2`

[2]This example is a variation of one in [Thompson, 1985].

(c) Finish the dolls with an antiqued look by lightly brushing on a thin coat of burnt umber oil-base paint.
[0-], [1-1], [2-1], [3-5]

3. To install a grab bar in a tub or shower enclosure, locate and mark the center of the appropriate wall studs with pencil marks on the ceiling before you tile. After the tile is up, locate the studs by following down the wall from the ceiling marks with a level. Screw the bars directly to the studs through holes drilled in the tile.

(a) So as to install a grab bar in a tub or shower enclosure, locate and mark the center of the appropriate wall studs with pencil marks on the ceiling before you tile. After the tile is up, locate the studs by following down the wall from the ceiling marks with a level. Screw the bars directly to the studs through holes drilled in the tile.
[0-2], [1-4], [2-], [3-1]

(b) In order to install a grab bar in a tub or shower enclosure, locate and mark the center of the appropriate wall studs with pencil marks on the ceiling before you tile. After the tile is up, locate the studs by following down the wall from the ceiling marks with a level. Screw the bars directly to the studs through holes drilled in the tile.
[0-], [1-], [2-2], [3-5]

(c) Install a grab bar in a tub or shower enclosure by locating and marking the center of the appropriate wall studs with pencil marks on the ceiling before you tile. After the tile is up, locate the studs by following down the wall from the ceiling marks with a level. Screw the bars directly to the studs through holes drilled in the tile.
[0-1], [1-3], [2-2], [3-1]

(d) [3] Install a grab bar in a tub or shower enclosure by locating and marking the center of the appropriate wall studs with pencil marks on the ceiling before you tile; after the tile is up, by locating the studs by following down the wall from the ceiling marks with a level; and by screwing the bars directly to the studs through holes drilled in the tile.
[0-2], [1-2], [2-3], [3-]

4. To hang the wallpaper, follow the steps on page 47.

(a) So as to hang the wallpaper, follow the steps on page 47.
[0-2], [1-5], [2-], [3-]

(b) In order to hang the wallpaper, follow the steps on page 47.
[0-], [1-1], [2-2], [3-4]

(c) Hang the wallpaper by following the steps on page 47.
[0-], [1-1], [2-], [3-6]

[3] This fourth paraphrase is due to the fact that, with a pattern such as *To do* β*, do* α_1*; do* α_2*; ... do* α_n, two paraphrases are possible: *Do* β *by doing* α_1*; do* α_2*, ... , do* α_n and *Do* β *by doing* α_1*,* α_2*, ... ,* α_n.

Appendix C

Knowledge Representation

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; The ROLES --- the flag "t" indicates  attributes

(cl-define-role 'achieved-role t)
(cl-define-role 'agent t)
(cl-define-role 'along-role t)
(cl-define-role 'at-role t)
(cl-define-role 'basic t)
(cl-define-role 'caused-role t)
(cl-define-role 'destination t)
(cl-define-role 'direction t)
(cl-define-role 'experiencer t)
(cl-define-role 'in-role t)
(cl-define-role 'instrument)
(cl-define-role 'lex-item t)
(cl-define-role 'location t)
(cl-define-role 'manner)
(cl-define-role 'means t)
(cl-define-role 'patient t)
(cl-define-role 'path-role t)
(cl-define-role 'polarity t)
(cl-define-role 'semfield-role t)
(cl-define-role 'source t)
(cl-define-role 'water-availability t)
(cl-define-role 'with-role t)


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
; The OBJECT hierarchy


(cl-define-concept 'semfield
                   '(one-of spatial control ident contact composition))

(cl-define-concept 'entity
                   '(disj-prim classic-thing type entity))

(cl-define-concept 'animate
                   '(disj-prim entity life animate))

(cl-define-concept 'inanimate
                   '(disj-prim entity life inanimate))

(cl-define-concept 'geometric-concept
                   '(disj-prim inanimate type geometric-concept))

; Clearly there are necessary and sufficient definitions for polygons,
; squares, diagonals etc. However, as they involve notions like "all
; sides have the same length", for simplicity I defined all these
; concepts as disjoint primitives.

(cl-define-concept 'geometric-figure
                   '(disj-prim  geometric-concept type geometric-figure))

(cl-define-concept 'diagonal
                   '(disj-prim  geometric-concept type diagonal))

(cl-define-concept 'perp-axis
                   '(disj-prim  geometric-concept type perp-axis))


(cl-define-concept 'polygon
                   '(disj-prim geometric-figure type polygon))

(cl-define-concept 'rectangle
                   '(disj-prim polygon type rectangle))

(cl-define-concept 'square
                   '(disj-prim polygon type square))

(cl-define-concept 'triangle
                   '(disj-prim polygon type triangle))
```

```
(cl-define-concept 'screw
                   '(disj-prim inanimate type screw))

(cl-define-concept 'event
                   '(disj-prim classic-thing type event))

(cl-define-concept 'state
                   '(disj-prim classic-thing type state))

(cl-define-concept 'property
                   '(and (disj-prim classic-thing type property)
                         (one-of loose tight clean dirty)))

; The PLACE subhierarchy

(cl-define-concept 'place
                   '(and (disj-prim classic-thing type place)
                         (all semfield-role semfield)))

(cl-define-concept 'spatial-place
                   '(and place (fills semfield-role spatial)))

(cl-define-concept 'control-place
                   '(and place (fills semfield-role control)))

(cl-define-concept 'comp-place
                   '(and place (fills semfield-role composition)))

(cl-define-concept 'ident-place
                    '(and place (fills semfield-role ident)))

(cl-define-concept 'at-sp-place
                   '(and spatial-place
                         (all at-role  entity)))

(cl-define-concept 'washing-site
                   '(and at-sp-place
                         (fills water-availability yes)))

(cl-define-concept 'in-sp-place
                   '(and spatial-place
                         (all in-role  entity)))

(cl-define-concept 'along-sp-place
                   '(and spatial-place
```

```
                    (all along-role  entity)))

(cl-define-concept 'along-diagonal
                   '(and along-sp-place
                         (all along-role  diagonal)))

(cl-define-concept 'along-perp-axis
                   '(and along-sp-place
                         (all along-role  perp-axis)))

(cl-define-concept 'at-control-place
                   '(and control-place
                         (all at-role  animate)))

(cl-define-concept 'at-comp-place
                   '(and comp-place
                         (all at-role  entity)))

(cl-define-concept 'at-comp-triangle
                   '(and comp-place
                         (all at-role  triangle)))

(cl-define-concept 'at-comp-rect
                   '(and comp-place
                         (all at-role  rectangle)))

(cl-define-concept 'at-comp-square
                   '(and comp-place
                         (all at-role  square)))

(cl-define-concept 'at-ident-place
                   '(and ident-place
                         (all at-role  property)))

(cl-define-concept 'at-ident-clean
                   '(and ident-place
                         (fills at-role clean)))

(cl-define-concept 'at-ident-dirty
                   '(and ident-place
                         (fills at-role  dirty)))

(cl-define-concept 'at-ident-loose
                   '(and ident-place
                         (fills at-role loose)))
```

```
(cl-define-concept 'at-ident-tight
                   '(and ident-place
                         (fills at-role  tight)))


; The PATH subhierarchy

(cl-define-concept 'path
                   '(and (disj-prim classic-thing type path)
                         (all semfield-role semfield)))

(cl-define-concept 'spatial-path
                   '(and path (fills semfield-role spatial)))

(cl-define-concept 'control-path
                   '(and path (fills semfield-role control)))

(cl-define-concept 'comp-path
                    '(and path (fills semfield-role composition)))

(cl-define-concept 'ident-path
                    '(and path (fills semfield-role ident)
                               (all source ident-place)
                               (all destination ident-place)))

(cl-define-concept 'tl-ident-path
                    '(and ident-path
                          (fills source at-ident-tight)
                          (fills destination at-ident-loose)))

(cl-define-concept 'lt-ident-path
                    '(and ident-path
                          (fills source at-ident-loose)
                          (fills destination at-ident-tight)))

(cl-define-concept 'clean-ident-path
                    '(and ident-path
                          (fills source at-ident-dirty)
                          (fills destination at-ident-clean)))

(cl-define-concept 'from-path-sp
                   '(and spatial-path
                         (all source spatial-place)))
```

```
(cl-define-concept 'to-path-sp
                   '(and spatial-path
                         (all destination spatial-place)))

(cl-define-concept 'to-at-path-sp
                   '(and to-path-sp
                          (all destination at-sp-place)))

(cl-define-concept 'to-path-control
                   '(and control-path
                          (all destination control-place)))

(cl-define-concept 'to-at-path-control
                   '(and to-path-control
                          (all destination at-control-place)))

(cl-define-concept 'from-to-path-sp
                  '(and from-path-sp to-path-sp))

(cl-define-concept 'circular-path
                   '(and path (same-as source destination)))

(cl-define-concept 'circular-path-sp
                  '(and circular-path from-to-path-sp))

(cl-define-concept 'from-path-comp
                  '(and comp-path
                        (all source comp-place)))

(cl-define-concept 'to-path-comp
                   '(and comp-path
                         (all destination comp-place)))

(cl-define-concept 'to-path-comp-inhalf
                   '(and to-path-comp
                         (fills destination in-half)))

(cl-define-concept 'from-to-path-comp
                  '(and from-path-comp to-path-comp))

; The STATE subhierarchy

; Notice: the name "experiencer" is a bit misleading to  indicate
; the role filled by the entity which is "in" that state.
```

```
(cl-define-concept 'be
                   '(and state
                         (all semfield-role semfield)
                         (all experiencer entity)
                         (all location place)
                         (same-as semfield-role
                                 (location semfield-role))))

(cl-define-concept 'be-spatial
                   '(and be
                         (fills semfield-role spatial)
                         (all location spatial-place)))

(cl-define-concept 'be-sp-washing-site
                   '(and be-spatial
                         (all location washing-site)))

(cl-define-concept 'be-control
                   '(and be
                         (fills semfield-role control)
                         (all location at-control-place)))

; The following definitions are required to approximate  Jackendoff's
; notion of verbs of material composition: the polarity is used to
; distinguish the cases in which the Theme is the whole and the parts
; the reference objects ("Sam assembled the house out of bricks" /
; "Sam broke the bowl into one thousand pieces"), from the reverse
; case ("Sam assembled the bricks into a house"). These definitions
; will be used in the recipe for "create triangles from cutting
; square". The same distinctions apply to  "go-comp"  below.


(cl-define-concept 'be-comp
                   '(and be
                         (fills semfield-role composition)
                         (all location at-comp-place)
                         (all polarity (one-of  yes no))))

(cl-define-concept 'be-comp-plus
                   '(and be-comp
                         (all location at-comp-place)
                         (fills polarity yes )))

(cl-define-concept 'be-comp-minus
```

```
                  '(and be-comp
                        (all location at-comp-place)
                        (fills polarity no)))

(cl-define-concept 'be-comp-sq-tr
                   '(and be-comp-plus
                         (all experiencer  square)
                         (all location at-comp-triangle)))

(cl-define-concept 'be-comp-sq-rect
                   '(and be-comp-plus
                         (all experiencer  square)
                         (all location at-comp-rect)))

(cl-define-concept 'be-ident
                   '(and be
                         (fills semfield-role ident)
                         (all location at-ident-place)))

(cl-define-concept 'be-ident-dirty
                   '(and be
                         (fills semfield-role ident)
                         (all location at-ident-dirty)))

(cl-define-concept 'be-ident-clean
                   '(and be
                         (fills semfield-role ident)
                         (all location at-ident-clean)))

(cl-define-concept 'be-ident-loose
                   '(and be
                         (fills semfield-role ident)
                         (all location at-ident-loose)))

(cl-define-concept 'be-ident-tight
                   '(and be
                         (fills semfield-role ident)
                         (all location at-ident-tight)))
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                               ;
;        The ACTION subhierarchy                ;
;                                               ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(cl-define-concept 'act-type
                   '(and event  (all experiencer entity)))

(cl-define-concept 'cause
                  '(and event
                              (all agent animate)
                              (all caused-role act-type)))

(cl-define-concept 'intentional-act
                   '(and act-type
                         (all experiencer animate)))

(cl-define-concept 'achieve-act
                   '(and act-type
                         (all achieved-role state)))

(cl-define-concept 'achieve-spatial
                   '(and achieve-act
                         (all achieved-role be-spatial)))

(cl-define-concept 'achieve-sp-wash
                   '(and achieve-act
                         (all achieved-role be-sp-washing-site)))

(cl-define-concept 'achieve-ident
                   '(and achieve-act
                         (all achieved-role be-ident)))

(cl-define-concept 'act-on
                   '(and act-type
                         (all patient entity)))

(cl-define-concept 'go
                   '(and act-type
                         (all path-role path)
                         (all semfield-role semfield)
                         (same-as semfield-role
                                  (path-role semfield-role))))
```

```
(cl-define-concept 'go-spatial
                   '(and go (fills semfield-role spatial)))

(cl-define-concept 'go-control
                   '(and go (fills semfield-role control)))

(cl-define-concept 'go-control-to-at
                   '(and go-control
                         (all path-role  to-at-path-control)))

(cl-define-concept 'go-ident
                   '(and go (fills semfield-role ident)
                            (all path-role ident-path)))

(cl-define-concept 'go-ident-loose
                   '(and go (fills path-role tl-ident-path)))

(cl-define-concept 'go-ident-tight
                   '(and go (fills path-role lt-ident-path)))

(cl-define-concept 'go-ident-clean
                   '(and go (fills path-role clean-ident-path)))

(cl-define-concept 'go-contact
                '(and go (fills semfield-role contact)))

(cl-define-concept 'go-comp
                   '(and go (fills semfield-role composition)
                            (all path-role comp-path)))

(cl-define-concept 'go-comp-plus
                   '(and go-comp (fills polarity yes)))

(cl-define-concept 'go-comp-minus
                   '(and go-comp (fills polarity no)))

(cl-define-concept 'go-comp-inhalf
                    '(and go-comp-plus
                          (all path-role to-path-comp-inhalf)))

(cl-define-concept 'go-spatial-with
                   '(and go-spatial
                         (all with-role inanimate)))
```

```
(cl-define-concept 'go-sp-from-to
                   '(and go-spatial
                         (all path-role from-to-path-sp)))

(cl-define-concept 'go-sp-circ
                   '(and go-spatial
                         (all path-role circular-path-sp)))

(cl-define-concept 'cause-and-acton
                   '(and cause act-on
                         (same-as experiencer agent)
                         (same-as patient (caused-role experiencer))))

(cl-define-concept 'cause-of-acton
                   '(and cause-and-acton
                         (all caused-role act-on)))

(cl-define-concept 'move-sth-swh
                   '(and cause-and-acton
                         (all caused-role go-sp-from-to)))

(cl-define-concept 'get-control
                    '(and cause-and-acton
                          (all caused-role go-control-to-at)
                          (same-as patient (caused-role experiencer))
                          (same-as experiencer
                                   (caused-role path-role
                                        destination at-role))))

(cl-define-concept 'reduce-to-pieces
                   '(and cause-and-acton
                         (all caused-role go-comp-plus)
                         (all lex-item (one-of cut break))))

(cl-define-concept 'cut-location
                   '(and reduce-to-pieces
                         (fills lex-item cut)
                         (all location spatial-place)))

(cl-define-concept 'cut-sq-inhalf-alongdiag
                   '(and cut-location
                         (all patient square)
                         (all caused-role go-comp-inhalf)
                         (all location along-diagonal)))
```

```
(cl-define-concept 'cut-sq-inhalf-alongaxis
                   '(and cut-location
                         (all patient square)
                         (all caused-role go-comp-inhalf)
                         (all location along-perp-axis)))


; "create" inherits "patient" from act-on. Patient is not
; the best name for the object which is being created.
; There are two types of "create" because  both "create whole
; from parts" or "create parts from whole" are possible.
; The latter one applies in the "cut square" case.

(cl-define-concept 'create-act
                   '(and cause-and-acton
                         (all caused-role go-comp)
                         (all lex-item (one-of create form))))

(cl-define-concept 'create-whole-from-parts
                   '(and create-act
                         (all caused-role go-comp-plus)))

(cl-define-concept 'create-parts-from-whole
                   '(and create-act
                         (all caused-role go-comp-minus)))

(cl-define-concept 'turn
                   '(and cause-and-acton
                         (all caused-role go-sp-circ)))

(cl-define-concept 'turn-direction
                   '(and turn
                         (all direction
                              (one-of clockwise counterclockwise))))

(cl-define-concept 'turn-clockwise
                   '(and turn (fills direction clockwise)))

(cl-define-concept 'turn-counterclockwise
                   '(and turn (fills direction counterclockwise)))

(cl-define-concept 'cause-state-change
                    '(and cause-and-acton
                          (all caused-role go-ident)
                          (same-as patient
```

```
                                (caused-role experiencer))
                          (all lex-item
                               (one-of loosen tighten wash))))

(cl-define-concept 'loosen-screw
                   '(and cause-state-change
                         (all caused-role go-ident-loose)
                         (all patient screw)
                         (fills lex-item loosen)))

(cl-define-concept 'tighten-screw
                   '(and cause-state-change
                         (all patient screw)
                         (all caused-role go-ident-tight)
                         (fills lex-item tighten)))

(cl-define-concept 'wash
                   '(and cause-state-change
                         (all caused-role go-ident-clean)
                         (fills lex-item wash)))

(cl-define-concept 'physical-wash
                   '(and wash
                         (fills basic yes)
                         (all location washing-site)))
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                        ;
;                      ACTION LIBRARY                                    ;
;                                                                        ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; ROLES

(cl-define-role 'header t)
(cl-define-role 'substep1 t)
(cl-define-role 'substep2 t)
(cl-define-role 'substep3 t)
(cl-define-role 'ann-relation t)
(cl-define-role 'ann-arg1 t)
(cl-define-role 'ann-arg2 t)
(cl-define-role 'annot1 t)
(cl-define-role 'annot2 t)
(cl-define-role 'annot3 t)
(cl-define-role 'qualif1 t)
(cl-define-role 'effect1 t)



; AUXILIARY CONCEPTS

(cl-define-concept 'poss-relations
                   '(one-of enables precedes during))

(cl-define-concept 'annotation-type
                   '(and classic-thing
                         (all ann-relation poss-relations)
                         (all ann-arg1 act-type)
                         (all ann-arg2 act-type)))

(cl-define-concept 'enab-annot
                   '(and annotation-type
                         (fills ann-relation enables)))

(cl-define-concept 'gsp-enab-getc
                   '(and enab-annot
                         (all ann-arg1 go-spatial)
                         (all ann-arg2 get-control)))
```

```
(cl-define-concept 'getc-enab-gospw
                   '(and enab-annot
                         (all ann-arg1 get-control)
                         (all ann-arg2 go-spatial-with)))

(cl-define-concept 'achsp-enab-physwash
                   '(and enab-annot
                         (all ann-arg1 achieve-spatial)
                         (all ann-arg2 physical-wash)))


; RECIPES

(cl-define-concept 'recipe
                    '(and classic-thing
                                   (all header
                                        intentional-act)))

(cl-define-concept 'recipe-1-step
                    '(and recipe
                       (all substep1 intentional-act)))


(cl-define-concept 'recipe-3-steps
                    '(and recipe
                       (all substep1 intentional-act)
                       (all substep2 intentional-act)
                       (all substep3 intentional-act)))
```

```
(cl-define-concept 'move-recipe
     '(and recipe-3-steps
           (all header move-sth-swh)
           (all substep1 go-spatial)
           (all substep2 get-control)
           (all substep3 go-spatial-with)
           (all annot1 gsp-enab-getc)
           (all annot2 getc-enab-gospw)
           (all qualif1 be-spatial)
           (all effect1 be-spatial)
           (same-as (header agent)(substep1 agent))
           (same-as (header agent)(substep2 agent))
           (same-as (header agent)(substep3 agent))
           (same-as (header patient)(substep2 patient))
           (same-as (header patient)(substep3 with-role))
           (same-as (header caused-role path-role source)
                    (substep1 path-role destination))
           (same-as (header caused-role path-role destination)
                    (substep3 path-role destination))
           (same-as substep1 (annot1 ann-arg1))
           (same-as substep2 (annot1 ann-arg2))
           (same-as substep2 (annot2 ann-arg1))
           (same-as substep3 (annot2 ann-arg2))
           (same-as (qualif1 experiencer)
                    (header patient))
           (same-as (effect1 experiencer)
                    (header patient))
           (same-as (qualif1 location)
                    (substep1 path-role destination))
           (same-as (effect1 location)
                     (substep3 path-role destination))
      )
)
```

```
(cl-define-concept 'wash-recipe
     '(and recipe-3-steps
           (all header wash)
           (all substep1 achieve-spatial)
           (all substep2 achieve-spatial)
           (all substep3 physical-wash)
           (all annot1 achsp-enab-physwash)
           (all annot2 achsp-enab-physwash)
           (all qualif1 be-ident-dirty)
           (all effect1 be-ident-clean)
           (same-as (header agent)(substep1 agent))
           (same-as (header agent)(substep2 agent))
           (same-as (header agent)(substep3 agent))
           (same-as (header agent)
                    (substep1 achieved-role experiencer))
           (same-as (header patient)
                    (substep2 achieved-role experiencer))
           (same-as (header patient)(substep3 patient))
           (same-as (substep1 achieved-role location)
                    (substep2 achieved-role location))
           (same-as (substep1 achieved-role location)
                    (substep3 location))
           (same-as substep1 (annot1 ann-arg1))
           (same-as substep3 (annot1 ann-arg2))
           (same-as substep2 (annot2 ann-arg1))
           (same-as substep3 (annot2 ann-arg2))
           (same-as (qualif1 experiencer)
                    (header patient))
           (same-as (effect1 experiencer)
                    (header patient))
      )
)
```

```
; The following two recipes have an obvious flaw, namely, they lack any
; reference to the cardinality of the triangle / rectangle set.


(cl-define-concept 'create-two-triangles
     '(and recipe-1-step
           (all header create-parts-from-whole)
           (all substep1 cut-sq-inhalf-alongdiag)
           (all effect1 be-comp-sq-tr)
           (same-as (header agent) (substep1 agent))
           (same-as (header patient)
                    (effect1 location at-role))
           (same-as (header caused-role path-role destination)
                    (effect1 location))
           (same-as (substep1 patient)
                    (effect1 experiencer))))

; The following recipe may be optionally loaded in or not:
; accordingly, the examples "cut the square in half to create two
; polygons", "cut the square in half along the perpendicular axis to
; create two polygons", etc will have different treatment. "cut the
; square in half along the perpendicular axis to create two triangles"
; will clearly always be judged incoherent.

(cl-define-concept 'create-two-rectangles
     '(and recipe-1-step
           (all header create-parts-from-whole)
           (all substep1 cut-sq-inhalf-alongaxis)
           (all effect1 be-comp-sq-rect)
           (same-as (header agent) (substep1 agent))
           (same-as (header patient)
                    (effect1 location at-role))
           (same-as (header caused-role path-role destination)
                    (effect1 location))
           (same-as (substep1 patient)
                    (effect1 experiencer))))
```

```
(cl-define-concept 'loosen-screw-recipe
                   '(and recipe-1-step
                         (all header loosen-screw)
                         (all substep1 turn-counterclockwise)
                         (same-as (header agent) (substep1 agent))
                         (same-as (header patient) (substep1 patient))
                         (all qualif1 be-ident-tight)
                         (all effect1 be-ident-loose)
                         (same-as (header patient)
                                  (qualif1 experiencer))
                         (same-as (header patient)
                                  (effect1 experiencer))))

(cl-define-concept 'tighten-screw-recipe
                   '(and recipe-1-step
                         (all header tighten-screw)
                         (all substep1 turn-clockwise)
                         (same-as (header agent) (substep1 agent))
                         (same-as (header patient) (substep1 patient))
                         (all qualif1 be-ident-loose)
                         (all effect1 be-ident-tight)
                         (same-as (header patient)
                                  (qualif1 experiencer))
                         (same-as (header patient)
                                  (effect1 experiencer))))
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                               ;
;               INDIVIDUALS                     ;
;                                               ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; A subset of the definitions of individuals used to test the
; algorithm.



(cl-create-ind 'hearer 'animate)

(cl-create-ind 'speaker 'animate)

(cl-create-ind 'mary 'animate)

(cl-create-ind 'urn1 'inanimate)

(cl-create-ind 'kitchen 'inanimate)

(cl-create-ind 'room 'inanimate)

(cl-create-ind 'sink 'washing-site)

(cl-create-ind 'ink (and in-sp-place
                         (fills in-role kitchen)))

(cl-create-ind 'inr (and in-sp-place
                         (fills in-role room)))

(cl-create-ind 'tok '(and to-path-sp
                         (fills destination ink)))

(cl-create-ind 'tor (and  to-path-sp
                         (fills destination inr)))

(cl-create-ind 'fromr '(and from-path-sp
                            (fills source inr)))

(cl-create-ind 'fromk-tor '(and from-to-path-sp
                                (fills source ink)
                                (fills destination inr)))
```

```
(cl-create-ind 'go-kitchen '(and go-spatial
                                 (fills experiencer hearer)
                                 (fills path-role tok)))

(cl-create-ind 'go-mary-kitchen '(and go-spatial
                                      (fills experiencer mary)
                                      (fills path-role tok)))

(cl-create-ind 'at-place1 '(and at-sp-place (fills at-role speaker)))

(cl-create-ind 'tospeaker '(and to-at-path-sp
                                 (fills destination at-place1)))

(cl-create-ind 'go-urn-tospeaker '(and go-spatial
                                       (fills experiencer urn1)
                                       (fills path-role tospeaker)))

(cl-create-ind 'move-exp '(and move-sth-swh
                               (fills experiencer mary)
                               (fills patient urn1)
                               (fills caused-role go-urn-tospeaker)))

(cl-create-ind 'go-fromk-tor '(and go-sp-from-to
                                   (fills path-role fromk-tor)))

(cl-create-ind 'move1 '(and move-sth-swh
                            (fills experiencer hearer)
                            (fills patient urn1)
                            (fills caused-role go-fromk-tor)))

(cl-create-ind 'go-from-room '(and go-spatial
                                   (fills experiencer urn1)
                                   (fills path-role fromr)))

(cl-create-ind 'move2 '(and move-sth-swh
                            (fills experiencer hearer)
                            (fills patient urn1)
                            (fills caused-role go-from-room)))

; the next command correctly fails, as the "inanimate" urn1 can't be
; in control of another entity.

(cl-create-ind 'at-place2 '(and at-control-place
                                (fills at-role urn1)))
```

```
(cl-create-ind 'at-place3 '(and at-control-place
                               (fills at-role hearer)))

(cl-create-ind 'topc3 '(and to-at-path-control
                            (fills destination at-place3)))

(cl-create-ind 'go-control1 '(and go-control-to-at
                                  (fills experiencer urn1)
                                  (fills path-role topc3)))

(cl-create-ind 'get-control1 '(and get-control
                                  (fills caused-role go-control1)
                                  (fills experiencer hearer)))

(cl-create-ind 'go-to-room '(and go-spatial
                                 (fills experiencer urn1)
                                 (fills path-role tor)))

(cl-create-ind 'move3 '(and move-sth-swh
                            (fills experiencer hearer)
                            (fills patient urn1)
                            (fills caused-role go-to-room)))


; The previous three definitions show how fillers are  propagated
; because of same-as restrictions. The two actions  to match are
; go-kitchen (the substep) and move3 (the goal): the  restriction
; (source kitchen) from go-kitchen is added to tor during  the
; matching process; therefore tor is reclassified  as a
; from-to-path-sp, and as a consequence also go-to-room is
; reclassified as a go-sp-from-to.

(cl-create-ind 'go-no-path '(and go-spatial (fills experiencer urn1)))

(cl-create-ind 'move4 '(and move-sth-swh
                            (fills experiencer hearer)
                            (fills patient urn1)
                            (fills caused-role go-no-path)))

(cl-create-ind 'wash1 '(and wash
                            (fills agent hearer)
                            (fills patient urn1)))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Individuals relative to "cut"                                        ;
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(cl-create-ind 'sq1 'square)

(cl-create-ind 'tr1 'triangle)

(cl-create-ind 'diagonal1 'diagonal)

(cl-create-ind 'perp-axis1 'perp-axis)

(cl-create-ind 'along-diagonal1
               '(and along-diagonal
                     (fills along-role diagonal1)))

(cl-create-ind 'along-perp1
               '(and along-perp-axis
                     (fills along-role perp-axis1)))

(cl-create-ind 'at-tr1 '(and at-comp-triangle
                             (fills at-role tr1)))

(cl-create-ind 'at-sq1 '(and at-comp-square
                             (fills at-role sq1)))

(cl-create-ind 'from-sq-to-tr '(and comp-path
                                    (fills destination at-tr1)
                                    (fills source at-sq1)))

(cl-create-ind 'go-comp1 '(and go-comp-minus
                               (fills path-role from-sq-to-tr)))

(cl-create-ind 'cut1 '(and reduce-to-pieces (fills lex-item cut)))

(cl-create-ind 'cut-sq '(and reduce-to-pieces
                             (fills experiencer hearer)
                             (fills patient sq1)
                             (fills lex-item cut)))

(cl-create-ind 'create-tr '(and create-parts-from-whole
                             (fills lex-item create)
                             (fills caused-role go-comp1)
                             (fills experiencer hearer)
                             (fills patient tr1)))

(cl-create-ind 'cut-sq-carefully '(and reduce-to-pieces
```

```
                                        (fills experiencer hearer)
                                        (fills patient sq1)
                                        (fills lex-item cut)
                                        (fills manner carefully)))

(cl-create-ind 'cut-sq-diag '(and cut-location
                              (fills experiencer hearer)
                              (fills patient sq1)
                              (fills lex-item cut)
                              (fills location along-diagonal1)))

(cl-create-ind 'cut-sq-perp '(and cut-location
                              (fills experiencer hearer)
                              (fills patient sq1)
                              (fills lex-item cut)
                              (fills location along-perp1)))

;;;;;;;;;;;;;;;;;;;;;;;;
; INDIVIDUALS FOR TURN  ;
;;;;;;;;;;;;;;;;;;;;;;;;

(cl-create-ind 'screw1 'screw)

(cl-create-ind 'turn1 '(and turn-counterclockwise
                            (fills experiencer hearer)
                            (fills patient screw1)))

(cl-create-ind 'loosen1 '(and loosen-screw
                              (fills experiencer hearer)
                              (fills patient screw1)))
```

Appendix D

The AnimNL project

The AnimNL project has as its goal the automatic creation of animated task simulations from NL instructions. Its agents are animated human figures, and the tasks they are to engage in are *assembly procedures* and *maintenance procedures*. AnimNL is intended to support advanced human factors analysis, to enable users of computer-aided design tools to simulate human agents' interactions with the artifacts they are designing and thereby to notice design flaws that may only become apparent when people get at them. (AnimNL may also have potential use in creating personalized training films, using animated agents whose size, strength and (assumed) skills mirror those of the apprentice being trained.)

The AnimNL project builds on an animation system, *Jack*^{TM}, that has been developed at the Computer Graphics Lab at the University of Pennsylvania. *Jack* provides articulated, animated human figures — both male and female — capable of realistic motion through model-based inverse kinematics [Badler *et al.*, 1993]. In addition, *Jack* agents can be anthropomorphically sized and given different "strengths", so as to vary agents' physical capabilities. Different spatial environments can also be set up and modified at will, so as to vary the situation in which tasks are carried out. Both types of flexibility enable designers to explore a wide range of situations of use, in which human agents will engage with the objects under design.

The problems that AnimNL addresses lie in how to map from *high-level task specifications* (specifying a structure of related goals to be achieved and constraints — positive and negative — on how to achieve them) to *plausible physical behaviors* performed 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 previously been considered in a world of fairly simple robots.

Figure D.1 shows a schematic diagram of the system. AnimNL's architecture reflects both what is necessary for an agent *understanding and acting* in accordance with assembly and maintenance instructions and what is necessary for *simulating and animating* that behavior.

The first thing to notice about the diagram is that it consists of two relatively independent sets of processes. One set produces commitments to *act* for a particular purpose – e.g.

Figure D.1: AnimNL System Architecture

- *goto(door1, open(door1))* "go to door1 for the purpose of opening it"
- *grasp(urn1, lift(urn1))* "grasp urn1 for the purpose of lifting it"

The other set of processes figures out how the agent should *move* in order to carry out the specified actions on their specified objects for their specified purposes. What we will describe here is how instructions lead to initial commitments to act and how actions once embarked upon allow further commitments to be made and acted upon. (While the 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.)

Instructions are given to AnimNL in *steps* consisting of one or more utterances, such as:

> 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)

A step specifies a continuous behavior that the agent must attend to. When the step is complete, the agent can direct its attention to the next step in the procedure.

Steps are processed by a parser that uses a combinatory categorial grammar (CCG) [Steedman, 1990]. The parser produces the logical form based on Conceptual Structures discussed in Ch. 6. The Action Library that I described at length in that chapter is what is labelled *Action Schemata* in the **Action KB** box in Fig. D.1.

The CS representation corresponding to a single instruction step is incrementally developed into the *plan graph* described in Sec. 7.1.7, via processes of *reference resolution*, *plan inference*, *reference grounding* and *planning*. An intention here is an intention to satisfy a given goal or act in a given way at a given time (or in a given situation). Specific intentions develop incrementally via the processes mentioned above. The algorithm that I described in Ch. 7 corresponds to the boxes **plan graph initialization** and **plan inference** in Fig. D.1.

When an intention becomes sufficiently specified for the agent to be ready to commit to it and temporal dependencies permit such commitment, the intention is gated, triggering other, low-level planning processes (see Figure D.1 below the "action gate"). The output of these processes may either be an indication that the agent's "body" is unable to carry out the behavior that its "mind" would like it to or a collection of *behaviors* to be executed (simulated) in parallel.[1] Actions change the world, as well as the agent's knowledge. Such changes trigger further elaboration of the agent's intentional structure and further commitments to action.

In the following, I will describe some of the other modules in AnimNL.

[1] Previous actions need not be completed before a new action is committed to: an agent can be (and usually is) doing more than one thing at a time.

Planning in Anticipation of Future Intentions

The relative benignity of the assumed environment encourages agents to seek behavioral efficiency by anticipating the future and choosing actions that allow them to not only satisfy current sub-goals but do so in a way that best positions them for satisfying future goals. This kind of look-ahead optimization cannot be achieved simply by post-hoc smoothing of the transition between one otherwise fixed behavior and another. At the highest conceptual level, the agents should choose a method for achieving a current goal that reduces the physical effort they will have to expend to achieve the next one, and to exploit possible overlapping of behavioral results. One goal of AnimNL's planner, in elaborating the initial Plan Graph created in response to the specified instructions, is to use bounded look-ahead so as to make current goal-decomposition decisions based on the curent state of the world and future intentions — for further details see [Geib, 1992].

Planning in Anticipation of Acquiring Knowledge

AnimNL assumes that an agent cannot have up-to-date knowledge of any part of its environment that is outside its direct perception. An AnimNL agent may know what non-visible parts of its environment were like, when it perceived them earlier, and have expectations about what they will be like, when it sees them next, but its knowledge is limited to general truths about the world and to its direct perceptions. As for the extent of an AnimNL agent's perception, it is assumed that an agent cannot see into any space that has no *portal* open into the space the agent occupies. Thus AnimNL agents have to open doors, closets, boxes, etc., if they want to know what is inside, or go into other rooms to find out what is there, namely, they have to explore their environment. This is done through search plans [Moore, 1993]: search plans give an agent the ability to try out hypotheses about the environment (e.g. cracking a safe by trying out various combinations, opening and looking into kitchen cabinets to find a bowl of sugar, etc.) and thereby expand its knowledge and ability to act in its environment.

Geometric and Functional Planning

Instructions may tell an agent to open the door of a cabinet and remove some object from inside. But the way the agent moves in order to open the door depends on both its geometry (if and how it is latched, its degrees of freedom – different for hinged doors than for sliding doors, etc.) and the goal the agent has in opening it.

Similarly, instructions may tell the agent to open a door, drawing his attention to a fixture on the wall. Up to that point, the agent may not have known whether the architectural feature in question was a door or a French window. Characterized as a door however, the agent develops different expectations about the object than if it had been characterized as a window (e.g., about constraints on its motion, locations of possible latches, etc.).

The point of AnimNL's object-specific planner [Levison, 1993] is to direct an agent's movements in accordance with the artifacts being manipulated and the purpose of the manipulation.

Simulator

An AnimNL agent possesses certain *low-level behaviors* that it performs "automatically" when the situation demands. For example, an agent will take a step forward if necessary for maintaining balance, or adjust its hip position while walking, to alter its direction to avoid an object. Some of these behaviors are built into *Jack* itself, as inherent procedural constraints on an agent's body. A second set are simple animal behaviors (such as attraction and avoidance) that can be specified to the *simulator* that sits over *Jack*. Different types of agents can be specified as acting in accordance with different types of behaviors. Finally, a third set of low-level behaviors are postural adjustments that are made to achieve positional or orientational goals of an end effector. Like *Jack*, the simulator exists independently of AnimNL, able to create scenarios of any number of agents acting in accordance with any number of these simple behaviors.

Above those low-level behaviors, an AnimNL agent has a certain range of *low-level actions* (such as walk and grasp) that it can be "told" to carry out. (Such actions are specified symbolically to the simulator, and multiple actions of multiple agents can specified simultaneously.)

An AnimNL agent can be given general knowledge about types of *simple actions* at the level of opening things, picking things up and carrying them. Such knowledge is specified declaratively, with the specifications mapped into a structure of low-level actions that can be input to the simulator.

Among the modules that the simulator can call upon in simulating the behavior of agents and their environment for eventual input to *Jack* is the *posture planner* [Jung, 1992]. Quantitatively calculating an effective posture for an articulated body with 70 joints and 136 degrees of freedom (not counting hands) is a daunting task. The posture planner discovers a final global posture by finding collision-avoiding intermediate postural constraints through reasoning about qualitative geometric relations between body parts and objects and about motion dependencies between body parts. (Postural constraints are spatial constraints on *control points* and *control vectors* previously defined on important body parts.) The global posture satisfying a set of postural constraints is computed from a robust inverse kinematics algorithm. Posture planning complements joint space planning, which starts with the strong assumption that a final global posture is given.

Bibliography

[Allen and Perrault, 1980] James Allen and Raymond Perrault. Analyzing Intentions in Utterances. *Artificial Intelligence*, 15:143–178, 1980.

[Allen, 1983] James Allen. Recognizing Intentions from Natural Language Utterances. In Michael Brady and Robert Berwick, editors, *Computational Models of Discourse*. MIT Press, 1983.

[Allen, 1984] James Allen. Towards a General Theory of Action and Time. *Artificial Intelligence*, 23:123–154, 1984.

[Allen, 1987] James Allen. *Natural Language Understanding*. The Benjamin/Cummings Publishing Company, 1987.

[Allen, 1990] James Allen. Two Views of Intention: Comments on Bratman and on Cohen and Levesque. In P. Cohen, J. Morgan, and M. Pollack, editors, *Intentions in Communication*. MIT Press, 1990.

[Alterman *et al.*, 1991] Richard Alterman, Roland Zito-Wolf, and Tamitha Carpenter. Interaction, Comprehension, and Instruction Usage. Technical Report CS-91-161, Dept. of Computer Science, Center for Complex Systems, Brandeis University, 1991.

[Badler *et al.*, 1990] Norman Badler, Bonnie Webber, Jeff Esakov, and Jugal Kalita. Animation from Instructions. In Badler, Barsky, and Zeltzer, editors, *Making them Move*. MIT Press, 1990.

[Badler *et al.*, 1993] Norman I. Badler, Cary B. Phillips, and Bonnie Webber. *Simulating Humans: Computer Graphics Animation and Control*. Oxford University Press, 1993.

[Balkanski, 1990] Cecile Balkanski. Modelling Act-Type Relations in Collaborative Activity. Technical Report TR-23-90, Center for Research in Computing Technology, Harvard University, 1990.

[Balkanski, 1992a] Cecile Balkanski. Action Relations in Rationale Clauses and Means Clauses. In *COLING92, Proceedings of the Fourteenth International Conference on Computational Linguistics*, pages 267–273, 1992.

[Balkanski, 1992b] Cecile Balkanski. Actions, Beliefs and Intentions in Rationale Clauses and Means Clauses. In *AAAI92, Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 296–301, 1992.

[Balkanski, 1993] Cecile Balkanski. *Actions, Beliefs and Intentions in Multi-Action Utterances*. PhD thesis, Harvard University, 1993. Technical Report TR-16-93.

[Borgida and Patel-Schneider, 1993] Alex Borgida and Peter F. Patel-Schneider. A Semantics and Complete Algorithm for Subsumption in the CLASSIC Description Logic. AT&T Bell Laboratories, June 1993.

[Borgida *et al.*, 1989] Alexander Borgida, Ronald Brachman, Deborah McGuinness, and Lori Alperin Resnick. CLASSIC: a Structural Data Model for Objects. In *ACM SIGMOD International Conference on Management of Data*, 1989.

[Brachman and Levesque, 1984] Ronald Brachman and Hector Levesque. The Tractability of Subsumption in Frame-Based Description Languages. In *AAAI84, Proceedings of the Fourth National Conference on Artificial Intelligence*, pages 34–37, 1984.

[Brachman and Schmolze, 1985] Ronald Brachman and James Schmolze. An Overview of the KL-ONE Knowledge Representation system. *Cognitive Science*, 9:171–216, 1985.

[Brachman *et al.*, 1979] R. Brachman, R. Bobrow, P. Cohen, J. Klovstad, B. Webber, and W. Woods. Research in Natural Language Understanding, Annual Report. Technical Report 4274, Bolt Beranek and Newman, 1979.

[Brachman *et al.*, 1983a] R. Brachman, R.Fikes, and H. Levesque. KRYPTON: A Functional Approach to Knowledge Representation. Technical Report FLAIR 16, Fairchild Laboratories for Artificial Intelligence, Palo Alto, California, 1983. Also in *Readings in Knowledge Representation*, Brachman, Levesque eds., Morgan-Kaufmann, 1985.

[Brachman *et al.*, 1983b] R. Brachman, R.Fikes, and H. Levesque. KRYPTON: Integrating Terminology and Assertion. In *AAAI83, Proceedings of the Third National Conference on Artificial Intelligence*, 1983.

[Brachman *et al.*, 1991] Ronald Brachman, Deborah McGuinness, Peter Patel-Schneider, Lori Alperin Resnick, and Alexander Borgida. Living with CLASSIC: When and How to Use a KL-ONE-like Language. In John F. Sowa, editor, *Principles of Semantic Networks — Explorations in the Representation of Knowledge*, pages 401–456. Morgan Kaufmann, 1991.

[Bratman, 1990] Michael Bratman. What is Intention? In P. Cohen, J. Morgan, and M. Pollack, editors, *Intentions in Communication*. MIT Press, 1990.

[Carberry, 1985] Sandra Carberry. *Pragmatic modeling in information system interfaces*. PhD thesis, University of Delaware, 1985.

[CC, 1988] Country Crafts, 1988.

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

[Charniak, 1988] Eugene Charniak. Motivation Analysis, Abductive Unification, and Nonmonotonic Equality. *Artificial Intelligence*, 34:275–295, 1988.

[Clark and Haviland, 1977] Herbert Clark and Susan Haviland. Comprehension and the Given-New Contract. In R. Freedle, editor, *Discourse Production and Comprehension*. Lawrence Erlbaum Associates, 1977.

[Cohen and Levesque, 1990] Philip Cohen and Hector Levesque. Rational Interaction as the Basis for Communication. In P. Cohen, J. Morgan, and M. Pollack, editors, *Intentions in Communication*. MIT Press, 1990.

[Cohen and Perrault, 1979] Philip Cohen and Raymond Perrault. Elements of a Plan-Based Theory of Speech-Acts. *Cognitive Science*, 3:177–212, 1979.

[Cohen *et al.*, 1990] Philip Cohen, Jerry Morgan, and Martha Pollack, editors. *Intentions in Communication*. MIT Press, 1990.

[CWP, 1989] Country Wood Projects, Summer 1989.

[Dale, 1989] Robert Dale. *Generating Referring Expressions in a Domain of Objects and Processes*. PhD thesis, University of Edinburgh, 1989.

[Dale, 1992] Robert Dale. *Generating Referring Expressions*. ACL-MIT Series in Natural Language Processing. The MIT Press, 1992.

[Davies, 1986] Eirlys Davies. *The English Imperative*. Croom Helm, 1986.

[Delin *et al.*, 1993] Judy Delin, Donia Scott, and Tony Hartley. Knowledge, Intention, Rhetoric: Levels of Variation in Multilingual Instructions. In *ACL Workshop on Intentionality and Structure in Discourse Relations*, pages 7–10, Columbus, OH, 1993.

[Devanbu and Litman, 1991] Premkumar Devanbu and Diane Litman. Plan-Based Terminological Reasoning. In *KR91, Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning*, pages 128–138, 1991.

[Dixon *et al.*, 1988] Peter Dixon, Jeremiah Faries, and Gareth Gabrys. The Role of Explicit Action Statements in Understanding and Using Written Directions. *Journal of Memory and Language*, 27(6):649–667, 1988.

[Dixon, 1987a] Peter Dixon. The Processing of Organizational and Component Step Information in Written Directions. *Journal of Memory and Language*, 26(1):24–35, 1987.

[Dixon, 1987b] Peter Dixon. The Structure of Mental Plans for Following Directions. *Journal of Experimental Psychology: Learning, Memory and Cognition*, 13(1):18–26, 1987.

[Dyer, 1983] M.G. Dyer. *In-Depth Understanding: a Computer Model of Integrated Processing and Memory for Narrative Comprehension*. MIT Press, 1983.

[Esakov and Badler, 1990] Jeffrey Esakov and Norman I. Badler. An Architecture for High-Level Human Task Animation Control. In P.A. Fishwick and R.S.Modjeski, editors, *Knowledge-Based Simulation: Methodology and Application*, pages 162–199. New York: Springer Verlag, 1990.

[Feldman and Sproull, 1977] Jerome Feldman and Robert Sproull. Decision Theory and Artificial Intelligence II: The Hungry Monkey. *Cognitive Science*, 1:158–192, 1977. Also in *Readings in Planning*, Allen, Hendler, Tate eds., Morgan-Kaufmann, 1990.

[FH, 1990] The Family Handiman, February 1990.

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

[Fillmore, 1985] Charles Fillmore. Frames and the Semantics of Understanding. *Quaderni di Semantica*, VI(2), 1985.

[Geib, 1992] Christopher Geib. Intentions in Means/End Planning. Technical Report MS-CIS-92-73, University of Pennsylvania, 1992.

[Genesereth and Nilsson, 1987] Michael R. Genesereth and Nils J. Nilsson. *Logical Foundations of Artificial Intelligence*. Morgan Kaufmann Publishers, 1987.

[Goldman, 1970] Alvin Goldman. *A Theory of Human Action*. Princeton University Press, 1970.

[Green, 1991] Georgia Green. Purpose Infinitives and their Relatives. Technical Report Cognitive Science CS-91-04, University of Illinois at Urbana-Champaign, 1991.

[Grosz and Sidner, 1986] Barbara Grosz and Candace Sidner. Attention, Intentions, and the Structure of Discourse. *Computational Linguistics*, 12:175–204, 1986.

[Grosz and Sidner, 1990] Barbara Grosz and Candace Sidner. Plans for Discourse. In P. Cohen, J. Morgan, and M. Pollack, editors, *Intentions in Communication*. MIT Press, 1990.

[Hallowell, 1988a] Alice Rich Hallowell, editor. *Tile, Remodeling Handbook*. Sunset Books. Lane Publishing Company, 1988.

[Hallowell, 1988b] Alice Rich Hallowell, editor. *Wall Coverings*. Sunset Books. Lane Publishing Company, 1988.

[Halpern and Moses, 1985] J.Y. Halpern and Y.O. Moses. A Guide to the Modal Logics of Knowledge and Belief. In *IJCAI85, Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, 1985.

[Hamblin, 1987] Charles Hamblin. *Imperatives*. Basil Blackwell, 1987.

[HC, 1988] Holiday Crafts, 1988.

[Hegarty, 1990] Michael Hegarty. Secondary Predication and Null Operators in English, 1990. Manuscript.

[Heim, 1982] Irene Heim. *The Semantics of Definite and Indefinite Noun Phrases*. PhD thesis, University of Massachussets, 1982.

[Herskovits, 1986] Annette Herskovits. *Language and Spatial Cognition: an Interdisciplinary Study of the Preposition in English.* Cambridge University Press, 1986.

[Hobbs, 1990] Jerry Hobbs. Artificial Intelligence and Collective Intentionality: Comments on Searle and on Grosz and Sidner. In P. Cohen, J. Morgan, and M. Pollack, editors, *Intentions in Communication.* MIT Press, 1990.

[Horn, 1989] Laurence Horn. *A Natural History of Negation.* The University of Chicago Press, 1989.

[Huettner *et al.*, 1987] Alison Huettner, Marie Vaughan, and David McDonald. Constraints on the Generation of Adjunct Clauses. In *ACL87, Proceedings of the 25th Annual Meeting of the Association for Computational Linguistics*, 1987.

[Iwanska, 1992] Lucja Iwanska. A General Semantic Model of Negation in Natural Language: Representation and Inference. In *KR92, Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning*, 1992.

[Jackendoff, 1983] Ray Jackendoff. *Semantics and Cognition.* Current Studies in Linguistics Series. The MIT Press, 1983.

[Jackendoff, 1990] Ray Jackendoff. *Semantic Structures.* Current Studies in Linguistics Series. The MIT Press, 1990.

[Jones, 1985] Charles Jones. Agent, Patient, and Control into Purpose Clauses. In *Chicago Linguistic Society, 21*, 1985.

[Jones, 1991] Charles Jones. *Purpose Clauses: Syntax, Thematics and Semantics of English Purpose Constructions.* Kluwer Academic Publishers, 1991.

[Joshi and Rosenschein, 1976] Aravind K. Joshi and Stanley J. Rosenschein. Some Problems of Inferencing: Relation of Inferencing to Decomposition of Predicates. In *COLING76, Proceedings of the Fifth International Conference on Computational Linguistics*, 1976.

[Joshi, 1978] Aravind K. Joshi. Some Extensions of a System for Inference of Partial Information. In D.A. Waterman and F. Hayes-Roth, editors, *Pattern-Directed Inference Systems*, pages 241–257. Academic Press, 1978.

[Jung, 1992] Moon Jung. *Human Body Posture Planning in Work Spaces.* PhD thesis, University of Pennsylvania, 1992.

[Kautz, 1990] Henry Kautz. A Circumscriptive Theory of Plan Recognition. In P. Cohen, J. Morgan, and M. Pollack, editors, *Intentions in Communication.* MIT Press, 1990.

[Kobsa and Wahlster, 1989] Alfred Kobsa and Wolfgang Wahlster, editors. *User Models in Dialog Systems.* Springer Verlag, 1989.

[Lambert and Carberry, 1991] Lynn Lambert and Sandra Carberry. A Tripartite Plan-Based Model of Dialogue. In *ACL91, Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics*, pages 47–54, 1991.

[Lambert and Carberry, 1992] Lynn Lambert and Sandra Carberry. Modeling Negotiation Subdialogues. In *ACL92, Proceedings of the 30th Annual Meeting of the Association for Computational Linguistics*, pages 193–200, 1992.

[Levin and Pinker, 1991] Beth Levin and Steven Pinker. Introduction. *Cognition, Special Issue on Lexical and Conceptual Semantics*, 41:1–7, 1991.

[Levison, 1993] Libby Levison. Object Specific Reasoning, 1993. Thesis Proposal, University of Pennsylvania. Forthcoming.

[Lewis, 1979] David Lewis. Scorekeeping in a Language Game. *Journal of Philosophical Language*, 8:339–359, 1979.

[Litman and Allen, 1990] Diane Litman and James Allen. Discourse Processing and Commonsense Plans. In P. Cohen, J. Morgan, and M. Pollack, editors, *Intentions in Communication*. MIT Press, 1990.

[Litman, 1985] Diane Litman. *Plan Recognition and Discourse Analysis. An Integrated Approach for Understanding Dialogues*. PhD thesis, University of Rochester, 1985.

[Lochbaum, 1991a] Karen Lochbaum. An Algorithm for Plan Recognition in Collaborative Discourse. In *ACL91, Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics*, pages 33–38, 1991.

[Lochbaum, 1991b] Karen Lochbaum. Plan Recognition in Collaborative Discourse. Technical Report TR-14-91, Center for Research in Computing Technology, Harvard University, 1991.

[Lyons, 1977] J. Lyons. *Semantics*. Cambridge University Press, 1977.

[Mac Gregor, 1988] Robert Mac Gregor. A Deductive Pattern Matcher. In *AAAI88, Proceedings of the Seventh National Conference on Artificial Intelligence*, 1988.

[Mac Gregor, 1991] Robert Mac Gregor. The Evolving Technology of Classification-Based Knowledge Representation Systems. In John F. Sowa, editor, *Principles of Semantic Networks — Explorations in the Representation of Knowledge*, pages 385–400. Morgan Kaufmann Publishers, Inc., 1991.

[Mann and Thompson, 1988] William C. Mann and Sandra Thompson. Rhetorical Structure Theory: toward a Functional Theory of Text Organization. *Text*, 8(3):243–281, 1988.

[McKeown *et al.*, 1992] K.R. McKeown, M. Elhadad, Y. Fukumoto, J. Lim, C. Lombardi, J. Robin, and F. Smadja. Language Generation in COMET. In O.Stock, E. Hovy, D. Rosner, and R. Dale, editors, *Sixth International Workshop on Natural Language Generation*, 1992.

[Merin, 1991] Arthur Merin. Imperatives: Linguistics vs. Philosophy. *Linguistics*, 29:669–702, 1991.

[Miller and Johnson-Laird, 1976] George Miller and Philip Johnson-Laird. *Language and Perception*. Harvard University Press, 1976.

[Moore, 1993] Michael B. Moore. Search Plans. Thesis Proposal, University of Pennsylvania, 1993.

[Moser *et al.*, 1983] M. G. Moser, J. Schmolze, D. Israel, M. Vilain, and D. McAllester. NIKL. Technical Report 5421, BBN Laboratories, Cambridge, MA, 1983.

[Moser, 1992] Margaret G. Moser. *The Negation Relation: Semantic and Pragmatic Aspects of a Relational Analysis of Sentential Negation*. PhD thesis, University of Pennsylvania, 1992.

[Nebel, 1990] Bernhard Nebel. *Reasoning and Revision in Hybrid Representation Systems*. Lecture Notes in Artificial Intelligence. Springer-Verlag, 1990.

[Ortiz, 1993] Charles L. Ortiz. The Semantics of Event Prevention. In *AAAI93, Proceedings of the Eleventh National Conference on Artificial Intelligence*, 1993.

[Perrault and Allen, 1980] Raymond Perrault and James Allen. A Plan-Based Analysis of Indirect Speech-Acts. *American Journal of Computational Linguistics*, 6:167–182, 1980.

[Polanyi, 1988] Livia Polanyi. A Formal Model of the Structure of Discourse. *Journal of Pragmatics*, 12:601–638, 1988.

[Pollack, 1986] Martha Pollack. *Inferring Domain Plans in Question-Answering*. PhD thesis, University of Pennsylvania, 1986.

[Pollack, 1990] Martha Pollack. Plans as Complex Mental Attitudes. In P. Cohen, J. Morgan, and M. Pollack, editors, *Intentions in Communication*. MIT Press, 1990.

[Prince, 1981] Ellen Prince. Toward a Taxonomy of Given-New Information. In P. Cole, editor, *Radical Pragmatics*. Academic Press, 1981.

[Quantz and Kindermann, 1990] J. Quantz and C. Kindermann. Implementation of the BACK System Version 4. Technical Report KIT 78, Technische Universitat Berlin, 1990.

[Ramshaw, 1989a] Lance A. Ramshaw. A Metaplan Model for Problem-Solving Discourse. In *EACL89, Proceedings of the Fourth Meeting of the European Chapter of the Association for Computational Linguistics*, 1989.

[Ramshaw, 1989b] Lance A. Ramshaw. *Pragmatic Knowledge for resolving Ill-Formedness*. PhD thesis, University of Delaware, 1989.

[Ramshaw, 1991] Lance A. Ramshaw. A Three-Level Model for Plan Exploration. In *ACL91, Proceedings of the 29th Meeting of the Association for Computational Linguistics*, pages 39–46, 1991.

[Sacerdoti, 1977] Earl Sacerdoti. *A Structure for Plans and Behavior*. Artificial Intelligence Series - Computer Science Library. Elsevier, 1977.

[Schank and Abelson, 1977] Roger C. Schank and Robert P. Abelson. *Scripts, Plans, Goals and Understanding*. Lawrence Erlbaum Associates, 1977.

[Schank, 1975] Roger C. Schank. *Conceptual Information Processing*. North Holland, 1975.

[Schirra, 1990] Jöerg Schirra. A Contribution to Reference Semantics of Spatial Prepositions: the Visualization Problem and its Solution in VITRA. Technical Report 75, SFB 314, VITRA, Universität des Saarlandes, Saarbrücken, 1990.

[Schmerling, 1982] Susan Schmerling. How Imperatives are Special, and how they aren't. In R. Schneider, K. Tuite, and R. Chametzy, editors, *Papers from the Parasession on Nondeclaratives, Chicago Linguistics Society*, 1982.

[Schoppers, 1988] Marcel Schoppers. *Representation and Automatic Synthesis of Reaction Plans*. PhD thesis, University of Illinois at Urbana-Champaign, 1988.

[Searle, 1975] John R. Searle. Indirect Speech Acts. In P. Cole and J.L. Morgan, editors, *Syntax and Semantics 3. Speech Acts*. Academic Press, 1975.

[Shapiro, 1992] Stuart C. Shapiro, editor. *Encyclopedia of Artificial Intelligence — Second Edition*. John Wiley and Sons, 1992.

[Sidner, 1985] Candace Sidner. Plan Parsing for Intended Response Recognition in Discourse. *Computational Intelligence*, 1, 1985.

[Steedman, 1990] Mark Steedman. Gapping as Constituent Coordination. *Linguistics and Philosophy*, 13:207–263, 1990.

[Steedman, 1991] Mark Steedman. Structure and Intonation. *Language*, 68(2):260–296, 1991.

[Suchman, 1987] Lucy Suchman. *Plans and Situated Actions*. Cambridge University Press, 1987.

[Suppes and Crangle, 1988] P. Suppes and C. Crangle. Context-Fixing Semantics for the Language of Action. In J. Dancy, J. Moravcsik, and C. Taylor, editors, *Human Agency: Language, Duty, and Value*, pages 47–76. Stanford University Press, 1988.

[Tate, 1987] Austin Tate. Generating Project Networks. In *IJCAI87, Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, 1987.

[Tenenberg, 1989] Josh Tenenberg. Inheritance in Automated Planning. In *KR89, Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*, 1989.

[Thomason, 1990] Richmond Thomason. Accommodation, Meaning, and Implicature: Interdisciplinary Foundations for Pragmatics. In P. Cohen, J. Morgan, and M. Pollack, editors, *Intentions in Communication*. MIT Press, 1990.

[Thompson, 1985] Sandra Thompson. Grammar and Written Discourse: Initial vs. Final Purpose Clauses in English. *Text*, 1-2:55–84, 1985.

[Vander Linden *et al.*, 1992a] Keith Vander Linden, Susanna Cumming, and James Martin. Using System Networks to Build Rhetorical Structures. In O.Stock, E. Hovy, D. Rosner, and R. Dale, editors, *Sixth International Workshop on Natural Language Generation*, 1992.

[Vander Linden *et al.*, 1992b] Keith Vander Linden, Susanna Cumming, and James Martin. The Expression of Local Rhetorical Relations in Instructional Text. Technical Report CU-CS-585-92, University of Colorado at Boulder, Department of Computer Science, 1992.

[Vander Linden, 1993a] Keith Vander Linden. Rhetorical Relations in Instructional Text Generation. In *ACL Workshop on Intentionality and Structure in Discourse Relations*, pages 140–143, Columbus, OH, 1993.

[Vander Linden, 1993b] Keith Vander Linden. *Speaking of Actions: Choosing Rhetorical Status and Grammatical Form in Instructional Text Generation.* PhD thesis, University of Colorado at Boulder, 1993. Technical Report CU-CS-654-93.

[Vere and Bickmore, 1990] Steven Vere and Timothy Bickmore. A Basic Agent. *Computational Intelligence*, 6:41–60, 1990.

[Vilain, 1985] Mark Vilain. The Restricted Language Architecture of a Hybrid Representation System. In *IJCAI85, Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, 1985.

[Wahlster *et al.*, 1989] Wolfgang Wahlster, Elisabeth Andre', Matthias Hecking, and Thomas Rist. WIP: Knowledge-Based Presentation of Information. Technical Report WIP-1, DFKI — German Research Center for Artificial Intelligence, 1989.

[Wahlster *et al.*, 1991] Wolfgang Wahlster, Elisabeth Andre', Son Bandyopadhyay, Winfried Graf, and Thomas Rist. WIP: The Coordinated Generation of Multimodal Presentations from a Common Representation. In Oliviero Stock, John Slack, and Andrew Ortony, editors, *Computational Theories of Communication and their Applications.* Berlin: Springer Verlag, 1991.

[Webber and Baldwin, 1992] Bonnie Webber and Breck Baldwin. Accommodating Context Change. In *ACL92, Proceedings of the 30th Meeting of the Association for Computational Linguistics*, pages 96–103, 1992.

[Webber and Di Eugenio, 1990] Bonnie Webber and Barbara Di Eugenio. Free Adjuncts in Natural Language Instructions. In *COLING90, Proceedings of the Thirteenth International Conference on Computational Linguistics*, pages 395–400, 1990.

[Webber *et al.*, 1991] Bonnie Webber, Norman Badler, Barbara Di Eugenio, Libby Levison, and Michael White. Instructing Animated Agents. In *Proceedings of the US-Japan Workshop on Integrated Systems in Multi-Media Environments.*, Las Cruces, NM, 1991.

[Webber *et al.*, 1992] Bonnie Webber, Norman Badler, F. Breckenridge Baldwin, Welton Becket, Barbara Di Eugenio, Christopher Geib, Moon Jung, Libby Levison, Michael

Moore, and Michael White. Doing What You're Told: Following Task Instructions In Changing, but Hospitable Environments. Technical Report MS-CIS-92-74, University of Pennsylvania, 1992. To appear in *Language and Vision across the Pacific*, Y. Wilks and N. Okada editors.

[Webber *et al.*, 1993] Bonnie Webber, Norman Badler, Barbara Di Eugenio, Christopher Geib, Libby Levison, and Michael Moore. Instructions, Intentions and Expectations. Technical Report MS-CIS-93-61, University of Pennsylvania, 1993. To appear in *Artificial Intelligence Journal*, Special Issue on Computational Theories of Interaction and Agency.

[Webber, 1991] Bonnie Lynn Webber. Structure and Ostension in the Interpretation of Discourse Deixis. *Language and Cognitive Processes*, 6(2):107–135, 1991.

[Weida and Litman, 1992] Robert Weida and Diane Litman. Terminological Reasoning with Constraint Networks and an Application to Plan Recognition. In *KR92, Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning*, 1992.

[Wellman, 1988] Michael P. Wellman. *Formulation of Tradeoffs in Planning under Uncertainty.* PhD thesis, Massachusetts Institute of Technology, 1988.

[White, 1992] Michael White. Conceptual Structures and CCG: Linking Theory and Incorporated Argument Adjuncts. In *COLING92, Proceedings of the Fourteenth International Conference on Computational Linguistics*, pages 246–252, 1992.

[Wilensky, 1982] Robert Wilensky. Points: A Theory of the Structure of Stories in Memory. In W. Lehnert and M. Rengle, editors, *Strategies for Natural-Language Processing.* Lawrence Erlbaum Associates, 1982.

[Wilensky, 1983] Robert Wilensky. *Planning and Understanding. A Computational Approach to Human Reasoning.* Addison-Wesley Publishing Company, 1983.

[Winograd, 1972] Terry Winograd. *Understanding Natural Language.* Academic Press, 1972.

[Woods and Schmolze, 1991] William Woods and James Schmolze. The KL-ONE Family. *Computers and Mathematics with Applications, Special Issue on Semantic Networks in Artificial Intelligence*, 1991. Also available as Technical Report TR-20-90, Aiken Computation Laboratory, Harvard University.

[Woods, 1991] William A. Woods. Understanding Subsumption and Taxonomy: A Framework for Progress. In John F. Sowa, editor, *Principles of Semantic Networks — Explorations in the Representation of Knowledge*, pages 45–94. Morgan Kaufmann Publishers, Inc., 1991.

[Zwarts and Verkuyl, 1993] Joost Zwarts and Henk Verkuyl. An Algebra of Conceptual Structure; an Investigation into Jackendoff's Conceptual Semantics. To be published in *Linguistics and Philosophy.* Forthcoming, 1993.