Computational Intelligence, Volume 14, Number 1, 1998

AN ACTION REPRESENTATION FORMALISM TO INTERPRET NATURAL LANGUAGE INSTRUCTIONS

Barbara Di Eugenio
Learning Research and Development Center
University of Pittsburgh
Pittsburgh, PA, 15260 USA
dieugeni@cs.pitt.edu

The focus of this paper is on an action representation formalism that encodes both *linguistic* and *planning* knowledge about actions, and that supports the interpretation of complex Natural Language instructions, and in particular, of instructions containing Purpose Clauses. The representation uses linguistically motivated primitives, derived from Jackendoff's work on Conceptual Semantics, and is embedded in the description logic based system CLASSIC. I will first motivate the characteristics of the formalism as needed to understand Natural Language instructions. I will then describe the formalism itself, and I will argue that the integration of a linguistically motivated lexical semantics formalism and of a description logic based system is beneficial to both. Finally, I will show how the formalism is exploited by the algorithm that interprets Purpose Clauses. The output of the algorithm is used in the *Animation from NL* project, that has as its goal the automatic creation of animated task simulations.

1. INTRODUCTION

The interpretation of Natural Language (NL) instructions, 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. However, another component to the inference processes that the agent performs is frequently overlooked — that component is the surface form that the agent is presented with. This may sound surprising, given that much work has been done on understanding NL instructions, and therefore on interpreting their surface forms — see for example (Winograd, 1972, Vere and Bickmore, 1990, Chapman, 1991). What I mean with *the surface form has not been given adequate attention* is that the linguistic input is often very simplified with respect to naturally occurring instructions. Most systems that interpret NL instructions, including the ones mentioned above, accept only simple main clauses, or main clauses modified by a very restricted set of subordinate clauses. Moreover, such systems directly map the input, often represented by means of a logical form à la Davidson (1982), onto the planning / script-like knowledge about tasks — this body of knowledge can be called the *plan library*. This approach has two shortcomings. First, it neglects the astonishing variety of constructions that occur in real NL instructions, and thus the constraints on actions and the relations between actions that they express. Second, it misses certain inferences that occur while mapping from the logical form to the plan library. In this paper, I will show how a representation that tries to take into account both linguistic and planning knowledge about actions constitutes a first step toward overcoming these two problems. While of course this does not completely solve the problem of interpreting NL instructions, I claim that an approach to action representation as I propose is an important step in the right direction.

Consider first some evidence from naturally occurring instructions.[1] First of all, NL instructions include a wide variety of linguistic features, such as instruments, tense and aspect features and manner modifiers. These characteristics provide essential information that is rarely taken into account. Compare:

(1a) *With the lug wrench, loosen the lug nuts that hold the tire on. Loosen means turn counterclockwise. Loosen each nut.*
(1b) *With the lug wrench, loosen the lug nuts that hold the tire on. Loosen means turn counterclockwise. Loosen each nut* **only one or two turns, not all the way.** (*Fixing cars — A People's Primer*)

It is unquestionable that the interpretations of *loosen each nut* in Examples (1a) and (1b) are very different: not only does the manner modifier *only one or two turns, not all the way* in Example (1b) constrain the number of repetitions of the *turning counterclockwise* action, but it also eliminates one possible interpretation of *loosen* as *loosen till the nut can be taken off / falls off*.

Furthermore, any action can be described in NL at any level of detail, as in

(2a) *Apply paste to the wall.*
(2b) *Using a paint roller or brush, apply paste to the wall.*
(2c) *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.* (Hallowell, 1988)

These first few examples already make apparent that the action representation formalism must be able to take into account all the linguistic information that the input provides. This in turn raises a problem, because it is unrealistic to expect the system to have knowledge about all possible ways that an action can be described in NL. My solution is to provide an action representation system flexible enough to *accommodate*[2] the input, namely, to support inferring the known action $\mathcal{KA}$ to which the action described in the input is most closely related. After $\mathcal{KA}$ has been inferred, and if the context provides further information, the system can still interpret the input by appropriately constraining $\mathcal{KA}$; otherwise, the system could engage the user in a repair dialogue in a perspicuous manner. Clearly, when the system can't intepret the input, there is an irreducible discrepancy between the knowledge on which the input is based, and the system knowledge: it would be the task of the repair module to decide what such discrepancy amounts to — a wrong instruction or incompleteness of the system knowledge — and to take appropriate action. However, the discrepancy has to be detected to start with, and this is the problem on which I focus.

Let's now consider the other problem, mapping from the logical form to the plan library. To start with, not all actions described in the input can and must necessarily map into an action to be executed: for instance, negative imperatives give rise to constraints on other actions rather than to independent actions (Di Eugenio, 1993a, Vander Linden and Di Eugenio, 1996). But even when all the actions described in the input have to be executed, there are subtleties arising from the interplay of the relation between the actions in question and the knowledge in the plan library.

[1]The examples that follow present a number of variations, one of which is naturally occurring, and the others constructed.

[2]The term *accommodation* is borrowed from (Lewis, 1979), and I use it somewhat informally.

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

(3a) *Place a plank between two ladders.*
(3b) *Place a plank between two ladders to create a simple scaffold.* (Hallowell, 1988)

In both (3a) and (3b), the action to be executed is *place a plank between two ladders*. However, (3a) would be correctly interpreted by placing the plank *anywhere* between the two ladders: this shows that in (3b) the agent must be inferring the proper position for the plank from the expressed goal *to create a simple scaffold*. Clearly, the logical form representations for *place a plank between two ladders* in (3a) and (3b) will be identical: how and when is the further constraint on *place a plank between two ladders* in (3b) computed? Clearly, *Place a plank between two ladders* can't be interpreted independently from the goal it is executed to achieve.

A more concrete (constructed) example of this is:

(4) *[Hold the cup under the spigot]$_\alpha$ [to fill it with coffee]$_\beta$.*

It is likely that the agent will have some general knowledge such as (5) in his plan library:

(5) *To fill a container, [hold it with its concavity directed upwards]$_\gamma$.*

Note that the action description α = *hold the cup under the spigot* in (4) is both more and less specific than the action description γ = *hold a container with its concavity directed upwards* in (5). α specializes γ, as it specifies the kind of container, *cup*, and the location the container must be held at, *under the spigot*; α is also less specific than γ, because of the missing modifier *with their concavity directed upwards*. The action to be executed has to include all these components, i.e. it could be described as *hold the cup under the spigot with its concavity directed upwards*. Even if there are well established computational mechanisms that can perform such inferences — for example, unification — as far as I know, no system that interprets NL instructions carries out this kind of reasoning.

Finally, let's consider the following minimal pair:[3] [4]

(6a) *Go into the kitchen and get me the coffee urn.*
(6b) *Go into the kitchen and wash the coffee urn.*

Following established linguistics and computational linguistics practice, most systems would map an instruction such as (6a) into a logical form (LF for short) such as (7), where S stands for Speaker and H for Hearer[5] — the representation for (6b) would have *wash(β)* replacing *get(β) $\wedge$ destination(β, S)*:

(7) $\mathrm{go}(\alpha) \wedge \mathrm{agent}(\alpha, \mathrm{H}) \wedge \mathrm{destination}(\alpha, x) \wedge \mathrm{kitchen}(x) \wedge \mathrm{unique}(x) \wedge \mathrm{get}(\beta) \wedge \mathrm{agent}(\beta, \mathrm{H}) \wedge \mathrm{patient}(\beta, y) \wedge \text{coffee-urn}(y) \wedge \mathrm{unique}(y) \wedge \mathrm{destination}(\beta, \mathrm{S}) \wedge \mathrm{and}(\alpha, \beta)$

Such LF's are built compositionally, and comprise elements of meaning for each

[3]Both these examples are constructed, but they illustrate a construction very common e.g. in directions.

[4]*Minimal pair* is the term used in Linguistics to denote two examples that differ only in the word of phrase of interest, here the verb in the second conjunct.

[5]For ease of exposition, I will use feminine gender for S/the instructor and masculine for H/the agent.

word in the surface utterance; in general, all elements of meaning need to be further specified through additional semantic and pragmatic processing. Part of pragmatic processing will consist in matching the LF in (7) to the plan library. Presumably it will then be inferred that α has to occur before β, thus refining *and* (α, β) to *and then* — *and*(α, β) in (7) doesn't refer to any logical connective, but simply derives from the conjunction *and* in the input. However, first, *and* should be expanded not just as *and then*, but as *in order to* (Doran, 1993). Second, under the latter interpretation, in (6a) an expectation arises that the coffee urn is in the kitchen. This expectation is strong enough that even if another coffee urn is visible in the initial context, namely, before *go into the kitchen* is executed, the agent won't bind the referent of y to such coffee urn; rather, he will wait to do so until *go into the kitchen* has changed his perceptual context. Also note that this expectation provides the context in which *unique(y)* holds: *unique(y)* is derived from the definite article *the*, but by itself it does not specify the context in which the definite Noun Phrase *the coffee urn* uniquely refers.

No such expectation arises in (6b). As I will discuss below, the presence or absence of such expectations depends on the fact that one action, α, is done in order to do the other, β, and on knowledge about the lexical semantics of the verbs involved and about acting in the world.

1.1. Purpose Clauses: a Case Study

The alert reader will have noticed that Examples (3b), (4), (6a) and (6b) all include a component that expresses the purpose for which a certain action is executed. More specifically, infinitival *to* constructions such as *to create a simple scaffold* in (3b) and *to fill it with coffee* in (4) are instances of *Purpose Clauses* (PC's). PC's are just one of the many ways that purposes may be expressed in NL: others are *for nominalizations*, *for gerunds* (Vander Linden, 1993, Vander Linden and Martin, 1995), subordinates introduced by *so that*, *such that*, and purposive *and* as in (6). I was driven to address the issue of action representation by my work on interpreting PC's in instructional text, more specifically, PC's in contexts such as *Do* α *to do* β, or *To do* β*, do* α.[6] [7] The notation, that I adopt throughout the paper, is as follows: α is the action that appears in the main clause, such as *Place the plank between two ladders* in (3b); β is the goal described by the infinitival clause, such as *to create a simple scaffold* in (3b). In the following, I will use α and β to refer both to the actual English text in the examples, and to the corresponding logical form — the context in which the symbols appear will disambiguate. Furthermore, below I will use the following symbols: $\mathcal{M}_l$ is the label for the l-th method (or recipe) that H (the system) has in its Plan Library; the index l has no meaning other than to distinguish one method from the other. δ_l is the header of $\mathcal{M}_l$, while $\gamma_{l,j}$ is the j-th component (substep) of $\mathcal{M}_l$ — Figure 5 shows an example of a recipe, with its header δ and substeps γ_1, γ_2, and γ_3. $\mathcal{R}$ is the relation holding between α and β. Finally, ρ_{perf} is the action description that is computed by the algorithm.

[6] My corpus consists of 164 consecutive occurrences of PC's, collected from two how-to-do books and from two craft magazines (Hallowell, 1988, TRH, 1988, CC, 1988, CWP, 1989).

[7] When the PC is preposed, it is common that the surface form is *To do* β*, do* $\mathcal{A}$, where $\mathcal{A}$ is a sequence of actions of length $n \geq 1$ (Thompson, 1985, Vander Linden, 1993). While I collected all these examples, and examined them from a linguistic point of view, I won't discuss them in this paper, because my algorithm only deals with one α and one β at a time.

While discrepancies between input and stored action descriptions are pervasive in NL instructions, purpose expressions facilitate understanding them. In fact, given an instruction such as *Do α "for the purpose of" doing β*, the agent will try to find the connection between α and β, by exploiting the fact that β describes the goal to be achieved. One possible translation of this interpretation process into computational terms is:[8]

(8a) use β as an index into the plan library;
(8b) find a collection of methods $\mathcal{M}_l$ that achieve β;
(8c) *match* α to an action $\gamma_{l,j}$ that appears as a component in $\mathcal{M}_l$.

These are typical *plan recognition* inferences as applied to NL, e.g. see (Wilensky, 1983, Charniak, 1988, Litman and Allen, 1990).[9] In general, the main issue in plan recognition is to understand which goal β an action α is meant to achieve (Kautz, 1990), while in the case of Purpose Clauses the goal β is explicitly given. However, in almost all work on plan recognition, *match* in step (8c) is taken to mean that α is an *instance* of $\gamma_{l,j}$, while as discussed above such a direct match is rare: α may be more or less specific than $\gamma_{l,j}$, as in (4), or even inconsistent with it — a problem of *structural consistency* arises; or a certain relation $\mathcal{R}$ may hold between α and β, but only under certain expectations, as in (6a). The relation between α and $\gamma_{l,j}$ and the existence of expectations also affect the action ρ_{perf} the agent adopts as the object of his intentions: only when α is *instance of* $\gamma_{l,j}$ is $\rho_{perf} = \alpha$, otherwise ρ_{perf} has to be computed.

The necessity of a *flexible* match in plan recognition arises independently from NL processing, and has been discussed by Kautz (1990)[10] and by Weida and Litman (1994).[11] I will come back to the their approaches in Section 3.4, after I have discussed my own formalism. Within NL interpretation, the only researcher who deals with flexible matches is Charniak (1988). He divides *motivation analysis*, i.e. understanding an agent's actions in terms of his motivations, into four parts: 1) find possible motivations; 2) see if a suggested motivation is consistent with the action in question; 3) determine the additional information needed to integrate the action with the suggested motivation; 4) if there is more than one possible motivation, select the most plausible. Charniak notices that steps 1 and 4 are the most important, but also the most difficult and least understood; however, he adds that steps 2 and 3, which correspond to (8c) above, have not been solved either. Charniak recasts them in terms of matching roles in the hypothesized explanation (e.g., "the instrument of the action") against objects in the story (e.g., `telescope-22`), i.e. of matching

[8] For a different approach to understanding PC's see (Balkanski, 1993) on computing beliefs and intentions of the speaker and of the agent in declarative sentences containing rationale and means clauses.

[9] Pollack (1986, 1990) has convincingly argued that there are two views of plans, the *mental phenomenon* view, in which agents reason about mental states of other agents, and the *data structure* view, in which agents reason about the propositions and actions that may be objects of such mental states: to explain cooperative interactions of agents, in particular as applied to discourse (Grosz and Sidner, 1990, Lochbaum, 1994), one has to adopt the *mental phenomenon* view. The processing in (8) apparently belongs to the *data structure* view, as there is no reference to the beliefs and intentions of the agent: thus one would conclude that (8) is not appropriate to understanding NL instructions. I rather view the processing in (8) as the component that deals with reasoning about the domain; certainly there must be one such component in both the *mental phenomenon* and the *data structure* approaches. Also Lochbaum (1994), while subscribing to the *mental phenomenon* view, includes one such module in her model.

[10] Kautz also includes an application of his formalism to recognition of speech acts.

[11] Also see (Joshi and Rosenschein, 1976) for some related issues in the context of partiality of information in rule systems.

a Skolem function against a constant. Charniak's solution is an extension to unification that allows two terms to unify if their equality is consistent with everything known about them. As the reader will realize, Charniak's approach and mine are very close; the difference between the two is that Charniak seems to represent only planning, i.e. frame-like, knowledge, and not also linguistic knowledge as I do. Moreover, Charniak's sort hierarchy is limited to primitive types, while I will show that a hierarchy defined by subsumption between complex terms is necessary to interpret NL instructions.[12]

Other researchers, most notably Alterman and his students (Alterman *et al.*, 1991), (Carpenter and Alterman, 1994), have dealt with different kinds of flexible matches, at a "higher" level. Their view is that an agent has a basic set of skills and plans that are routinely used while acting; instructions are used as a resource only when the stored knowledge about plans cannot be adapted to the situation at hand. Thus, they concentrate on understanding the relation of the instructions, augmented with the current context, to the known plan, so that a new plan minimally different from the known plan can be derived. While the instructions dealt with are indeed complex, some linguistic information such as determiners appears not to be interpreted. Even if addressing every possible component of meaning is beyond any current system's coverage, the system's design should at least be flexible enough that these different sources of information can be easily integrated into it.

Before giving a preliminary description of my solution, I will provide some more detail on the two kinds of discrepancy I've studied so far.

Structural Consistency. Suppose that H knows a method, call it $\mathcal{M}_{37}$, to *create two triangles* by *cutting a square in half along the diagonal*:[13]

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

Suppose that the agent has to interpret instructions containing Purpose Clauses and "referring" to the knowledge expressed in $\mathcal{M}_{37}$, as in the following examples:

(10a) *[Cut the square in half along the diagonal with scissors]*$_{\alpha_1}$ *[to create two triangles]*$_{\beta_1}$.
(10b) *[Cut the square in half]*$_{\alpha_2}$ *[to create two triangles]*$_{\beta_2}$. (CC, 1988)
(10c) *[Cut the square in half with scissors]*$_{\alpha_3}$ *[to create two triangles]*$_{\beta_3}$.
(10d) *[Cut the square in half along a perpendicular axis]*$_{\alpha_4}$ *[to create two triangles]*$_{\beta_4}$.

If the agent applies the plan recognition paradigm sketched above in (8), $\mathcal{M}_{37}$ is retrieved through the match of β_i to δ_{37} in each of the cases in (10). Then, in each case, α_i and γ_{37} have to be compared, to compute the action ρ_{perf} to be performed. More specifically,

1. In (10a), α_1 is more specific than γ_{37}. ρ_{perf} can be taken to be α_1, but only *after* the added modifier *with scissors* is checked for consistency with γ_{37}.

[12] Such a hierarchy could presumably be built within an extended unification framework such as the one Charniak proposes: however, the description logic based systems that I advocate provide a full development environment that is generally not available in prototypes based on unification.

[13] This is just an informal approximation to how knowledge is really represented — see the actual representation of $\mathcal{M}_{37}$ in Figure 8.

2. In (10b), α_2 is less specific than γ_{37}, and is therefore consistent with it. The question remains of what ρ_{perf} should be, α_2 or γ_{37}. I propose that ρ_{perf} should be taken to be γ_{37}. β_2 is an instance of δ_{37}, which lends support to the conclusion that $\mathcal{M}_{37}$ is referred to; therefore, the action to be performed is γ_{37}, rather than the underspecified α_2 in the input instruction.
3. In (10c), α_3 lacks the modifier describing where to cut, but adds the instrument modifier *with scissors*, which is consistent with everything else we know about γ_{37} and β. ρ_{perf} is then inferred to be a composite description, derived from both α_3 and γ_{37}.
4. Finally, in (10d), α_4 is inconsistent with γ_{37}, and therefore ρ_{perf} cannot be computed.

Expectations. In the previous section, I illustrated some possible structural relations holding between α and $\gamma_{l,j}$. In all cases, ρ_{perf}, the action to be performed, is a constrained version of α. Additionally, the relation $\mathcal{R}$ between α and β — *generation* or *enablement* [14] — may be constrained to $\mathcal{R}'$ through the derivation of specific conditions. These conditions arise because of a complex interplay between the input instruction and the knowledge the agent has about the action to execute. In the *Animation from NL (AnimNL)* system, we call such conditions *expectations*. As argued in (Webber *et al.*, 1995), *expectations* are an important component of the complex process of translating from language to behavior in the environment. Instructions raise expectations that complement an agent's current perceptions in influencing its behavior; in turn, expectations can lead to further perceptual activities — not just observations, but activities that enable observations.

Obviously, instructions engender general expectations. The agent will assume that an instruction reliably describes a way of accomplishing a certain task in the world, and therefore, that whatever is not mentioned in the instruction is either obvious or not relevant to the task at hand. However, other more specific types of expectations may arise. (Webber *et al.*, 1995) describes expectations concerning processes, more specifically their termination conditions or their outcomes, and others regarding object locations. Here I will discuss the latter. An example was given above: in (6a), but not in (6b), an expectation $\mathcal{E}$ that the referent of the coffee urn is in the kitchen is developed.

Before discussing these examples and the expectations they engender in more detail, I will situate them in a broader framework, in which planning knowledge comes into play. In many planning formalisms, a distinction is drawn between two types of conditions, *preconditions* and *qualifiers*.[15] For an action to be executable, both kinds of conditions have to hold: however, if preconditions don't hold prior to execution, they must be achieved; on the contrary, qualifiers must hold to start with, and are not meant to be achieved. Procedurally, preconditions give rise to subgoaling, qualifiers don't. There are two reasons why qualifiers are not meant to be achieved: either the agent has no control over them, or it would be irrational to set out to achieve them. For example, *being sunny* is a qualifier on *recharge solar battery* on which (obviously) the agent has no control; *dirty object* is a qualifier on *wash* — see Figure 14 — that usually agents do not want to achieve just for the

[14] Informally, *generation* is a relation between actions that holds between α and β if β is done *by* executing α. *Generation* is often contrasted with *enablement*, which holds between α and β if α brings about conditions necessary to execute β — see (Goldman, 1970, Pollack, 1986, Balkanski, 1993). In (Di Eugenio, 1993b) I show that purpose clauses do express generation and enablement.

[15] Also called *applicability conditions* (Schoppers, 1988) or *constraints* (Litman and Allen, 1990).

sake of *washing* the object in question. Upon being given an instruction, if the agent trusts the instructor, all the qualifiers belonging to the agent's common sense plan for that action will give rise to expectations: if the agent is told (6b), his expectation is that the coffee urn is indeed dirty, otherwise, why would S tell him to wash it?

The process of computing the relation between α and $\gamma_{l,j}$ takes qualifiers into account as well, and in certain contexts, such as (6a), more complex expectations are generated. One qualifier on getting an object from x to y is that the object be at x to start with. Given that α specifies x as *the kitchen*, the locational expectation $\mathcal{E}_1$ = *urn in kitchen* develops. Also *wash* imposes a condition on the location of the object to be washed (that it be at the washing site): however, this condition is expressed as a precondition rather than as a qualifier, because if the object to be washed is not at the washing site, it should be brought there. Furthermore, *wash* will presumably have a qualifier constraining the location of the *wash* action itself to be a washing site; thus, in (6b) the expectation $\mathcal{E}_2$ = *washing site in kitchen* arises.

Locational expectations don't simply derive from knowledge about the action to be performed, namely, the qualifier in itself is not sufficient for such an expectation to arise. For example, neither (11a) nor (11b) convey much information about the location of the coffee urn or of the washing site.

(11a) *Get me the coffee urn.*
(11b) *Wash the coffee urn.*

Moreover, such expectations can't simply be attributed to *world knowledge*. For example, one explanation of the oddness of (12) is that one doesn't normally find washing equipment in the living room. As such, (12) indeed appears to convey $\mathcal{E}_2$ = *washing site in kitchen*, which goes against world knowledge.

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

In general, locational expectations arise if α is executed in order to do β, and if α changes the *perceptual space* $\mathcal{S}$ the agent has access to into $\mathcal{S}$'.[16] If there are objects that β's qualifiers require to be at $\mathcal{S}$', the expectation arises that they are at or in $\mathcal{S}$'. There are many naturally occurring examples of this phenomenon. One such (not involving a PC) is:

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

The action *remove access cover* opens up a new perceptual space, inside which the agent finds the cartridge.

Note the importance of deriving locational expectations. They are necessary for the system to ground referring expressions correctly — *grounding* is the process of linking a referring expression to an entity in the world (Webber and Baldwin, 1992). If the right expectation is not derived, grounding may be incorrectly carried out. Suppose that both S and H are in the living room ($\mathcal{S}$) when (6a) is issued. If no coffee urn is currently perceivable, the agent will derive $\mathcal{E}_1$, and assume that *the coffee urn* denotes an urn currently in the kitchen, one whose existence the agent may not even be aware of. Conversely, if a coffee urn is currently perceivable, the agent will

[16]The assumption is 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.

most likely be puzzled, and ask for clarification. Thus, if $\mathcal{E}_1$ were not derived, the system could wrongly ground the referring expression *the coffee urn* to a coffee urn in sight in the perceptual space $\mathcal{S}$ in which the instruction is uttered. In the same situation, in which S and H are in the living room, (6b) engenders different behavior. If a coffee urn is currently perceivable, H will ground the referring expression *the coffee urn* to the urn in sight; otherwise, he will develop the same expectation as in (6a) and assume the coffee urn is in the kitchen. The point is that (6b) doesn't convey such a necessary expectation $\mathcal{E}_1$ as (6a) does, and thus, my algorithm does not derive $\mathcal{E}_1$ when interpreting (6b) — $\mathcal{E}_1$ will be derived by other modules of *AnimNL*, if the current context warrants it.

In Section 4.2, I will show how the information provided by qualifiers, together with that provided by subsumption, is sufficient to derive the right expectations.

1.2. Proposed Solution

To summarize so far, I am interested in devising an action representation system that can support the interpretation of complex action descriptions; one of the problems to be addressed is that NL action descriptions seldom exactly match the knowledge that an agent has about actions and their characteristics. My solution is a formalism that consists of two components, the *Action Taxonomy* and the *Plan Library*, both implemented by means of the description logic based system[17] CLASSIC (Brachman *et al.*, 1991). The Action Taxonomy stores lexical semantic knowledge about actions, using primitives derived from Jackendoff's work (1990, 1991) on Conceptual Semantics. As Jackendoff's decompositional representation is partially motivated by the need to account for syntactic reflexes of conceptual structure, it is particularly useful to capture linguistic distinctions such as those illustrated above in Examples (1) and (2). The action terms defined in the Action Taxonomy are the components of the *recipes*, i.e. *common sense* plans about actions (Pollack, 1990),[18] stored in the Plan Library. The recipes in the Plan Library are the knowledge the system has about actually planning / acting in the world, real or simulated.

The choice of a description logic based system (DLBS) like CLASSIC is neither a lucky accident nor simply motivated by implementation concerns. I claim that DLBS's provide the right Knowledge Representation tools to support an algorithm that must compare different action descriptions, and deal with those that are not known to the system: in fact subsumption, the partial order on concepts DLBS's are based on, captures the correct hierarchical relation between action descriptions; moreover, the concept classifier, the algorithm that places a new concept in the appropriate location in the hierarchy by computing subsumption relations, provides the appropriate inferential basis to deal with different and possibly unknown action descriptions. Thus, using a DLBS helps to partly relax the assumption of completeness

[17] Once known, within the Knowledge Representation community, as *hybrid* systems or as systems based on *terminological logics*.

[18] I'm using the term *recipe* in the sense of Pollack's distinction between *recipes* and *plans* (1990, p. 79):

> There are plans and there are plans. There are plans that an agent "knows": essentially recipes for performing particular actions or for achieving particular goal states. And there are plans that an agent adopts and that subsequently guide his action. The distinction is between knowing that a plan for assassinating the president is shooting him, and actually planning to assassinate the president by shooting him. To keep matters straight, we can refer to what one knows when one knows a way to do something as a *recipe-for-action*.

of the Knowledge Base. The assumption is relaxed only as concerns processing the individual action descriptions derived from the logical form of the input: they can be unknown to the system, but they must be described in terms of the primitives that have been defined in the Action Taxonomy. Moreover, the Plan Library is still considered complete.

While I chose Jackendoff's theory of lexical semantics and DLBS's independently, I argue that integrating the two is beneficial to both: a linguistically motivated theory of lexical semantics is a first step towards making a general purpose knowledge representation formalism attuned to the needs of natural language semantics; conversely, embedding a linguistic theory into a computational framework with a clear semantics makes the linguistic theory usable in computational systems, and much more easily testable and extendible.

I will describe the action representation formalism in detail in Section 3, which also contains a review of related work on representing actions. Before that, in Section 2, I will describe the context of my work, the *AnimNL* project. Section 4 shows how the formalism supports the implementation of the inferences described in Section 1.1. Further details on all the topics discussed in this paper, and a linguistic analysis of Purpose Clauses, can be found in (Di Eugenio, 1993b). A linguistic analysis of other expressions that can convey *purpose*, such as *free adjuncts* and *until clauses*, can be found in (Di Eugenio and Webber, 1996).

2. THE ANIMNL SYSTEM

To contextualize my work, I will give a brief overview of its setting, the *AnimNL* (*Animation from NL*) system. The *AnimNL* project (Webber *et al.*, 1992, Webber *et al.*, 1995) 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 and maintenance procedures. The underlying animation system, *Jack*^{TM}, developed at the Center for Human Modeling and Simulation at the University of Pennsylvania, provides articulated, animated human figures — both male and female — capable of realistic motion through model-based inverse kinematics (Badler *et al.*, 1993). *Jack* agents can be anthropomorphically sized[19] at will and given different "strengths", so as to vary agents' physical capabilities. Different spatial environments can also be set up and modified, so as to vary the situation in which tasks are carried out. *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, thereby noticing design flaws that may only become apparent when people use the artifacts.

Figure 1 shows a schematic diagram of the system, which is still at the prototype stage.

Instructions are given to *AnimNL* in *steps*, where 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. While *AnimNL* now accepts as

[19]Namely, they are given properly sized attributes such as stature, lengths of limbs, weight, joint limits, fatigue rates etc. Each attribute can be independently varied, as long as they remain within permissible combinations for the target population: for example, 5^{th} percentile legs might never be found on anybody with 95^{th} percentile arms, even though the population allows such sizes individually. Values are extracted from anthropometric databases such as the Society of Automotive Engineers "standard people" database.

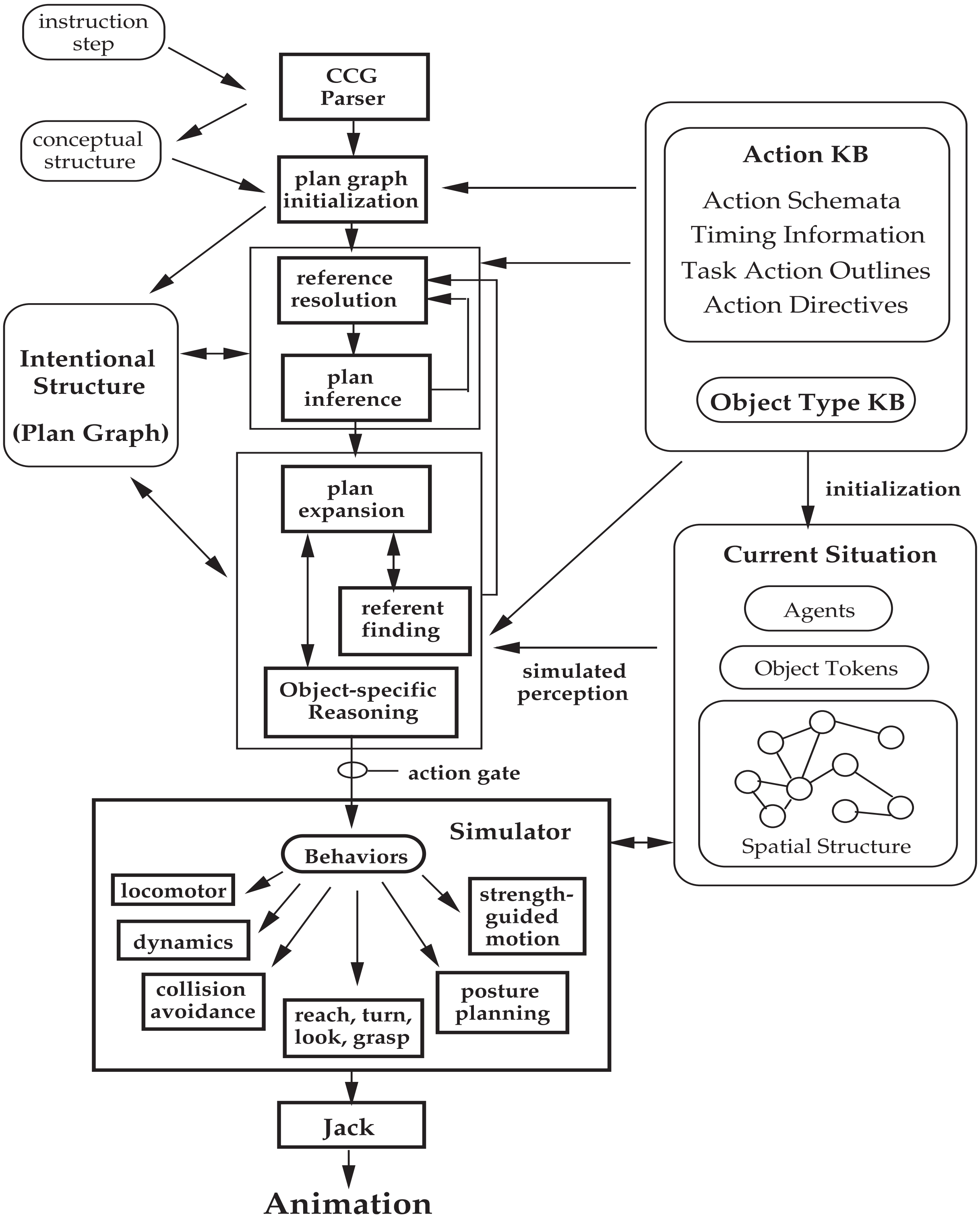

Figure 1. AnimNL System Architecture

a step only one (possibly complex) sentence at a time, a step should ultimately consist of one or more utterances. Steps are processed by a parser (White, 1992) that uses a Combinatory Categorial Grammar (Steedman, 1990) to produce a Logical Form (LF). The LF for the current step is incrementally developed into the *plan graph*, via processes of *reference resolution*, *plan inference*, *plan expansion*, *referent finding* and *object specific reasoning*. The *plan graph* represents *intentions*, where *intention* is meant as an intention to satisfy a given goal or act in a given way in a given situation, for a particular purpose, e.g. *goto(door1, open(door1))* "go to door1 for the purpose of opening it".[20] The lower-level planning processes below the action gate in Figure 1 determine how the agent should *move* in order to carry out the specified intention on the specified objects for the specified purposes. Their output may either be a collection of *behaviors* to be executed (simulated) in parallel [21] or an indication that the agent's "body" is unable to carry out the behavior that its "mind" would like it to.

With respect to Figure 1, the Action Taxonomy and Plan Library that I will describe in Section 3 is collectively labeled *Action Schemata* in the *Action KB* box. The algorithm discussed in Section 4 performs all *plan graph initialization* and a subset of the *plan inference* processes. It produces the initial *plan graph*, namely, it models what intentions the agent adopts simply based on the input instruction and the stored knowledge, but without taking the current situation into account. As an example, consider the unambiguous variation of (6a), *Go into the kitchen* **to** *get me the coffee urn* — *AnimNL* interprets *to* rather than *and*, which is multiway ambigous. My algorithm builds a *plan graph* that contains nodes derived from the substeps of the recipe in Figure 5, as shown in Figure 13. This is a first step in the hierarchical expansion of the plan, which will be augmented by the *plan expansion* module (Geib, 1995) e.g. with a step *open the door* if the door to the kitchen is closed. The *object specific reasoner* (Levison, 1996) has the knowledge and performs the reasoning necessary to generate the correct action decomposition, for example with regard to the type of door to be opened: different movements have to be performed to open a swinging door or a sliding door. The *referent finder* (Moore, 1993) grounds referential NP's to objects in the world, and actively looks for objects when they are expected to be in one place, but are not immediately visible: in case the coffee urn of Example (6a) is not visible to the agent after he enters the kitchen, this module will generate a recursive search through possible containers such as cabinets.

A caveat before proceding. The issue we concentrated on in *AnimNL* was the vertical integration of all the different modules in Figure 1, including the animation package. Thus, of all the examples that I'll discuss, only variations of (6a), in the form *Go into the X to get me the Y*, where X is a room, and Y is an object, have been processed to produce animations. Variations on this single example are sufficient to produce a number of different animations. Just varying X and Y requires several issues to be addressed, such as how to plan the agent's path, which movements the agent performs to actually get control of Y, etc. Moreover, variations in the input interact with variations in the current situation, such as the agent's location when the instruction is uttered, and the specific location of the object Y within the room

[20]The term *intention* in *AnimNL* is used to refer to a mental state that is admittedly simplified with respect to the notion of intention in e.g. (Bratman *et al.*, 1988, Pollack, 1990, Grosz and Sidner, 1990). Such simplification is warranted because (at least for the time being) a *master-slave* relation between instructor and agent is assumed.

[21]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.

X, as discussed above with respect to the coffee urn and the kitchen. Furthermore, a simplified version of *AnimNL*, *SodaJack* (Geib *et al.*, 1994) — so called because Jack is shown working at a soda fountain, serving sodas and ice-creams — produces several other animations. *SodaJack* is simplified in that it accepts only simple main clauses such as *Give me a soda* via a menu system.

To conclude, it is clear that my work cannot be evaluated in terms of how many examples are correctly animated by *AnimNL*: to animate every example for which my module produces a PlanGraph, the other parts of the symbolic KB, and in certain cases even the simulated behaviors, should have been greatly expanded — altogether an infeasible feat. Thus, in Section 4.3 I will evaluate my module independently of *AnimNL*. This perhaps makes the evaluation less convincing. On the other hand, it was worthwhile to develop my work within *AnimNL*, and it is worthwhile to present it within that framework. Much semantics and / or discourse work is developed independently of larger systems or applications.[22] Legitimate questions then arise, regarding the treatment of many problems impinging on the work — parsing, the interpretation of referring expressions and of verbal aspect, just to name a few. I could safely abstract away from those, as they are dealt with elsewhere in the system.

3. THE REPRESENTATION FORMALISM

3.1. Motivations

From the observations and examples provided so far, the following should be clear. First, as the number of possible action descriptions in NL is very large, it is impossible to define all possible action descriptions a priori; thus, it must be possible to compare action descriptions one to the other. This is why I chose a flexible action representation formalism that provides a partial order on descriptions and that is able to deal with new action descriptions.

Second, while planning notions such as *substep* or *qualifier* are necessary to understand NL instructions, they are not sufficient, and linguistic knowledge must also be represented. I adopted a lexical semantics approach to the representation of linguistic knowledge and I keep linguistic and planning knowledge separate.

Types of knowledge. In general, at least four types of knowledge are possible. They depend on:[23] the type of knowledge itself, whether *linguistic* or regarding *acting in the world*; and the objects the knowledge concerns, namely, *actions* (considered atomic) or *plans* (composed of actions). Apparently, I concentrate on only two of the four possible types: *linguistic* knowledge about *actions* and *acting* knowledge about *plans*. The former will be equated with *lexical semantics* knowledge about action verbs; this will make apparent that from a linguistic point of view actions are not atomic. As regards knowledge about acting in the world, it actually includes *actions* as well: a notion of *basic-act-type*, i.e. an atomic action which is not decomposed into substeps, is provided — see Figure 4 and the discussion in Section 3.3. I will use the expression *planning knowledge about actions* to refer to *acting* knowledge about both *actions* and *plans*.

The issue of *linguistic* knowledge about *plans* is far more complex. Naturally,

[22] As the time goes by, this is less and less common.
[23] An anonymous reviewer raised this issue.

language has tools for describing plans, and in fact, purpose expressions like the PC's I studied are a means of referring to components of plans. One of the goals of instructional text is that agents acquire new plans, whose descriptions include e.g. causal and temporal relations between actions. However, it seems that there is no clear repository for such knowledge in a computer system, but that it should be distributed among different knowledge sources and modules: the grammar will associate certain possible meanings with certain connectives, the discourse processor will recognize certain hierarchical embeddings between actions, subplans and plans, etc. The multifaceted nature of *plans* as objects in a discourse is discussed in (Traum *et al.*, 1996).

Lexical semantics. The Action Taxonomy contains linguistic, i.e. *lexical semantics*, knowledge about action verbs. It is clear that *linguistic* knowledge about a certain lexical item has many different facets, of which lexical semantics is just one. However, focusing on lexical semantics makes sense for the current enterprise.

First, as I'm interested in interpreting NL instructions, the meanings of the words that compose an instruction seem to be a natural starting point for interpretation.

Second, the rationale behind much current research in lexical semantics (Jackendoff, 1990), (Levin and Rappaport Hovav, 1992) is that components of meanings affect syntactic structure. For example, the *conative construction*, exemplified in (14b), is possible only with verbs that specify the *manner* of an action that results in a change of state of an object, such as *cut*, but not with those that leave the manner unspecified, such as *break*:

(14a) *Fran* **cut / broke** *the bread.*
(14b) *Fran* **cut at / * broke at** *the bread.*

Given that, as we saw e.g. in Examples (1) and (2), different surface structures provide different information, the surface structure itself can hint at the relevant meaning component, even if the relation between meaning components and their syntactic reflexes is many to many.

Within lexical semantics, work on *lexical decomposition* is the closest to my concern of providing a principled representation for action verbs. According to Fass and Pustejovsky in (Shapiro, 1992, p.806), lexical decomposition is that branch of lexical semantics concerned with the internal semantic structure of lexical and conceptual items within a lexicon. I chose Jackendoff's theory of Conceptual Semantics (1990, 1991), discussed below.

Using linguistic theories of lexical decomposition, and Jackendoff's in particular, may make some AI practitioners cringe. For example, Wilks in (1992) dismisses Conceptual Semantics as a variation on AI decompositional formalisms for NL, such as Schank's Conceptual Dependency (1975), and thus concludes that "it reinvents the wheel". In reality, the two enterprises are quite different. In his answer to Wilks' criticisms, Jackendoff (1992) notes that linguistic theories of lexical decomposition try to account for a wide variety of linguistic data, while many AI implementations focus on a limited corpus. More importantly, one essential criterion to evaluate a linguistic theory is that it must be constrained enough to explain the ungrammaticality of sentences that are in principle plausible. As Dorr and Voss point out in (1993, p.379, fn.14), this means that the current research challenges in lexical semantics include what can be represented as a predicate, the definition of what constitutes a valid argument, the number of arguments, the obligatory or optional nature of those arguments. These issues were not addressed in earlier work, neither by linguists working on case like (Fillmore, 1968), nor by AI practitioners.

Lexical semantics and planning knowledge. The requirement to keep linguistic and planning knowledge separate derives partly from linguistic considerations, and partly from issues of modularity. While this distinction is intuitively plausible, it is difficult to find clear criteria to distinguish lexical and world knowledge, and different researchers carve up the two kinds of knowledge differently. For example, Jackendoff (1990) claims that the geometric features that distinguish verbs of manner of motion such as *run* and *jog* do not pertain to Conceptual Semantics, but to a 3D-model that is accessed by the visual faculty; on the other hand, B. Levin in (1993) includes the sound type (shrill, low or high) in the lexical entries for verbs of sound.

Within Computational Linguistics, the clearest arguments in favor of such distinction come from the Machine Translation literature. L. Levin and Nirenburg in (1992, 1994) propose two types of lexical semantics, syntax-driven (SDLS) and ontology-driven (ODLS). SDLS pertains to the relationship between predicates and their arguments, and it is composed of Lexical Conceptual Structures (LCS) and Linking Rules. LCS's, similar to Jackendoff's Conceptual Structures, represent syntactically relevant semantic information, such as the roles and the number of arguments to a predicate. Linking rules associate grammatical functions with arguments in an LCS: for example, the *experiencer* argument of *to like* is its surface subject, while the same argument of its Italian counterpart, *piacere*, is expressed as the indirect object. ODLS specifies meanings of lexical units, and suggests how they contribute to the overall representation of text meaning. Both SDLS and ODLS are necessary. SDLS is not sufficient because radically different LCS's encode the same phenomenon in different languages, and therefore it is not possible to map both source and target language onto the same LCS: for example, in English main and auxiliary verbs encode important aspects of meaning, such as phase — *begin, continue, finish* — or modality — *must, expect, try*; languages like Japanese encode the same phenomena through suffixes on the verb. On the other hand, ODLS is not sufficient either, as it doesn't encode the predicate argument structure, which is necessary for both interpretation and generation.

In my work, I adopt a position similar to L. Levin and Nirenburg's. My action taxonomy can be considered equivalent to their SDLS, although I haven't encoded linking rules yet, and my plan library, to their ODLS. My rationale for doing so is to keep distinct the NL encoding of action descriptions, and their conceptualization as elements of acting in the world. I distinguish between the two by drawing on the one hand on what linguists such as Jackendoff and B. Levin include in the definitions of verbs, and on the other, on the representation of action operators in the AI planning tradition. Future work will undoubtedly show whether such division needs further refinement.

Description Logics. I propose a KR system based on a description logic such as CLASSIC (Brachman *et al.*, 1991) as the underlying backbone of my formalism. DLBS's — descendants of the research on KL-ONE (Brachman and Schmolze, 1985), (Woods and Schmolze, 1992) — are composed of a *Terminological Box*, or T-Box, that is used to define terms, and of an *Assertional Box*, or A-Box, used to assert facts or beliefs. The T-Box has its own *term-definition language*, which provides primitives to define *concepts* and *roles*, which express relations between concepts. The T-box 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 (KB). The *realizer* in the A-Box also computes subsumption, in this case between a concept and an individual, to infer which concepts a certain individual asserted in the A-Box is an instance of.

I chose a DLBS because subsumption induces the right partial order on action descriptions, and having a virtual lattice makes it possible not to define all possible action descriptions a priori. Moreover, CLASSIC is used to represent planning knowledge as well.[24] Thus, the action terms defined in the Action Taxonomy can be directly used as primitives in the recipes in the Plan Library, and recipes can be naturally kept organized in a hierarchical structure.

The CLASSIC query language is used within the algorithm to interpret PC's. First, the individual action descriptions which are part of the logical form are asserted into the A-Box; for each such description, the A-Box recognizer computes the set of concepts the description instantiates. Computing subsumption is exploited to retrieve the $\texttt{Recipe}_l$'s of whose header the goal β is an instance, and then, to compare α to $\gamma_{l,j}$, as I will describe in Section 4. Incidentally, the role of the A-Box in my system is fairly limited: it is only used to assert the descriptions of actions in the input instruction, and to compute of which known concepts such descriptions are instances.[25]

Finally, note that while I advocate DLBS's in general, I won't assess CLASSIC in particular. To do so would require a careful comparison between CLASSIC and other systems such as LOOM (Mac Gregor, 1988) or BACK (Quantz and Kindermann, 1990). In reality, the choice of CLASSIC over other systems was mainly due to practical reasons: CLASSIC was (and is) readily available, easy to install, and written in CommonLisp, the language in which the top level part of *AnimNL* is also written.

3.2. The Action Taxonomy

The theory of lexical semantics that I chose is Jackendoff's Conceptual Semantics, that he formalizes by means of constructs called Conceptual Structures (CS's). CS's are well suited to represent action descriptions. First, Conceptual Semantics takes the syntactic structure of the sentence seriously and tries to explain facts of linguistic distribution in terms of meaning components, as discussed above with respect to Example (14). Second, CS's are suitable to express the logical form of an instruction, as they make apparent which information may be missing from the utterance; this information will have to be provided by inference. This is particularly important as *AnimNL* is concerned with interpreting an action description down to its (simulated) execution. Moreover, CS's are well suited as the semantic representation associated to the Combinatory Categorial Grammar used by the *AnimNL*'s parser (White, 1992). Finally, the CS primitives capture generalizations about action descriptions and their relationships to one another, such as that *carry* is *move object* augmented with a specific physical means of moving the object. This makes them suitable for a hierarchical representation.

Before showing how the T-Box employs CS's, I will illustrate CS's by showing how the CS for *Go into the kitchen* is built.

[24] Thus, the distinction between *Action Taxonomy* and *Plan Library* is conceptual rather than real.

[25] In CLASSIC the distinction between T-Box and A-Box is somewhat blurred, as the language to describe terms is the same as the language to describe individuals.

Conceptual Semantics postulates the existence of the following types: *Thing, Place, Path, Event, State, Manner* and *Property*.[26] Each entity belongs to one of the types. The CS for a KITCHEN is shown in (15):

(15) $[_{\text{Thing}}\ \text{KITCHEN}]_k$

Square brackets indicate an entity of type *Thing* meeting the enclosed featural description. Indexes such as k are used to distinguish instances of a type. Small caps indicate atoms in CS, which serve as links to other systems of representation; for example, the CS for a room would differ 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, as already mentioned above with regard to manner of motion verbs. In the context of my work, some of these distinctions, such as that between KITCHEN and ROOM, can be partly captured by means of the taxonomy, in that it can at least be encoded that a KITCHEN is a type of ROOM.

Complex CS's are generated by conceptual functions applied to other CS's.

The place *in the kitchen*, shown in (16a), is derived by applying the conceptual function IN: Thing → Place. Analogously, (16b) represents a path whose end point is in the kitchen, and is obtained via the function TO: Place → Path:

(16a) $[_{\text{Place}}\ \text{IN}([_{\text{Thing}}\ \text{KITCHEN}]_k)]_l$

(16b) $[_{\text{Path}}\ \text{TO}([_{\text{Place}}\ \text{IN}([_{\text{Thing}}\ \text{KITCHEN}]_k)]_l)]_m$

The event of *going into the kitchen* is derived by applying the function
GO: Thing × Path → Event to (16b):

(17) $[_{\text{Event}}\ \text{GO}([\text{YOU}]_i,\ [_{\text{Path}}\ \text{TO}([\text{IN}([\text{KITCHEN}]_k)]_l)]_m)]_p$

Among other functions yielding events, CAUSE will be used in the CLASSIC definition of action:

(18) $[_{\text{Event}}\ \text{CAUSE}([_{\text{Thing}}]_i,\ [_{\text{Event}}\ ]_j)]$

An added level of generality is obtained in Conceptual Semantics via *semantic fields*. 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.

For example, a CS primitive like GO represents the generic concept of change of state,[27] while the semantic field specifies the domain to which the change applies. In

[26] This set of types is derived from (Jackendoff, 1990). (Jackendoff, 1991) includes *Time* and *Amount* among the types.

[27] See (Jackendoff, 1990, p. 91-95) for other change of state primitives such as INCHoative.

(19a), which represents a sentence like *Jack went into the kitchen*, the change takes place in the spatial domain; (19b), which is part of the meaning of a sentence such as *Tom sold a car to Jack*, represents a change of ownership.

(19a) $[\mathrm{GO}_{\mathrm{Sp}}([\textsc{jack}], [\mathrm{TO}([\mathrm{IN}([\textsc{kitchen}])])])]$

(19b) $[\mathrm{GO}_{\mathrm{Poss}}([\textsc{car}], [\mathrm{FROM}([\mathrm{AT}([\textsc{tom}])])], [\mathrm{TO}([\mathrm{AT}([\textsc{jack}])])])]$

For our *AnimNL* application, another semantic field — *Control* — turned out to be necessary. As the instructions of interest in *AnimNL* concern direct physical manipulation of objects, it is important to represent the notion of *having control over* some object. (20a) and (20b) show the difference between *Bill has a car* and *Bill has the ball*, as in a soccer match:

(20a) $[\mathrm{BE}_{\mathrm{Poss}}([\textsc{car}], [\mathrm{AT}([\textsc{bill}])])]$

(20b) $[\mathrm{BE}_{\mathrm{Ctrl}}([\textsc{ball}], [\mathrm{AT}([\textsc{bill}])])]$

CS's are directly representable in CLASSIC. Figures 2 and 3 show part of the Action Taxonomy I have implemented. The two figures follow well established graphical conventions used to represent KB's based on DLBS's (Woods and Schmolze, 1992, p. 9). *Concepts* are represented by ellipses, *individuals* by shaded ellipses, and *roles*, i.e. relations between concepts, by small circles containing inscribed squares. *Primitive concepts* are marked by asterisks. Attached to concepts are roles and structural conditions, namely, relations between fillers of different roles. Attached to roles are value restrictions and other role "facets", like names and number restrictions, that represent the arity of the relation — note that, in both Action Taxonomy and Plan Library, every role other than `manner` is an *attribute*; thus, the number restriction, often omitted, is `(1,1)`. Value restrictions on roles are indicated by directed arrows pointing from the role to the concept that constrains possible fillers. SAME-AS conditions are indicated by diamond shaped lozenges enclosing an equal sign: they represent equality of role fillers — this is the restricted type of structural condition that CLASSIC allows. For example, the SAME-AS restriction associated to the concept *circular-path* in Figure 2 restricts the fillers of the two roles `source` and `destination` to be the same. Finally, 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; thick dotted lines connect an individual to the concepts of which it is an instance.

Figure 2 includes concepts belonging to three taxonomies: *entity*, *place* and *path*, Figure 3 adds a portion of the *action* taxonomy. Note that the roots of these taxonomies — *entity*, *place*, *path*, *event* and also *state*, not shown here — are all primitive daughters of the predefined concept *classic-thing*. Other than these five concepts, all the ones belonging to the *entity* taxonomy, but none of those appearing in the other taxonomies, are primitive. It is not surprising that subconcepts in the *entity* taxonomy are primitive. Many such concepts correspond to natural kinds, for which it is often impossible to provide necessary and sufficient conditions: this explains the standard DBLS practice of defining them as primitive. I also define geometric figures as primitive, even if necessary and sufficient conditions exist for them. However, encoding definitions such as *A square is a polygon with exactly 4*

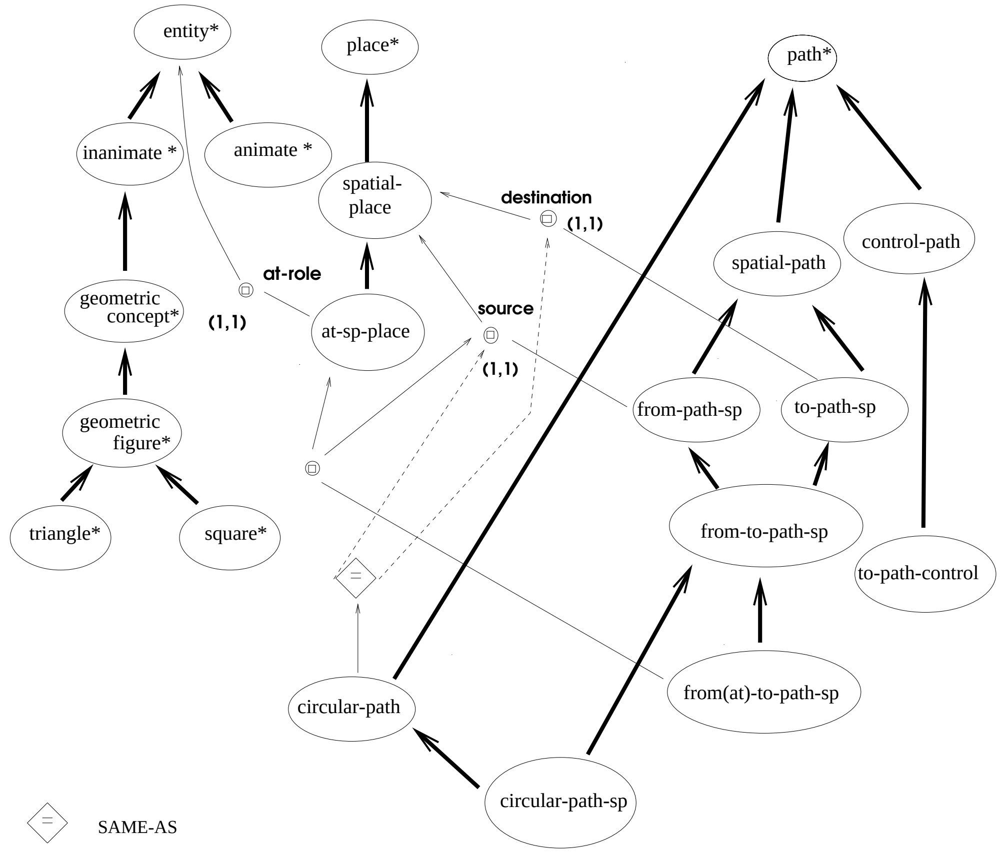

Figure 2. Portions of the *Entity, Place* and *Path* Hierarchies

sides of equal length and 4 right angles is not completely straightforward; so, given that my primary interest was in defining actions, rather than entities, I simplified the *entity* taxonomy. Providing definitions for all the concepts belonging to the other subtaxonomies rather than regarding them as primitive is made possible by the fact that these concepts encode decompositional semantics definitions of words, rather than the words themselves.

I will now provide more detail about the different subtaxonomies, starting with the one rooted in *place*. The concepts belonging to this subhierarchy represent conceptual functions of the form F: Thing → Place, such as AT, IN, ON. Different semantic fields determine different kinds of places (and paths).[28] Figure 2 only includes the concept *spatial-place*, and its subordinate *at-sp-place*, which represents the AT conceptual function. The single role `at-role` on *at-sp-place* has exactly one filler, of type *Entity*.

[28] I represented semantic fields by means of a role `semfield-role` — not shown in these or later figures — whose value restriction is the concept *semfield* defined by enumeration.

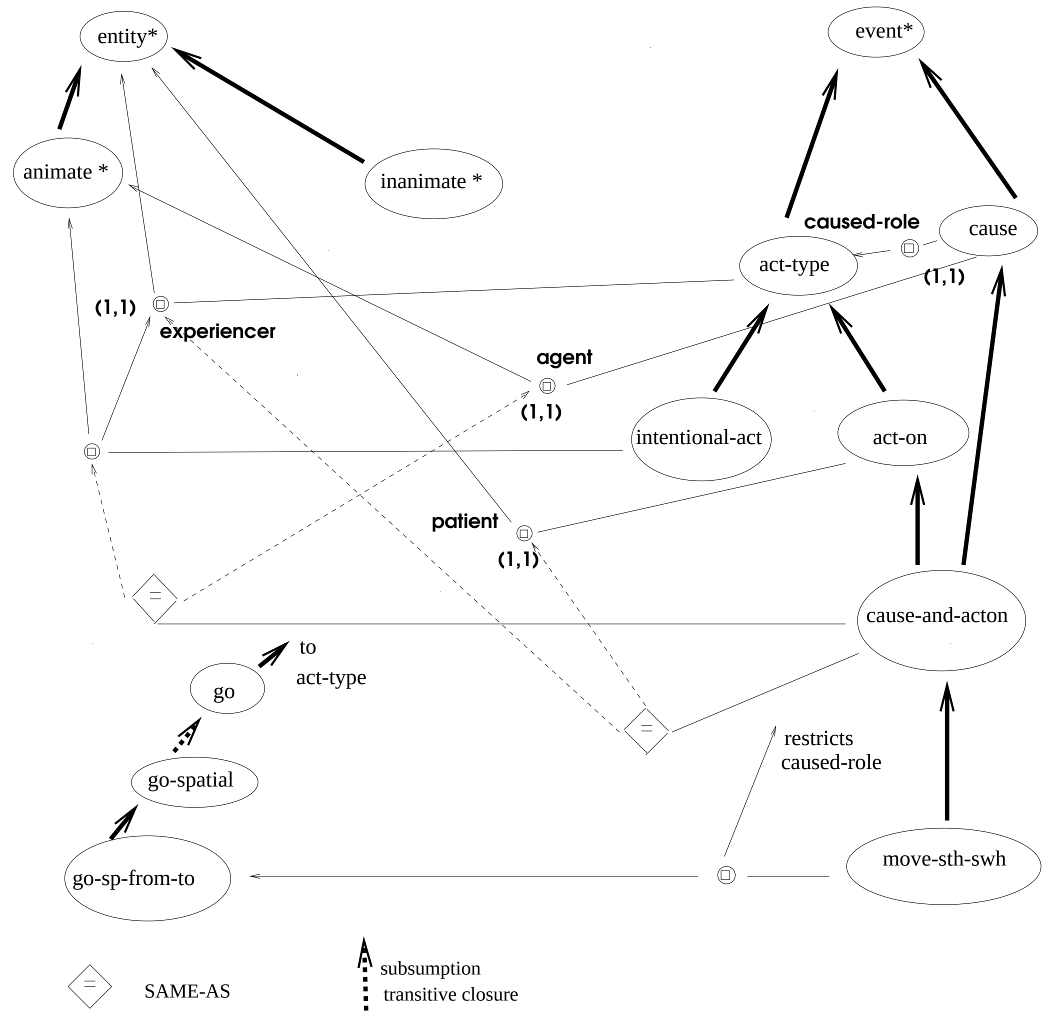

FIGURE 3. A portion of the *Action* Hierarchy

Functions yielding *Path*s are represented by the concepts under *Path*. *from-to-path-sp(atial)* demonstrates multiple inheritance, as it has two parent concepts, *to-path-sp* and *from-path-sp*. *from-to-path-sp* has two roles, `source` and `destination`, which both have a *spatial-place* as filler. The concept *from(at)-to-path-sp* restricts the role `source` inherited from *from-to-path-sp* to be filled by *at-sp-place*. It therefore corresponds to the following CS:

$$\begin{bmatrix} \text{FROM}_{\text{Sp}}([\text{AT}_{\text{Sp}}([_{\text{Thing}}])]) \\ \text{TO}_{\text{Sp}}([_{\text{Place}}]) \end{bmatrix}$$

Figure 3 shows a small portion of the action subhierarchy. The two *thematic* and *action* tiers discussed in (Jackendoff, 1990) are kept distinct, and included in the definition of actions by defining *cause-and-acton* as daughter of both *cause* and *act-on*. The concept *cause-and-acton* corresponds to the intuitive notion of *action*. *cause* has two roles, the `agent`, restricted to be *animate* and the `caused-role`, restricted to be an *act-type*. This further restricts Jackendoff's definition of CAUSE in (18), which allows the "agent" to be a *Thing*, therefore also an inanimate entity, and the "theme" to be an *Event*. I introduced these restrictions just for the sake of simplicity, and they could easily be removed. They reflect the characteristics of my application domain, that deals with actions performed by intentional agents only. Also note that the definition of *cause-and-acton* constrains the role `agent` inherited from *cause* to be the SAME-AS the role `experiencer` inherited from *act-type* via *act-on*; and the role `patient` inherited from *act-on* to be SAME-AS the role `experiencer` on the filler of the role `caused-role` inherited from *cause*. The corresponding CLASSIC definition makes this clear:

```
(cl-define-concept 'cause-and-acton
                   '(and cause act-on
                         (same-as experiencer agent)
                         (same-as patient (caused-role experiencer))))
```

move-sth-swh — for *move something somewhere* — is subsumed by *cause-and-acton*. Its individuating restriction is the filler of the `caused-role`, *go-sp-from-to*. The latter is an *act-type* of type *go-spatial*, with the role *path-role* restricted to be *from-to-path-sp*; the `experiencer` role on *go-sp-from-to* is the moved object. The CLASSIC definition of *move-sth-swh* is a direct translation of its CS representation, shown in the header in Figure 5.

Note that, while I chose CS's and DLBS's for different reasons, the integration of the two formalisms is beneficial to each. The usage in the T-Box of linguistically sound primitives transforms the T-Box into a real lexicon, or at least, into a KB onto which logical forms can be easily mapped to; conversely, the DLBS provides CS's with a computational basis, by supplying a hierarchical organization and the possibility of dynamically extending the lexicon.

3.3. The Plan Library

So far I have only discussed how to represent *lexical semantics* knowledge about action verbs. *Planning* knowledge is encoded in the Plan Library as simple recipes — the term *recipe* is defined in footnote 18. The Plan Library is implemented in CLASSIC as well. This has the following advantages, besides providing a uniform knowledge representation language for the whole KB: first, classification can be exploited to provide a taxonomy of action recipes; second, the CLASSIC query language is used to index recipes through the goal β.

The syntax of recipes is described in Figure 4, and one instance of a recipe, which illustrates a method for *move something somewhere*, is shown in Figure 5. Figure 5 uses the CS syntax rather than the corresponding CLASSIC definition, shown in Figure 6, for readability. In Figure 4, the terminals that are not relations — *basic-act-type, act-type, state* — are concepts belonging to the Action Taxonomy, and thus should be interpreted as type restrictions on components of the recipes.[29]

[29] *basic-act-type* doesn't appear in Figure 3 which contains just a small portion of the action hierarchy. As far as *state* is concerned, a subtaxonomy rooted in *state*, a primitive concept immediately subsumed by *classic-thing*, has been defined. The sentence *state belongs to the Action Taxonomy* above of course doesn't

RECIPE	$\rightarrow$	BASIC-RECIPE | NON-BASIC-RECIPE
BASIC-RECIPE	$\rightarrow$	BASIC-HEADER QUALIFIER* EFFECT$^+$
NON-BASIC-RECIPE	$\rightarrow$	HEADER BODY QUALIFIER* EFFECT$^+$
BASIC-HEADER	$\rightarrow$	basic-act-type
HEADER	$\rightarrow$	act-type
BODY	$\rightarrow$	act-type$^+$ ANNOTATION*
ANNOTATION	$\rightarrow$	act-type$_1$ enables act-type$_2$ | act-type$_1$ TEMP-REL act-type$_2$
QUALIFIER	$\rightarrow$	state
EFFECT	$\rightarrow$	state
TEMP-REL	$\rightarrow$	before | meets | overlaps ...

FIGURE 4. The Recipe BNF

Following established planning practice, recipes comprised a *header*, *body*, *qualifiers* and *effects*. However, the relations between these components are expressed in terms of *enablement* and *generation*, e.g. the body *generates* its header. Of course, there may be many recipes with the same header; the recipe in Figure 5 is just one of those possible for moving an object from one location to another (it is the only one I have included so far).

Basic-recipe and *non-basic-recipe* are kept separate as a consequence of having to provide a base case for the representation, via *basic act-types*. As (Pollack, 1986, p.59) points out, it is rather difficult to pinpoint what a *basic act-type* is. In my work, I assume as basic all those CS Event functions that don't have another event as argument, such as GO and all its descendants. Furthermore, those distinctions between act-types that cannot be captured in terms of CS representations, such as that between *run* and *jog* discussed in Section 3.1, and PHYSICAL-WASH in Figure 14, give rise to basic act-types. Ideally, for all of them *AnimNL* should provide further decompositions into lower level actions.

A glaring difference with traditional planning representations is the absence of preconditions. In the literature, preconditions are mainly used to express conditions that have to be true before an action is executed, and that give rise to subgoaling. In my recipes, preconditions are indirectly represented through some of the substeps in the body that generates a certain header: those substeps are the actions that will make those conditions true. Preconditions have also been used for auxiliary functions, such as ordering substeps in a plan: these auxiliary functions are performed by the *annotations* on the body, that specify relations between substeps, such as *enablement* or simply temporal order — see Figures 5 and 14 (the temporal relations are based on (Allen, 1984)). Note that the temporal order between substeps need not be total, as shown in Figure 14.

There are several reasons behind the choice of not having preconditions. The

mean that a *state* is an *act-type*, but that the *state* taxonomy belongs to the part of the KB that deals with the lexical semantics of verbs.

Header	$[\mathrm{CAUSE}([\textsc{agent}]_i, [\mathrm{GO}_{\mathrm{Sp}}(j,k)])]_\delta$ $\left[\begin{array}{l} \mathrm{FROM}(m) \\ \mathrm{TO}(l) \end{array}\right]_k$
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 5. A *Move Something Somewhere* Recipe

first is to keep the mapping between surface form and stored knowledge more direct, given that NL instructions generally describe actions and not states. Second, it is difficult to distinguish in a principled way between preconditions and substeps in the body of a recipe. One could argue that the body of a (physical) *Move* recipe simply consists of a *transfer* of an object from one place to another, and that a precondition for a *Move* is having control over that object. However, consider a heavy object: the agent will start exerting force to lift it, and then will 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.[30] Other motivations behind this choice have to do with the *AnimNL* system as a whole — see (Webber *et al.*, 1992, Geib, 1995).

Finally, the other components of recipes are *qualifiers* and *effects*. *Qualifiers*, already discussed in Section 1.1, are those conditions on an action that must hold to start with; they are not meant to be achieved, either because the agent has no control over them, or because it would be irrational to do so. *Effects* are what must be true after an action has been executed.[31] As I mentioned, NL instructions describe

[30]Other researchers are indeed trying to capture the difference between preconditions and substeps, and therefore they are encoding them both in their plan representation (Delin *et al.*, 1994, Young and Moore, 1994).

[31]It is clear that recipes cannot be complete, as in general both the qualifier and the effect lists only

```
(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 go-spatial.enables.get-control)
           (all annot2 get-control.enables.go-spatial-with)
           (all qualif1 be-spatial)
           (all effect1 be-spatial)
           (same-as (header agent)(substep1 experiencer))
           (same-as (header agent)(substep2 agent))
           (same-as (header agent)(substep3 experiencer))
           (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 annot-argument1))
           (same-as substep2 (annot1 annot-argument2))
           (same-as substep2 (annot2 annot-argument1))
           (same-as substep3 (annot2 annot-argument2))
           (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))
      ))
```

FIGURE 6. The *Move* Recipe in CLASSIC

actions rather states; however, *effects* are explicitly encoded as they are necessary for plan execution. Most effects could actually be inferred on demand, if axioms relating *act-types* and the *states* they bring about were available — also (Jackendoff, 1990) mentions inference rules of this sort. However, it is more efficient to encode effects explicitly, as if they were compiled out in advance.

The CLASSIC implementation. A few words now on the CLASSIC implementation of the BNF in Figure 4 — note that when the work described in this paper was being carried out, only CLASSIC 1.0 was available, and thus these implementation notes refer to that release. Figure 6 shows the CLASSIC definition for the *Move* recipe concept.[32] The most natural way of translating the BNF for *recipe* into CLASSIC would be to define a topmost concept *recipe*, with roles `header`,

include some necessary conditions, and are not is exhaustive. (Genesereth and Nilsson, 1987), among others, discusses the related issues of the *qualification* and *frame* problems in AI.

[32]Number restrictions are missing in Figure 6 because, as mentioned in Section 3.2, all roles are attributes.

`substeps`, `annotations`, `qualifiers` and `effects`, properly restricted. For example, `substeps` would be restricted as follows:
(**ALL** `substeps intentional-act`) (**AT-LEAST** `1 substeps`)
Unfortunately, as CLASSIC 1.0 didn't provide for role *differentiation*, it was impossible to express that each substep of the *Move* recipe must have a different value restriction, *go-spatial*, *get-control* and *go-spatial-with* respectively. Thus, the *recipe* concept was defined as having only a `header`, with the substeps appearing only on the subconcepts of *recipe*, e.g. *recipe-1-step* and *recipe-3-steps*. Similar considerations hold for annotations, qualifiers and effects.[33] [34] Coreference between "parameters", such as fillers of roles on a qualifier and on a substep, is expressed by means of `SAME-AS` restrictions, as shown in Figure 6. Relations between substeps — for example, the *enablement* relations between $\texttt{substep}_i$ and $\texttt{substep}_{i+1}$ — are expressed by means of $\texttt{annot}_i$. Incidentally, *enablement* entails *temporal precedence*, therefore the three substeps are totally ordered, $\texttt{substep}_1 < \texttt{substep}_2 < \texttt{substep}_3$. Thus note that the correspondence between the ordering and the substep names is a coincidence: relations holding between substeps are encoded *only* by means of annotations, not implicitly via the lexicographic order of the substep names.

New recipes can be added to the Plan Library by exploiting the classifier to maintain the hierarchical organization: for example, a $Move_2$ recipe with further qualifiers, or with a more specific *substep1*, would be correctly classified as a descendant of the *Move* recipe in Figure 5.

All in all, the current KB comprises about 130 concepts, of which 14 are state concepts, 44 action concepts, and 9 recipe concepts, and 35 roles.

3.4. Related work

Above, I already commented on the difference between linguistic theories of lexical decomposition, and AI decompositional theories of meaning à la Schank (1975). I pointed out that the two approaches put their emphasis on different issues, and that AI theories didn't bother too much to account for facts of linguistic distribution. Such observations also apply to the scripts and plans representation based on Schank's Conceptual Dependency, which was the basis for the theory of story understanding developed e.g. in (Schank and Abelson, 1977, Wilensky, 1983). The recipes in the Plan Library, if taken by themselves, namely, without the Action Taxonomy, can certainly be seen as very similar to scripts, and in fact as providing less information.

A formalism that has been very influential on both linguistic and computational theories is Fillmore's case grammar (1968). Fillmore introduced the notion of *case frame*, a predicate containing arguments that are a set of obligatory and optional cases. His theory is at the somewhat high level of cases such as *agentive* or *instrumental*, and doesn't offer any other insight into the decomposition of lexical items.

[33]For the reader familiar with the CLASP (Devanbu and Litman, 1991) definition of *Action*: there all the required roles, such as `precondition`, can be described 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 `substeps`, which describes actions whose descriptions are mutually exclusive.

[34]As mentioned above, this problem may just be a consequence of using CLASSIC 1.0. More recent versions of CLASSIC, that provide for *role hierarchies*, may offer a way of solving the problem. If not even CLASSIC 2.3 is expressive enough, other systems such as LOOM (Mac Gregor, 1988) do provide role differentiation: however, the usual trade-off between expressive power and complexity of the classifier has to be taken into account.

Abstraction hierarchies and action representation. Many researchers have noted that taxonomies of actions and/or plans are necessary, especially for plan recognition. However, they have either just noted that action descriptions form a lattice, but not exploited the power of the virtual lattice as I have here (Balkanski, 1993), (Lochbaum, 1994); or they have focussed on representing only planning, and not also linguistic knowledge (Kautz, 1990), (Tenenberg, 1989). Linguistic knowledge is absent also in those systems that exploit subsumption to deal with hierarchical organizations of complex objects, such as plans (Devanbu and Litman, 1991), (Swartout *et al.*, 1991), (Wellman, 1988), constraint networks (Weida and Litman, 1994), or rules (Yen *et al.*, 1991). Among the systems that exploit subsumption to deal with plans, CLASP (Devanbu and Litman, 1991)[35] is the closest to my concerns. CLASP provides a rich plan language. Plans are built out of actions, defined as classic STRIPS planning operators. Plan concepts are defined from action and state concepts using operators such as SEQUENCE, LOOP, TEST. The ordinary subsumption algorithm is extended to deal with plan classification.

Even if linguistic knowledge is absent, some of the research just mentioned is very relevant to my concerns, as it introduces flexible matches in plan recognition. More specifically, work by Kautz (1990) and by Weida and Litman (1994) addresses the notion of *compatibility*: a set of observations is *compatible* with a plan if it can *potentially* instantiate the plan.

Kautz defines abstraction / decomposition hierarchies of *events*, which can be considered equivalent to recipes: they comprise *Components*, i.e. substeps, and *Constraints*, which in turn include *Temporal Constraints*, *Preconditions* and *Effects*. (Kautz, 1990, p. 110) introduces the notion of event compatibility:

> Two event predicates E_1 and E_2 are compatible if there is an event type E_3 such that both E_1 and E_2 abstract* E_3

where E_1 *abstract*'s* E_2 if E_1 is the same or abstracts E_2. Event compatibility is then used by Kautz's algorithm to build *explanation graphs*, namely, explanations of observed events. The notion of structural consistency I discussed earlier is clearly similar to Kautz's notion of event compatibility, but goes beyond it in at least two ways. First, all the terms in Kautz's taxonomy are defined a priori; thus E_3 cannot be inferred if it has not been included in the hierarchy to start with. This is exactly what the use of a DLBS's makes possible — see in particular Ex. (23c) below. Moreover, there is no structure to event predicates: while Kautz does provide role functions that yield the parameters of the events, such as the filler of the *agent* role, such parameters are basic terms, instead of structured objects as in my work.

Work by Weida and Litman (1994) is even closer than Kautz's to mine, as they define *compatibility* in the context of a DLBS. Their system, T-REX, can represent and reason with complex constraint networks — T-REX computes subsumption between network concepts, and compatibility between network instances and network concepts. Both my recipes and Kautz's events could easily be represented in T-REX. While the inferences I will discuss in the next section could presumably all be performed by T-REX, it is not clear how expectations would arise and, if they did, what role they would play. However, if I were to reimplement the Plan Library in order to more fully exploit the capabilities of a DLBS, I would definitely embed it in either

[35]Note that two research groups have independently chosen the name *CLASP* for their systems. The CLASP system in question is the one by Devanbu and Litman (1991), not the one by Yen et al. (1991).

CLASP or T-REX.

Turning now to linguistic applications of DLBS's, the first one was PSI-KLONE (Brachman *et al.*, 1979), that encoded syntactic productions as terms in the ontology. Semantic representations based on DLBS's, in particular on LOOM (Mac Gregor, 1988), have been used in NL generation systems, such as (Bateman, 1990, Reiter, 1992, Moore, 1995). (Stede and Rösner, 1994) proposes various approaches to encoding event decomposition into LOOM, but planning knowledge doesn't seem to be included. (Franconi, 1994) proposes a rich description logic for NL processing, within which it is possible to represent complex constructs, such as collections and plural quantifiers. Planning knowledge is also represented (Artale and Franconi, 1994): however, action terms appearing in plans seem to be considered as primitives, in accordance to planning practice. Finally, Dorr and Voss in (1996) extend previous work by Dorr (1993) that applies a lexical representation based on Jackendoff's CS's to Machine Translation. The main novelty in (Dorr and Voss, 1996) is that the representation is embedded in a DLBS (LOOM). Each root form (e.g. uninflected form for verbs) in the lexicon for a certain language is represented by means of a RLCS (Root Lexical CS). The lexicons for the different languages are then hierarchically structured by computing subsumption among different RLCS's. The DLBS is also used to resolve translation *mismatches*. These are cases in which a word w_{L_1} in language $\mathcal{L}_1$ has a certain representation RLCS_1, but no word w_{L_2} in $\mathcal{L}_2$ conveys exactly the same RLCS_1. Thus, when lexicon lookup fails to find w_{L_2}, the DLBS is queried to return the word in $\mathcal{L}_2$ whose RLCS_2 is closest to RLCS_1, where "closest" is defined based on subsumption. My work and Dorr and Voss's are clearly related. We both use Jackendoff's CS's and DLBS's: however, because of the different applications we pursue, Dorr and Voss are not concerned with representing and using planning knowledge; moreover, it is not clear what an inconsistency would amount to in their system.

4. INTERPRETING PURPOSE CLAUSES

Having laid down the description of the formalism, I will describe now how the representation is exploited by the algorithm that interprets PC's shown in Figure 7 — the steps affected by the representation are 2 and 3. Before describing the algorithm in Figure 7, I will discuss three of its limitations.

The first concerns the input the algorithm can process. The algorithm can interpret sentences consisting of a single main clause, but Figure 7 does not include this case for the sake of simplicity. As regards complex sentences, they must be in the form *Do* α *"for the purpose of" doing* β. Although this allows the algorithm to process connectives other than the infinitival *to* characteristic of PCs, it requires that such *purpose* meaning of the connective be recognized prior to the algorithm itself. As mentioned in Section 1.1, connectives other than *to* can convey *purpose*. Some, such as *so that, such that*, *by* in Means Clauses — *Do* α_1 *by doing* α_2 — do so unambiguously, and can be directly processed by the algorithm. Other connectives, such as *and*, are ambiguous in the relation they convey, and the algorithm can't process them. In these cases, the surface form is not specific enough to individuate whether the connective conveys a *purpose* meaning; even if *purpose* were recognized, the algorithm still wouldn't have enough information to choose the goal β among the actions described in the input. Ideally, it should be the algorithm itself that recognizes whether a *purpose* relation holds, and in which direction, through the

INPUT: Logical form LF of input sentence.
OUTPUT: Initial `PlanGraph`.

1. **(PREPROCESSING)**
   Choose β.
2. **(COMPUTE Recipe Candidate Set)**
   $\texttt{RSet} = \{R \mid \text{Recipe}(R) \wedge (\text{cl-instance?}\ \beta, \text{header}(R)) = T\}$
3. FOR each $\texttt{Recipe}_l \in \texttt{RSet}$ DO

   FOR each $\gamma_{l,j} \in \texttt{Body(Recipe}_l\texttt{)}$ DO

   Check the consistency of α and $\gamma_{l,j}$.
   (As a side-effect, a set of expectations $\{\mathcal{E}\}$ may be computed)

   IF there is **no** $\gamma_{l,j}$ such that α and $\gamma_{l,j}$ are consistent
   THEN $\texttt{RSet} = \texttt{RSet} \setminus \texttt{Recipe}_l$
4. **(GENERATE Output)**
   IF $\texttt{RSet} = \emptyset$
   THEN Signal *Can't process instruction*
   ELSE
   IF $|\texttt{RSet}| > 1$
   THEN $\texttt{Result} =$ Choose best match $\texttt{Recipe}_l \in \texttt{RSet}$
   ELSE $\texttt{Result} = \texttt{Recipe}_l \in \texttt{RSet}$
   Generate(`PlanGraph`, α, β, `Result`)

FIGURE 7. The Algorithm top level

inferences it performs. Moreover, the algorithm currently only deals with pairs of actions, namely, with LF's of the sort *Do* α_1 *"connective" do* α_2. Thus, as mentioned in fn. 7, PCs of the form *To do* β, *do* $\mathcal{A}$, where $\mathcal{A}$ is a sequence of actions, cannot be interpreted.

The second limitation concerns the fact that, while a flexible match is performed between α and $\gamma_{l,j}$, the match between β and recipe headers in step 2 is constrained to pure instantiation. This limitation is due to the need of circumscribing the problem (and the implementation) rather than to any theoretical reason. β could be allowed to vary with respect to the stored knowledge: namely, β could trigger the retrieval of recipes whose headers are subconcepts of β. α could then be used to select among those recipes, if it were specific enough — however, my corpus of PC's contains no examples in which this kind of inference is necessary, another reason why I chose a direct match between β and the headers of recipes. However, nothing in the algorithm or in its implementation prevents adding this further degree of flexibility, provided that the selection among candidate recipes is modified as well. See (Di Eugenio, 1993b, p.87,147) for further details.

The third, and in my opinion more serious limitation of the algorithm, is that the previous context is not taken into account. Namely, the examples dealt with are not embedded in longer instructional texts. Thus, the `PlanGraph` returned by the algorithm is always a newly initialized one, not an updated version of a preexisting one. I didn't take the general discourse context into account because I was focusing on a different problem, and because plan recognition for discourse is ex-

tremely complex. However, it is clear that the general discourse context should be taken into account, both in *AnimNL* in general and in my work in particular. For example, the context could be exploited to constrain the choice of the appropriate recipe, as I will discuss below. Luckily, there is a sizable body of research on this topic on which to draw for further work, including (Litman and Allen, 1990), (Carberry, 1990), (Lambert and Carberry, 1991), (Lochbaum, 1994). Incidentally, the implementation already provides for keeping a larger context, as a preexisting `PlanGraph` can be included as one of the parameters to the algorithm.
I will now turn to the description of the algorithm substeps.

Input and Preprocessing. The input to the algorithm is the LF of the input sentence. The LF contains one action description α_i per clause, plus interclausal connectives (i.e., each clause describing an action or an event is reified). Each α_i is an instance, asserted in the A-Box, of a CLASSIC action concept expressed in terms of CS's. One of the α_i's is then designated as β, and used to index into the Plan Library. The choice of β depends on the actual surface form: in PC's — *Do* α_1 *to do* α_2 — β is α_2; in Means Clauses — *Do* α_1 *by doing* α_2 — β is α_1. Namely, while in Purpose Clauses the goal β is the action described in the infinitival clause, in Means Clauses β is the action described in the main clause.

Compute Recipe Candidate Set. The purpose of Step 2 is to retrieve the recipes of whose header the goal β is an instance. For each recipe, the value restriction δ of the `header` role is retrieved, and then the CLASSIC query `(cl-instance?` β δ`)` is executed. Step 2 returns only the recipes with the most specific headers among those for which the query succeeds: namely, given two recipes $\texttt{Recipe}_\theta$ with header θ, and $\texttt{Recipe}_\eta$ with header η, such that β is an instance of both θ and η, and η is a descendant of θ, only $\texttt{Recipe}_\eta$ will be included in `RSet`.

Computing Consistency. At an abstract level, Step 3 can be concisely described as examining the concept ρ `= (and` α^{conc} $\gamma_{l,j}$`)`, where α^{conc} is the most specific concept of which α is an instance. Note that α^{conc} may be a virtual concept, as it may be more specific than the concept(s) of which the A-Box recognizes that α is an instance. In fact, in CLASSIC roles are declared independently of the concepts they may appear on; thus an instance can include roles that don't appear on any of its recognized parents. The algorithm computes α^{conc}, starting, of course, from α's recognized parents.

To examine ρ, the following queries are posed in the given order, until one returns t or all return *nil*:

(21a) `(and` α^{conc} $\gamma_{l,j}$`)` $\stackrel{?}{=}$ α^{conc}, i.e., does $\gamma_{l,j}$ subsume α^{conc}?

(21b) `(and` α^{conc} $\gamma_{l,j}$`)` $\stackrel{?}{=}$ $\gamma_{l,j}$, i.e., does α^{conc} subsume $\gamma_{l,j}$?

(21c) `(and` α^{conc} $\gamma_{l,j}$`)` $\neq$ `NIL?`, i.e., are α^{conc} and $\gamma_{l,j}$ consistent?

Both (21a) and (21b) are subsumed by (21c), but they are are processed separately, and before (21c): first, they are less complex to check; second, the particular relation holding between α and $\gamma_{l,j}$ affects how ρ_{perf} is computed. In fact, the queries in (21a) through (21c) correlate with progressively more complex ways in which ρ_{perf} must be computed, as shown in Table 1. In the first case, no computation is performed, as ρ_{perf} is α; in the second case, ρ_{perf} is an instance of $\gamma_{l,j}$, with the roles inherited from α^{conc} being filled with the fillers from α; finally, in the third case ρ_{perf} is an instance of the virtual concept ρ = `(and` α^{conc} $\gamma_{l,j}$`)`, with the roles inherited from α^{conc} being filled with the fillers from α, as in the previous case. `(create-instance (and act-type (fillers` α`)))` uses the CLASSIC command `cl-create-ind` to create a new instance of `act-type`; restrictions on the roles are taken from α by `(fillers` α`)`.

(21a)	$\rho = \alpha^{conc}$	$\Longrightarrow$	$\rho_{perf} = \alpha$
(21b)	$\rho = \gamma_{l,j}$	$\Longrightarrow$	ρ_{perf} = `(create-instance (and` $\gamma_{l,j}$ `(fillers` α`))`
(21c)	$\rho \neq \alpha^{conc} \wedge$ $\rho \neq \gamma_{l,j} \wedge$ $\rho \neq$ NIL	$\Longrightarrow$	ρ_{perf} = `(create-instance (and` ρ `(fillers` α`))`

TABLE 1. Computing ρ_{perf}

Note that my implementation of *structural consistency* is directly related to the notion of *structural subsumption*. Woods in (1991) points out that the term *subsumption* is used in many different ways in the literature. He lists five of those, *extensional, structural, recorded, axiomatic,* and *deduced* subsumption. In particular (Woods, 1991, p. 68-69):

> *structural subsumption*[36]—the subsuming concept is determined to be more general than the subsumed concept by virtue of a formally specified subsumption criterion applied to the structures of descriptions — preferably by an algorithm that is computationally tractable or at least more efficient than general deduction.
> *recorded subsumption*—the more general concept is explicitly recorded as subsuming the more specific concept in a stored taxonomic structure— either directly or indirectly, i.e., either by a direct link or by the transitive closure of such links. (Woods, 1991, p. 68-69)

Woods points out that the notion of *structural subsumption* comes into play when setting up the taxonomy or when classifying new concepts with respect to it, while *recorded* subsumption is the result of the taxonomic organization. Thus, all queries in (21) exploit *recorded* subsumption, and may have to compute *structural* subsumption.

Finally, the notion of consistency in (21) is actually still too restrictive, as it is purely structural. Consider

(22a) $[_{\text{Place}}\,\text{IN}([\text{KITCHEN}_1])]_{\mu_1}$

(22b) $[_{\text{Place}}\,\text{AT}([\text{TABLE}_2])]_{\mu_2}$

While neither of the μ_i's subsumes the other, and `(and` μ_1 μ_2`)` is inconsistent, they can be considered consistent if TABLE$_2$ is in KITCHEN$_1$: thus at least

[36] Emphasis in original.

some simple notions of geometric compatibility must be included in the algorithm.[37] Therefore, while (21a) is directly implemented by `(cl-instance?` α $\gamma_{l,j}$`)`, (21b) and (21c) cannot be — candidate CLASSIC queries would be `(cl-subsumes?` α^{conc} $\gamma_{l,j}$`)` for (21b) and `(not (cl-disjoint?` α^{conc} $\gamma_{l,j}$`))` for (21c). I provide a procedure `(check-real-comp` α^{conc} $\gamma_{l,j}$`)` that supplies the more flexible notion of consistency necessary to account for limited geometric reasoning. `check-real-comp` coincides with `(cl-subsumes?` α^{conc} $\gamma_{l,j}$`)` and `(not (cl-disjoint?` α^{conc} $\gamma_{l,j}$`))` if no role value restrictions on α or $\gamma_{l,j}$, and recursively, on their subcomponents, involve concepts or individuals of type *spatial path* or *spatial place*. When this happens, `check-real-comp` isolates the "spatial" roles and checks them separately, still using CLASSIC queries to the extent possible. Thus, this limited geometric reasoning can't be guaranteed to be sound, as it is not performed by CLASSIC inference engine.

Note that in Figure 7 there is no explicit step that implements **Computing Expectations**. In fact, such expectations arise from what is known about certain parameters within the current $\texttt{Recipe}_l$. Parameters that appear in a *qualifier* will appear in at least one other component in $\texttt{Recipe}_l$; thus, they will appear in at least one `SAME-AS` restriction, as `SAME-AS` restrictions are used to encode coreference — see Figure 6. As `SAME-AS` restrictions are taken into account while computing consistency between α and $\gamma_{l,j}$, expectations "emerge" from the processing.

Generate Output. Step 4 is executed only if there is at least one $\texttt{Recipe}_l$ left in `RSet`, the recipe candidate set initially computed by step 2.

There are several reasons why `RSet` may be empty. First, there may be no $\texttt{Recipe}_l$ whose header is instantiated by β. As discussed above, this constraint could be relaxed. Second, β may select at least one $\texttt{Recipe}_l$, but there may be no $\gamma_{l,j}$ such that α and $\gamma_{l,j}$ match. This means that the input instruction and the stored knowledge are inconsistent. The reasons for the inconsistency could be that the instruction is wrong, or that the Plan Library is incomplete. This is where the repair module should take over.

In case `RSet` contains more than one $\texttt{Recipe}_l$ at the beginning of step 4, the algorithm has to choose the one that matches best — this choice corresponds to step (4) in Charniak's motivation analysis discussed in Section 1. Recall that a first selection among recipes occurs in Step 2, which returns only the matching recipes with the most specific headers. The simple heuristic I adopted correlates the fit of match between recipe and input with the complexity of computing the corresponding ρ_{perf}, as shown in Table 1. If this still results in a tie, one $\texttt{Recipe}_l$ is chosen randomly.

As mentioned above, an important factor that affects this choice, namely, the current context, is not taken into account: a $\texttt{Recipe}_l$ that is dispreferred according to the complexity of ρ_{perf} may actually fit the context best. In fact, there is a whole body of work — among the most recent, (Litman and Allen, 1990), (Carberry, 1990), (Lambert and Carberry, 1991) — that exploits a variety of contextual constraints to decide which candidate plan should be pursued. These constraints are based on various features of linguistic and / or rational behavior on the part of agents. For example, the focusing heuristics in (Carberry, 1990) and (Lambert and Carberry, 1991),

[37]Geometric reasoning is a complex process, both as regards interpreting language that refers to space (Herskovits, 1986), and as regards how geometric constraints interact with acting in the world (Moore, 1993, Levison, 1996). Thus, there are plans to develop a full fledged geometric reasoner for *AnimNL* to address the problem in general. The examples of geometric reasoning I discuss are simple enough that they can be solved within CLASSIC.

and the coherence heuristics in (Litman and Allen, 1990) — informally, *plans related to the active ones are preferred to unrelated plans* — stem from the fact that agents are generally coherent in their discourses. Other heuristics have to do with rational behavior — for example, *If the planning agent wants an action and proposition P is an effect of the action, then he might want P* (Carberry, 1990, p.34).

Some of these heuristics are appropriate for dialogues between two rational agents, but seem too complex for instructional texts that mainly describe actions. However, heuristics similar to Carberry's focusing or Litman's coherence would indeed be appropriate if I took the context into account. Moreover, linguistic clues could be exploited, as done by (Litman and Allen, 1990) or by (Lambert and Carberry, 1991). However, the linguistic clues that have been studied mainly concern how to identify e.g. different speech acts such as *rejections* or *expressions of doubt* in dialogues, rather than substeps in domain-level plans.

To conclude the description of how the best recipe is chosen, note that if in a certain $\texttt{Recipe}_l$ α matches more than one substep $\gamma_{l,j}$, the algorithm also has to choose which $\gamma_{l,j}$ matches best — this tie is again solved according to Table 1; examples will be discussed in Section 4.2. Ideally, the choices of best $\texttt{Recipe}_l$ and of best matching $\gamma_{l,j}$ should inform each other.

Finally, after choosing the best match $\texttt{Recipe}_l$, the algorithm produces the initial `PlanGraph`, that represents the intentions and expectations the agent adopts as a consequence of the input instruction and the available planning knowledge. The `PlanGraph` is composed of nodes that contain individual action descriptions, and edges that denote relations such as *generation* and its generalization *is-substep*, *enablement*, and various temporal relations based on (Allen, 1984). This step creates the nodes corresponding to β, α, and to the $\gamma_{l,j}$'s; and the edges either deriving from the `Body` and `Annotations` on the $\texttt{Recipe}_l$, or computed during step 3. The *expectations* derived in step 3 are associated with the corresponding edge. An example of `PlanGraph` is shown in Figure 13.

I will now discuss how the inferences described in the abstract in Section 1.1 are performed.

4.1. Structural Consistency revisited

Let's go back to Examples (10a) through (10d), repeated here for convenience:

(23a) *[Cut the square in half along the diagonal with scissors]*$_{\alpha_1}$ *[to create two triangles]*$_{\beta_1}$.
(23b) *[Cut the square in half]*$_{\alpha_2}$ *[to create two triangles]*$_{\beta_2}$. (CC, 1988)
(23c) *[Cut the square in half with scissors]*$_{\alpha_3}$ *[to create two triangles]*$_{\beta_3}$.
(23d) *[Cut the square in half along a perpendicular axis]*$_{\alpha_4}$ *[to create two triangles]*$_{\beta_4}$.

In each of these examples, β is always the same, *create two triangles*. Therefore, the recipe R_{tr} in Figure 8 is retrieved (this is the real representation of $\mathcal{M}_{37}$ from Section 1.1). The header of R_{tr} can be glossed as *the agent causes the triangles to be composed starting from l (the square)*; the body as *the agent causes the square to be decomposed into halves*. *Comp* is the semantic field *Composition*, the superscript $+/-$ represents the mapping of whole and parts to thematic relations — for further details see (Jackendoff, 1990, p.116—122). γ is the CLASSIC concept

Header	$[\mathrm{CAUSE}(i, \mathrm{GO}^{-}{}_{\mathrm{comp}}(\mathrm{TRIANGLE}_j, k))]$ $\left[\ \mathrm{FROM}(\mathrm{AT}(l))\ \right]_k$
Body	$\left[\begin{array}{l} \mathrm{CAUSE}(i, \mathrm{GO}^{+}{}_{\mathrm{comp}}(\mathrm{SQUARE}_l, m)) \\ [_{Place}\mathrm{ALONG\text{-}DIAGONAL}] \\ \left[\ \mathrm{TO}(\mathrm{IN\text{-}HALF})\ \right]_m \end{array}\right]_\gamma$
Effects	$[\mathrm{BE}^{+}{}_{\mathrm{comp}}(l,\ \mathrm{AT}(j))]$

FIGURE 8. A *Create Triangles* Recipe

`cut-sq-inhalf-alongdiag`, and its position in the hierarchy is shown in Figure 9.[38] Let's derive now the four cases in (23) — the reader should keep in mind that α_i^{conc} is the most specific, possibly virtual concept of which α_i is an instance.

(23a) is the easiest case, as α_1 is simply an instance of γ, thus (21a) returns t. Given that no expectations are derived, ρ_{perf} is inferred to be α_1.

The second case, (23b), is more interesting. As shown in Figure 10, α_2^{conc} is an ancestor of γ. ρ_{perf} is inferred to be, not α_2, but rather, an instance of γ. The roles inherited from α_2^{conc} on ρ_{perf} are filled with the fillers from α_2. Incidentally, note that `cut-sq-inhalf` is not subsumed by `cut-location`. The role `location` refers to the physical location on the patient where the *cut* takes place, namely, its value restriction is `spatial-place`; instead, `in-half` is an instance of `comp-place`, a `place` with semantic field *Composition*. `in-half` is found at the end of the role chain informally described as: `(destination (path-role (caused-role (`α_2`))))`.

(23c) and (23d) both illustrate query (21c), but return different results.

In (23c), α_3 is an instance of a concept, α_3^{conc}, which is neither an ancestor, nor a descendant of γ — see Figure 11. In this case, the algorithm checks whether the virtual concept ρ `= (and` α_3^{conc} γ`)` is consistent: as it is, the action to be performed is taken to be an instance of ρ, with the roles inherited from α_3^{conc} filled with the fillers on α_3. Finally, in (23d), γ, *cut the square in half along the diagonal* and α_4, *cut the square in half along a perpendicular axis* are structurally exactly the same, but they are inconsistent. This is detected when examining the fillers of the role *location* on the two different concepts, namely, by asking CLASSIC whether ρ `= (and` α_4^{conc} γ`)` is consistent — see Figure 12. The value restrictions of the role `location` on α_4^{conc} and on γ are respectively, `along-perp-axis` and `along-diagonal`; in turn, these two are inconsistent concepts, because their value restrictions, `perp-axis` and `diagonal`,

[38]One important issue missing in recipe R_{tr} is *cardinality*, i.e. how to represent that such action will produce *exactly two* triangles.

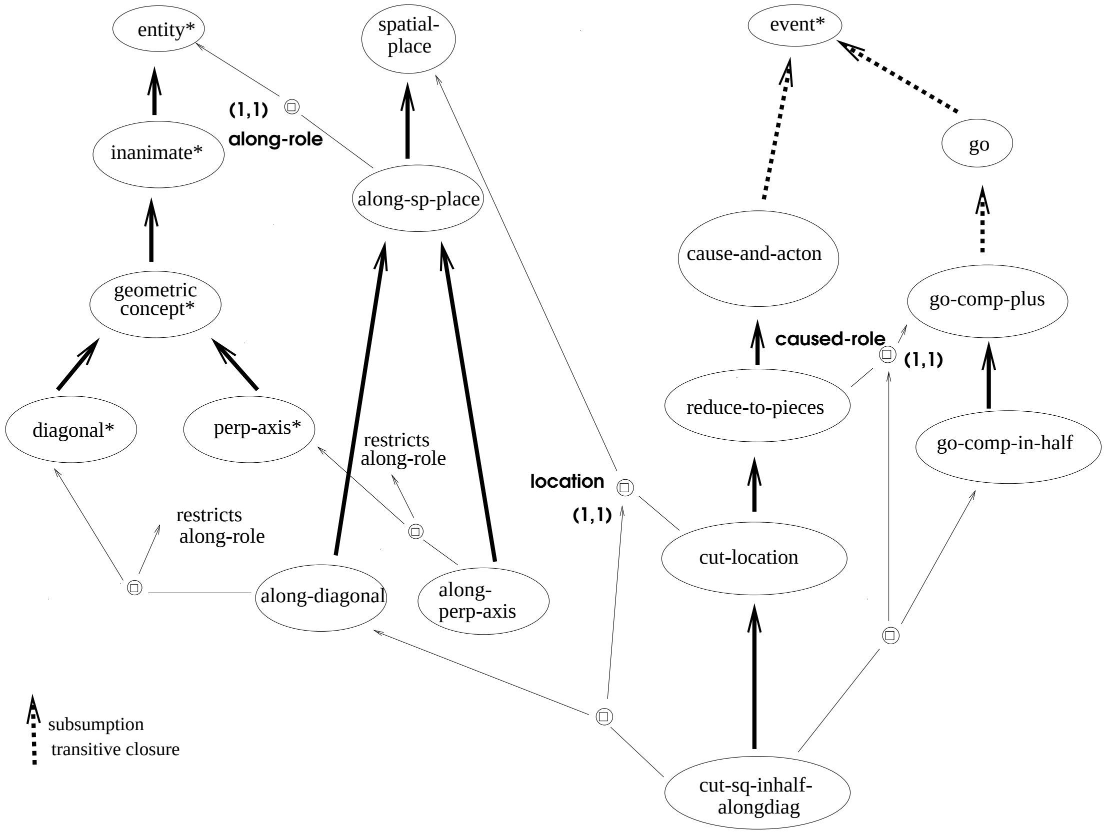

FIGURE 9. The *cut* hierarchy in network form

have been defined as disjoint.[39]

Finally, note that Examples (23a) and (23c) give rise to a potential problem for *AnimNL*: ρ_{perf}, that the other modules in *AnimNL* have to further process, has features such as the instrument that the stored description $\gamma_{l,j}$ doesn't include. Two observations mitigate this problem. First, even if ρ_{perf} has features that $\gamma_{l,j}$ doesn't include, ρ_{perf} has been computed within the context of $\texttt{Recipe}_l$, on the basis of which the `PlanGraph` has been constructed. $\texttt{Recipe}_l$ in a sense acts as a default, as all the actions in $\texttt{Recipe}_l$ can be processed by the other modules in *AnimNL* — in the worst case, the added modifiers could be ignored. However, they can in general be taken into account: in fact, while the action description per se may be unknown to *AnimNL*, its components are not. For example, knowledge about instruments, and about how certain actions such as *cut* may require instruments to be executed, is contained in the *Object KB* — cf. Figure 1. This information can then be used by the *object specific reasoner* (Levison, 1996) to further process Examples (23a) and

[39] While I realize that this is not the most perspicuous way of representing such geometric concepts, my focus in this work was on how to verify structural consistency, rather than on building a geometric reasoner.

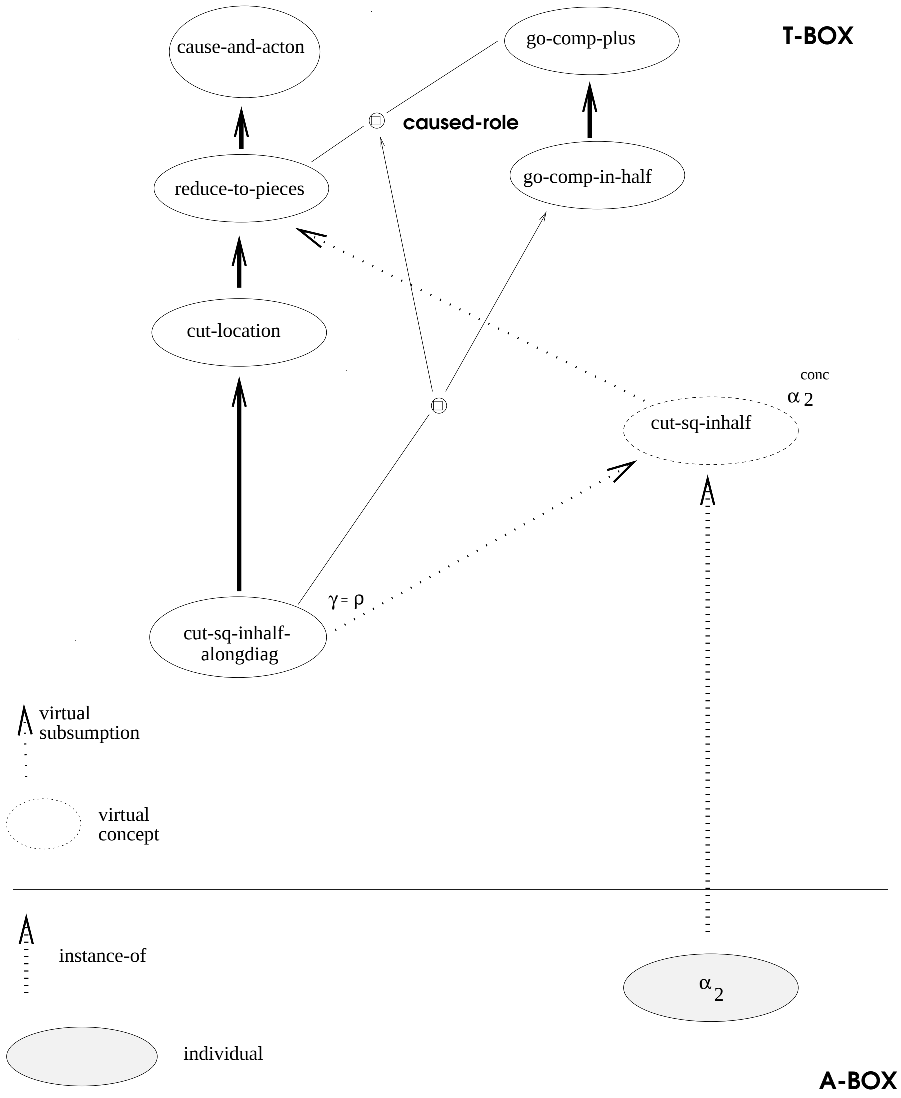

Figure 10. A less specific *cut*

(23c). It is true, though, that the full integration of different KB's in *AnimNL* is still an open problem, and that the particular knowledge about different ways of cutting according to the instrument to be used is not available yet.

4.2. Computing Expectations revisited

I will now show how the algorithm computes $\mathcal{E}_1$, *coffee urn in kitchen*, in the context of (6a), while in the context of (6b) it builds $\mathcal{E}_2$, *washing site in kitchen*. In this section, I will use the CS notation to describe actions and recipes as it is more

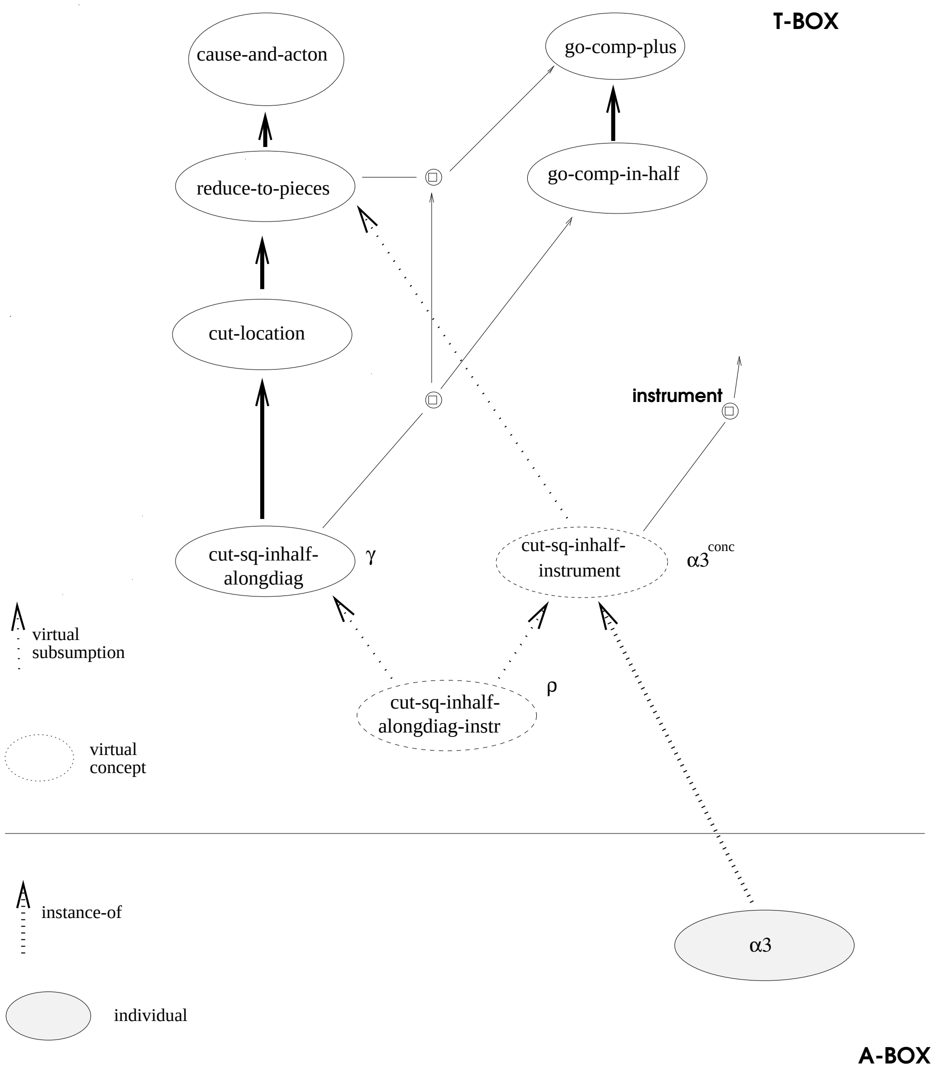

FIGURE 11. A more and less specific *cut*

succinct than CLASSIC definitions, but the reader should keep in mind that every Conceptual Structure shown here corresponds to a CLASSIC concept or individual; and that expressions such as *parameter* `i` *is bound to value* `x` are simply shorthands to refer to *filler* `x` *of role* `i` (of the appropriate concept / individual).

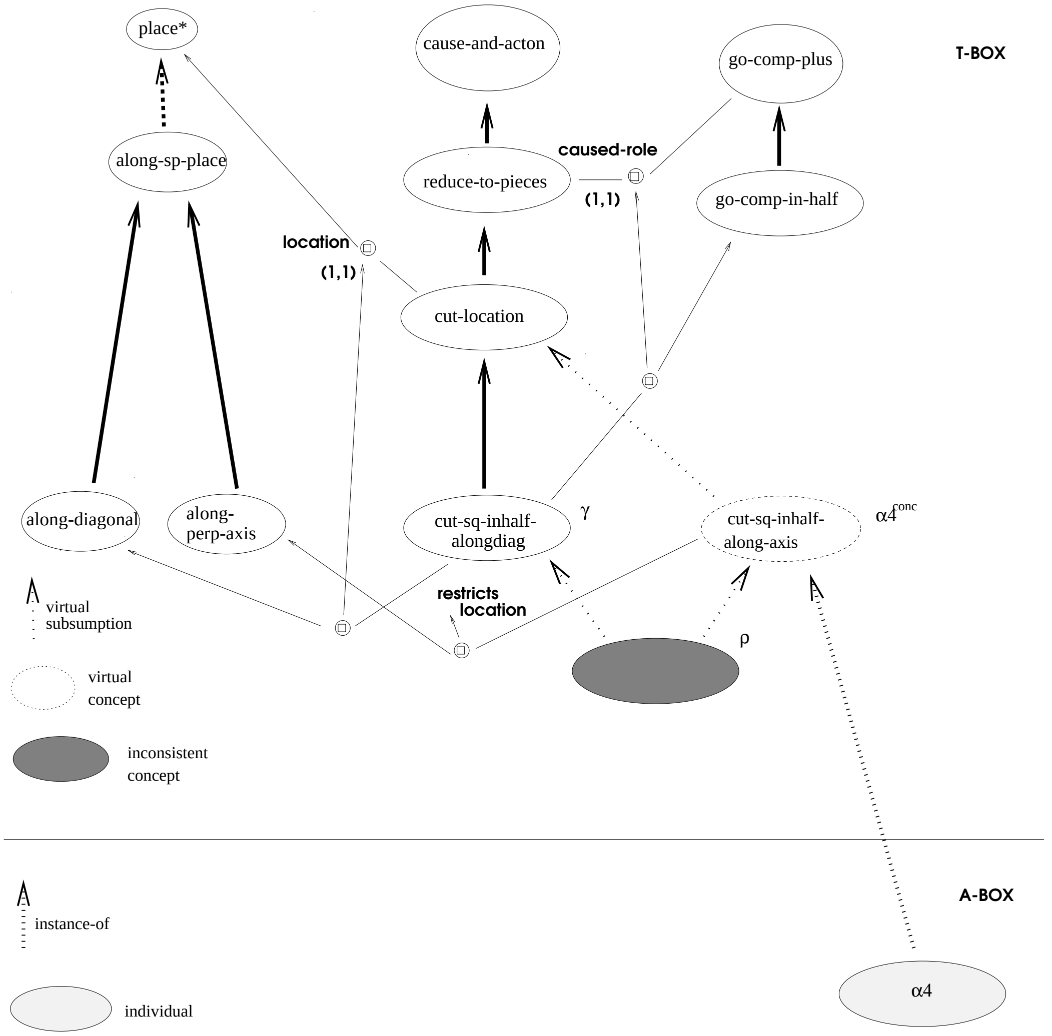

Figure 12. An inconsistent *cut*

Also note that while the connective in (6a) and (6b) is *and*, as mentioned previously, the algorithm works on *to*. Thus the examples dealt with by the algorithm are:

(24a) *[Go into the kitchen]*$_{\alpha_1}$ *[to get me the coffee urn]*$_{\beta_1}$.
(24b) *[Go into the kitchen]*$_{\alpha_2}$ *[to wash the coffee urn]*$_{\beta_2}$.

The logical form produced by the parser for (24a) is as follows — *get me* is interpreted as *cause to go to me*, and *to* as *FOR(β_1)*:

$$\left[\begin{array}{l}\mathrm{GO_{Sp}}([\textsc{you}]_r, [\mathrm{TO}([\mathrm{IN}([\textsc{kitchen}])])]) \\ \mathrm{FOR}(\beta_1)\end{array}\right]_{\alpha_1}$$

$$[\mathrm{CAUSE}(r, [\mathrm{GO_{Sp}}([\textsc{urn-of-coffee}]_s, w)])]_{\beta_1}$$
$$\left[\begin{array}{l}\mathrm{FROM}(t) \\ \mathrm{TO}([\mathrm{AT}([\textsc{me}])])\end{array}\right]_{w}$$

In step 2, the goal β_1 is found to be an instance of the header of the *Move* recipe shown in Figure 5, and of no other recipe. In step 3, α_1 and the three substeps $\gamma_1, \gamma_2, \gamma_3$ are checked for consistency. α_1 and γ_1 are consistent, since α_1 is an instance of γ_1, with i bound to YOU and m to [IN([KITCHEN])]:

(25a) $[\mathrm{GO_{Sp}}([\textsc{you}]_r,\ [\mathrm{TO}([\mathrm{IN}([\textsc{kitchen}])])])]_{\alpha_1}$

(25b) $[\mathrm{GO_{Sp}}(i,\ [\mathrm{TO}(m)])]_{\gamma_1}$

The algorithm still has to verify whether there is any other information on i or m in the *Move* recipe that makes i's or m's bindings untenable, such as $m = [\textsc{outdoors}]$. The only other constraint on these parameters comes from the qualifier $[\mathrm{BE_{Sp}}(j,\ m)]$. Thus we can conclude:

(26) $\mathcal{E}_1 = [\mathrm{BE_{Sp}}([\textsc{coffee-urn}],\ [\mathrm{IN}([\textsc{kitchen}])])]$

As α_1 and γ_1 are consistent, the conclusion is that α_1 *generates* γ_1 under $\mathcal{E}_1$.[40]

The algorithm still checks whether α_1 may be consistent with other substeps of the *Move* recipe. α_1 and γ_2 are inconsistent, as α_1 is a simple GO_{sp}, while γ_2 involves CAUSE — cf. Figure 5. Instead, the same inferences that yield that α_1 is consistent with γ_1 under $\mathcal{E}_1$, could conclude that α_1 is consistent with γ_3 under

(27) $\mathcal{E}_2 = [\mathrm{BE_{Sp}}([\mathrm{AT}([\textsc{me}])],\ [\mathrm{IN}([\textsc{kitchen}])])]$

However, according to Table 1, α_1 and γ_1 match "better" than α_1 and γ_3. In fact, while α_1 is an instance of γ_1, α_1^{conc} is more abstract than γ_3. Furthermore, in naturally occurring examples in general α corresponds to the first action of a sequence generating β. Therefore, the match between α_1 and γ_1 is preferred.

Finally, in step 4, the various nodes and edges corresponding to the actions and the relations between them — α_1, γ_1, γ_2, γ_3, the edge labeled *generation* between α_1 and γ_1 — are created. The *is-substep* and *enables* edges deriving respectively from the body and the annotations of the *Move* recipe are also created and included in the `PlanGraph`. We obtain the structure shown in Figure 13, where the labels on the nodes are clearly only suggestive of their contents.

[40] $\mathcal{E}_1$ can be considered as a condition on the associated conditional generation relation, a condition that the surface form has singled out.

Note that the algorithm would generate the same kind of expectation in examples that minimally differ from (24a), such as *?Go into the living room to get me the car*, which could perhaps be acceptable under the circumstance that the car keys are in the living room. The following $\mathcal{E}_{1'}$ would be inferred:

(28) $\mathcal{E}_{1'} = [\mathrm{BE}_{\mathrm{Sp}}(\textsc{car}, [\mathrm{IN}([\textsc{living-room}])])]$

$\mathcal{E}_{1'}$ is indeed correctly inferred from the syntactic context and the available knowledge about actions; a detailed geometric reasoner is required to act as a filter on the locational expectations the algorithm derives, and then to possibly trigger other more complex inference processes.

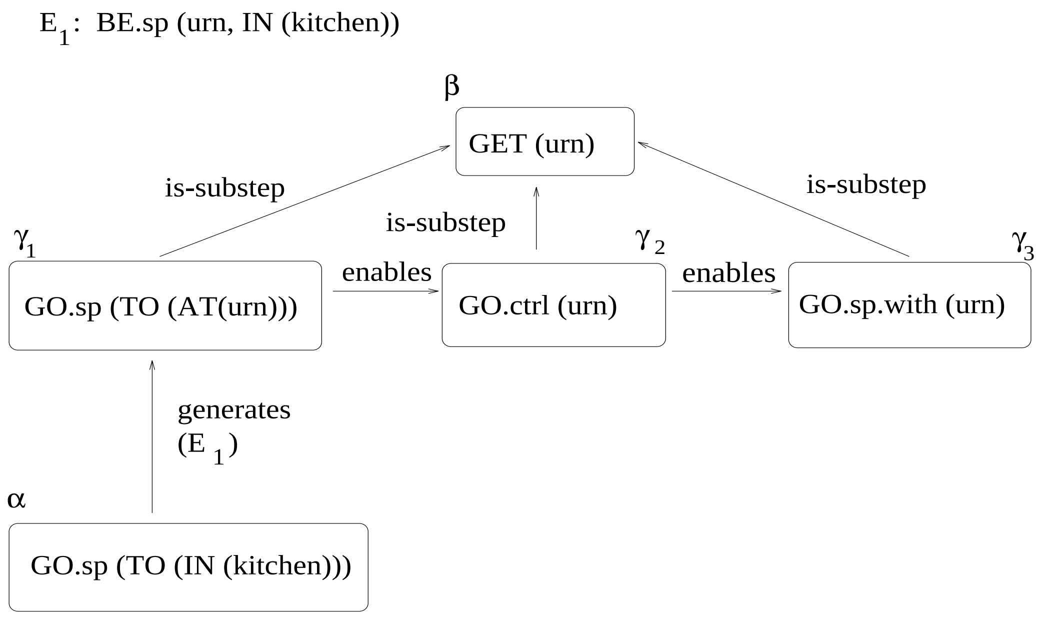

FIGURE 13. PlanGraph for *Go into the kitchen to get me the coffee urn*

It remains to be shown how in (24b), *Go into the kitchen to wash the coffee urn*, the expectation concerns the washing site, not the urn, being in the kitchen. Figure 14 shows the recipe for *Wash*.
Some characteristics of this recipe need explanation.

First, its header can be paraphrased as *the agent causes the patient to become clean*, i.e. *to "go" along the path whose end point is the property* `clean`. The function GO in this case has the semantic field `Identificational`.

Second, γ_1 and γ_2 are both expressed in terms of the ACHIEVE operator, defined as a function from states to actions (Pollack, 1986). It seems proper to express the first two substeps of the *Wash* recipe in these terms, rather than by means of a more specific action, such as $\mathrm{GO}_{\mathrm{sp}}$.

Header	$[\mathrm{CAUSE}([\textsc{agent}]_i, [\mathrm{GO}_{\mathrm{ident}}(j, k)])]_\delta$ $\left[\ \mathrm{TO}([\mathrm{AT}([\textsc{clean}])])\ \right]_k$
Body	- $[\mathrm{ACHIEVE}(i,\ \mathrm{BE}_{\mathrm{Sp}}(i,\ [\mathrm{AT}(t)]))]_{\gamma_1}$ - $[\mathrm{ACHIEVE}(i,\ \mathrm{BE}_{\mathrm{Sp}}(j,\ [\mathrm{AT}(t)]))]_{\gamma_2}$ - $[\mathrm{PHYSICAL\text{-}WASH}(i,\ j,\ [\mathrm{AT}(t)])]_{\gamma_3}$ **Annotations** - γ_1 *enables* γ_3 - γ_2 *enables* γ_3
Qualifiers	- $[\mathrm{BE}_{\mathrm{Ident}}(j,\ \mathrm{AT}([\textsc{dirty}]))]$ - $[\textsc{washing-site}]_t$
Effects	- $[\mathrm{BE}_{\mathrm{ident}}(j,\ \mathrm{AT}([\textsc{clean}]))]$

FIGURE 14. A *Wash* Recipe

Note that there is a difference between (29a) and (29b):

(29a) $[\mathrm{GO}_{\mathrm{sp}}(i,\ [\mathrm{TO}([\mathrm{AT}(j)])])]$

(29b) $[\mathrm{ACHIEVE}(i,\ \mathrm{BE}_{\mathrm{sp}}(i,\ [\mathrm{AT}(j)]))]$

For (29a) we want the agent to perform an action of $\mathrm{GO}_{\mathrm{sp}}$; while for (29b), we are only interested in the end result, that the agent is at j.

Given this, expressing γ_1 in the *Move* recipe as (29a), while γ_1 in the *Wash* recipe is expressed as (29b), may seem arbitrary. However, the *Move* recipe basically encodes one of the senses of *fetch*, namely, *to go or come after and return with* (The American Heritage Dictionary), and therefore, what is encoded is that the agent must physically go to the location of the object. 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.

Third, γ_3 is simply denoted as PHYSICAL-WASH, which is a basic act-type: as mentioned in Section 3.3, it is left to other processes to specify a situationally appropriate sense of physical washing.

Finally, *wash* has two qualifiers, one requiring the object to be washed to be dirty, and the other specifying that the action site must be a washing site.

Referring back to the algorithm in Figure 7, steps 1 and 2 are the same as for (24a), except that the goal β_2 matches the header of the *Wash* recipe. For step 3, we try to match α_2 with a substep γ_i of the *Wash* recipe. To deal with the ACHIEVE operators, the algorithm first retrieves the effects of $\mathrm{GO}_{\mathrm{Sp}}$, that are then matched to the argument of type *state* of the ACHIEVE operator.
α_2 and γ_1 are respectively:

(30a) $[\mathrm{GO}_{\mathrm{Sp}}([\textsc{you}]_r, [\mathrm{TO}([\mathrm{IN}([\textsc{kitchen}])])])]_{\alpha_2}$

(30b) $[\mathrm{ACHIEVE}(i, [\mathrm{BE}_{\mathrm{Sp}}\ (i, [\mathrm{AT}(t)])])]_{\gamma_1}$

The effects of (30a) are

(31) $[\mathrm{BE}_{\mathrm{Sp}}([\textsc{you}]_r, [\mathrm{IN}([\textsc{kitchen}])])]$

Therefore, given the qualifier $[\textsc{washing-site}]_t$, α_2 and γ_1 are found to be consistent under the expectation

(32) $\mathcal{E}_2 = [\mathrm{BE}_{\mathrm{Sp}}([\mathrm{AT}([\textsc{washing-site}])], [\mathrm{IN}([\textsc{kitchen}])])]$

Given that the only other unbound parameter in (30b) is i, which is not further constrained, and that IN and AT are both *place* functions, and therefore consistent with each other, we can assume that (32) is consistent with the rest of the knowledge we have. As mentioned above, some more detailed geometric reasoning is needed to understand that the place $[\mathrm{AT}([\textsc{washing-site}])]$ is contained in the place $[\mathrm{IN}([\textsc{kitchen}])]$.

A match between α_2 and γ_1 is found, while no match between α_2 and γ_2 or γ_3 is found. Note that a second expectation, concerning the coffee urn being dirty, is derived as well. Finally, a `PlanGraph` is generated similarly to what discussed for Example (24a).

4.3. Evaluation

If *AnimNL* were more than a prototype, and if all the other modules in the system had been thoroughly tested, then the ideal evaluation for the work described in this paper would be end-to-end: namely, my module should be evaluated in terms of the animations produced by *AnimNL*. However, as discussed in Section 2, conducting such an evaluation is infeasible: to animate every example for which my module produces a PlanGraph, the other parts of the symbolic KB's, and in certain cases even the simulated behaviors, should be greatly expanded — altogether an impossible feat, and one involving many other researchers besides the author.

Thus, I had to evaluate my module independently from *AnimNL*. I conducted the evaluation by identifying equivalence classes of examples, and testing the algorithm on a few examples per class. The results were evaluated in terms of the plan graph the algorithm builds: I verified it was correct, and I traced the algorithm to make sure that correct plan graphs were not produced for the wrong reasons.

Equivalence classes of examples are constructed in terms of the triggered inferences, more specifically, in terms of the outcomes of each of the three queries in (21), repeated here for convenience:

(33a) `(and` α^{conc} $\gamma_{l,j}$`)` $\stackrel{?}{=}$ α^{conc}, i.e., does $\gamma_{l,j}$ subsume α^{conc}?

(33b) `(and` α^{conc} $\gamma_{l,j}$`)` $\stackrel{?}{=}$ $\gamma_{l,j}$, i.e., does α^{conc} subsume $\gamma_{l,j}$?

(33c) `(and` α^{conc} $\gamma_{l,j}$`)` $\neq$ `NIL?`, i.e., are α^{conc} and $\gamma_{l,j}$ consistent?

Each query may have one of the following four outcomes:

- structural consistency is verified, and no expectation are derived: *success*
- structural consistency is verified, and expectations are derived. This case is subcategorized into
  - consistent expectations: *success*
  - inconsistent expectations: *failure*
- structural consistency is not verified: *failure*

Note the difference between the two outcomes labeled as *failure*. For queries (33a) and (33b), only inconsistent expectations trigger a real failure, while a failure due solely to a structural inconsistency triggers the next query in the sequence. Therefore, there are 10 possible distinct outcomes for the algorithm, corresponding to 10 equivalence classes.[41] For instance, one equivalence class characterizes examples for which query (33b) produces a *success* outcome with no derived expectations. Two examples belonging to this class are (23b), *Cut the square in half to create two triangles*, discussed above, and the following:

(34) *Turn the screw to loosen*

Example (34) triggers the same inference as Example (23b), in that α is less specific than the stored γ, *Turn the screw counterclockwise.*

It should be clear that equivalence classes don't depend on surface similarities between examples or on the head verb appearing in the example: examples headed by the same verb may belong to different classes, while examples headed by different verbs may belong to the same class. For instance, compare (24a), repeated here as (35a), and (35b).

(35a) *Go into the kitchen to get me the coffee urn*
(35b) *Go into the kitchen to get me the coffee urn from the basement*

For both examples, structural consistency is verified, and the expectation $\mathcal{E}_1$ = *coffee urn in kitchen* is derived by query (33a). However, $\mathcal{E}_1$ is consistent for Example (35a), but inconsistent for Example (35b), given that (35b) explicitly refers to the location of the coffee urn, *the basement*. Also note that Example (35b) does not belong to the same equivalence class as e.g. (23d), *Cut the square in half along a perpendicular axis to create two triangles*, on which the algorithm (correctly) fails as well. In

[41] These 10 equivalence classes are actually further subdivided, according to the specific paths that the algorithm follows. However, this abstraction is sufficient for the current purpose.

the latter example, *failure* is detected by query (33c) because of a purely structural inconsistency between two role fillers, `along-diagonal` and `along-perp-axis`, rather than because of an inconsistent expectation, as in Example (35b).

The evaluation I just described is limited in at least two ways. First, the training set was small, as the KB and algorithm were developed to address the examples listed in (6) and in (10). Once those first examples were dealt with correctly, I added further knowledge, such as the recipes concerning loosening and tightening screws; I then tested the algorithm on examples both in the same equivalence classes as the ones used in development, and outside these equivalence classes. A second limitation is more serious and definitely worrisome, at least at first sight: on this evaluation, the success rate of the algorithm was 100%, namely, it always returned a correct result, be it a *success* or a *failure*. This result is not as surprising if one takes into account the assumption of completeness for the KB: if this had not been the case, the algorithm would have failed, and it would have been necessary to add, possibly, new semantic primitives, but more likely, new recipes. Of course this is straightforward in CLASSIC *once* the new concept has been derived: the problem is to derive the missing semantic primitive or the missing recipe from the input. Machine Learning techniques could perhaps help.

The assumption of completeness of the KB clearly needs to be relaxed. However, the focus of the work discussed in this paper concerns the algorithm's ability to provide flexible matching once the knowledge is present, and to that extent, the evaluation I discussed here is at least suggestive that the algorithm is correct.

5. SUMMARY AND FUTURE WORK

In this paper, I have argued that interpreting Natural Language instructions requires flexibility in the match between the instruction in input and the stored knowledge. I have presented an action representation formalism that supports such flexibility. I have discussed how the formalism represents both linguistic and planning knowledge about actions, in the Action Taxonomy and Plan Library respectively; and how both components are implemented by means of the DLBS CLASSIC.

The advantages of the proposed formalism are as follows. Jackendoff's CS representations capture many linguistic distinctions that are of import for instruction interpretation and execution as well. The DLBS provides, first, a mechanism to maintain an organized lattice of action descriptions; second, an inference engine able to deal with discrepancies between the action descriptions in the input, even those unknown to the system, and the descriptions the system has knowledge about. I also argued that integrating the two formalisms is beneficial to both. It gives a principled linguistic basis to a Knowledge Base built for the purpose of interpreting NL; conversely, it provides a computational basis for a linguistic theory that can be computationally used in real systems, tested and extended.

I have also demonstrated how the formalism is exploited by an algorithm, implemented within the larger framework of the *AnimNL* system, that interprets Purpose Clauses. In particular, CLASSIC's classifier provides the basis for checking structural consistency and computing expectations.

Future work includes first of all extending the kind of data that my representation and algorithm account for. I believe that the approach is useful for any kind of instruction, as the need to match input descriptions with stored knowledge will always arise. However, in the case of PC's the problem is simplified by having the goal explicitly given, while the goal must be inferred in plan recognition algorithms.

As regards possible inputs, the algorithm can already interpret other purpose connectives, such as *so that, such that*, and Means Clauses. However, to deal with the latter, and to distinguish them from PC's, some more perspicuous treatment of the agent's mental state, such as in (Balkanski, 1993), may be needed. The algorithm needs to be extended to other connectives that may convey purpose, such as *and* and *until*.

Some *negative imperatives* can be handled with the same techniques (Di Eugenio, 1993a), (Vander Linden and Di Eugenio, 1996). For example, given a hierarchical representation of *cleaning* actions, it would be easy to recognize that *scrub* and *wet-mop* are not viable alternatives of *cleaning the parquet*: this could lead to learning proper methods for cleaning a parquet.

(36) *Caring for the floor. Dust-mop or vacuum your parquet floor as you would carpeting.* **Do not scrub or wet-mop the parquet.**

More complex hierarchical relations between recipes should be provided. For example, different temporal orders on the substeps of recipes that are otherwise the same should impose a hierarchical relation between the recipes in question. For instance, the *Move* recipe in Figure 5 imposes a total order on its substeps. Suppose we were to define a $Move_2$ recipe in which the temporal order of the substeps were BEFORE(γ_1, γ_3) and BEFORE(γ_2, γ_3), without any explicit ordering relating γ_1 and γ_2. The $Move_2$ recipe should be recognized as subsuming the *Move* recipe. This type of reasoning is performed in both CLASP (Devanbu and Litman, 1991) and T-REX (Weida and Litman, 1994): either formalism could be used to reimplement the Plan Library in order to more fully exploit the capabilities of the DLBS.

Finally, the algorithm could be expanded in a number of different ways.

First, a *recursive step* should be added to examine the expansions of the substeps of a recipe, if no match between the input action and any of the substeps has been found.

Second, more complex inferences should be implemented. Consider Example (4), *Hold the cup under the spigot to fill it with coffee.* In Section 1, it was hypothesized that the more specific *hold the cup under the spigot* **upright** could be derived from explicit knowledge such as (5); however, in absence of (5), the same description could still be derived from some other knowledge, such as the conditions on a generation relation between *depress the lever* and *fill cup with coffee*.

Third, it is necessary to deal with actions whose execution achieves more than one of the substeps of a certain `Recipe`: in *Take the coffee urn into the kitchen to wash it*, *Take the coffee urn into the kitchen* achieves both γ_1 and γ_2 in Figure 14.

Finally, the discourse context must be added, and it should be represented by the `PlanGraph` built so far. Among others, this would affect the evaluation function that chooses the best match `Recipe`.

ACKNOWLEDGEMENTS

This research was carried on while the author was at the Department of Computer and Information Science, University of Pennsylvania, and supported 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. Various drafts of the paper were written while the author was with the Computational Linguistics program, Carnegie Mellon University. Many thanks to Bonnie Webber for her guidance during the years in graduate school, to the members of the *AnimNL* group for many discussions over interpreting instructions, to Marilyn Walker, Susan Haller, Pamela Jordan and three anonymous reviewers for comments on earlier versions.

REFERENCES

(Allen, 1984) James Allen. Towards a General Theory of Action and Time. *Artificial Intelligence*, 23:123–154, 1984.

(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.

(Artale and Franconi, 1994) Alessandro Artale and Enrico Franconi. A Computational Account for a Description Logic of Time and Action. In *KR94, Principles of Knowledge Representation and Reasoning: Proceedings of the Fourth International Conference*, 1994.

(Badler *et al.*, 1993) Norman I. Badler, Cary B. Phillips, and Bonnie Lynn Webber. *Simulating Humans: Computer Graphics Animation and Control.* Oxford University Press, 1993.

(Balkanski, 1993) Cecile Balkanski. *Actions, Beliefs and Intentions in Multi-Action Utterances.* PhD thesis, Harvard University, 1993. Available as Technical Report TR-16-93, Center for Research in Computing Technology.

(Bateman, 1990) John A. Bateman. Upper modeling: organizing knowledge for natural language processing. In *Proceedings of the Fifth International Workshop on Natural Language Generation, Pittsburgh, Pennsylvania*, 1990.

(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. L. Webber, and W. Woods. Research in Natural Language Understanding, Annual Report. Technical Report 4274, Bolt Beranek and Newman, 1979.

(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 *et al.*, 1988) Michael Bratman, David Israel, and Martha Pollack. Plan and resource-bounded practical reasoning. *Computational Intelligence*, 4:349–355, 1988.

(Carberry, 1990) Sandra Carberry. *Plan Recognition in Natural Language Dialogue.* ACL-MIT Press Series on Natural Language Processing. MIT Press, 1990.

(Carpenter and Alterman, 1994) Tamitha Carpenter and Richard Alterman. Reading Instructions to improve Agent Activity. In *Working Notes, AAAI Spring Symposium on Active NLP*, 1994.

(CC, 1988) Country Crafts, 1988.

(Chapman, 1991) David Chapman. *Vision, Instruction and Action.* MIT Press, 1991.

(Charniak, 1988) Eugene Charniak. Motivation Analysis, Abductive Unification, and Nonmonotonic Equality. *Artificial Intelligence*, 34:275–295, 1988.

(CWP, 1989) Country Wood Projects, Summer 1989.

(Davidson, 1982) Donald Davidson. *Essays on Actions and Events.* Oxford University Press, 1982.

(Delin *et al.*, 1994) Judy Delin, Anthony Hartley, Cecile Paris, Donia Scott, and Keith Vander Linden. Expressing Procedural Relationships in Multilingual Instructions. In *Seventh International Workshop on Natural Language Generation*, pages 61–70, Kennebunkport, Maine, 1994.

(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.

(Di Eugenio and Webber, 1996) Barbara Di Eugenio and Bonnie Lynn Webber. Pragmatic Overloading in Natural Language Instructions. *International Journal of Expert Systems, Special Issue on Knowledge Representation and Reasoning for Natural Language Processing*, 9(1), March 1996.

(Di Eugenio, 1993a) Barbara Di Eugenio. Speaker's Intentions and Beliefs in Negative Imperatives. In *ACL Workshop on Intentionality and Structure in Discourse Relations*, pages 11–14, Columbus, OH, 1993.

(Di Eugenio, 1993b) Barbara Di Eugenio. *Understanding Natural Language Instructions: a Computational Approach to Purpose Clauses.* PhD thesis, University of Pennsylvania, December 1993. Available as Technical Report MS-CIS-93-91.

(Doran, 1993) Christine Doran. Purposive AND Clauses in Spoken Discourse, 1993. Unpublished manuscript, University of Pennsylvania.

(Dorr and Voss, 1993) Bonnie J. Dorr and Clare R. Voss. Machine translation of spatial expressions: defining the relation between an interlingua and a knowledge representation system. In *AAAI93, Proceedings of the 12th Conference of the American Association for Artificial Intelligence*, 1993.

(Dorr and Voss, 1996) Bonnie J. Dorr and Clare R. Voss. Multi-Level Approach to Interlingual MT: Defining the Interface between Representational Languages. *International Journal of Expert Systems, Special Issue on Knowledge Representation and Reasoning for Natural Language Processing*, 9(1), March 1996.

(Dorr, 1993) Bonnie J. Dorr. *Machine Translation: a view from the lexicon.* MIT Press, 1993.

(Fillmore, 1968) Charles J. Fillmore. The case for case. In E. Bach and R. T. Harms, editors, *Universals in Linguistic Theory.* Holt, Rinehart and Winston, 1968.

(Franconi, 1994) Enrico Franconi. Description Logics for Natural Language Processing. In *Working Notes, AAAI Fall Symposium on Knowledge Representation for NLP in Implemented Systems*, November 1994.

(Geib *et al.*, 1994) Chris Geib, Libby Levison, and Michael Moore. SodaJack: An architecture for agents that search for and manipulate objects. Technical Report MS-CIS-94-16, Department of Computer and Information Science, University of Pennsylvania, 1994.

(Geib, 1995) Christopher Geib. *The Intentional Planning System (ItPlans).* PhD thesis, University of Pennsylvania, May 1995.

(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.

(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, 1988) Alice Rich Hallowell, editor. *Wall Coverings.* Sunset Books. Lane Publishing Company, 1988.

(Herskovits, 1986) Annette Herskovits. *Language and Spatial Cognition: an Interdisciplinary Study of the Preposition in English.* Cambridge University Press, 1986.

(Jackendoff, 1990) Ray Jackendoff. *Semantic Structures.* Current Studies in Linguistics Series. MIT Press, 1990.

(Jackendoff, 1991) Ray Jackendoff. Parts and boundaries. *Cognition*, 41:9–45, 1991.

(Jackendoff, 1992) Ray Jackendoff. What is Semantic Structures about? *Computational Linguistics*, 18(2), 1992. Letters to the Editor.

(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.

(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.

(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.

(Levin and Nirenburg, 1994) Lori Levin and Sergei Nirenburg. The Correct Place of Lexical Semantics in Interlingual MT. In *COLING94, Proceedings of the Fifteenth International Conference on Computational Linguistics*, 1994.

(Levin and Rappaport Hovav, 1992) Beth C. Levin and Malka Rappaport Hovav. Wiping the slate clean: a lexical semantic exploration. In Beth C. Levin and Steven Pinker, editors, *Lexical and Conceptual Semantics, Special Issue of Cognition: International Journal of Cognitive Science*. Blackwell Publishers, 1992.

(Levin, 1993) Beth C. Levin. The contribution of linguistics. In Madeleine Bates and Ralph Weischedel, editors, *Challenges in Natural Language Processing*. Cambridge University Press, 1993.

(Levison, 1996) Libby Levison. *Connecting planning and acting via object-specific reasoning*. PhD thesis, University of Pennsylvania, 1996. MS-CIS-96-08.

(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.

(Lochbaum, 1994) Karen E. Lochbaum. *Using Collaborative Plans to Model the Intentional Structure of Discourse*. PhD thesis, Harvard University, 1994. Available as Technical Report TR-25-94, Center for Research in Computing Technology.

(Mac Gregor, 1988) Robert Mac Gregor. A Deductive Pattern Matcher. In *AAAI88, Proceedings of the Seventh National Conference on Artificial Intelligence*, 1988.

(Moore, 1993) Michael B. Moore. Search Plans. Technical Report MS-CIS-93-56, Department of Computer Science, University of Pannsylvania, 1993. Thesis Proposal.

(Moore, 1995) Johanna D. Moore. *Participating in Explanatory Dialogues*. MIT Press, 1995.

(Nirenburg and Levin, 1992) Sergei Nirenburg and Lori Levin. Syntax-Driven and Ontology-Driven Lexical Semantics. In James Pustejovsky and Sabine Bergler, editors, *Lexical Semantics and Knowledge Representation*. Springer Verlag, 1992.

(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.

(Quantz and Kindermann, 1990) J. Quantz and C. Kindermann. Implementation of the BACK System Version 4. Technical Report KIT 78, Technische Universitat Berlin, 1990.

(Reiter, 1992) Ehud Reiter. Using classification to generate text. In *ACL92, Proceedings of the 30th Meeting of the ACL*, pages 265–272, 1992.

(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.

(Schoppers, 1988) Marcel Schoppers. *Representation and Automatic Synthesis of Reaction Plans*. PhD thesis, University of Illinois at Urbana-Champaign, 1988.

(Shapiro, 1992) Stuart C. Shapiro, editor. *Encyclopedia of Artificial Intelligence — Second Edition*. John Wiley and Sons, 1992.

(Stede and Rösner, 1994) Manfred Stede and Dietmar Rösner. Mapping Knowledge to Language with LOOM. In *Working Notes, AAAI Fall Symposium on Knowledge Representation for NLP in Implemented Systems*, November 1994.

(Steedman, 1990) Mark Steedman. Gapping as Constituent Coordination. *Linguistics and Philosophy*, 13:207–263, 1990.

(Swartout *et al.*, 1991) William Swartout, Cécile Paris, and Johanna Moore. Design for a Explainable Expert Systems. *IEEE Expert*, 6(3):58–64, 1991.

(Tenenberg, 1989) Josh Tenenberg. Inheritance in Automated Planning. In *KR89, Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*, 1989.

(Thompson, 1985) Sandra Thompson. Grammar and Written Discourse: Initial vs. Final Purpose Clauses in English. *Text*, 1-2:55–84, 1985.

(Traum *et al.*, 1996) David Traum, Lenhart K. Schubert, Nathaniel G. Martin, Chung Hee Hwang, Peter Heeman, George Ferguson, James F. Allen, Massimo Poesio, and Marc Light. Knowledge Representation in the TRAINS-93 Conversation System. *International Journal of Expert Systems, Special Issue on Knowledge Representation and Reasoning for Natural Language Processing*, 9(1), March 1996.

(TRH, 1988) Tile - Remodeling Handbook. Sunset Books. Lane Publishing Company, 1988.

(Vander Linden and Di Eugenio, 1996) Keith Vander Linden and Barbara Di Eugenio. A corpus study of negative imperatives in Natural Language instructions. In *COLING96, Proceedings of the Sixteenth International Conference on Computational Linguistics*, Copenhagen, Denmark, August 1996.

(Vander Linden and Martin, 1995) Keith Vander Linden and James Martin. Expressing Local Rhetorical Relations in Instructional Text. *Computational Linguistics*, 21(1):29–57, 1995.

(Vander Linden, 1993) Keith Vander Linden. *Rhetorical Relations 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.

(Webber and Baldwin, 1992) Bonnie Lynn 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 *et al.*, 1992) Bonnie Lynn 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.

(Webber *et al.*, 1995) Bonnie Lynn Webber, Norman Badler, Barbara Di Eugenio, Christopher Geib, Libby Levison, and Michael Moore. Instructions, Intentions and Expectations. *Artificial Intelligence, Special Issue on Computational Theories of Interaction and Agency*, 73, February 1995.

(Weida and Litman, 1994) Robert Weida and Diane Litman. Subsumption and Recognition of Heterogeneous Constraint Networks. In *CAIA94, Proceedings of the Tenth IEEE Conference on Artificial Intelligence for Applications*, 1994.

(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, 1983) Robert Wilensky. *Planning and Understanding. A Computational Approach to Human Reasoning*. Addison-Wesley Publishing Company, 1983.

(Wilks, 1992) Yorick Wilks. Review of 'Semantic Structures' by Ray Jackendoff. *Computational Linguistics*, 18(1):95–97, 1992.

(Winograd, 1972) Terry Winograd. *Understanding Natural Language*. Academic Press, 1972.

(Woods and Schmolze, 1992) William Woods and James Schmolze. The KL-ONE Family. *Computers and Mathematics with Applications, Special Issue on Semantic Networks in Artificial Intelligence, Part 1*, 23(2–5):133–178, 1992.

(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.

(Yen *et al.*, 1991) John Yen, Robert Neches, and Robert MacGregor. CLASP: Integrating Term Subsumption Systems and Production Systems. *IEEE Transactions on Knowledge and Data Engineering*, 3(1), 1991.

(Young and Moore, 1994) R. Michael Young and Johanna D. Moore. DPOCL: A Principled Approach to Discourse Planning. In *Seventh International Workshop on Natural Language Generation*, pages 13–20, Kennebunkport, Maine, 1994.

LIST OF TABLES

LIST OF FIGURES