1 Introduction

In this paper, I discuss a formalism that I devised and implemented to deal with complex instructions, in particular those containing Purpose Clauses. I argue that such formalism supports inferences that are an important pragmatic aspect of natural language, and that at the same time are related to surface reasoning, based on the syntactic structure of Natural Language; and moreover, that by using the kind of approach I propose, namely, first define linguistic terms and then use them in the part of the KB concerning the semantics of the domain, we can start bridging the gap between the two representation languages that the organizers of the symposium mention, the first used to capture the semantics of a sentence, the second used to capture general knowledge about the domain. Details on all the topics discussed here can be found in [Di Eugenio, 1993; Di Eugenio, 1994].

2 Motivations for the representation language

The characteristics of the formalism I propose derive from an analysis of an extensive corpus of Purpose Clauses, infinitival to constructions as in Do $\alpha$ to $\beta$; as some of these characteristics stem from the inferences necessary to interpret Purpose Clauses, it is with such inferences that I will start.

Interpreting $Do \alpha$ to $\beta$, where $\beta$ describes the goal to be achieved, in computational terms amounts to:

(1a) use $\beta$ as an index into the KB;
(1b) find a collection of methods $M_I$ that achieve $\beta$;
(1c) try to match $\alpha$ to an action $\gamma_{i,j}$ that appears as a component in $M_I$.

These are typical plan recognition inferences, eg see [Wilensky, 1983; Pollack, 1986; Charniak, 1988; Litman and Allen, 1990].

In all the work on plan recognition I know of, with the exception of [Charniak, 1988], match in step (1c) is taken to mean that $\alpha$ is instance-of $\gamma_{i,j}$. However, given the variability of NL action descriptions, we can’t assume that the input description exactly matches the knowledge that an agent has about actions and their mutual relations: my research focuses on computing a more flexible match between $\alpha$ and $\gamma_{i,j}$.

The two kinds of discrepancy between input and stored action descriptions I have examined so far concern structural consistency, and expectations that may need to be satisfied for a certain relation $R$ to hold between $\alpha$ and $\gamma_{i,j}$.

As regards structural consistency, consider an example such as

(2) $[Turn$ the screw]$_\alpha$ $[to loosen]_\beta$.

Presumably, the agent has some knowledge about the fact that $\gamma = Turn$ screw counterclockwise results in $\delta = Loosen$ screw.\footnote{This is true of the more common kind of screw.} There must be some way to understand that $\alpha = Turn$ screw is an underspecified version of $\gamma$, and that in fact $\gamma$ is the action to be performed. The same kind of reasoning should detect when $\alpha$ is more specific, or inconsistent with respect to $\gamma$, such as in the case of Turn the screw clockwise.

Now I will turn to computing expectations, that I will use as the main running example in the paper.
The expectations I have dealt with so far regard the location of objects that \( \beta \) manipulates, and arise when \( \alpha \) changes the perceptual space \( S \) the agent has access to. In particular, when \( \alpha \) results in the agent going to a place \( S' \) with the purpose of doing \( \beta \), one can infer \( S' \) to be the site of \( \beta \); thus, if there are objects that \( \beta \) manipulates, expect them to be at \( S' \). Consider

\[(3a) \quad [\text{Go into the kitchen}]_\alpha, [\text{to get me the coffee urn}]_\beta_1.\]

\[(3b) \quad [\text{Go into the kitchen}]_\alpha, [\text{to wash the coffee urn}]_\beta_2.\]

In both cases, the agent goes into the kitchen, which is then expected to be the location of \( \beta_1/\beta_2 \). In (3a), but not in (3b), a further expectation that the referent of the coffee urn is in the kitchen is developed. The difference between the two may be explained by appealing to the planning notion of qualifiers [Schoppers, 1988; Litman and Allen, 1990], conditions that must hold for an action to be relevant, and are not meant to be achieved. If \( \beta \) has among its qualifiers that an argument be at \( S' \) for \( \beta \) to even be relevant, then a locational expectation develops as in (3a). If not, a weaker expectation arises, as in (3b).

I hope that the previous examples give the reader the gist of my approach to NL instructions, which is to take into account both their surface form and the domain knowledge needed to interpret them: in many systems, the former is in a sense subordinate to the latter. In Ex. (3a), it is not just the domain knowledge, but the purpose relation between \( \alpha \) and \( \gamma \) that gives rise to the expectation — in this, my system performs surface reasoning, based on the syntactic structure of the NL.

To support this approach, it is necessary to represent action descriptions from both linguistic and planning points of view.

Linguistic characteristics include:

1. Like individuals, sets of individuals, etc., actions should be part of the ontology.
2. The formalism must be able to represent not only the usual participants such as agent or patient, but also means, manner, extent, etc which are pervasive in NL instructions.
3. The action described in the input may not exactly match the actions the system knows about, and it is impossible to define all action descriptions a priori. Therefore, the formalism must be able to support inferences to understand the relation between the input action descriptions and the stored ones.

As regards planning knowledge about actions:

1. Classic planning notions, such as action decomposition into substeps, qualifiers, effects, must be provided.
2. The formalism must be able to represent various relations among actions, such as at least temporal relations, generation and enablement. Intuitively, generation holds between \( \alpha \) and \( \beta \) if \( \beta \) is done by executing \( \alpha \); enablement holds between \( \alpha \) and \( \beta \) if \( \alpha \) brings about conditions necessary to execute \( \beta \). See [Goldman, 1970; Pollack, 1986; Balkanski, 1993]. In [Di Eugenio, 1993] I show that purpose clauses do express generation and enablement.
3. The formalism must be either usable by “lower level” processes or be easily interfaced to them. I believe the latter is true both of my KBs, and of the plan graph, the output of my interpretation module, that is used as input by other AnimNL modules — AnimNL is the system my work is embedded in, see below.

To address these issues, I have devised and implemented a formalism composed of two KBs. The first, the action taxonomy, stores linguistic knowledge about actions, the second, the plan library, contains action recipes [Pollack, 1986; Balkanski, 1993], ie common sense planning knowledge about actions. Both are implemented by means of the hybrid system CLASSIC [Brachman et al., 1991]: the terms defined in the action taxonomy are used in the recipes in the plan library.

The framework for the work presented here is the AnimNL (Animation from Natural Language) project [Webber et al., 1992; Webber et al., 1993]. AnimNL has as its goal the automatic creation of animated task simulations from NL instructions. Its agents are animated human figures, engaged in assembly procedures and maintenance procedures. The underlying animation system, Jack/TM, provides human figures capable of realistic motion through model-based inverse kinematics [Badler et al., 1993]. Both agents and spatial environments can be modified, so as to vary agents' physical capabilities and the situation in which tasks are carried out.

3 The representation formalism

3.1 The action taxonomy

A hybrid formalism by itself is not sufficient to provide a sound lexical decomposition for verbs, as it lacks a well-defined set of semantic primitives.\(^3\) I found a source for such semantic primitives in

\(^2\)Thus, the distinction between action taxonomy and plan library is conceptual rather than real.

\(^3\)In the sense of the linguistic notion of semantics.
Conceptual Structures — CSs for short [Jackendoff, 1990]. CSs are well suited to represent action descriptions, as they capture generalizations about actions and their relationships to one another; moreover, CS representations are suitable to express the logical form of an instruction, as they reveal where information may be missing from an utterance and has to be provided by inference.

The advantages of integrating the two formalisms are: using linguistically sound primitives in the T-Box transforms it into a real lexicon; a KL-ONE style representation endows CSs with a hierarchical organization and with the possibility of extending the lexicon.

A CS entity may be of ontological type Thing, Place, Path, Event, State, Manner or Property. CSs may also contain complex features generated by conceptual functions over other CSs. For instance, the function TO: Place → Path describes a Path that ends in the specified Place:

\[
\text{(4)} \quad \text{[Path TO([Place IN([Thing kitchen]_k)])]_m}
\]

Among functions that yield Events, we have GO: Thing x Path → Event:

\[
\text{(5)} \quad \text{[Event GO ([YOU]_i, [Path TO([IN([KITCHEN]_k)])]_m)]}
\]

CSs are readily integrated into CLASSIC, as shown in Fig. 1, that presents the portion of T-Box relative to the CS primitive GO — the different kinds of GO's shown vary on the semantic field, such as spatial or control, and on the kind of path. Jackendoff introduces semantic fields to capture the similarities between sentences like Jack went into the kitchen and The gift went to Bill. The semantic field control was introduced in [Di Eugenio and White, 1992] to represent the notion of having control over some object.

3.2 The plan library

A network representation of recipe concepts is shown in Fig. 2. It illustrates the move-recipe, which represents a method for (physically) moving an object from one location to another; the same recipe is shown, in perhaps a more readable format, and with more details, in Fig. 3. In Fig. 2, SAME-AS constraints, necessary to express coreference, e.g. between the arguments of an annotation and the substeps of the recipe it belongs to, are not shown.

Recipes have a header, body, qualifiers and effects. The terminology, especially header and body, is reminiscent of STRIPS, but the relations between these components are expressed in terms of enablement and generation, e.g. the body generates its header. The representation does not employ preconditions, because it is very difficult to draw the line between what is a precondition and what is part of the body of the action — see [Di Eugenio, 1993; Geib, 1992]. Action recipes express what is traditionally expressed by means of preconditions by means of actions, which are substeps in the body that generates a certain header. Other components of a recipe are the annotations on the body, that specify the relations between the substeps, e.g. enablement and temporal relations, the latter derived from [Allen, 1984]; qualifiers; and effects, what must be true after an action has been executed.

3.3 The role of the hybrid system

I adopted CLASSIC as the underlying backbone of the whole system because I found the classification and realization inferences that hybrid systems provide both powerful enough and restricted enough to compute the relation between the action described in the input and the stored ones.

Hybrid systems have often been used in the literature to maintain an organized KB of action concepts and plans, to facilitate adding new concepts and to infer of which plan type a certain sequence of events in the world is an instance. For example, in CLASP [Devanbu and Litman, 1991], which is the closest to my work among the systems that exploit subsumption to deal with plans, networks or rules, plans are built out of actions, defined as classic STRIPS planning operators. I go one step further by including linguistic knowledge, totally absent from CLASP, and by exploiting classification also in cases in which the input action description is an instance of a virtual concept.

The hybrid system inference engine is used as follows.

First of all, the individual action descriptions contained in the logical form⁴ are asserted into the A-Box, whose recognizer computes the set of types of which such descriptions are instances.

Then, the goal β is used as an index into the plan library, namely, the CLASSIC query language is used to find those headers of which β is an instance: the corresponding recipes Ri’s are then retrieved.⁵

At this point, for each retrieved recipe Ri, α is checked against each substep γj for consistency, namely to understand whether α is less or more specific, or inconsistent with γj.

In the case of (2), the recipe relating γ = Turn screw counterclockwise and δ = Loosen screw is re-

---

⁴The logical form is generated by a Combinatory Categorial Parser [White, 1992].
⁵The match between β and headers is constrained to be instance-of; however, see [Di Eugenio, 1993] on possible relaxations of this restriction, in which β is allowed to be less specific than the header of a recipe.
Figure 1: The go Hierarchy
Figure 2: The Recipe Hierarchy
<table>
<thead>
<tr>
<th>Header</th>
</tr>
</thead>
<tbody>
<tr>
<td>[CAUSE([AGENT]_i, [GOstellar(j, k)])]</td>
</tr>
<tr>
<td>[FROM(m)]</td>
</tr>
<tr>
<td>[TO(l)]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Body</th>
</tr>
</thead>
<tbody>
<tr>
<td>- [GOstellar(i, [TO(m)])]_{\gamma_1}</td>
</tr>
<tr>
<td>- [CAUSE(i, [GOcontrol(j, [TO([AT(i)])])])]_{\gamma_2}</td>
</tr>
<tr>
<td>- [GOstellar(i, k)] [WITH(j)]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Annotations</th>
</tr>
</thead>
<tbody>
<tr>
<td>- (\gamma_1) enables (\gamma_2)</td>
</tr>
<tr>
<td>- (\gamma_2) enables (\gamma_3)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Qualifiers</th>
</tr>
</thead>
<tbody>
<tr>
<td>- [BEstellar(j, m)]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Effects</th>
</tr>
</thead>
<tbody>
<tr>
<td>- [BEstellar(j, l)]</td>
</tr>
</tbody>
</table>

Figure 3: A Move Something Somewhere Recipe
retrieved; then $\alpha = \text{Turn the screw}$ is found to subsume $\gamma$, thus the action to be performed is taken to be $\gamma$. If $\alpha$ were $\text{Turn the screw carefully}$, the algorithm would infer that $\alpha$ and $\gamma$ are mutually consistent, and that the action to be performed is $\text{Turn the screw counterclockwise carefully}$; if $\alpha$ were $\text{Turn the screw clockwise}$, $\alpha$ and $\gamma$ would be found to be inconsistent. In this latter case, the instruction $\text{Turn the screw clockwise to loosen}$ could actually be felicitous, if the screw were of the appropriate, less common type; it is left for future work how to deal with default assumptions.

Let's now turn to the case of computing expectations, which arise from consistency checks that are more complex than those discussed so far.

In the case of (3a), after move-recipe has been retrieved through $\beta_1$, $\alpha_1 = \text{go(you, to(in(kitchen2)))}$ and the three substeps $\gamma_1, \gamma_2, \gamma_3$ are checked for consistency. $\alpha_1$ is found to be an instance of $\gamma_1$, with $i$ bound to $\text{you}$ and $m$ to $\text{in(kitchen2)}$.

Then, the algorithm checks whether there is any other information on $i$ or $m$ in Recipe that makes $i$'s or $m$'s bindings untenable, such as $m = \text{outdoors}$. The only other constraint on these parameters comes from the qualifier, instantiated to $\text{BE(coffee-urn3,m)}$. As the binding $\text{in(kitchen2)}$ for $m$ is consistent with the type restriction place placed on $m$ by $\text{BE}$, we can conclude $\text{BE(coffee-urn3, in(kitchen2))}$, and produce the plan graph in Fig. 4.

As regards (3b), the recipe for wash doesn't include such a qualifier, thus the expectation about the coffee urn being in the kitchen doesn't arise; however, the weaker expectation $\text{BE(washing-site, in(kitchen2))}$ is derived as a side effect of the consistency checks.

4 Conclusions

I have presented a formalism in which both linguistic and planning knowledge about actions are represented; the two KBs are unified by using CLASSIC as their common backbone.

On the other hand, I believe that this is the right direction to go, as it is necessary, first, to represent both linguistic and planning knowledge about actions, second, to provide flexibility of match between the logical form and the stored knowledge. Moreover, notice that complex objects — in this case, of type action — find a very natural treatment through the concept forming language of the hybrid system.

On the other hand, I am fully aware that I have totally avoided important problems such as quantifiers. Among the questions I would like to explore at the symposium is how far this formalism can go, more specifically as far as representing quantified expressions and negation: negation is very frequent in instructions, to be found, either implicitly or explicitly, in both warnings and (side) effects to be avoided.

References


Figure 4: The plan graph for Go into the kitchen to get me the coffee urn


