# Converting Natural Language Route Instructions into Robot Executable Procedures

S. Lauria T. Kyriacou G. Bugmann
Robotic Intelligence Laboratory
School of Computing
University of Plymouth
Drake Circus, Plymouth PL4 8AA
United Kingdom

J. Bos E. Klein
ICCS
Division of Informatics
University of Edinburgh
Buccleuch Place, Edinburgh EH8 9LW
Scotland, United Kingdom.

**Abstract**

*Humans explaining a task to a robot use chunks of actions that are often complex procedures for robots. An instructable robot needs to be able to map such chunks to existing pre-programmed primitives. We investigate an architecture for spoken dialogue systems able to extract executable robot procedures from user instructions. A suitable representation of the dialogue is introduced, then a Procedure Specification Language (PSL) is described that allows to extract from the semantic representation of the dialogue the robot executable procedures and their parameters*

## 1 Introduction

This paper presents an semantically based approach for human-robot dialogue understanding, as part of a project than envisages "Instruction-Based Learning" (IBL) [5], where robots acquire user-specific skills based on verbal instructions given by the user. In particular, we will focus on mapping the human language commands to for the robot executable instructions, using an intermediate semantic representation.

Our IBL system operates according to the following scenario. A user engages in a dialogue with the robot, where spoken instructions are mapped to semantic representations, natural language ambiguities are resolved, and the functional parameters are extracted from that representation [9]. The robot, having a database with previously learned tasks at its disposal, will now either perform the given instruction (if it knows how to do it), or if the task is unknown, ask the user to explain how to perform the task. The user then explains the task step by step. At the end of this learning process, the robot will have built a new procedure that becomes part of its knowledge base.

The requirements of natural language understanding induce the internal model of a route as a sequence of high-level task specifications (primitives). Hence it is necessary to provide the robot with a set of pre-programmed primitives corresponding to action chunks referred to by users. For more details about these aspects see [7].

A typical example in our scenario is the following (example `u8_GC_HD` extracted from the IBL corpus):

> *Instructor*: Go to the post office!
> *Robot*: How do I get to the post office?
> *Instructor*: Er head to the end of the street. Turn left. Take the first left. Er go right down the road past the first right and it's the next building on your right.

One of the issues in the project is the mapping from action chunks used in natural language to actions executable by the robot for both of the possible situations: either the system already knows how to perform a request, or it has to learn how to perform it. The first case corresponds to a successful mapping from the semantic analysis of the request to a sequence of executable robot actions. The second case corresponds to the creation of a sequence of executable robot actions for the unknown request through a user-robot dialogue.

Previous approaches to interpreting natural language instructions for mobile robots assume a application specific semantic representation [4]. However, we argue that there is a need for a domain-independent intermediate representation. This representation captures the meaning of the dialogue between user and robot and is used to resolve ambiguities inherent in natural language (for instance the reference of the pronoun it the example above). In addition, we use an application specific mapping from the intermediate representation to obtain robot executable scripts. Using this extra layer results in an overall system that is much easier to adapt the robot to new scenarios or tasks.

The paper is structured as follows. First we introduce the intermediate semantic representations known as Discourse Representation Structures (Section 2). In Section 3 we presents the Procedure Specification Language (PSL) used for the interpretation of the DRS. Section 4 illustrate the conversion of basic program components found in verbal instructions into robot-executable procedures. In Section 5 ongoing work covering the reuse of previously explained routes is discussed.

## 2 Understanding Natural Language Instructions

We will use Discourse Representation Structures (DRSs) to represent the meaning of the dialogue between user and system. There are three reasons that motivate this choice of formalism. First and foremost, DRT is a well understood framework and covers a wide variety of linguistic phenomena [6, 11]. These phenomena include context-sensitive expressions such as pronouns and presuppositions. To our knowledge, there is no other semantic formalism that comes close to the empirical coverage of DRT. Second, there now exist computational implementations that provide means to extend existing linguistic grammars with DRS-construction tools, and there are efficient algorithms available that implement Van der Sandt's presupposition projection algorithm for DRT [2]. Third, there is a direct link between DRT and first-order logic—there is a translation from DRSs to formulas of first-order logic that behaves linear on the size of the input [1].

### 2.1 Representing Instructions

DRT was initially designed to deal with texts, so we will use an extension of standard DRT that enables us to cope with instructions such as given in the example above. This extension introduces *actions* and modal operators into the DRS-language.

Let us first define the syntax of the DRS language. Basic DRSs have two components: a set of *discourse referents*, and a set of *conditions* upon those referents. Discourse referents stand for objects mentioned in the course of the dialogue. Conditions constrain the interpretation of these discourse referents. More formally, DRSs and merge of DRSs are defined in the usual way:

**Syntax of DRSs:**

1. If $\{x_1, \ldots, x_n\}$ is a set of discourse referents, and $\{\gamma_1, \ldots, \gamma_m\}$ is a set of DRS-conditions, then the ordered pair $\langle\{x_1, \ldots, x_n\}, \{\gamma_1, \ldots, \gamma_m\}\rangle$ is a DRS;
2. If $B_1$ and $B_2$ are DRSs, then so is $(B_1 \oplus B_2)$.

Following Lascarides [8], we extend the DRS language with action terms. Atomic action terms are identified by the $\delta$-operator. Complex action-terms are composed out of other action terms by either ; (sequence) or | (free choice).

**Syntax of DRS-action-terms:**

1. If B is a DRS, then $\delta$B is a DRS-action-term;
2. If $A_1$ and $A_2$ are DRS-action-terms, then so are $(A_1;A_2)$ and $(A_1 \mid A_2)$.

The DRS-condition subsume those of standard DRT. Further we have the modal operators $\Box$ and $\Diamond$ (clauses 3 and 6), hybrid DRS-conditions formed by discourse referents and DRSs (clause 5), and the command operator (clause 7).

**Syntax of DRS-conditions:**

1. If $R$ is a relation symbol for an $n$-place predicate and $x_1 \ldots x_n$ are discourse referents then $R(x_1,\ldots,x_n)$ is a DRS-condition;
2. If $x_1$ and $x_2$ are discourse referents, then $x_1 = x_2$ is a DRS-condition;
3. If B is a DRS, then $\neg$B, $\Box$B, $\Diamond$B are DRS-conditions;
4. If $B_1$ and $B_2$ are DRSs, then $B_1 \vee B_2$, $B_1 \Rightarrow B_2$ are DRS-conditions;
5. If x is a discourse referent and B a DRS, then x:B is a DRS-condition;
6. If A is a DRS-action-term, and B a DRS, then [A]B and $\langle$A$\rangle$B are DRS-conditions;
7. If A is a DRS-action-term then !A is a DRS-condition.

One of the theoretical motivations behind the internal structure of DRSs is the analysis of pronouns and other anaphoric expressions. Pronouns are interpreted in DRT by binding a previously introduced *accessible* discourse referent. Accessibility is governed by the way DRSs are nested into each other, and hence narrows down the choice of an antecedent in the process of pronoun resolution.

## 2.2 Example Representations

We will now illustrate the formal syntax definition by giving some examples that show how instructions can be modelled. We will use the more convenient box notation for DRSs in the examples that follow. Recall that we use the $\delta$-operator to form action-DRSs from DRSs and the ! operator to express that an action is commanded. So, the directive Go to the post office! translates to the following DRS:

$$\begin{array}{|l|} \hline \text{x y} \\ \hline \text{robot(x)} \\ \text{postoffice(y)} \\ !\delta \begin{array}{|l|}\hline \text{e} \\ \hline \text{go(e)} \\ \text{to(e,y)} \\ \text{agent(e,x)} \\ \hline \end{array} \\ \\ \hline \end{array}$$

This DRS states that the plan in the actual world contains the action for the robot to go to the post office. Semantically, actions relate two possible worlds: the world (or state) in which the action is issued, and the world in which the effects of the action hold. Because actions themselve can be of complex nature, we will use an additional world that describes what constitutes the action. This enables us to reason about possible outcomes of actions (not only for the purpose of planning, but also to verify that the resulting states are desired) and to check the preconditions of actions.

## 2.3 Interpretating Instructions

One way to interpret DRSs is to translate them to ordinary first-order logic. This is the approach that Bos & Oka follow [3], and they use classical first-order theorem provers and model builders to automate inference. The translation they use is based on the relational translation for modal logic to first-order formulas and essentially similar to the standard translation from DRT to first-order logic [6], extended with rules to deal with the modal operators and DRS-action-terms. The example DRS above would get the following translation in first-order logic:

$$\exists w\ \exists x\ \exists y\ (possible\_world(w) \wedge robot(w,x) \wedge postoffice(w,y) \wedge \exists v\ \exists a\ (action(w,a,v) \wedge \exists e\ (go(a,e) \wedge to(a,e,x) \wedge agent(a,e,y))))$$

Note that the translation increases the arity of all predicates symbols with one, where the additional argument position denotes a possible world. A miminal first-order model satisfying this formula (and further background knowledge in the form of meaning postulates describing the pre-conditions and effects of actions) could contain the following information:

```
D={d1,d2,d3,d4,d5,d6,d7,d8}
F(possible_world)={d1,d2,d3}
F(robot)={(d1,d4),(d2,d4),(d3,d4)}
F(postoffice)={(d1,d5),(d2,d5),(d3,d5)}
F(action)={(d1,d2,d3)}
F(go_from_to)={(d2,d4,d6,d5)}
F(at_loc)={(d1,d4,d6),(d3,d4,d5)}
```

Bos & Oka [3] actually emply automated model builders to generate models of these kind, and use these to extract actions for the dialogue manager. Since models are essentially flat structures without recursion, they are easy to process. For instance, all quantification and boolean structures are explicit in models. This makes models ideal to function as a database-lookup table to find out whether there are actions to be performed by the system.

However, the state-of-the-art in automated model building is not in a stage yet where it leads itself easily to integration in efficient implementations. Although the model building methods performs well for examples up to a few utterances, in general the instructions in the IBL corpus are much larger than that and sometimes reach ten to twenty utterances in a learning dialogue. Therefore we use an alternative rule-based method to extract executable primitives from DRSs. This technique is much more efficient and will be presented in the next section.

## 3 Procedure Specification Language

The internal representation of a route is a sequence of high-level task specifications (primitives). For this reason a production-rule based approach has been used to interpret the DRS as a sequence of procedure names.

The Procedure Specification Language (PSL) provides a common interchange language to describe resources. A list of robot executable procedures are extracted from the DRS and saved as a new procedure -the result of Instruction Based Learning-. The PSL provides the skeletal syntax used to compose the procedure names and the required parameters.

The PSL terms are either *special characters* or regular *string literals*, where string literals are made of sequences of characters excluding the special characters. The complete set of special characters that cannot appear as part of a string literal is:

```
&     |     #     %     $     ->
```

These characters can only be used for the special syntactic forms described in the above RSL syntax overview

The core syntax of the PSL syntax is the *rule* `a->b`. Rules associate the *condition* `a` on the left of the special syntax `->` with the string on the right of `->` corresponding to the robot procedure. For example, the rule

$$\texttt{event(X)\&go(X)\&to(X,Z)\&\$landmark(Z)}->\\ \texttt{go(prep='to',landmark = \$landmark(Z))} \quad (1)$$

will generates the procedure

```
go(prep='to',landmark='postoffice')
```

from the DRS in Section 2.2.

In each PSL rule, the condition is a conjunction of terms separated using the special syntax `&`, where each term can be a one or two place predicate symbol, a variable predicate, a variable action. In (1) `event(X)` is an example of a one place predicate while `to(X,Z)` is a two place predicate. The upper case symbol in parenthesis (i.e `X` for `event(X)`) is the variable associated with the predicate. In the example in Fig 1, only the `event,go,to` predicates with the same value for `X` can be considered to satisfy the condition for rule (1).

A variable predicate is indicated as the special symbol `$` followed by a string litteral. A variable predicate indicates a class of possible predicates. In the rule example (1), `$landmark(Z)` specifies that the predicate must be of landmark type.

A list containing all the predicate belonging to each class defined must be included with the PSL. The syntax to specify a class and all the members belonging to it is:

```
class_name:predicate_1|predicate_2|....|...
```

where `class_name` is the string litteral indicating the class and `predicate_1|predicate_2` is the list of terms `predicate_1,predicate_2` belonging to the class separated by the special symbol `|`.

An action predicate is indicated as the special symbol `#` followed by a predicate pointing to an action. For example, an action predicate can indicate an action to be executed while executing another action (for example sure from the hospital er go forwards until you come to dixons extracted from `u9_GC_HW` in the IBL corpus)

The end of both a rule and a class list is indicated by the the special syntax `%`. The list of the defined class is preceded by the string `#parameters#`,while the list of the rules is preceded by the string `#rules#`.

The PSL rule based approach facilitates the interpretation of a user command into a call to a procedure with the correct parameter associated to it. The introduction of parametrised primitives allows it to generalise the use of the procedure. For instance, the procedure designed for *turn left after the tree* should also work if the value *tree* for the parameter `landmark` is replaced by the value *church*. It is also possible to pass different combinations of parameters to the primitive procedure.

While, as explained in more detail in [9], the choice of the initial set of primitives is corpus based (that is it has been driven by the way users express themselves), both the parameter combinations and the interpretation of predicates into a parameter value is mainly robot driven as explained in more detail in [7].

The PSL rule syntax allows to establish the desired mapping between the predicates from the DRS representation and their interpretation into the correct value for the correct parameter. For example the user utterances: turn right and take a right should produce the same precedure call despite being represented as two different types of events in the DRS (i.e. as a turn and a take action rispectively). Table 1 shows two possible rules allowing to obtain the same procedure call for both utterances.

```
event(X)&turn(X)&in(X,Z)&$direction(Z)->
turn($direction(Z))

event(X)&take(X)&$direction(Z)->
turn($direction(Z))
```

Table 1: PSL rules. Example of two rules mapping different symbolic representation of an action into the same procedure.

Not all the information present in the DRS is used in detecting the condition components of the rule. One aspects still not yet fully implemented is the use of negation in the condition part of the rule. This would allow the designer to exclude undesired combination of predicates to be mapped into a rule.

## 4 Basic Program Components

A requirement in Instruction Based Learning is that components such as conditionals, loops, sequences found in instructions are correctly converted into robot executable procedures. Utterances containing conditional expressions have not been found in this corpus. Here, instructions consist mainly of sequences and loops.

### 4.1 Sequences

Since the order of the actions in the utterance is preserved by the DRS, extracting a properly ordered sequence of primitives and building the corresponding procedure code is straight forward. For instance, the pseudocode for the user explanation (example extract from `u22_GB_CD` in the IBL corpus):

> *Instructor*: er you have to take right and then again the first right

is shown in Table 2.

```
def action():
....
    take(direction='right')
    take(direction='right',ordinal='first')
.....
```

Table 2: Sequential Instructions. Pseudocode for the sequence of procedures obtained from a sequence of user actions

## 4.2 Loops

References to loops where an action has to be executed a fixed number of times are not found in the corpus. However, while-loops and do-until-loops are frequently found. These can either be explicit or implicit.

Implicit while-loops are found in actions such as in the example extracted from `u22_GB_CD` in the IBL corpus:

> *Instructor*: er you have to take right

This action implicity requires from the robot to search for the landmark *right_turning* while following the road. Such implicit loops are handled inside pre-programmed procedures (e.g. `take(direction='right',ordinal='first')`. See [7] for more details.

However, an action can be explicitly described as loop by the user in utterances such the one from `u20_GB_EP` extracted from the IBL corpus:

> *Instructor*: ..keep turning right until you ve got the grand hotel on your left..

With the PSL it is possible to define a suitable rule which allows to introduce the loop explicitly on the right hand side of it. For example, the rule in Table 3 will produce the pseudocode in table 4 for the utterance from `u20_GB_EP`. As a result, everytime the program in Table 4 is called the loop will be executed.

```
event(X)&turn(X)&in(X,Z)&$direction(Z)
&until(X,C)&#proposition(C) ->
while !(#proposition(C)):
        turn(direction=$direction(Z))
```

Table 3: Loop.Example of a rule extracting an explicit while-loop.

```
def action():
....
    while !(near(landmark='grand_hotel'):
           turn(direction='right')
....
```

Table 4: Loop. Pseudocode for the utterance `u20_GB_EP` .

## 5 Reusing Previously Learnt Procedures

One of the key features of IBL is the reuse of previously explained procedure as part of explanations of new more complex procedure. In the corpus of route instructions, this takes the form of previously explained routes being reused in later route explanations. Three possible ways of reusing previous routes can be found in the corpus. In the first case, the user explicitly refers to the use of the whole of a previously explained route followed by a series of actions. The following example is extracted from `u12_GA_EG` in the IBL corpus:

> *Instructor*: go to the post office at the post office turn left take a right at the crossroads tescos is on the left hand side of the street

In the second case, the user still explicitly refers to a previously explained route. However, this time the route has to be used only partially since at a given point (e.g. a landmark) a diversion is introduced by the user. The following example `u6_GC_CM` is extracted from the IBL corpus:

> *Instructor*: right erm head as though you re going towards the post office so you go over the bridge but instead of carrying straight on take a right carry on down that road until it bears round to the right slightly and at the end of the road the museum is there

In the third case, the user does not explicitly refers to a previously explained route, but only refers to a landmark used in it. Thus that route has to be inferred. The following `u13_GA_CL` example is extracted from the IBL corpus:

> *Instructor*: go to the bridge mentioned previously continue over the crossroads immediately after the bridge and follow the road to its end on your right you ll find the queens pub

In all these cases the system must be able to correctly link previously learnt sequences of actions with new instructions. As explained in more detail in [10], during the execution of a sequence of actions, the final state of the robot after an action must be compatible with the initial state of the next action. As a consequence, the recalled procedure has to be *tailored* so that the next procedure can be successfully executed.

For example in the utterance `u12_GA_EG`, the first action recalls the procedure to go to the post office, which ends with the robot facing the entrance of the post office, but this is not a suitable initial state for the next action (i.e turn left). So for the robot to succeed, the procedure `go_postoffice()` should not be executed entirely. However, to determine which elements of a previously learnt sequence must be kept is not an easy problem.

A solution to this problem could be the rule in Table 5. In this case the procedure `go_postoffice()` is executed until the condition `near(landmark='postoffice')` is verified, where `near(landmark='postoffice')` is a vision based procedure that check whether the robot is near the post office.

```
event(X)&go(X)&to(X,Y)&postoffice(Y)->
while  !(near(landmark='postoffice'):
        go_postoffice()
```

Table 5: Linking Sequences. The procedure `go_postoffice()` is executed until the robot is near the post office.

Then, the part of the procedure `go_postoffice()` which drives the robot into a position facing the post office, will not be executed. Note that this implies a concurrent execution of the two procedures.

Future work will cover the resolution of the problem above also for more complex cases such as the `u13_GA_CL` example and will evaluate the efficacy of the various components presenteded in this paper in converting Natural Language instructions into robot procedures.

## Acknowledgments

This work is supported by EPSRC grants GR/M90023 and and GR/M90160.

## References

[1] Patrick Blackburn, Johan Bos, Michael Kohlhase, and Hans de Nivelle. Inference and Computational Semantics. In Harry Bunt, Reinhard Muskens, and Elias Thijsse, editors, *Computing Meaning*, volume 2, pages 11–28. Kluwer, 2001.

[2] Johan Bos. Implementing the binding and accommodation theory for anaphora resolution and presupposition projection. *Computational Linguistics*, to appear.

[3] Johan Bos and Tetsushi Oka. An Inference-based Approach to Dialogue System Design. In *Proceedings of Coling 2002*, 2002.

[4] C. Crangle and P. Suppes. *Language and Learning for Robots.* CSLI Lecture Notes 41. Chicago University Press, Stanford, 1994.

[5] `http://www.tech.plym.ac.uk/soc/staff /guidbugm/ibl/index.html`.

[6] Hans Kamp and Uwe Reyle. *From Discourse to Logic; An Introduction to Modeltheoretic Semantics of Natural Language, Formal Logic and DRT.* Kluwer, Dordrecht, 1993.

[7] Theocharis Kyriacou, Guido Bugmann, and Stanislao Lauria. Personal Robot Training via Natural-Language Instructions. In *Proceedings of IROS 2002*, 2002. to appear.

[8] Alex Lascarides. Imperatives in Dialogue. In *Proceedings of the 5th International Workshop on Formal Semantics and Pragmatics of Dialogue (BI-DIALOG)*, pages 1–16, Bielefeld, Germany, 2001.

[9] Stanislao Lauria, Guido Bugmann, Theocharis Kyriacou, Johan Bos, and Ewan Klein. Training Personal Robots Using Natural Language Instruction. *IEEE Intelligent Systems*, pages 38–45, September/October 2001.

[10] Stanislao Lauria, Guido Bugmann, Theocharis Kyriacou, and Ewan Klein. Mobile Robot Programming Using Natural Language. *Robotics and Autonomous Systems*, pages 171–181, 2002.

[11] Rob A. Van der Sandt. Presupposition Projection as Anaphora Resolution. *Journal of Semantics*, 9:333–377, 1992.