Situation-Aware Interpretation, Planning and Execution of User Commands by Autonomous Robots

Michael Brenner
Institute for Computer Science
Albert-Ludwigs-University
Freiburg, Germany

***Abstract*— For a robot to be able to first understand and then achieve a human's goals, it must be able to reason about a) the context of the current situation (with respect to which it must interpret the human's commands) and b) the future world state (as intended by the human) and how to achieve it. Since humans express their intentions and plans using qualitative symbolic representations, robots must be enabled to reason and interact on the same representational level. In this paper, we describe the use of classical AI Planning techniques for situation-aware interpretation and execution of human commands. We show how, based on a Planning domain, a robot can be enabled to understand commands in natural language, plan for their situation-dependent realization and revise its plans based on new perceptions. We show the effectiveness of this approach in several HRI scenarios modeled as Planning domains as well as with examples from a real robot system developed in the EU-funded CoSy project.**

I. INTRODUCTION

Intelligent service robots are expected, in the near future, to understand natural language (NL) commands and to find situation-dependent ways to perform these tasks autonomously. This vision can only become true if robots can reason about tasks on the same high level of abstraction that the humans they interact with use to specify them. In particular, a robot must be able

- to interpret an NL command as a goal it must achieve,
- to intelligently plan *how* to achieve that goal based on its knowledge about the current situation,
- and to execute the planned behaviour
- or realize that it lacks information, needs help in performing the task or simply does not understand the command.

Additionally, to enable the broadest possible range of application domains for such robots and to ease changing and extending them,

- these capabilities should be *independent* of a particular application domain,
- rather, the general NL and problem-solving knowledge of a robot should be easily applicable to any particular domain of expertise.

Consider, for example, the commands in Ex. 1. They belong to largely differing application domains, each of which requires situation-aware task interpretation and planning.

(1) a. *Please put the dirty plates into the dishwasher.*
b. *Show me the exit, please.*
c. *Put the small cube near the red ball on the big blue cube.*
d. *Would you open the windows in the bedroom, please?*

In this paper, we present a generic approach for enabling robots to talk, reason and act in such different domains. It is based on modelling a robot's capabilities as an AI Planning domain. The main new contributions of our approach are

- a method for analysing a given planning domain and *automatically* generating a basic NL parser and lexicon for talking about it,
- an algorithm for the semantic interpretation of a NL command as a goal formula in the corresponding planning domain.

Together, these methods enable a robot to understand and execute NL commands that correspond to complex situation-dependent activities. If, for example, the robot is given the command in Ex. 1a and has just sensed two dirty plates, it may devise the following plan: ⟨*pick-up plate1; pick-up plate2; move-to kitchen; open dishwasher; put plate1 dishwasher; put plate2 dishwasher*⟩. If, during execution of this plan, the robot perceives more dirty plates, it will *update* the plan automatically in order to achieve the human's goal.

Our approach has been implemented and tested for various planning domains. For example, our implemented system can understand and find goal descriptions for all commands in Ex. 1. It can also detect, e.g., that Ex. 1c is *ambiguous* and resolve that ambiguity depending on the context of the current situation – i.e. should the small cube be put near the red ball or is it there already and should be put on the big blue cube?

The paper is structured as follows. In Sec. II we discuss the application of AI Planning to robotics, especially in a situation-aware *continual* planning process. Sec. III describes how a Planning domain can be used to enable a robot to syntactically parse NL commands for that domain. For being able to execute these task, the robot must be able to interpret the command semantically. This is described in Sec. IV. An outlook to current and future work is given in Sec. V.

II. AI PLANNING FOR INTELLIGENT ROBOT BEHAVIOUR

The purpose of an AI Planning system, i.e. a *planner*, can, in its most basic form, be described as follows: to find

Fig. 1 The *household* domain in MAPL (excerpt)

```
(:types
   phys˙obj property - object
   size dirtiness - property
   agent movable unmovable room - phys˙obj
   kitchen living˙room - room
   dishwasher table - unmovable
   plate cup - movable)

(:constants
   big small medium - size
   clean dirty - dirtiness)

(:state-variables
   (colour ?o - phys˙obj : ?c - colour)
   (size ?o - phys˙obj : ?s - size)
   (dirtiness ?o - phys˙obj : ?d - dirtiness)
   (pos ?o - phys˙obj : ?p - phys˙obj))

(:action open-dishwasher
 :agent (?a - agent)
 :parameters (?d - dishwasher)
 :variables (?r - room)
 :precondition (and (not (open ?d))
     (pos ?d : ?r) (pos ?a : ?r))
 :effect (open ?d))

(:action put
 :agent (?a - agent)
 :parameters (?obj - movable ?p - unmovable)
 :precondition (pos ?obj : ?a)
 :effect (pos ?obj : ?p))

(:action pick-up
 :agent (?a - agent)
 :parameters (?obj - movable ?p - unmovable)
 :precondition (pos ?obj : ?p)
 :effect (pos ?obj : ?a))
```

Fig. 2 A state in the *household* domain (excerpt)

```
(:objects
   robot - agent
   k - kitchen
   lr - living˙room
   plate1 plate2 plate3 - plate
   dw - dishwasher)

(:init
   (pos robot : lr) (pos dw : k)
   (pos plate1 : lr) (dirtiness plate1 : dirty)
   (pos plate2 : lr) (dirtiness plate2 : dirty)
   (pos plate3 : lr) (dirtiness plate3 : clean))
```

a sequence of actions that transform a given *initial state* into another state which satisfies some *goal condition*. A state can be regarded as a databases of logical facts and actions as transactions manipulating that database. The power of Planning lies in its *domain independence*: states, actions and goals are described in a formal language (nowadays this is usually PDDL [1]) and all given as input to a general-purpose Planning algorithm.

Integrating an AI planner into a robot is key to making it "intelligent": it enables the robot reason about how to achieve its goals on its own, instead of having to use pre-programmed solutions. Fig. 1 shows part of a Planning domain for a household robot. The formal language used is a slight variant of PDDL called MAPL that was developed for modelling multiagent environments and interactions between agents [2]. The robot is intended to deal with commands like Ex. 1a. The domain defines an ontology of concepts (the *types* hierarchy), a number of *state variables* for describing states and goals

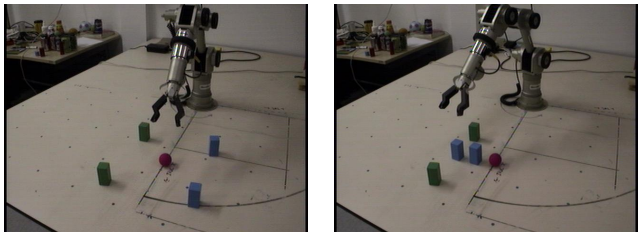
Algorithm 1 Continual planning agent

```
function CONTINUAL-PLANNING-AGENT(S, G)
  while S does not satisfy G do                          (1)
    if res(S,P) does not satisfy G                       (2)
      REMOVEOBSOLETESUFFIXFROM(P)
      P' = PLANNER(A, res(S,P), G)
      P = CONCAT(P, P')
    if P = ∅ then
      return "cannot achieve goal G"
    a = REMOVEFIRSTLEVELACTION(P)
    EXECUTE(a)                                           (3)
    perc = GETSENSORDATA()
    S = UPDATESTATE(S, perc)                             (4)
  return "goal reached"
```

in the domain, and a set of *actions* for altering states. Fig. 2 contains the description of one such state. While the planner enables the robot to reason about achieving its goals intelligently, what of course still must be programmed is the individual behaviours (or *skills*) that the robot is capable of performing. For each planning action there must be a skill (or pre-programmed combination of skills) that is guaranteed to achieve the desired effects. Additionally, a mapping must be defined between the continuous real world (as perceived by the robot's sensors and as manipulated by its actuators) and the qualitative Planning representation [3].

Alg. 1 shows a basic algorithm for including a planner into an autonomous agent. It is a Continual Planning algorithm, i.e. it integrates planning, execution and revision of plans in light of changing situations. Continual Planning is essential for robotic agents (while not being restricted to them), since both the robot's own as well as other agents' actions may turn out unexpectedly. We will explain the algorithm only briefly here: As long as the current situation does not satisfy the robot's goal (step 1) it executes its plan (step 3) and revises it if it believes the plan to no longer achieve the goal (step 2). During execution it constantly updates its beliefs about the current state (step 4). Because of the continual integration new perceptions and the immediate verification of the plan Alg. 1 can be said to be situation aware. This makes it suitable for use in a robotic system. Consider, for example, the robot shown in Fig. 3, which was developed in the EU project CoSy[1] and can be commanded in NL to to manipulate objects. The Continual Planning algorithm allows the robot to adapt its plans even when the situation is externally changed, e.g. if a cube is removed from or added to the scene.

Fig. 3 "Put the blue cubes to the left of the red ball."



(a) The initial situation. (b) Task achieved.

[1] www.cognitivesystems.org

However, in such situations it is not enough that the robot revises its beliefs about the current state, it must also re-interpret the task it was given, i.e. the *goal* it wants to achieve. If, for example, the robot detects a third blue cube in the scene that it was previously not aware of, it must be able to realize that it should move this cube, too. Re-interpreting a goal, however, requires that it is represented in a way enabling it. In essence, this means that referring expressions (REs) used by the human are not resolved once and for all before starting the Continual Planning process. Instead, the planner must be enabled to find situation-dependent interpretations and, consequently, plans for achieving them. To illustrate this with another example, consider Ex. 2a where the robot is told to put dirty plates into the dishwasher. If for the robot the current situation consists of two such plates *plate1* and *plate2* and a dishwasher *dw*, a correct representation of the command would be the simple formula in Ex. 2b.

(2) a. *Please put the dirty plates into the dishwasher.*

b. (and (in plate1 dw) (in plate2 dw))

c.
```
(forall (?v1 - plate)
  (implies (dirtiness ?v1 : dirty)
           (exists (?v2 - dishwasher) (in ?v1 ?v2))))
```

However, Ex. 2b does not allow any re-interpretation, since the RE used in the original command is lost when *binding* it to the two plates known to be dirty in the beginning. Ex. 2c provides a much more flexible goal description that allow the *planner* to resolve the RE on its own. It also does not rely on fixed object names, i.e. IDs, that may change quickly in a robotic system because vision may not be able to guarantee object constancy. Alg. 1 does not need to be changed to deal with more complex RE-preserving planning goals like Ex. 2c. However, it is not obvious how the robot can generate them from an NL command. The remainder of this paper provides an answer to this question.

III. TALKING ABOUT PLANNING DOMAINS

In this section, we describe how concepts defined in a MAPL planning domain can be used for automatically generating a vocabulary and grammar for talking about that domain. The components in PDDL and MAPL domains are fairly close to grammatical concepts: *types* roughly correspond to nouns and *actions* to verbs. *State variables* describe objects or inter-objects relations, i.e. they correspond to adjectives and prepositions. In fact, it is because of this correspondence that components of planning domains are usually intuitively *named* in analogy to their NL counterparts. For example, action names are conventionally chosen as verbs and object types as nouns. In order to generate grammars and lexica automatically from planning domains, we stipulate these conventions formally as follows:

A) Subtypes t of the MAPL type noun are *nouns*.

B) A state variable v referring to two or more *nouns* is a *preposition*.

C) A MAPL object (or constant) o of a subtype of adjective_category is an *adjective*.

Fig. 4 Auto-generated lexicon for the *household* domain (excerpt)

Adjective → dirty | clean | small | big | medium
Noun → plate | cup | dishwasher | table | room | kitchen
PrepositionalExpression → *Pos_PE* | *Left_Of_PE*
Pos_PE → in | from | into | on
Left_Of_PE → to the left of | left of

The types noun and adjective_category are pre-defined in MAPL. Note that since MAPL (like PDDL) supports multiple inheritance, declaring a type as a noun or adjective_category does not restrict the domain definition in any way. In particular, the basic MAPL type phys_obj (used for declaring physically existing objects) is a subtype of noun; so subtypes of phys_obj need not be declared as nouns explicitly. Adjective categories, e.g. *colour* or *size*, group several mutually exclusive adjectives describing an object. For each adjective category cat declared in the type ontology of a MAPL domain there must be a state variable of the same name and whose value domain is cat. This state variable is used to describe the specific properties of an object. See, e.g. the declaration of the state variables colour and colour in Fig. 1.

Based on these conventions, we have implemented a tool that parses a MAPL domain, analyses the definitions found there and translates them to grammar rules as shown in Fig. 4. Since the tool performs a straightforward mapping according to the conventions stated above, we omit a formal algorithmic description. Note, however, that the grammar rules produced are trivially simple (in comparison to grammar formalism used in Computational Linguistics). There are two reasons for this: firstly, the generated grammar is not intended for NL *generation*; therefore it can be allowed to be over-general, but simple. Secondly, it is domain-specific and thus can often *enumerate* constructs rather than generally model them in the grammar.

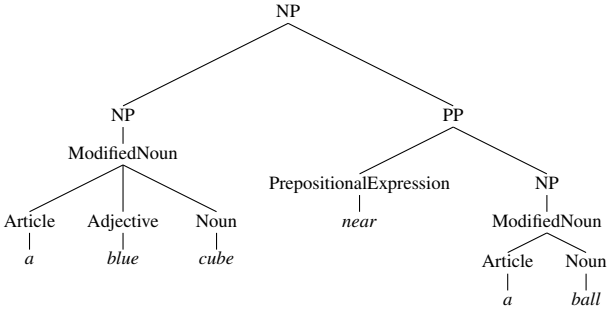
Our tool also allows to annotate the planning domain with *synonyms* for particular expressions. These annotations are given as MAPL *comments* so that they are ignored by planners. For the *household* domain, *from*, *into*, *on* and *in* have been defined as synonyms for pos. A similar concept exists for multi-word expressions: for example, "to the left of" will be treated as a synonym for the preposition "left_of" that corresponds to the state variable left_of. Again, since the grammar is not used to NL generation, it does not matter that the preposition "left_of" does not exist.

Our treatment of adjectives merits some brief explanation. While individual properties of objects could be modeled by boolean state variables or *propositions* (e.g. (blue cup), (red cup), etc.) instead of state variables corresponding to *adjective categories* (e.g. (colour cup : blue)), the latter model enables interpretation of *wh-questions* like "What colour is the cup?" or "Where is the dishwasher?". Such questions can be modeled as a goal to know the value of the corresponding state variable. Our implementation does not yet support questions, so we will postpone discussions of this issue to future work.

***Fig. 5** The base grammar (excerpt)*

S → Command | Statement | Question | S Conjunction S
Command → VP
Statement → NP VP
NP → Pronoun | Modified_Noun | NP RelClause | NP PP | NP Conjunction NP
Modified_Noun → Noun | Article Noun | Adjective Noun | Article Adjectives Noun
Noun → Noun_Singular | Noun_Plural
PP → PrepositionalExpression NP
RelClause → RelPronoun VP

Fig. 6 Parse tree for "a blue cube near a ball".

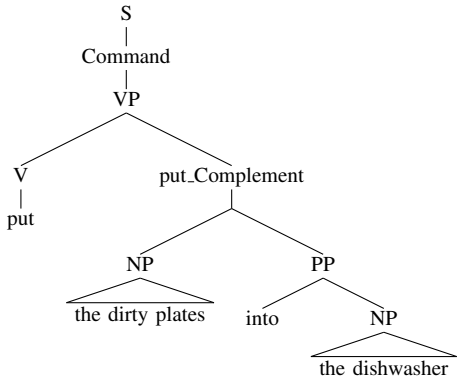


The generated grammar rules are complemented with rules from a simple base grammar, parts of which are shown in Fig. 5. Using a simple chart parser [4] our system is now able to parse NPs describing objects from a given planning domain. For example, in the object manipulation scenario from Fig. 3, this may result in the parse tree shown in Fig. 6. The only part of the grammar that must be explicitly specified for a given domain is the individual VP structure for a given verb. For example, *open* must be followed by an NP (what), but *put* by and NP and PP (what and where). While we could easily describe the most common VP structures in the base grammar, we enforce a domain-dependent definition for two reasons: firstly, verb-specific VP structures largely reduce ambiguity of parsing; secondly, we later need the individual VP complements anyway, in order to match them with parameters of the corresponding planning operator. So, similarly to synonyms, we use annotations actions in the planning domain to define the corresponding grammar rules. For example the action *put* from 1 is annotated "Put_Complement = NP PP". Based on this annotation, a couple of new grammar rules are automatically generated that, e.g., allow the parser to generate the parse tree shown in Fig. 7. In the next section, we describe an algorithm for translating such parse trees into planning goals.

IV. MAPPING NL COMMANDS TO PLANNING GOALS

Tab. I contrasts the way humans express what they intend another agent to do with how AI Planning systems are presented with the goal they must achieve: Firstly, while humans talk about *actions*, planning goals are *formulae* constraining acceptable states. Secondly, objects in the real world usually do not have a unique, commonly known name

Fig. 7 Parse tree for "Put the dirty plates into the dishwasher."



or ID, so humans use REs to (hopefully) refer to a distinctive object unambiguously. We will discuss both differences in this section. Based on our insights, we will then present an algorithm to translate NL commands to goal formulae.

	Human Dialogue	**Planning**
Goal description	Imperative commands *Ex. "Wash ..."*	logical formula *Ex.* $\forall x.(clean\ x)$
Object references	referring expressions *Ex. "the dirty plate"*	unique object names *Ex.* $plate17$

TABLE I
EXPRESSION OF INTENTIONS: HUMAN DIALOGUE VS. AI PLANNING

When a human issues a command, this usually does not mean that another agent can directly execute it. Ex. 1a, for example, involves first picking up plates and moving to the dishwasher before the only action mentioned in the command "put" can be carried out. Moreover, by referring to several objects, the command implicitly also refers to several actions that the human wants to be performed. Thus, one command implicitly corresponds to a whole plan. This plan, however, is not specified, since the robot must devise it on its own, depending on the situation, its knowledge and capabilities. In other words, all the human can do is to describe a *goal state* – yet in the much more compact form of a command: the name of an action substitutes the more complex enumeration of its effects. Alg. 2 is based on this insight: the desired goal state will satisfy the *effects* of the action the robot is commanded to perform.

The second difference between NL commands and planning goals is the use of REs in NL. Since in a Planning problem (e.g. Fig. 2) all objects are uniquely named, REs are usually not needed when specifying a goal state. However, this need arises in Human-Robot Collaboration where the unique object names are usually artificial IDs generated by the robot's vision system that a human can not refer to directly. Thus, the robot must be enabled to resolve REs itself. However, binding the RE to concrete objects is not always possible at the time the robot is given the command, but sometimes only during plan execution. Thus, to enable the robot to find a plan in the first place, the semantics of REs must be *preserved* in the planning goal to allow for

a situation-aware re-interpretation during execution. How to capture the semantics of REs is an important field of study in (Computational) Linguistics, Philosophy and AI; we take a rather simple approach here and model them using first-order predicate logic, respectively its Planning counterpart, the ADL formalism [5]. Luckily, most modern planners support ADL, in particular its ability to express *quantified* goal formulae. Quantified formulae allow to keep bindings with actual objects unresolved until, for a given situation, a plan is constructed of verified. We call this *lazy reference resolution*. This process is situation aware in the sense that it can lead to different reference bindings in different situations which may prompt changes of plans or clarification questions.

Algorithm 2 Translating parse trees to MAPL goal formulae

```
function TRANSLATEPT(pt)
  o = FINDMATCHINGOPERATOR(pt, D)                                  (1)
  e = effect(o)                                                    (2)
  formula = e
  forall free variables v in e do
    match v to NP np in pt                                         (3)
    find type and constraints in np                                (4)
    if ARTICLE(np) = definite ∧ NUMBER(np) = plural then           (5)
      formula = ∀v − type.constraints → formula                    (6)
    else
      formula = ∃v − type.constraints ∧ formula                    (7)
  return formula
```

Alg. 2 provides a high-level description of our translation algorithm. Called with a possible parse tree for a command, it first matches that parse tree to an operator (step 1). FINDMATCHINGOPERATOR() implements a pattern matching algorithm that looks for specific syntax patterns in the parse tree and, if one is found, maps it to an operator from the planning domain. These patterns can refer to arbitrary levels of the parse tree and may even use concrete lexicon entries. For example, in the *household* domain as presented in Fig. 1, the syntax pattern *"put NP PP"* will always map to the put operator. If there were more specific operators, like put-on, put-to-the-left-of as needed, e.g. in the CoSy object manipulation scenario, more distinctive patterns could be used for this mapping, e.g. *"put NP on NP"*. When a matching operator was found, step 2 of Alg. 2 extracts its effect and uses it as the goal formula, e.g. (pos ?obj : ?p) in the case of put. Since this formula contains free variables, it must be embedded in a quantified formula which constrains the values for the variables – in other words: a formula containing referring expressions. The remainder of the algorithm is thus concerned with finding and translating the REs corresponding to operator parameters. In step 3 an NP corresponding to the parameter is detected. Note how the planning operator can provide important guidance for this, e.g. by constraining the parameter to a particular type or by demanding preconditions that might appear in the NP, too.

Once the NP is detected it must be translated into a quantified formula (step 4). The details of this process are omitted from the algorithm, but we will explain them briefly here. There are basically three kinds of constraints that can be derived from an NP: adjective constraints, type constraints and constraints from subordinate phrases, i.e. relative clauses and prepositional clauses. Adjectives can be mapped back straightforwardly to the state variable they are a value of, e.g. if the NP characterising parameter ?obj is modified by *dirty*, this maps to the constraint (dirtiness ?obj : dirty). Nouns are directly mapped to types. Generating constraints from sub-phrases involves *recursively* translating the NPs in the sub-phrase into a sub-formula that is linked to the outer formula by a constraint depending on verb of the relative clause or the preposition of the PP. For example, the PP "on the table" applied to the parameter ?obj would give the constraint (exists (?v2 - table) (on ?obj ?v2)). The last important pieces of information that can be extracted from an NP is the *definiteness* of the article and *number* of the noun used. They determine the kind of quantification to be used in the final formula describing the NP. Step 5 of Alg. 2 shows the simple distinction made in our current implementation: only nouns used with definite articles and in plural form, like "the cubes" will be universally quantified. See Sec. V for a discussion of our use of quantifiers.

While the REs are translated to quantified formulae, the global formula is iteratively extended (steps 6 and 7). There is no general rule to determine the order in which this must be done [4, p. 815], so we currently assume that NPs appearing earliest in the command, will appear in the outermost scope of the formula.

The translation algorithm, as described so far, does not account for an important distinction: REs usually describe the present whereas the result of executing a command describes the future. This can lead to ambiguous or even contradictory goal formulae: For example, Ex. Ex. 3a describes the plate both as being on the table and in the dishwasher – the former describing the current situation, the latter the future. If this distinction is not made explicit, the generated formula Ex. 3b is self-contradictory. Therefore MAPL MAPL supports referring back to the initial state in the goal description with the initially keyword, as shown in Ex. 3c. All sub-formulae modelling REs will be in the scope of initially, whereas the operator effect describing the future goal state will be left unmodified.

(3) a. *Put the plate on the table into the dishwasher.*

```
b. (exists (?v1 - plate)
     (and (exists (?v2 - table) (pos ?v1 ?v2))
          (exists (?v3 - dishwasher) (pos ?v1 ?v3))))

c. (exists (?v1 - plate)
     (and (initially (exists (?v2 - table) (pos ?v1 ?v2)))
          (exists (?v3 - dishwasher) (pos ?v1 ?v3))))
```

Usually, NL commands are syntactically ambiguous. The semantic interpretation provided by Alg. 2 for each possible parse can support the situation-aware *disambiguation* of the command [6]. For example, if no plan at all can be found for a particular goal formula this can be interpreted as a possible misunderstanding of the command in the first place. This is of particular importance on a real robot that is communicated with by *speech*. There the commands given to our system are often ungrammatical and word detection is unreliable.

But the planner can also be used to resolve semantic ambiguities in a situation-aware manner. Consider Ex. 4a

which is syntactically ambiguous. However, it is also semantically ambiguous: Alg. 2 produces too logically different goal formulae for the two parse trees (Ex. 4b and Ex. 4c).

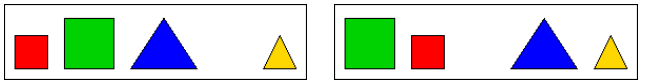
(4) a. *Put the big block to the left of the big pyramid to the left of the small pyramid.*

b. (exists (?v1 - cube) (and (initially (and (size ?v1 : big)
(exists (?v2 - pyramid) (and (size ?v2 : big) (left$_o$f ?v1 ?v2)))))
(exists (?v3 - pyramid) (and (size ?v3 : small) (left$_o$f ?v1 ?v3)))))

c. (exists (?v1 - cube) (and (initially (size ?v1 : big))
(exists (?v2 - pyramid) (and (size ?v2 : big) (exists (?v3 - pyramid)
(and (size ?v3 : small) (left$_o$f ?v2 ?v3))) (left$_o$f ?v1 ?v2)))))

Interestingly, both formulae are correct interpretations of the command – yet for different situations. Fig. 8 shows two such situations. In each of them, only one of the goal formulae makes sense (formula 4b in Fig. 8(a) and formula 4c in Fig. 8(b)). When both are given to the planner as goals, the "correct" one for the situation can be determined, since the planner is not able the resolve the RE for the other one, let alone find a plan for achieving the goal.[2]

Fig. 8 "Put the big block to the left of the big pyramid to the left of the small pyramid."



(a) Situation 1 (b) Situation 2

V. CONCLUSION AND FUTURE WORK

In this paper, we have described new methods enabling robots to understand a natural language command, autonomously plan how to achieve it, and execute the plan – all of these in a situation aware manner that allows to re-interpret the task in light of changing circumstances. The paper has contributed a method to analyse a formal planning domain in such a way that a robot can understand statements in natural language about this domain. Our approach is orthogonal to many existing HRI systems, since it is deliberately kept quite simplistic in terms of Computational Linguistics. However, as a result, it provides a number of important advantages:

- It is completely *domain independent* in the sense that for any existing Planning domain little or no additional information is necessary to make an artificial agent understand commands describing tasks in that domain.
- On the other hand, by virtue of the generated grammars and semantic interpretations being *domain-dependent*, the potential ambiguity of interpretations is reduced.
- In addition, since the generated interpretations have meaning for a powerful causal-temporal reasoning system, i.e. the planner, this system can be used to possibly further disambiguate possible interpretations semantically and pragmatically.

There are many avenues for future work. An important one is the further relaxation of the grammar rules used, i.e. to allow grammars to accept even more phrases as syntactically correct. While this may sound unintuitive, it is necessary for understanding *spoken* input which is often ungrammatical. It is our hope that ambiguities arising from (mildly) ungrammatical sentences can be resolved on the semantic-pragmatic level provided by the planner.

The formulae created by the current system are overgeneral in the sense that singular definite references ("the ball") are quantified with $\exists$ instead of the more common $\exists!$. We believe that definite references do not directly map to a logical restriction like "exactly one". Instead, we believe it to indicate that the speaker (believes she) has given additional information that permits unambiguous reference resolution. We aim to extend the system such that it can make use of such additional information (deixis, discourse context, etc.) if it is indeed available. Such an extension could consist in a artifical restriction of the domain of certain types, i.e. its possible instances.

If the robot does not have the necessary information for resolving references, we want to enable it to formulate appropriate *clarification* questions. To that end, the system must be able to generate distinguishing REs. Since the plan state provides all necessary information, integration of an algorithm for generating referring expressions (GRE), like e.g. [7], is straightforward and has already been partly accomplished. Integration of a GRE algorithm will also be the first step to enable the robot with more sophisticated *dialogue* capabilities. For example, the robot may verbalise its plan in order to give the human user feedback about the robot's understanding of the command.

[2]While in this example an external module for reference resolution would also be able to determine the correct interpretation, we can also imagine a situation where the semantic ambiguity is not resolvable in the *current* situation, but applies to the *goal* state, i.e. for one interpretation there is no way to achieve the goal. This kind of ambiguity is *only* resolvable by causal-temporal reasoning as provided by a planner.

ACKNOWLEDGMENTS

This work was supported by the EU FP6 IST Cognitive Systems Integrated Project "CoSy" FP6-004250-IP.

REFERENCES

[1] D. McDermott, "PDDL – the planning domain definition language," Yale Center for Computational Vision and Control, Tech. Rep. TR-98-003, 1998.

[2] M. Brenner, "Planning for multiagent environments: From individual perceptions to coordinated execution," in *Proc. Wsh. Multiagent Planning and Scheduling, ICAPS '05*, Monterey, USA, 2005.

[3] M. Brenner, N. Hawes, J. Kelleher, and J. Wyatt, "Mediating between qualitative and quantitative representations for task-orientated human-robot interaction," in *Proc. IJCAI-07*, 2007.

[4] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 2nd ed. Englewood Cliffs, NJ: Prentice-Hall, 2003.

[5] E. P. D. Pednault, "ADL: Exploring the middle ground between STRIPS and the situation calculus," in *Proc. KR'89*, 1989.

[6] G.-J. Kruijff and M. Brenner, "Modelling spatio-temporal comprehension in situated human-robot dialogue as reasoning about intentions and plans," in *AAAI Spring Symposium on Intentions in Intelligent Systems, Stanford, CA*, 2007.

[7] E. Krahmer, S. van Erk, and A. Verleg, "Graph-based generation of referring expressions," *Computational Linguistics*, vol. 29, no. 1, 2003.