Plan Failure Analysis: Formalization and Application in Interactive Planning Through Natural Language Communication

Chitta Baral[1], Tran Cao Son[2(✉)], Michael Gelfond[3], and Arindam Mitra[1]

[1] Department of Computer Science and Engineering, Arizona State University, Tempe, AZ, USA
[2] Department of Computer Science, New Mexico State University, Las Cruces, NM, USA
tson@cs.nmsu.edu
[3] Department of Computer Science, Texas Tech University, Lubbock, TX, USA

Abstract. While most robots in human robot interaction scenarios take instructions from humans, the ideal would be that humans and robots collaborate with each other. The Defense Advanced Research Projects Agency Communicating with Computer program proposes the collaborative blocks world scenario as a testbed for this. This scenario requires the human and the computer to communicate through natural language to build structures out of toy blocks. To formulate and address this, we identify two main tasks. The first task, called the *plan failure analysis*, demands the robot to analyze the feasibility of a task and to determine the reasons(s) in case the task is not doable. The second task focuses on the ability of the robot to *understand communications via natural language*. We discuss potential solutions to both problems and present prototypical architecture for the integration of planning failure analysis and natural language communication into an intelligent agent architecture.

Keywords: Human-robot interaction (HRI) · Planning · Plan failure analysis · Natural language communication

1 Introduction

Human-robot interaction is an important field where humans and robots collaborate to achieve tasks. Such interaction is needed, for example, in search and rescue scenarios where the human may direct the robot to do certain tasks and at times the robot may have to make its own plan. Although there has been many works on this topic, there has not been much research on interactive planning where the human and the robot collaborate in making plans. For such interactive planning, the human may communicate to the robot about some goals and the robot may make a plan to achieve them. If planning fails the robot may explain the failure, and the human may use the explanation to make suggestions for

© Springer International Publishing Switzerland 2016
M. Baldoni et al. (Eds.): PRIMA 2016, LNAI 9862, pp. 376–386, 2016.
DOI: 10.1007/978-3-319-44832-9_25

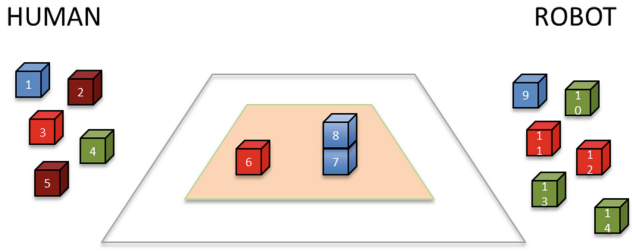

Fig. 1. A blocks world example

overcoming the problem. This interaction may continue until a plan is made and executed. Let us consider an example of such an interactive planning.

Consider the block world domain in Fig. 1 [9]. The robot has its own blocks and the human has some blocks. The two share a table and some other blocks on the table. Suppose that the human communicates to the robot the sentence "*Add another blue stack of the same height!*."

Even if we assume that the robot is able to recognize the color of the blocks, create, and execute plans for constructions of stacks of blocks, such a communication presents several challenges to a robot. Specifically, it requires that the robot is capable of understanding natural language, i.e., it requires that the robot is able to identify that

- the human refers to stacks of only blue blocks;
- the human refers to the height of a stack as the number of blocks on the stack;
- there is a blue stack of the height 2 on the table; and
- it should use its two blue blocks to build a new stack of two blue blocks.

In addition to the above, it is easy to see that the robot cannot accomplish the task in the situation in Fig. 1 if it is limited to the typical actions in the block domain (e.g., pick up, put down, or un/stack blocks) and uses only its own blocks. It is because of its planning process—looking for a plan to create a stack of two blue blocks—fails. *What should the robot do? Can it reply back that it needs an additional blue block? Can it ask for the blue block of the human?* In this paper, we address this task of plan failure analysis.

Previous approaches to dealing with the planning failure problem have been proposed. *Partial satisfaction planning* (PSP) identifies a maximal subset of the goal (or a sub-formula; or a collection of subgoals with a maximal aggregated ultility) that can be satisfied (e.g., [4]). This approach does not take into consideration the fact that some actions/fluents are not considered in the planning process and/or the presence of a human user interacting with the planner. Applications of planning system in the literature such as mixed-initiative (e.g., [1,2,14]) emphasize the need for interaction between a human user and a planning system. However, the human user plays the primary role in the generation of a working plan. *Assumption-based planning* [10] assumes that the planning agent does not have complete information about the world and thus

focuses on asserting additional facts so that the resulting planning problem has a solution. Similar to PSP, ABP does not consider the presence of a human user interacting with the planner. *Diagnostic planning* and/or *replanning* (e.g., [5,11,12]) also needs to deal with planning failure. The focus of approaches in this line of research is to address discrepancies between what is observed and what is hypothetically true after the execution of a sequence of actions.

In this paper, we propose an orthogonal approach to deal with planning failure within the context of HRI applications. In this type of applications, the robot understands a basic set of vocabularies for communication with the human user and is actively engaged in dealing with the failure of the planning process. It will identify possible reasons for the failure and possible course of actions for recovering from the failure. To achieve this goal, we formalize the *planning failure analysis problem* and introduce a notion called a *failure analysis* of a planning problem with respect to an extension (Sect. 2). We propose a potential solution for the natural language understanding problem and describe a system that translates communications from a human user into an answer set programming representation (Sect. 3). We discuss how the proposed components can be integrated into an intelligent agent architecture that allows agents to interactively planning with humans through natural language communication (Sect. 4).

2 Planning Failure Analysis: Formalization

A planning problem is specified by a tuple $\mathcal{P} = \langle F, A, I, G\rangle$ where F is a set of fluents (time-dependent Boolean variables), A is a set of actions with their preconditions and (conditional) effects, I describes the initial state, and G describes the goal state. $\langle F, A\rangle$ is often referred to as the domain of the planning problem.

A fluent literal is a fluent f or its negation $\neg f$. A set of fluent literals S is consistent if it does not contain f and its negation $\neg f$. S is complete if for every $f \in F$, either $f \in S$ or $\neg f \in S$. Each *action* $a \in A$ is associated with a consistent set of literals, $pre(a)$, called the precondition of a; and a set $e(a)$ of conditional effects of the form $\varphi \rightarrow \psi$ where φ and ψ are consistent sets of literals. Intuitively, $\varphi \rightarrow \psi$ says that when a is executed in a state satisfying φ then ψ is true in the resulting state.

A *state* is an interpretation of F, i.e., a complete and consistent set of fluent literals. Truth value of a fluent formula in a state is evaluated in the standard way. If φ is true in a state s, we write $s \models \varphi$. A set of fluent literals is viewed as conjunction of literals belonging to it. For an action a and a state s, let $es(a, s) = \bigcup_{\varphi\rightarrow\psi\in e(a), s\models\varphi} \psi$. The semantics of $\mathcal{P}$ is defined by a transition function Φ that maps each pair of a state s and an action a into a state denoted by $\Phi(a, s)$. Formally, $\Phi(a, s)$ is defined as follows: $\Phi(a, s) = s \setminus \overline{es(a, s)} \cup es(a, s)$ if $s \models pre(a)$where $\overline{es(a, s)} = \{\bar{l} \mid l \in es(a, s)\}$ and for every $f \in F$, $\bar{f} = \neg f$ and $\overline{\neg f} = f$; otherwise, $\Phi(a, s)$ is undefined. This function is extended to define $\Phi^*([a_1, \ldots, a_n], s)$ as (i) $\Phi^*([], s) = s$; and (ii) $\Phi^*([a_1, \ldots, a_n], s) = \Phi(a_n, \Phi^*([a_1, \ldots, a_{n-1}], s))$.

A state is called the initial state of $\mathcal{P}$ if it satisfies I. A plan of length n for G is a sequence of actions $[a_1, \ldots, a_n]$ (or a solution of $\mathcal{P}$) if for every state s_0 satisfying I, $\Phi^*([a_1, \ldots, a_n], s_0)$ is defined and satisfies G.

Example 1. The block domain in Fig. 1—in the view of the robot—can be represented by the planning domain $\langle F_b, A_b \rangle$ with a set of constants[1] denoting the human (h), the robot (r), the set of blocks $Blks = \{b_1, \ldots, b_{14}\}$, $F_b = \{on(B_1, B_2), has(X, B), ontable(B), holding(X, B), clear(B), handempty, color(B, C)\}$ and $A_b = \{pick(B), putdown(B), stack(B_1, B_2), unstack(B_1, B_2), takes(B)\}$ where $X \in \{h, r\}$, C is one of the colors, and $B, B_1, B_2 \in Blks$ are blocks with $B_1 \neq B_2$. Preconditions and effects of actions are defined as usually, e.g., for the action of picking up a block from the table or taking its own block, they are:

- $pre(pick(B))$=$\{handempty, ontable(B), clear(B)\}$
- $e(pick(B)) = \{\emptyset \rightarrow \{holding(B), \neg ontable(B), \neg handempty, \neg clear(B)\}\}$
- etc.

We omit the description of other actions for brevity. The planning problem discussed in the first section can be represented by $\mathcal{P}_b = \langle F_b, A_b, I_b, G_b \rangle$ with I_b encodes the configuration in Fig. 1 and G_b be the following formula:

$$\bigvee_{x \neq y \in Blks, \{x,y\} \cap \{b_7, b_8\} = \emptyset} \begin{bmatrix} color(x, blue) \wedge color(y, blue) \wedge \\ ontable(x) \wedge on(y, x) \wedge clear(y) \end{bmatrix}$$

We will now formally define the planning failure analysis problem. The above discussion leads us to the definition.

Definition 1. *Let $\mathcal{P} = \langle F, A, I, G \rangle$ be a planning problem. We say that $\mathcal{P}$ needs a* failure analysis *if it has no solution.*

We can easily check that $\mathcal{P}_b$ does not have a solution and thus needs a failure analysis. Realizing that a planning problem needs a failure analysis is identical with checking whether it has a solution or not. As such, the complexity of the problem of identifying whether or not a planning problem needs a failure analysis is the same as that of planning. It is known that the complexity of planning is undecidable in the general case [13]. Under certain assumptions (e.g., finite and deterministic domains), it reduces to PSPACE-complete [6]. For planning problems considered in this paper, if we limit the length of plans then the complexity reduces to NP-complete. We will next focus on answering the question of why the planning process fails and what can one do when the planning problem has no solution. *We start with the assumption that changes can be caused only by actions.* Thus, one way to rectify this problem is to provide additional actions that could be used in creating a plan.

[1] Fluents/actions with variables are shorthands for collections of their ground instantiations. The formalization used in this example is a variant of the block world domain representation in planning benchmarks and assumes that each block has a unique color. The goal is simplified to be "build a stack of two blue blocks."

Definition 2. *An* extension to a planning problem $\mathcal{P} = \langle F, A, I, G\rangle$ *is a pair* $(Afs, Acts)$ *where* Afs *is a set of fluents and* $Acts$ *is a set of actions such that* $F \cap Afs = \emptyset$ *and* $A \cap Acts = \emptyset$.

Intuitively, $Acts$ is the set of actions that the planning agent (robot) could execute and Afs could be the set of fluents, which occur in the definition of $Acts$ and do not belong to F. Observe that $Acts$ could change the fluents in F and their definitions could introduce new fluents. It can contain actions that the robot cannot use without permission and/or execute independently. Note that this is different from *planning with preferences*, which considers that the actions are available at the robot's disposal but the robot prefers not to use them.

In our running example, $Acts$ could be the set with a single action $ask\_permission$ whose precondition is empty (true) and whose effect is that the robot can use the human's blocks in solving the problem[2], i.e., $e(ask\_permission) = \{\{has(h, B)\} \rightarrow \{has(r, B)\} \mid B \in Blks\}$; $Afs = \emptyset$ since every fluent occurring in the definition of $ask\_permission$ also belongs to F. For later reference, we will denote with $\mathcal{E}_1$ the extension $(\emptyset, \{ask\_permission\})$ to $\mathcal{P}_b$. Notice that this extension can be refined by the extension $\mathcal{E}_2 = (\emptyset, \{ask\_permission(B) \mid B \in Blks\})$ where, for each $b \in Blks$, $ask\_permission(b)$ has the precondition $\{has(h, b)\}$ and an effect $\emptyset \rightarrow \{has(r, b)\}$.

Given an extension $\mathcal{E} = (Afs, Acts)$ to a planning problem $\mathcal{P} = \langle F, A, I, G\rangle$, what could be a failure analysis for $\mathcal{P}$ w.r.t. $\mathcal{E}$? The above discussion suggests that it could be a pair (Af, Ac) with $Af \subseteq Afs$ and $Ac \subseteq Acts$ such that $\mathcal{P}^* = \langle F \cup Af, A \cup Ac, I \cup Af^*, G\rangle$ has a solution where Af^* is an interpretation of the set of fluents Af. By this definition, $\mathcal{E}_1$ is an analysis for $\mathcal{P}_b$ w.r.t. $\mathcal{E}_1$ since $\mathcal{P}^* = \langle F, A \cup \{ask\_permission\}, I, G\rangle$ has a solution. Similarly, we can check that for every $X \subseteq \{ask\_permission(B) \mid B \in Blks\}$ such that $ask\_permission(b_1) \in X$, $(\emptyset, X)$ is an analysis for $\mathcal{P}_b$ w.r.t. $\mathcal{E}_2$.

The above definition appears reasonable for $\mathcal{E}_1$ and $\mathcal{E}_2$. However, it allows the possibility of *wishful analysis* as in the next example. Let us consider the extension $\mathcal{E}_3 = (\{color(b_{10}, blue)\}, \emptyset)$, i.e., the robot wishes that the color of another block in its possession is blue. If we were to use the above definition then $\mathcal{E}_3$ is also an analysis for $\mathcal{P}_b$ w.r.t. $\mathcal{E}_3$. Clearly, this analysis is not practical for $\mathcal{P}_b$ since the robot cannot change the color of a block. On the other hand, $\mathcal{E}_4 = (\{color(b_{10}, blue), paint\_available\}, \{paint\_blue(b_{10}), buy\_paint\})$ where buy_paint causes $paint\_available$ to be true and $paint\_blue(b_{10})$ causes the block 10 to be blue is an extension of $\mathcal{P}_b$ and is itself an analysis for $\mathcal{P}_b$.

Definition 3. *Let* $\mathcal{P} = \langle F, A, I, G\rangle$ *be a planning problem and* $\alpha = [a_1, \ldots, a_n]$ *be one of its solution. We say that a fluent* f changes its value during the execution of α, *denoted by* $\pm f \xrightarrow{\alpha} \mp f$, *if there exist some* $1 \leq j < n$ *such that* f *is true (resp. false)* $\Phi^*([a_1, \ldots, a_{j-1}], s_0)$ *and false (resp. true) in* $\Phi^*([a_1, \ldots, a_j], s_0)$.

[2] Observe that the action $ask\_permission(.)$ refers to a communication between the robot and the human user and thus is not included in the initial planning domain of $\mathcal{P}_b$. Furthermore, we assume that the human is collaborative and thus would grant the robot the permission to use his blocks.

If we require that the fluent that are added to the domain change their values in the plans that could be generated, then $\mathcal{E}_3$ cannot be used to provide an analysis for $\mathcal{P}_b$. Observe that there are situations in which wishful analyses might be useful. For example, when the robot does not have complete information about the world (e.g., its sensors are imperfect and the robot know that the information about the initial state can be incorrect; or the robot executes a plan and observes something unexpected). This issue has been investigated in [10] or in the literature on diagnostic planning and/or replanning. As discussed, the focus of the present work is different.

Definition 4. *Let $\mathcal{E} = (Afs, Acts)$ be an extension to the planning problem $\mathcal{P} = \langle F, A, I, G\rangle$. A pair (Af, Ac) such that $Af \subseteq Afs$ and $Ac \subseteq Acts$ is called a* plan failure analysis *for $\mathcal{P}$ w.r.t. $\mathcal{E}$ (or an* analysis *for $\mathcal{P}$, for short) if there exists an interpretation Af^* of the set of fluents Af and $\mathcal{P}^* = \langle F \cup Af, A \cup Ac, I \cup Af^*, G\rangle$ has a solution α such that $\pm f \xrightarrow{\alpha} \mp f$ for every $f \in Af$.*

When $Af = Ac = \emptyset$, we say that it is a no-fault analysis*; otherwise, it is a* non-trivial *analysis.*

By the above definition, we can see that $\mathcal{E}_1$ is an analysis for $\mathcal{P}_b$ w.r.t. $\mathcal{E}_1$ but $\mathcal{E}_3$ is not an analysis for $\mathcal{P}_b$ w.r.t. $\mathcal{E}_3$. Definition 4 characterizes analyses as those that introduce new fluents and actions into the planning problem so that they will be useful in creating a solution for the new planning problem. In general, we prefer analyses that change the planning problem in a minimal way. For example, $(\emptyset, \{ask\_permission(b_1)\})$ could be viewed as better than $(\emptyset, \{ask\_permission(b_1), ask\_permission(b_2)\})$ w.r.t. $\mathcal{E}_2$ as the former asks for less from the human.

Definition 5. *Let $\mathcal{E} = (Afs, Acts)$ be an extension to the planning problem $\mathcal{P} = \langle F, A, I, G\rangle$. Let (Af, Ac) and (Af', Ac') be two plan failure analyses for $\mathcal{P}$ w.r.t. $\mathcal{E}$. We say that (Af, Ac) is a* more preferred failure analysis *than (Af', Ac'), denoted $(Af, Ac) \prec (Af', Ac')$, if $Ac \subsetneq Ac'$.*

An analysis (Af, Ac) is said to be a most preferred analysis *for $\mathcal{P}$ w.r.t. $\mathcal{E}$ if there exists no other analysis that is more preferred than (Af, Ac).*

By Definition 5, it is easy to see that $(\emptyset, \{ask\_permission(b_1)\})$ is the most preferred failure analysis for $\mathcal{P}_b$ w.r.t $\mathcal{E}_2$. We prove that the relation $\prec$ is a partial order over the set of analyses for a planning problem.

Proposition 1. *For a planning problem $\mathcal{P} = \langle F, A, I, G\rangle$ and an extension $\mathcal{E} = (Afs, Acts)$ to $\mathcal{P}$, the following holds: (*i*) $\prec$ defines a partial ordering over the set of failure analyses for $\mathcal{P}$ w.r.t. $\mathcal{E}$; and (*ii*) if $(\emptyset, \emptyset)$ is an analysis for $\mathcal{P}$ then it is the unique most preferred analysis for $\mathcal{P}$ w.r.t. $\mathcal{E}$.*

It is reasonable to assume that given a planning problem $\mathcal{P} = \langle F, A, I, G\rangle$ there exists an extension $\mathcal{E}$ to $\mathcal{P}$ that covers all possible failure analyses for $\mathcal{P}$ should it need a failure analysis. For example, the robot in our running example should be able to assume that it can ask the human for permission to use the human's blocks in responding to the command (as in $\mathcal{E}_1$ or $\mathcal{E}_2$); or it has, at its

disposal, actions for changing the color of a block (as in $\mathcal{E}_4$); or all of these (as in $\mathcal{E}_1 \cup \mathcal{E}_2 \cup \mathcal{E}_4$). However, the fact that the robot cannot make unreasonable assumptions such as the color of a block could change by itself (as in $\mathcal{E}_3$) should be taken into consideration. Specifying $\mathcal{E}$ is therefore problem-dependent and is out of the scope of this paper. We note that the proposed notion of plan failure analysis has been proposed by in our earlier work [3]. The formalization proposed in this paper only shares Definition 1 with the earlier one.

3 Natural Languge to Answer Set Programming (ASP)

The previous section deals with the planning failure analysis problem, assuming that the robot understands the commands from the human user. Computing planning failure analyses can be done using *logic programming under answer set semantics* [15] and can be implemented similar to the proposal in [3]. For space reason, we omit the discussion on computing planning failure analyses. In this section, we focus on providing this capability to the robot. Specifically, we describe a translation system that represents the communicative signals of a human in machine understandable terms. It is a relatively well-studied problem in Natural Language Processing (NLP) and popularly known as *semantic parsing* [17]. A semantic parser maps natural language sentences to a formal representation language (such as) to allow automated inference and processing. Recently several powerful systems have been developed to build a semantic parser for varied target representations [7,16,18–20,22,23]. In this research, we have trained the NL2KR system [22] for the task of translation.

In NL2KR, the meaning of words and phrases are expressed as λ-calculus [8] expressions. The meaning of a sentence is built from the semantics of constituent words through appropriate λ-calculus applications. The parse tree of

Table 1. A set of human commands and their representation in an intermediate language. NL2KR has been trained on these sentences to translate new sentences similar to these sentences. A small program is written to convert the intermediate formal representation to the syntax of ASP.

Sentence	Meaning
Add another blue stack of the same height.	λx. *goal_cond(x, op, add)* $\wedge$ *goal_cond(x, is, stack)* $\wedge$ *goal_cond(x, color, blue)* $\wedge$*goal_cond(x, height, same)* $\wedge$*goal_cond(x, type, another).*
Take my blocks.	λx. *add_block(has_robot(x), has(human, x)* $\wedge$ *block(x)).*
How about a red stack of the same height as the blue stack?	λx. *goal_cond(x, is, stack)*$\wedge$ *goal_cond(x, color, red)* $\wedge$ *goal_cond(x, height, same)*$\wedge$ *goal_cond(x, origin, blue)* $\wedge$ *goal_cond(x, type, another).*
Add another stack of the same height as the tallest stack.	λx. *goal_cond(x, op, add)* $\wedge$ *goal_cond(x, is, stack)* $\wedge$ *goal_cond(x, height, tallest)* $\wedge$ *goal_cond(x, type, another).*

the sentence in Combinatory Categorial Grammar [21] (henceforth CCG) directs how the words are combined to produce the meaning of the sentence. During training NL2KR takes as input: (1) a set of training sentences and their target formal representation, and (2) an initial lexicon or dictionary consisting of some words, their CCG categories and meanings in terms of λ-calculus expressions. It then produces a bigger lexicon that is used in translation. For this work, the training set contained the sentences shown in Table 1. For the translation of a new sentence into the syntax of ASP, the λ-calculus expression of the sentence is first obtained. A small program then adds syntactic sugar to the λ-calculus expression to make it a valid ASP statement.

Consider the sentence "*Take my blocks*" from the Table 1. Let us say that the initial dictionary contains two entries as shown in Table 2. The first entry says that the word "take" with the CCG category "S/NP" has the meaning $\lambda p.\lambda x.$ *use(has_robot(x), p @x)*. The second entry provides a meaning of the word "blocks" for the category "NP".

Table 2. A sample *initial* dictionary containing two entries.

Word	CCG	Meaning
Take	S/NP	$\lambda p.\lambda x.$ *add_block(has_robot(x), p @x)*
Blocks	NP	$\lambda x.$ *block(x)*

Given this information, NL2KR then learns the meaning of the unknowns i.e. the phrase "my blocks" and the word "my" in the following way. It first obtains the CCG parse tree (Fig. 2) of the input sentence. A CCG parse tree shows how the words are combined together to characterize the meaning of the sentence. For example, the CCG in Fig. 2 indicates that the determiner (NP/N) "my" takes the noun (N) "blocks" as the input to produce the meaning of the noun phrase (NP) "my blocks".

The verb (S/NP) "take" then scoops the noun phrase (NP) "my blocks" to produce the meaning of the sentence (S) "take my blocks". With the CCG parse tree in hand, NL2KR then uses an operation called "Inverse-λ" to obtain the meaning of the phrase "my blocks" as "$\lambda x.has(human, x) \wedge block(x)$" from the meaning of "take" and "take my blocks". The "Inverse-λ" operator takes two λ-expressions, H (the root node) and G (the left child or the right child) and returns the λ-expression F for the other child such that either $H = G@F$ (when G is the left child) or $H = F@G$ (when G is the right child). Finally, it obtains the meaning of the word "my" $\lambda x8.\lambda x6.has(human, x6) \wedge x8@x6$ by "Inverse-λ" from the meaning of "my blocks" and "blocks" and saves it in the final dictionary for use during the translation of new sentences.

Furthermore, NL2KR uses a operation called "generalization" to handle unknown words. For example, given a new sentence "Use my blocks" NL2KR will generalize the meaning of "use" by the meaning of "take" to translate it

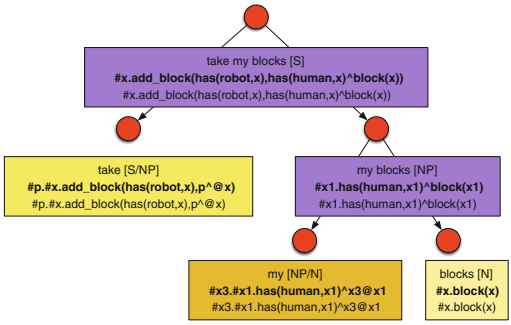

Fig. 2. An augmented CCG parse tree for the sentence "take my blocks" obtained from NL2KR. Each node shows the CCG category, the text description and the λ-expression associated with that node. NL2KR treats '#' as 'λ'.

to "$\lambda x.$ *add_block(has_robot(x), has(human, x)* $\wedge$ *block(x))*". A λ-to-ASP procedure will then convert the λ expression to the desired ASP representation, *add_block(has_robot(S)* $\leftarrow$ *has(human, S), block(S)).*

4 Discussion

The components described in the previous sections are developed in response to the challenge given in [9]. Since a non-trivial analysis represents a reason for the failure of the planning process, an intelligent agent (robot) working interactively with a human user can use failure analyses to explain to him/her why it fails to achieve the goal. In fact, we envision that the proposed components can be integrated into a general architecture for a robot to interactively working with a human user as in Fig. 3.

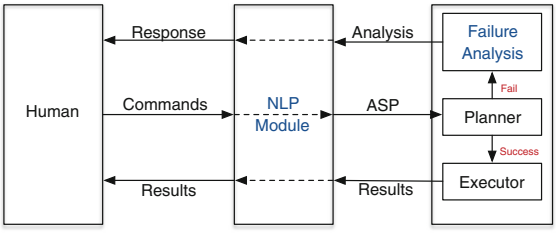

Fig. 3. Schematic integration of planning failure analysis and NLP module into an intelligent agent architecture

We will next discuss additional tasks that are required for an end-to-end implementation of the architecture in Fig. 3. The first task is to compute the plan failure analyses that can be computed using answer set programming.

The second task is related to the planning component of the robot. In general, the planner is supposed to plan for goals of arbitrary formulae over the set of fluents of the planning domain. Yet, it is expected that a communication between the human and the robot would sometimes contain directives that do not belong to the language of the planning domain (e.g., 'same', 'height', 'tallest', 'another', etc.). As demonstrated, our translation system can learn new vocabularies[3] and translates these directives to new statements representing in the language of the planner. We believe that if other state-of-the-art off-the-shelf planning systems (e.g., FF, Fast Downward, etc.) are used, converting the new directives to a formula might be more suitable. How to train our translation system for this task and what is the impact on the performance of the system are two very interesting questions that we leave for future research.

5 Conclusions

We discuss challenges in development of intelligent agents that interact with humans in planning. We introduce the notion of a planning failure analysis for a planning problem given its extension and also show that failure analyses can be computed using ASP and used for generating responses. We describe a translation system that can convert natural language communications to ASP goals. We discuss how the proposed components can be integrated into an overall architecture for developing intelligent agents that interact with human in problem solving and describe additional tasks that need to be completed.

References

1. Ai-Chang, M., Bresina, J., Charest, L., Chase, A., Hsu, J.C.J., Jonsson, A., Kanefsky, B., Morris, P., Rajan, K., Yglesias, J., Chafin, B.G., Dias, W.C., Maldague, P.F.: Mapgen: mixed-initiative planning and scheduling for the Mars Exploration Rover Mission. IEEE Intell. Syst. **19**, 8–12 (2004)
2. Allen, J.F., Ferguson, G.: Human-machine collaborative planning. In: Proceedings of 3rd International Workshop on Planning and Scheduling for Space (2002)
3. Baral, C., Son, T.C.: "Add another blue stack of the same height!": ASP based planning and plan failure analysis. In: Calimeri, F., Ianni, G., Truszczynski, M. (eds.) LPNMR 2015. LNCS, vol. 9345, pp. 127–133. Springer, Heidelberg (2015)
4. Benton, J., Do, M.B., Kambhampati, S.: Anytime heuristic search for partial satisfaction planning. Artif. Intell. **173**(5–6), 562–592 (2009)
5. Brafman, R.I., Shani, G.: Replanning in domains with partial information and sensing actions. J. Artif. Intell. Res. **45**, 565–600 (2012)
6. Bylander, T.: The computational complexity of propositional strips planning. Artif. Intell. **69**(1–2), 165–204 (1994)
7. Chen, D.L., Mooney, R.J.: Learning to interpret natural language navigation instructions from observations (2011)

[3] If it cannot deal with a new word, the system should respond by asking for an alternative or about the meaning of the word.

8. Church, A.: An unsolvable problem of elementary number theory. Am. J. Math. **58**(2), 345–363 (1936)
9. DARPA: Communicating with Computers (CwC) (2015)
10. Davis-Mendelow, S., Baier, J.A., McIlraith, S.A.: Assumption-based planning: generating plans and explanations under incomplete knowledge. In: Proceedings of AAAI (2013)
11. De Giacomo, G., Reiter, R., Soutchanski, M.: Execution monitoring of high-level robot programs. In: Principles of Knowledge Representation and Reasoning, pp. 453–465. Morgan Kaufmann Publishers (1998)
12. Erdem, E., Patoglu, V., Saribatur, Z.G.: Integrating hybrid diagnostic reasoning in plan execution monitoring for cognitive factories with multiple robots. In: ICRA (2015)
13. Erol, K., Nau, D., Subrahmanian, V.: Complexity, decidability and undecidability results for domain-independent planning. Artif. Intell. **76**(1–2), 75–88 (1995)
14. Ferguson, G., Allen, J.F.: Trips: An integrated intelligent problem-solving assistant. In: Proceedings of AAAI, pp. 567–572 (1998)
15. Gelfond, M., Lifschitz, V.: Logic programs with classical negation. In: Proceedings of 7th International Conference on Logic Programming, pp. 579–597 (1990)
16. Ge, R., Mooney, R.J.: Learning a compositional semantic parser using an existing syntactic parser. In: JCAMACL and IJCNLP, pp. 611–619. Association for Computational Linguistics (2009)
17. Kate, R.J., Wong, Y.W., Mooney, R.J.: Learning to transform natural to formal languages. In: Proceedings of AAAI 2005 (2005)
18. Kwiatkowski, T., Zettlemoyer, L., Goldwater, S., Steedman, M.: Inducing probabilistic CCG grammars from logical form with higher-order unification. In: EMNLP, pp. 1223–1233. ACL (2010)
19. Kwiatkowski, T., Zettlemoyer, L., Goldwater, S., Steedman, M.: Lexical generalization in CCG grammar induction for semantic parsing. In: EMNLP, pp. 1512–1523. ACL (2011)
20. Mooney, R.J.: Learning for semantic parsing. In: Gelbukh, A. (ed.) CICLing 2007. LNCS, vol. 4394, pp. 311–324. Springer, Heidelberg (2007)
21. Steedman, M.: The Syntactic Process. MIT Press, Cambridge (2000)
22. Vo, N.H., Mitra, A., Baral, C.R.: The NL2KR platform for building natural language translation systems. In: Association for Computational Linguistics (ACL) (2015)
23. Zettlemoyer, L.S., Collins, M.: Online learning of relaxed CCG grammars for parsing to logical form. In: EMNLP-CoNLL, pp. 678–687 (2007)