Assumption-Based Planning

Damien Pellier, Humbert Fiorino

Laboratoire Leibniz (CNRS - INPG - IMAG), Equipe MAGMA
46, avenue Félix Viallet, F-38031, Grenoble
Tel: +33 4 76 57 47 76 – Fax : +33 4 76 57 50 81
Email: {Damien.Pellier, Humbert.Fiorino}@imag.fr

Abstract— **The purpose of the paper is to introduce a new approach of planning called Assumption-Based Planning. This approach is a very interesting way to devise a planner based on a multi-agent system in which the production of a global shared plan is obtained by conjecture/refutation cycles. Contrary to classical approaches, our contribution relies on the agents reasoning that leads to the production of a plan from planning domains. To take into account complex environments and the partial agents knowledge, we propose to consider the planning problem as a defeasible reasoning where the agents exchange proposals and counter-proposals and are able to reason about uncertainty. The argumentation dialogue between agents must not be viewed as a negotiation process but as an investigation process in order to build a plan. In this paper, we focus on the mechanisms that allow an agent to produce "reasonable" proposals according to its knowledge.**

I. INTRODUCTION

This paper tackles the problem of devising an intelligent agent able to elaborate plans under partial knowledge and/or to produce plans that partially contradict its knowledge. In other words, in order to reach a goal, such an agent is able to provide a plan *which could be executed if certain conditions were met*. Unlike "classical" planners, the planning process does not fail if some conditions are not asserted in the knowledge base, but rather proposes an Assumption-Based Plan or *conjecture*. Obviously, this conjecture must be *reasonable*: the goal cannot be considered "achieved" and the assumptions must be as few as possible because they become new goals for the other agents. For instance, suppose that a door is locked: if the agent seeks to get into the room behind the door and the key is not in the lock, the planning procedure fails even though the agent is able to fulfill 100% of its objectives behind the door. Another possibility is to suppose for the moment that the key is available and then plan to open the door whereas finding the key might become a new goal to be delegated. To that end, we designed a planner that relaxes some restrictions regarding the applicability of planning operators.

The Assumption-Based Planning process is based on the concept of proof validity that can be considered as an exchange of proposals and counter-proposals. According to [1], a correct proof does not exist in the absolute. At any time, an experimentation or a test can refute a proof. If one single test leads to a refutation, the proof is reviewed and it is considered as mere conjecture which must be repaired in

order to reject this refutation and consequently becomes less questionable. The new proof can be subsequently tested and refuted anew. Therefore, the proof elaboration is an iterative non monotonous process of conjectures/refutations/repairs.

The same goes for our approach: each agent can refine, refute or repair the current conjecture. If the reparation of a previously refuted plan succeeds, it becomes more robust but it can still be refuted later. If the reparation of the refuted plan fails, agents leave this part of the reasoning and explore another conjecture: "bad" conjectures are ruled out because there is no agent able to push the process further. Finally, as in an argumentation with opponents and proponents, the current conjecture is considered as an acceptable solution when the proposal/counter-proposal cycles end and all assumptions have been removed.

The conjecture – refutation cycles can be illustrated by the following informal dialog:

Ag_1: "If I had fuel, I could load the passenger *fred* at *downtown*, move the taxi *t* from *downtown* to *park* and unload it at *park*, but I have no fuel": (1) initial refinement of the goal: the lack of fuel does not lead to the planning failure but becomes an assumption to be removed;

Ag_2: "I can provide you fuel": (2) refinement of the conjecture: "has fuel" is no more an assumption.

Ag_1: "Thank you!" (3)

Ag_2: "But you need to pay the taxi to move the passenger *fred* from *downtown* to *park*": (4) refutation of the conjecture;

Ag_2: "Therefore, you can load the passenger *fred* at *donwtown*, pay the taxi *t* and move it from *downtown* to *park*": (5) repairing of the conjecture by adding actions to execute;

Ag_1: "Yes, if I had money... But sorry, I cannot pay": (6) refutation of the conjecture;

Ag_3: "OK, I'll pay the taxi for you": (7) refinement of the conjecture, "has money" is no more an assumption.

This informal example shows how agents iteratively refine (1, 2, 7), refute (4, 6) or repair (5) the current conjecture in order to produce an acceptable plan: Ag_1 loads *fred* in *t* at *donwtown*, Ag_2 refuels *t*, Ag_3 pays the taxi, Ag_1 moves *t* from *downtown* to *park*, and Ag_1 unloads *fred* at *park*. As a matter of fact, Assumption-Based Planning raises many

challenging issues: how to plan with incomplete information? Which reasonable assumptions can be put forward in order to reach a given goal? How to define the conjecture/refutation protocol so as to converge to an acceptable solution?

In this paper, we focus on the Assumption-Based Planning algorithm, i.e. on how one agent elaborates a conjecture. In section II, we briefly summarize the classical planning approach to introduce our Assumption-Based Planning model. In sections III and IV, we describe our own planning algorithm. Then, we discuss (section V) the properties of our approach. The last section is dedicated to related works (section VI).

II. CLASSICAL PLANNING VERSUS ASSUMPTION-BASED PLANNING

A. Classical planning model overview

Classical planning can be defined by a tuple $\langle \mathcal{G}, \mathcal{E}, \mathcal{A} \rangle$: $\mathcal{G}$, is a goal description (i.e., a set of world states), $\mathcal{E}$ is a partial description of the world (i.e., the agent's knowledge) and $\mathcal{A}$ is a description of the actions that an agent can execute. $\mathcal{E}$ and $\mathcal{G}$ are described in propositional logic. For instance, the description of the world state can be written as follows:

$$\{ \, at(cab38, downtown), at(fred, downtown),$$
$$hasfuel(cab38, 10) \, \}$$

The goal of the agent is described by a set of knowledge defining the world state to be reached after a plan execution. In our example, the goal is reduced to a set containing only one proposition:

$$\{ \, at(fred, park) \, \}$$

In general, an action is described by an operator defined by:

- a *name*, with parameters;
- a *precondition list* (i.e, the knowledge that must hold to apply the action);
- a *del list* (i.e., the knowledge that does not hold after the action execution);
- an *add list* (i.e., the knowledge that holds after the action execution).

For example, consider a taxi *cab38* at *downtown* and a passenger *fred* at *downtown* too. The goal submitted to the team is to move *fred* from his initial location to *park*. Considering the following actions:

- load a passenger p in a taxi t at a specific location x: $load(p, t, x)$;
- unload a passenger p from a taxi t at a specific location x: $unload(p, t, x)$;
- move a taxi t from a location x to an other y: $move(t, x, y)$.

The action, $move(t, x, y)$, can be executed if and only if there is a taxi t at x and t has enough fuel:

move(t, x, y)
pre $\{at(t, x), hasfuel(t, q), (q \geq 10)\}$
del $\{at(t, x), hasfuel(t, q)\}$
add $\{at(t, y), hasfuel(t, (q - 10))\}$

The action, $load(p, t, x)$, can be executed if and only if there is a taxi t and a passenger p located at the same place x:

load(p, t, x)
pre $\{at(p, x), at(t, x)\}$
del $\{at(p, x)\}$
add $\{in(p, t)\}$

The action, $unload(p, t, x)$, can be executed if and only if there is a taxi t containing a passenger p at x:

unload(p, t, x)
pre $\{in(p, t), at(t, x)\}$
del $\{in(p, t)\}$
add $\{at(p, x)\}$

An action $\alpha \in \mathcal{A}$ is described by a transformation operator:

$$\langle Pre_\alpha, Del_\alpha, Add_\alpha \rangle$$

- Pre_α is the set of predicates that defines the *preconditions* of the action α;
- Del_α is the set of predicates that defines the knowledge that becomes false after the execution of α (del list);
- Add_α is the set of predicates that defines the knowledge that becomes true after the execution of α (add list).

A planning problem is defined by a tuple:

$$\langle \mathcal{E}, \mathcal{O}, \mathcal{G} \rangle$$

- $\mathcal{E}$ defines the knowledge of an agent;
- $\mathcal{O} = \{\langle Pre_\alpha, Del_\alpha, Add_\alpha \rangle \mid \alpha \in \mathcal{A}\}$ defines the description of the actions that an agent can execute (i.e., an operators set);
- $\mathcal{G}$ defines the goal of an agent, (i.e, a set of predicates).

A plan π is an ordered list of actions:

$$\pi = (\alpha_1, \ldots, \alpha_n)$$

where each action α_i is an action in $\mathcal{A}$.

Considering a planning problem $\langle \mathcal{E}, \mathcal{O}, \mathcal{G} \rangle$, a plan $\pi = (\alpha_1, \ldots, \alpha_n)$ defines a sequence of $n + 1$ *world states*

$$\pi = \mathcal{E}_0, \mathcal{E}_1 \ldots, \mathcal{E}_n$$

with

- $\mathcal{E}_0 = \mathcal{E}$ and
- $\mathcal{E}_i = (\mathcal{E}_{i-1} - Del_{\alpha_i}) \cup Add_{\alpha_i} \; (1 \leq i \leq n)$.

A plan $\pi = (\alpha_1, \ldots, \alpha_n)$ is a solution of a planning problem $\langle \mathcal{E}, \mathcal{O}, \mathcal{G} \rangle$ if and only if:

1) the preconditions of all actions hold in the previous world state, $\mathcal{E}_{i-1} \models Pre_{\alpha_i}$ for $1 \leq i \leq n$;

2) the goal is reached in the final state generated by the plan, $\mathcal{E}_n \models \mathcal{G}$.

In our example (paying the taxi is not considered here), a solution plan π is

$$\begin{aligned}
\pi = \quad & (load(fred, cab38, downtown), \\
& move(cab38, downtown, park), \\
& unload(fred, cab38, downtown))
\end{aligned}$$

B. Assumption-Based Planning model

The classical planning model presented in the previous section cannot produce plans with assumptions or conjectures. We define a conjecture as a plan that can be executed if some assumptions hold. That leads us to explain the main difference between classical planning and our approach. This difference relies on the action semantics. Indeed, it is not necessary that all preconditions hold in the current state to execute an action. The preconditions that do not hold, build a set of knowledge about the world $\mathcal{H}$ called *assumptions*.

A conjecture χ is defined as an ordered list of couples

$$\chi = (\langle \mathcal{H}_{\alpha_1}, \alpha_1 \rangle, \ldots, \langle \mathcal{H}_{\alpha_n}, \alpha_n \rangle)$$

where

- $\mathcal{H}_{\alpha_i}$ describes the assumptions that must hold before executing α_i. If $\mathcal{H}_{\alpha_i}$ is an empty set, no assumption is needed to apply α_i;
- α_i is an action in $\mathcal{A}$.

Considering a planning problem $\langle \mathcal{E}, \mathcal{O}, \mathcal{G} \rangle$, a conjecture χ is an ordered list of $n+1$ world states

$$\chi = \mathcal{E}_0, \mathcal{E}_1, \ldots, \mathcal{E}_n$$

with

- $\mathcal{E}_0 = \mathcal{E} \cup \mathcal{H}_{\alpha_1}$ and
- $\mathcal{E}_i = (\{\mathcal{E}_{i-1} \cup \mathcal{H}_{\alpha_{i-1}}\} - Del_{\alpha_i}) \cup Add_{\alpha_i}$ $(1 \leq i \leq n)$.

In our example, if the taxi has not fuel, a possible conjecture is:

$$\begin{aligned}
\chi = \quad & (\langle \{\}, load(fred, cab38, downdown) \rangle, \\
& \langle \{hasfuel(cab38, 10)\}, \\
& \qquad move(cab38, downtown, park) \rangle, \\
& \langle \{\}, unload(fred, cab38, park) \rangle)
\end{aligned}$$

To reach its goal, an agent must check all assumptions made by the conjecture. It can count on its teammates competences to make those assumptions become true. In other words, assumptions made by one agent become additional goals to be satisfied. The Assumption-Based Planning is justified by this possible collaborative reasoning in a multi-agent context.

III. ASSUMPTIONS GENERATION

In classical planning model, operators are applicable if and only if the preconditions of the operators are unifiable with the agent's knowledge base. In order to elaborate conjectures (i.e, plans with assumptions), this constraint is relaxed. We consider that an operator is always applicable even if all preconditions do not hold. Therefore, the application of an operator involves the computation of the lacking facts. This computation is based on the unification algorithm. That is, at least one substitution that makes the preconditions match with some agent's knowledge must be founded. A substitution σ is a finite set of the form $\sigma = \{x_1 \rightarrow t_1, \ldots, x_n \rightarrow t_n\}$ where every x_i is a variable, every t_i is a term not equal to x_i, and $x_i \neq x_j$ for any $i \neq j$. Let σ be a substitution and p be the preconditions of an operator. Then $p\sigma$ is an expression obtained from p by replacing simultaneously each occurrence of the variable x_i with the term t_i.

A. Substitutions computation

The computation of the set of possible assumptions is described by the algorithm 1. Let Pre_α be the preconditions of the action α, $\mathcal{E}$ be the agent's knowledge and σ an empty substitution at the first step of the algorithm. The *FindSubstitutions* procedure computes all possible substitutions from $\mathcal{E}$ to execute α. If Pre_α is empty, then α can always be applied and the procedure returns an empty set (line 2). Otherwise, the procedure tries to find recursively the substitution in order to unify Pre_α with $\mathcal{E}$.

In this case, the *FindSutitutions* procedure unstacks the first precondition p contained in Pre_α (line 5) and tries to unify p for each knowledge of the agent (line 7). If the unification process succeeds, there is a substitution σ that unifies p with a knowledge of $\mathcal{E}$. Therefore, the algorithm is recursively launched on R, the remaining preconditions of α (line 12).

For example, consider the move operator described as follows:

move(t, x, y)
pre $\{at(t,x), hasfuel(t,q), (q \geq 10)\}$
del $\{at(t,x), hasfuel(t,q)\}$
add $\{at(t,y), hasfuel(t,(q-10))\}$

The knowledge of the agent $\mathcal{E}$ is:

$$\begin{aligned}
& \{at(cab38, downtown), isloaded(cab38), \\
& hasfuel(cab38, 10), at(cab74, downtown), \\
& (not(isloaded(cab74))), hasfuel(cab74, 5), \\
& \qquad at(cab73, downtown)\}
\end{aligned}$$

In this case, there are three substitutions:

- $\sigma_1 = \{t \rightarrow cab38, x \rightarrow downtown, q \rightarrow 10\}$
- $\sigma_2 = \{t \rightarrow cab74, x \rightarrow downtown, q \rightarrow 5\}$
- $\sigma_3 = \{t \rightarrow cab73, x \rightarrow downtown\}$

Algorithm 1: FindSubstitutions(Pre_α, $\mathcal{E}$, σ)

1 result $\leftarrow$ an empty set of substitutions ;
2 **if** Pre_α *is empty* **then**
3   **return** result ;
4 **end**
5 $p \leftarrow$ the first term of Pre_α;
6 $R \leftarrow$ the remaining terms in Pre_α;
7 **foreach** *term* $e \in \mathcal{E}$ **do**
8 $\theta \leftarrow$ Unify(p, e, σ);
9   **if** θ == *Failure* **then**
10   **continue** ;
11   **else**
12   **foreach** *substitutions* s in
            FindSubstitutions(R, $\mathcal{E}$, $\sigma\theta$) **do**
13   add a substitution s in result ;
14   **end**
15   **end**
16 **end**
17 **return** result ;

B. Assumptions generation

Each substitution defines a possible executable action in which the parameters are instantiated by the value contained in the substitution. The *GenerateAssumption* procedure is given by the algorithm 2. Let α be an action, σ a substitution and $\mathcal{E}$ a set of knowledge. The algorithm applies for each precondition p of α the substitution σ and check if $p\sigma$ is contained in the agent's knowledge (line 3). If $p\sigma$ is contained in $\mathcal{E}$, $p\sigma$ is not an assumption (line 5). In this case, the algorithm has the same behaviour as a classical planner. Otherwise, $p\sigma$ is an assumption needed to execute α (line 8). In the previous example, the application of the three substitutions σ_1, σ_2 and σ_3 on the preconditions of the *move* operator produces the following assumptions sets (assumptions are labeled with a *):

A) { $at(cab38, downtown)$, $hasfuel(cab38, 10)$, $isloaded(cab38)$ }
B) { $at(cab74, downtown)$, $hasfuel(cab74, 5)$, $isloaded(cab74)*$, $(q \geq 10)$ }
C) { $at(cab73, downtown)$, $hasfuel(cab73, q)*$, $isloaded(cab73)*$, $(q \geq 10)*$ }

The result A of the substitution $\sigma1$ application is a subset of the agent's knowledge. Therefore, no assumption is needed to execute the *move* action. However, the results B and C of the σ_2 and σ_3 applications contain assumptions. The *GenerateAssumptions* procedure distinguishes three kinds of assumptions:

1) **Hypothesis generation:** the substitution can generate literals that do not belong to the current state, e.g. in C) *isloaded(cab73)*. Then, those literals are added to the current state as *hypothesis*. This means that expressions missing from the current state are not

considered as false but rather as unknown. Hypothesis can also contain variable symbols not instantiated if no instance can be found by the substitution function (e.g., *?q* in C) *hasfuel(cab73, ?q)*). This means that *cab73* has an unknown quantity of fuel, but this unknown does not prevent the planning process to proceed (e.g. assessment of *?q* remains an open issue to be fixed by another agent);

2) **Fact negation:** if an atom in the substitution is the negation of a fact in the current state, then this fact is withdrawn and replaced by its negation (e.g., the ground atom in B) *isloaded(cab74)*). In that case, the agent knows that it is reasoning by contradiction and that it bets on its teammates ability to change the world consistently (in that example, by loading *cab74*);

3) **Constraint violation:** as in fact negation, constraints can be violated e.g. in B) *(?q >= 10.00)*.

Algorithm 2: GenerateAssumptions(α, σ, $\mathcal{E}$)

1 result $\leftarrow$ an empty set of assumptions ;
2 $Pre_\alpha \leftarrow$ the preconditions of α ;
3 **foreach** *precondition* $p \in Pre_\alpha$ **do**
4   `// pσ is an agent's knowledge`
5   **if** $p\sigma \in \mathcal{E}$ **then** **continue** ;
6   `// pσ is an assumption`
7   **else**
8   add $p\sigma$ in result ;
9   **end**
10 **end**
11 **return** result ;

IV. PLANNING ALGORITHM

The Assumption-Based Planning algorithm principle relies on a domain independent planning mechanism, HTN (Hierarchical Transition Network). In HTN planner [2], the objective is not to achieve a set of goals but to perform some sets of *tasks*. The agent's input includes a set of operators similar to those used in classical planning [3] and also a set of *methods*, each of which is a prescription on how to decompose some tasks into some sets of subtasks. The agent proceeds by decomposing *non-primitive tasks* recursively into smaller and smaller subtasks, until *primitive tasks*, that can be performed directly by planning operators, are reached.

A. Primary notions

A *primitive action* α is described by an operator $\langle name_\alpha, Pre_\alpha, Del_\alpha, Add_\alpha \rangle$ where $name_\alpha$ is the name of the primitive action, Pre_α is the preconditions set needed to execute α, Del_α and Add_α define respectively the set of effects to delete and to add to the agent's knowledge. An operator is executed when all preconditions Pre_α are

satisfied in the current knowledge state of the agent. The operator execution involves the modification of the current state according to the effects contained in Del_α et Add_α. An operator can be applied if there is a substitution σ for Pre_α such as $Pre_\alpha\sigma$ is instantiated. $n_\alpha\sigma$ defines the action that can be executed for each variable replaced by a value contained in σ. For example, the operator $load(p, t, x)$:

load(p, t, x)
pre $\{at(p, x), at(t, x)\}$
del $\{at(p, x)\}$
add $\{in(p, t)\}$

and the agent's knowledge:

$$\{ at(cab38, downtown), at(fred, donwtown) \}$$

There is a substitution σ that binds the variables p, t et x to the constant values $fred$, $cab38$ and $downtown$. Therefore, the action $load(fred, cab38, downtown)$ can be executed.

A *compound action* α is described by a method $\langle n_\alpha, Pre_\alpha, Act_\alpha \rangle$ where n_α is the name of α, Pre_α is the preconditions set needed to apply α, Act_α defines a list of actions to execute (i.e., the method body). A method can be executed when all preconditions Pre_α are satisfied in the current state. A method execution involves the execution of all actions contained in the ordered actions list Act_α. A method can be applied if there is a substitution σ for Pre_α with $Pre_\alpha\sigma$ instantiated. $Act_\alpha\sigma$ defines the list of actions to execute the method. For example, let a method *move-passenger(p, x, y)* that move a passenger p from a location x to y:

move-passenger(p, x, y)
pre $\{at(p, x), at(t, x)\}$
act $\{load(p, t, x), move(t, x, y), unload(p, t, y)\}$

and the agent's knowledge:

$$\{ at(cab38, downtown), at(fred, downtown) \}$$

There is a substitution σ that binds each variables p, x, y and t with the constant values $fred$, $downtown$, $park$ and $cab38$. Therefore, the action *move-passenger$(fred, downtown, park)$* can be executed. This execution involves the execution of three actions contained in the body of the operator:

1) $load(fred, cab38, downtown)$;
2) $move(cab38, downtown, park)$;
3) $unload(fred, cab38, park)$.

A conjecture is an ordered list of primitive actions. If χ is a conjecture and $\mathcal{E}$ a knowledge state, $\chi(\mathcal{E})$ is the state reached after the execution of χ from $\mathcal{E}$.

A planning problem is defined by $\langle \mathcal{E}, \mathcal{O}, \mathcal{G} \rangle$:
- $\mathcal{E}$ defines the agent's knowledge. This knowledge is described by a set of propositions;
- $\mathcal{O}$ defines the set of operators or methods;

- $\mathcal{G}$ defines the ordered list of goals that must be reached by the agent.

The set of solution conjectures $\mathcal{C}(\mathcal{E}, \mathcal{O}, \mathcal{G})$ of a planning problem can be recursively defined:
- If $\mathcal{G}$ is an empty set, the empty conjecture is returned.
- Otherwise, let α be the first task or goal of $\mathcal{G}$, and $\mathcal{R}$ be the remaining goals:
  1) If α is a primitive action and there is a conjecture χ_1 to reach α then,

  $$\mathcal{C}(\mathcal{E}, \mathcal{O}, \mathcal{G}) = \{append(\chi_1, \chi_2) \mid \chi_2 \in \mathcal{C}(\mathcal{E}, \mathcal{O}, \mathcal{R})\}$$

  2) If α is a primitive action and there is not a conjecture χ to reach α then,

  $$\mathcal{C}(\mathcal{E}, \mathcal{O}, \mathcal{G}) = \emptyset$$

  3) If α is a composed action then,

  $$\mathcal{C}(\mathcal{E}, \mathcal{O}, \mathcal{G}) = \mathcal{C}(\mathcal{E}, \mathcal{O}, append(Act_\alpha, \mathcal{R}))$$

  where Act_α defines the actions list to be executed in order to realise α.

B. Principle

Until now, we have considered that a method or an operator was always executable, even if all their preconditions were not completely held and we have presented the algorithm to compute assumptions. The number of possible assumptions is potentially unlimited. Therefore, it is not possible to compute all possible conjectures. The purpose of the Assumption-Based Planning algorithm is to find conjectures that make the fewest assumptions. To that end, the Assumption-Based Planning algorithm is based on a reachable states search space. This states space is stored in a tree called the *conjecture tree*.

This tree contains the different steps of the agent's reasoning. Each node stands for a state that can be reached after the execution of an action α_i in a conjecture $\chi = (\alpha_1, \ldots, \alpha_n)$ A node n_i is defined by $\langle \mathcal{E}_i, \mathcal{A}_i, w_i \rangle$:
- $\mathcal{E}_i$ is a world state (i.e., a set of propositions fully or partially instantiated);
- $\mathcal{A}_i$ is a list of remaining actions to execute at this step of the reasoning;
- w_i is the valuation of the node (i.e., the number of assumptions given from the root node of the conjecture tree to this node).

The edges define the possible transitions between the different world states. A transition is labeled by a method or operator name and the possible assumptions needed to reach it.

The general procedure to produce a conjecture is given by the algorithm 3. The *FindConjecture* procedure takes as parameters a planning problem: $\mathcal{E}$ the initial state (i.e., the agent's knowledge), $\mathcal{O}$ a list of operators and $\mathcal{G}$ the ordered

list of goals. The root node of the conjecture tree is defined by the node $\langle \mathcal{E}, \mathcal{G}, 0 \rangle$, $\mathcal{E}$ is the initial set of the agent's knowledge, $\mathcal{G}$ the list of goals and null valuation.

The algorithm can be split in two different steps: the conjecture tree expansion (line 3) which represents the reachable states space and the conjecture extraction (line 4).

Algorithm 3: FindConjecture($\mathcal{E}$, $\mathcal{O}$, $\mathcal{G}$)

---

1   root $\leftarrow \langle \mathcal{E}, \mathcal{G}, 0 \rangle$;
2   CTree $\leftarrow$ create a conjecture tree with root node root ;
3   ExpandConjectureTree(*CTree*, *root*) ;
4   **return** ExtractConjecture(*CTree*) ;

C. Conjecture tree expansion

Unlike HTN recursive algorithm, the conjecture tree expansion is not a simple depth first exploration: the computation of the conjecture with the fewest assumptions is equivalent to a minimization problem. In order to solve this problem, the expansion algorithm is based on the "branch and bound" algorithm. The nodes stored in the conjecture tree are valuated by the number of assumptions made (see algorithm 4) to reach it and the node with the weakest valuation is recursively chosen at the expansion step to expand the conjecture tree until a leaf is found.

The expansion procedure is described by the algorithm 4. From the current exploration node $n_i = \langle \mathcal{E}_i, \mathcal{A}_i, w_i \rangle$, the *ExpandConjecture* procedure tries to apply the first action α contained in $\mathcal{A}_i$ (line 1). Then it computes the possible substitutions Σ for the preconditions of α according to the knowledge $\mathcal{E}_i$ (line 6). For each substitution, the procedure generates the assumptions $\mathcal{H}_\alpha$ needed to apply α (line 10). If α is a primitive action (line 14), then it adds a child node n_{i+1} to the current node such as:

$$n_{i+1} = \langle\ (\{\mathcal{E}_i \cup \mathcal{H}_\alpha\} - Del_\alpha) \cup Add_\alpha, \mathcal{R}, w_i + |\mathcal{H}_\alpha|\ \rangle$$

where $|\mathcal{H}_\alpha|$ defines the number of assumptions needed to apply α. Otherwise (line 20), α is a composed action and the procedure adds a child node to the current node such as:

$$n_{i+1} = \langle\ \{\mathcal{E} \cup \mathcal{H}_\alpha\}, append(Act_\alpha, \mathcal{R}), w_i + |\mathcal{H}_\alpha|\ \rangle$$

Finally, the procedure evaluates the new current node by computing the node with the lowest valuation in the conjecture tree (line 30) and runs recursively the expansion procedure with the new node (line 31).

When assumptions are generated by the algorithm, the expansion procedure checks if the assumptions are legal (line 11). An assumption is legal if its assumption predicates was described in the planning domain as hypothetical. The legality notion was added to our algorithm to specify states of the world that can be hypothetical. This mechanism allows to

---

Algorithm 4: ExpandConjectureTree($CTree$, n_i)

---

1   `// Extract the first action` α `contained in`
    `// the current node` n_i
2   $\alpha \leftarrow$ the first action contained in n_i ;
3   $\mathcal{R} \leftarrow$ the remaining actions contained in n_i ;

4   `// Compute the possible substitutions from`
5   `//` Pre_α`,` α `preconditions, and` $\mathcal{E}_i$`, the world`
6   `// of the current node` n_i `and` σ `an empty`
7   `// substitution`
8   $\Sigma \leftarrow$ FindSubstitutions(Pre_α, $\mathcal{E}_i$, σ) ;

9   `// Create a new node` n_{i+1} `for each`
10   `// each applicable action`
11   **foreach** *substitution* $\sigma \in \Sigma$ **do**
12   `// Compute the assumptions needed to`
13   `// apply` α
14      $\mathcal{H}_\alpha \leftarrow$ GenerateAssumptions(α, σ, $\mathcal{E}_i$) ;
15   `// Check if the assumption generated`
16   `// are legal`
17   **if** $\exists\, h \in \mathcal{H}_\alpha$ with h a legal assumption **then**
     continue ;
18   `//` α `is a primitive action`
19   **if** α *is primitive* **then**
20        $\mathcal{E}_{i+1} \leftarrow (\{\mathcal{E}_i \cup \mathcal{H}_\alpha\} - Del_\alpha) \cup Add_\alpha$;
21        $\mathcal{A}_{i+1} \leftarrow \mathcal{R}$;
22        $w_{i+1} \leftarrow w_i + |\mathcal{H}_\alpha|$ avec $|\mathcal{H}_\alpha|$ the number of elements in $\mathcal{H}_\alpha$;
23   **end**

24   `//` α `is a composed action`
25   **else**
26        $\mathcal{E}_{i+1} \leftarrow \mathcal{E}_i \cup \mathcal{H}_\alpha$;
27        $\mathcal{A}_{i+1} \leftarrow$ Append(Act_α, $\mathcal{R}$) ;
28        $w_{i+1} \leftarrow w_i + |\mathcal{H}_\alpha|$ with $|\mathcal{H}_\alpha|$ the number of elements in $\mathcal{H}_\alpha$;
29   **end**

30   `// Add a new node` n_{i+1} `to the`
31   `// conjecture tree`
32   add a new node $n_{i+1} = \langle \mathcal{E}_{i+1}, \mathcal{A}_{i+1}, w_{i+1} \rangle$;
33   **end**

34   `// Compute the node with few assumptions`
35   `// and run recursively the procedure`
36   `// ExpandConjectureTree`
37   bestNode $\leftarrow$ the node with few assumptions in $CTree$;
38   ExpandConjectureTree(*bestNode*) ;

---

reduce the states space to explore, limiting the size of the conjecture tree.

D. Conjecture Extraction

The second step is the conjecture extraction from the conjecture tree (see Algo. 5). A conjecture is represented by a branch (i.e. a path from the root node to a leaf). First,

the *ExtractConjecture* procedure computes among the set of solution nodes (i.e., the leaf of the conjecture tree) the node that makes the fewest assumptions (line 2). For each edge of the branch from the chosen solution node to the root node (line 3), the procedure checks the type of transition (i.e., primitive or composed action). If the edge is labeled with a primitive one, then the action and their assumptions are added to the conjecture (line 5). Otherwise, only the assumptions are added to the conjecture (line 7).

Algorithm 5: ExtractConjecture($CTree$)

1   $\chi \leftarrow$ an empty conjecture ;
2   $n \leftarrow$ the leaf solution node with the fewest assumptions ;
3   **while** n *is not the root node of* **CTree** **do**
4   **if** n *was built with a primitive action* α_i **then**
5   add action $\langle \mathcal{H}_{\alpha_i}, \alpha_i \rangle$ to χ;
6   **else**
7   add the assumptions $\mathcal{H}_{\alpha_i}$ to the action α_{i-1} contains in χ ;
8   **end**
9     $n \leftarrow$ the father node of n ;
10 **end**
11 **return** χ ;

V. DISCUSSION

Soundness and completeness: The algorithm tries to decompose the initial goal in an ordered list of primitive tasks. As in SHOP, for a finite search space, the construction of the conjecture tree is sound and complete. For infinite search spaces, it is also complete due to the iterative-deepening conjecture tree construction. However, our algorithm is more greedy because, when assumptions must be done, more nodes are created. The primary tests on our JAVA implementation highlight results in the same order of magnitude than the JSHOP algorithm (the JAVA implementation of SHOP algorithm).

Search limitation: The number of allowed assumptions can be bounded in order to end the search process at an arbitrary limit. When the limit is set to 0, the algorithm is equivalent to SHOP. This can be used to adapt our algorithm to the system capabilities and find conjecture with more and more assumptions.

Choice of a planning system: There are many different planning systems (e.g., planning based on Binary Decision Diagrams [4], Mutex [5], heuristic search [6], constraints satisfaction [7] and so forth). However, SHOP is well-suited for assumptions generation thank to the substitution procedure that allows to compare the agent's knowledge with the preconditions necessary to trigger an operator or a method. We found out that assumptions generation turned out to be much more difficult in planners like GRAPHPLAN [5].

Adequacy with interactions: In teamwork context, a conjecture can be refuted by another agent; if no repair is found, the conjecture must be abandoned. In this case, the cost of providing another conjecture is low because agents can rely on the conjecture tree already computed and resume their exploration.

VI. RELATED WORK

The problem of constructing plans in a distributed environment has been approached from two different directions. One approach makes an emphasis on the problem of controlling and coordinating the actions of multiple agents in a shared environment. The others approaches focus on planning and how it can be extended into distributed environment, where the process of formulating a plan could involve actions and interactions of many participants. The planning approach is the nearest to Assumption-Based Planning.

The first approach objective is not to form a good collective plan, but rather to ensure that the agent's local objective will be met by this plans. This approach based on BDI models formalizes the distributed planning process using the mental states of the agents [8], [9], [10]. These approaches emphasize the necessity for a group of agents to share a *joint intention* in order to reach a goal [11], [12]. They have been validated in projects such as STEAM [13]. Another cognitive approach [14] formalizes the coordination process by two kinds of intentions: the agent's *intention to* do an action or a plan and the agent's *intention that* some propositions hold. Then, shared plans are generated by combining predefined plans or "recipes".

The second approach [15], [16] places the problem of forming a plan as the ultimate objective and is typically carried out by agents that have been endowed with shared goals and representation. This approach can be divided in three distinct steps: the decomposition step of tasks into subtasks [17], [18], the allocation step [19] and the execution and conflict resolution step. This formal division is sometimes difficult to implement because the three steps are not independent in many cases: conflict resolution can imply to re-allocate several subtasks or to seek another decomposition and, therefore, deadlocks must be carefully dispelled. Several frameworks focus on the detection of relations between plans [20]. Two kinds of relations are identified: positive relations (e.g., redundant tasks) or negative relations (e.g., resources conflicts). Conflicts are solved in many ways: by negotiation [21], by argumentation [22], [23], by synchronization [24], [25] etc. More recently, the systems DSIPE [26] that is a distributed version of SIPE-2 or the dMARS project [27] based on a PRS architecture, are interested in solving real-world planning problems and, to that end, argue for the use of domain knowledge in planning.

Although these coordination mechanisms bring us a number of answers to make a group of agents work together as a team, they show limitations to solve cooperative task when the goals of the agents are not well proportioned. These limitations can be explained by the fact that coordination mechanisms are not interleaved in the planning process and often use predating plans or recipes libraries. Another limitation is due to the difficulty to take into account the necessary uncertainty in real-world because of the increasing planning complexity coming from incomplete information.

VII. CONCLUSION

The Assumption-Based Planning model outlined in this paper relies on plan production and revision by conjecture/refutation cycles. We presented one of the most important elements of this project: the Assumption-Based Planner. This process is based on a HTN planner, SHOP. For a given initial task, the algorithm seeks the less hypothetical plan (i.e, conjecture). It breaks down the initial task recursively into simple operators. At each step, if no complete substitution is found, the operators or methods constraints are relaxed in order to compute the assumptions necessary to push the plan elaboration further. The construction of the conjecture tree is guided by the valuation of each node in terms of assumptions guaranteeing to find the most reasonable conjecture. As far as teamwork is concerned, each time a conjecture is definitively abandoned, a new one can be proposed from the ongoing conjecture tree.

Thus, Assumption-Based Planning model merges in the collaborative plan generation, the decomposition and the coordination steps. Moreover, the Assumption-Based Planning includes in the agents' reasoning the notion of uncertainty and allows to compose the agents competences. The argumentation is used to structure the multi-agent reasoning as a collaborative investigation process and not as a negotiation one. From our point of view, this approach is suitable for applications in which agents share a common goal and in which the splitting of the planning and the coordination steps (when agents have independent goals, they locally generate plans and then solve their conflicts) becomes difficult due to the agents strong interdependence. Our target applications are the composition of web services, the cooperation of video games characters and the dynamic reconfiguration of GUI components.

Our future research direction will be the formalization of the conjecture/refutation protocol: the corresponding dialog games have to define interaction rules ensuring the convergence of the multi-agent planning process towards a mutually accepted conjecture; and the identification of different kinds of domain-independent plan refutation to build robust plans.

REFERENCES

[1] I. Lakatos, *Proofs and Refutations*. Cambridge University Press, 1976.

[2] D. Nau, T. Au, O. Ilghami, U. Kuter, W. Murdock, D. Wu, and F. Yaman, "Shop2: An HTN planning system," *Journal of Artificial Intelligence Research*, vol. 20, pp. 379–404, 2003.

[3] R. Finkes and N. Nilsson, "Strips: A new approach to the application of theorem proving to problem solving," *Artificial Intelligence*, vol. 3-4, no. 2, pp. 189–208, 1971.

[4] R. M. Jensen and M. M. Veloso, "OBDD-based universal planning for multiple synchronized agents in non-deterministic domains," in *Artificial Intelligence Planning Systems*, 2000, pp. 167–176.

[5] A. Blum and M. Furst, "Fast planning through planning graph analysis," *Artificial Intelligence*, vol. 90(1-2), pp. 279–298, 1997.

[6] M. B. Do and S. Kambhampati, "Sapa: A domain-independent heuristic metric temporal planner," in *Proceedings of the European Conference on Planning*, 2001.

[7] H. Kautz and B. Selman, "Unifying SAT-based and graph-based planning," in *Workshop on Logic-Based Artificial Intelligence*, J. Minker, Ed., College Park, Maryland, USA, 1999.

[8] M. E. Bratman, *Intention, Plans, and Practical Reason*. Cambridge, MA, USA: Harvard University Press, 1987.

[9] M. Pollack, "Plans as complex mental attitudes," in *Intentions in Communication*, 1990, pp. 77–103.

[10] A. S. Rao and M. P. Georgeff, "Modeling rational agents within a bdi-architecture," in *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*, San Mateo, CA, USA, 1991, pp. 473–484.

[11] P. R. Cohen and H. J. Levesque, "Intention is choice with commitment," *Artificial Intelligence*, vol. 42, no. 2-3, pp. 213–261, 1990.

[12] H. J. Levesque, P. R. Cohen, and J. H. T. Nunes, "On acting together," in *Proceedings of the National Conference on Artificial Intelligence*, Boston, MA, USA, 1990, pp. 94–99.

[13] M. Tambe, "Towards flexible teamwork," *Journal of Artificial Intelligence Research*, vol. 7, pp. 83–124, 1997.

[14] B. J. Grosz and S. Kraus, "Collaborative plans for complex group action," *Artificial Intelligence*, vol. 86, no. 1, pp. 269–357, 1996.

[15] D. Corkill, "Hierarchical planning in distributed environment," in *Proc. IJCAI*, Cambrige, MA, USA, 1979, pp. 168–179.

[16] V. Lesser and D. Corkill, "Functionally-accurate cooperative distributed systems," *IEEE Transaction on Systems, Man, and Cybernetics*, vol. 11, no. 1, pp. 295–310, 1981.

[17] E. H. Durfee and V. R. Lesser, "Using partial global plans to coordinate distributed problem solvers," in *Proc. IJCAI*, Milan, Italy, 1987, pp. 875–883.

[18] A. Lansky and L. Getoor, "Scope and abstraction: Two criteria for localized planning," in *Proceedings of the International Joint Conference on Artificial Intelligence*, Montreal, Canada, 1995, pp. 1612–1618.

[19] R. G. Smith and R. Davis, "Frameworks for cooperation in distributed problem solving," *Distributed Artificial Intelligence*, pp. 61–70, 1988.

[20] F. von Martial, "Interactions among autonomous planning agents," in *Proceedings of the European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, 1990, pp. 105–119.

[21] G. Zlotkin and J. S. Rosenschein, "Negotiation and conflict resolution in non-cooperative domains," in *Proc. AAAI*, Boston, MA, USA, 1990, pp. 100–105.

[22] S. Kraus, M. Nirkhe, and K. P. Sycara, "Reaching agreements through argumentation: a logical model (preliminary report)," in *Proceedings of the International Workshop on Distributed Artificial Intelligence*, Hidden Valley, Pennsylvania, USA, 1993, pp. 233–247.

[23] M. Tambe and H. Jung, "The benefits of arguing in a team," *The Artificial Intelligence Magazine*, vol. 20, no. 4, pp. 85–92, 1999.

[24] B. Bradley, A. Barrett, and H. Durfee, "Theory for coordinating concurrent hierarchical planning agents using summary information," in *Proc. AAAI*, 1999, pp. 495–502.

[25] B. Bradley and A. Barrett, "Continual coordination through shared activites," in *Proc. AAMAS*, Melbourne, Australia, 2003, pp. 57–67.

[26] M. desJardins and M. Wolverton, "Coordinating planning activity and information flow in a distributed planning system," *Artificial Intelligence Magazine*, 2000.

[27] M. D'Inverno, M. Luck, M. Georgeff, D. Kinny, and M. Wooldridge, "The dmars architecture: A specification of the distributed multi-agent reasoning system," *Autonomous Agents and Multi-Agent Systems*, vol. 9, no. 1-2, pp. 5–53, 2004.