First Principles Planning in BDI Systems*

Lavindra de Silva
RMIT University
Melbourne, Australia
ldesilva@cs.rmit.edu.au

Sebastian Sardina
RMIT University
Melbourne, Australia
ssardina@cs.rmit.edu.au

Lin Padgham
RMIT University
Melbourne, Australia
linpa@cs.rmit.edu.au

ABSTRACT

BDI (Belief, Desire, Intention) agent systems are very powerful, but they lack the ability to incorporate planning. There has been some previous work to incorporate planning within such systems. However, this has either focussed on producing low-level plan sequences, losing much of the domain knowledge inherent in BDI systems, or has been limited to HTN (Hierarchical Task Network) planning, which cannot find plans other than those specified by the programmer. In this work, we incorporate classical planning into a BDI agent, but in a way that respects and makes use of the procedural domain knowledge available, by producing abstract plans that can be executed using such knowledge. In doing so, we recognize an intrinsic tension between striving for abstract plans and, at the same time, ensuring that unnecessary actions, unrelated to the specific goal to be achieved, are avoided. We explore this tension, by first characterizing the set of "ideal" abstract plans that are non-redundant while maximally abstract, and then developing a more limited but feasible account in which an abstract plan is "specialized" into a new abstract plan that is non-redundant and preserves abstraction as much as possible. We describe an algorithm to compute such a plan specialization, as well as algorithms for the production of a valid high level plan, by deriving abstract planning operators from the BDI program.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Intelligent Agents, Languages and structures*

General Terms

Theory, Algorithms

1. INTRODUCTION

BDI (Belief, Desire, Intention) systems are a widely used platform for developing agent systems, and have been claimed to provide a more than 300% efficiency gain in developing complex systems [1]. It is also recognised that they provide robustness and flexibility. Nonetheless, they generally lack any planning functionality. In [12], a principled approach to incorporating HTN (Hierarchical Task Network) planning into BDI systems was provided, by using the knowledge contained in the BDI plan library. Still, such an approach only provides look-ahead planning to assist in choosing among existing plans. It is unable to create any new plan structures.

In this work, we consider the problem of using *classical first-principles* planning to find plans not currently available as part of the plan library. Unlike some earlier work on adding classical planning to BDI systems (e.g., [10, 3]), we focus on producing *abstract*, so-called *hybrid plans*, using the knowledge encoded in the BDI system as much as possible. Hybrid-plans may include *abstract operators*, which can be mapped back to BDI goals, thus allowing the agent to execute the plan using its BDI plan library. This has two substantial advantages as compared to plans made up of primitive actions only. Firstly, it respects what Kambhampati et al. refer to as the *user-intent* principle [8]: "good" plans are ones whose primitive actions can be parsed in terms of goals whose primary effects support the goal state, as such plans respect relevant encoded domain knowledge. Our approach of directly using BDI goals as abstract operators intuitively also ensures that the resulting sequence of primitive actions is parseable in terms of higher level goals. Secondly, providing an abstract plan allows for flexibility and robustness during execution, since standard BDI plan selection and failure recovery can be used to adapt to environmental changes.

1. *Navigate(rock1, rock2)*
2. *PerformSoilExperiment(rock2)*
3. *Navigate(rock2, rock3)*
4. *PerformSoilExperiment(rock3)*

(a) Hybrid-solution h

1. *Navigate(rock1, rock2)*
2. *ObtainSoilResults(rock2)*
3. *EstablishConnection*
4. *SendResults(rock2)*
5. *Navigate(rock2, rock3)*
6. *ObtainSoilResults(rock3)*
7. *SendResults(rock3)*
8. *BreakConnection*

(b) Hybrid-solution h'

1. *Navigate(rock1, rock2)*
   (A) *CalibrateViaGPS*
   (B) *Move(rock1, rock2)*
2. *PerformSoilExperiment(rock2)*
   (A) *ObtainSoilResults(rock2)*
      (i) *PickSoilSample(rock2)*
      (ii) *AnalyseSoilSample(rock2)*
         (a) *GetMoistureContent(rock2)*
         (b) *GetSoilParticleSize(rock2)*
         (c) *GetSoilDensity(rock2)*
      (iii) *DropSoilSample*
   (B) *TransmitSoilResults(rock2)*
      (i) *EstablishConnection*
      (ii) *SendResults(rock2)*
      (iii) *BreakConnection*
3. *Navigate(rock2, rock3)*
   (A) *CalibrateViaGPS*
   (B) *Move(rock2, rock3)*
4. *PerformSoilExperiment(rock3)*
   (A) *ObtainSoilResults(rock3)*
      (i) *PickSoilSample(rock3)*
      (ii) *AnalyseSoilSample(rock3)*
         (a) *GetMoistureContent(rock3)*
         (b) *GetSoilParticleSize(rock3)*
         (c) *GetSoilDensity(rock3)*
      (iii) *DropSoilSample*
   (B) *TransmitSoilResults(rock3)*
      (i) *EstablishConnection*
      (ii) *SendResults(rock3)*
      (iii) *BreakConnection*

(c) Execution trace of hybrid-solution h

Figure 1: A redundant hybrid-solution h, a non-redundant hybrid-solution h', and the execution trace of h.

*We would like to thank the anonymous reviewers for their helpful suggestions. We would also like to acknowledge the support of Agent Oriented Software and the Australian Research Council (under grant LP0882234), and of the National Science and Engineering Research Council of Canada (under a PDF fellowship).

Cite as: First Principles Planning in BDI Systems, Lavindra de Silva, Sebastian Sardina, Lin Padgham, *Proc. of 8th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2009)*, Decker, Sichman, Sierra and Castelfranchi (eds.), May, 10–15, 2009, Budapest, Hungary, pp. 1105–1112
Copyright © 2009, International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org), All rights reserved.

One problem that must be addressed in developing abstract plans is that of avoiding overly abstract operators, which, while achieving the desired goal, also yield additional unwarranted actions. Consider the example of a Mars Rover agent exploring the surface of Mars. At some point, the agent invokes a planner, which returns the hybrid-solution h shown in Figure 1(a). Its execution involves carrying on primitive and non-primitive (i.e., abstract) steps, as can be seen in the execution trace shown in Figure 1(c) (e.g., navigating is an abstract step involving two primitive steps). Now, notice that in this execution, breaking the connection after sending the results for $rock2$, and then establishing the connection again before sending the results for $rock3$ are (unwarranted) redundant steps. Such redundancy is brought about by the overly abstract task *PerformSoilExperiment*. What one would prefer to have is the *non-redundant* hybrid-solution h' of Figure 1(b). This plan avoids the redundancy inherent in the initial solution h, yet still retains much of the structure of the abstract plans provided by the programmer. In particular, it retains the abstract tasks *Navigate* and *ObtainSoilResults*, which would allow their achievement in an alternative manner to that shown here, if such existed and was warranted by the situation at execution time. The replacement of tasks *PerformSoilExperiment* and *TransmitSoilResults* with a subset of their components is thus motivated in order to remove redundancy.

We note that, in addition to the redundant primitive actions observed in this example, it is possible for a hybrid-plan to contain unwarranted abstract operators, that is, abstract tasks whose corresponding primitive actions are all redundant. In such cases, it may be sensible to remove the abstract task altogether.

Interestingly, it turns out that there is an intrinsic tension between the use of maximally-abstract tasks and the removal of redundancy. After describing, in Section 3, how to obtain a valid hybrid-plan by using a classical planner and domain information derived from the BDI program, we explore how to strike a principled and well-defined balance between abstraction and removal of redundancy. In Section 4, hence, we provide an *ideal definition* of a minimal, non-redundant, maximally-abstract hybrid-plan. Since finding such solutions is computationally difficult, we develop, in Section 5, a weaker notion and an algorithm to "improve" a given hybrid-solution w.r.t. non-redundancy and maximal-abstraction.

2. PRELIMINARIES

The goal of our work is the synthesis of (new) abstract plans, using existing BDI event-goals, such that the abstract plans can be executed by the BDI execution engine in the same way as a plan from the plan library. As has been previously noted (e.g., [4, 13, 12]), HTN planners use similar hierarchical structures to those of BDI systems, though for different objectives. Basically, HTN compound tasks map to BDI event-goals, while HTN methods map to BDI plan-rules (plans). As the technical machinery that we need is already well developed for HTN planners, we will review, and then use this in our work, following the notation of [5].

The central concept in HTN planning is that of a *task*. There are two kinds of tasks. A *primitive task* is an action $act(\vec{x})$ which can be directly executed by the agent in the environment (e.g., $drive(x_1, x_2)$). A (high-level) *compound task* (also called a *non-primitive task*) $t(\vec{x})$ is one that cannot be executed directly (e.g., $travel(origin, dest)$). A *task network* $d = [s, \phi]$ is a non-empty collection s of labeled tasks of the form $(n : t)$ (labels are unique in d) that need to be accomplished, and a boolean formula of constraints ϕ. Constraints impose restrictions on the ordering of tasks $(n \prec n')$, on the binding of variables, and on what literals must be true before or after a task $(l, n), (n, l)$, or between two tasks (n, l, n'). A *method* $\langle t, d\rangle$ encodes a legal way of decomposing a compound task t into other tasks; in particular, a method specifies in task network d the legal sub-tasks for accomplishing t, as well as the constraints (if any) that must hold among those sub-tasks. HTN methods thus provide the procedural knowledge of the domain. For example, method m_{travel} below encodes one way of travelling from x to y, when those locations are close.

$$m_{travel} = \langle travel(x, y),\ d_{taxi}\rangle;$$
$$d_{taxi} = [\{t_1 : getTaxi,\ t_2 : ride(x, y),\ t_3 : payDriver\}, \phi];$$
$$\phi = t_1 \prec t_2 \wedge t_1 \prec t_3 \wedge ((t_1, Flat) \vee t_2 \prec t_3) \wedge (Close(x, y), t_1).$$

Notice that, when traveling by taxi, one should always pay at the end of the trip, unless the tariff found after booking the taxi is flat.

An *HTN planning domain* $\mathcal{D} = \langle Op, Me\rangle$ consists of a library Me of methods and a library Op of STRIPS operators. An *HTN planning problem* $\mathcal{P}_{htn}$ is a triple $\langle d, \mathcal{I}, \mathcal{D}\rangle$, where d is the task network to be accomplished, $\mathcal{I}$ is the initial state (i.e., the set of all ground atoms that are true initially), and $\mathcal{D}$ is a planning domain. A *primitive plan* σ is a sequence $act_1 \cdot \ldots \cdot act_k$ of ground primitive tasks (i.e., actions). A *labeled* primitive plan $\tau = (n_1 : act_1) \cdot \ldots \cdot (n_k : act_k)$ is like a primitive plan but with labeled tasks. We shall sometimes use them interchangeably with the obvious meaning.

Given a planning problem $\mathcal{P}_{htn} = \langle d, \mathcal{I}, \mathcal{D}\rangle$, the planning process involves repetitively selecting and applying an applicable reduction method from $\mathcal{D}$ to some compound task in d. This results in a new, and typically more "primitive," task network d'. This reduction process is repeated until only primitive tasks are left. Formally, if $d = [s, \phi]$ is a task network, $(n : t) \in s$ is a labeled non-primitive task occurring in d, and $m = \langle t', d'\rangle$ is a method that may be used to decompose t (that is, t and t' unify), then *reduce*(d, n, m) denotes the set of task networks that result from *decomposing* task $(n : t)$ in network d using method m. Informally, such decomposition involves updating both the set s, by replacing $(n : t)$ with the tasks in d' (by arbitrarily renaming task labels), and the constraints ϕ to account for the constraints in d'. We then define the set of all possible reductions of d as follows:[1]

$$\mathit{red}(d, \mathcal{D}) = \{d' \mid d' \in \mathit{reduce}(d, n, m), (n : t) \in s, m \in Me\}.$$

A primitive plan σ is a *completion* of primitive task network d (i.e., one containing only primitive tasks) at state $\mathcal{I}$, denoted $\sigma \in \mathit{comp}(d, \mathcal{I}, \mathcal{D})$, if σ is a total ordering of the primitive tasks in a ground instance of d that satisfies the constraint formula in d.

Finally, by using sets $\mathit{red}(d, \mathcal{D})$ and $\mathit{comp}(d, \mathcal{I}, \mathcal{D})$, one can easily define the set of plans $\mathit{sol}(d, \mathcal{I}, \mathcal{D})$ that solves an HTN planning problem $\mathcal{P}_{htn} = \langle d, \mathcal{I}, \mathcal{D}\rangle$ (see [5]). Note that all these definitions generalize trivially to *labeled* plans τ.

We point out that the process of HTN decomposition can also be seen as *traces*. So, we say that a *decomposition trace* for task network d is a sequence of task networks $\lambda = d_1 = d \cdot \ldots \cdot d_k$ such that $d_{i+1} \in \mathit{red}(d_i, \mathcal{D})$, for all $1 \leq i < k$. A *complete* trace is one whose last task network is primitive. A plan σ, hence, is a solution for a task network d if it is a completion of the final task network d_k in a complete decomposition trace of d.

HTN planners, like BDI systems, focus on "goals-to-do," that is, how to achieve a given task in an initial state, by making use of domain specific procedural information. Classical planners, on the other hand, focus on "goals-to-be," that is, how to bring about a specific *goal state* from first-principles. A classical planning problem is thus defined as a tuple $\mathcal{C} = \langle \mathcal{I}, \mathcal{G}, Op\rangle$, where $\mathcal{I}$ is the initial state, $\mathcal{G}$ is the specification of the goal state to be achieved, and Op is the model of actions, generally as STRIPS operators. A primitive plan σ solves $\mathcal{C}$, denoted $\sigma \in \mathit{sol}(\mathcal{I}, \mathcal{G}, Op)$, iff σ is executable in $\mathcal{I}$

[1] We have omitted the state in function *red* as it is not necessary.

(i.e., all actions' preconditions are satisfied) and the state resulting from the execution of σ satisfies $\mathcal{G}$.

In this paper, we investigate what we refer to as *hybrid-planning*: synthesizing *abstract* plans that can bring about a certain state of affairs (as in classical planning) by making use of available domain knowledge (as in HTN planning). This type of planning is particularly appealing in the context of BDI agent systems. Hybrid-planning may enhance BDI agents by providing *new* plans that were not encoded by the programmer. It will do so, though, by re-using whatever procedural knowledge is available in the system. By obtaining plans at a high-level of abstraction, we support the flexibility and robustness of these systems: if an abstract step in a plan happens to fail, another option may be tried to achieve the step. Solving a hybrid planning problem, then, involves constructing an *abstract* plan that can be decomposed using the domain knowledge into a primitive plan that brings about the goal state.

A *hybrid planning problem* is a tuple $\mathcal{H} = \langle \mathcal{I}, \mathcal{G}, \mathcal{D} \rangle$, where $\mathcal{I}$ is the initial state, $\mathcal{G}$ is the goal state, and $\mathcal{D}$ is an HTN planning domain. A *hybrid-plan* $h = [s, \phi]$ is a task network where ϕ stands for a conjunction of ordering constraints. Thus, hybrid-plans are what is often referred to as *partially-ordered* plans [11]. Finally, a hybrid-plan h is a *hybrid-solution* for a hybrid planning problem $\mathcal{H}$ iff $sol(h, \mathcal{I}, \mathcal{D}) \cap sol(\mathcal{I}, \mathcal{G}, Op) \neq \emptyset$, that is, if there exists an HTN solution for h—a primitive plan—that achieves the goal.

3. OBTAINING A HYBRID-SOLUTION

In order to obtain a hybrid-plan that achieves a given goal state, given an initial state, we follow three basic steps: *(i)* we first transform event-goals in the BDI system into abstract operators; *(ii)* we then call the classical planner of choice with the current (initial) state, the required goal state, and the abstract operators obtained in the first step; and finally, *(iii)* we check the correctness of the plan obtained to ensure that there is a viable decomposition.[2] This final step is indeed necessary due to the incompleteness of the representation used in the first transformation step.

3.1 Obtaining the Abstract Operators

Similarly to Kambhampati et al. [8], we view the plan library as capturing valuable domain specific information which should be used/respected when obtaining new plans, as the plan library preserves what Kambhampati et al. refer to as *user-intent*. In order to use event-goals as (abstract) operators for our planning domain, we require both pre-conditions and post-conditions associated with event-goals. Typically, pre-conditions are specified on particular plans in a BDI system as a context condition. The pre-condition of an event-goal can then be obtained straightforwardly as the disjunction of the context conditions of the associated plans in the plan library. If post-conditions are already specified for event-goals or plans (as they are in some systems [3]), then these can be used directly. In the case that they are not specified, we compute *definite effects* of an event-goal, based on the structure of event-goals and plans, combined with knowledge of the effects of basic actions. This is done using an adaptation and extension of the summarisation algorithm of Clement et al. [2]. The effects obtained in this manner are (potentially) a superset of the *primary effects* of an event-goal supplied by the programmer in the work of [8], as our computed definite effects will include any necessary side effects.

Intuitively, *definite effects* of an event-goal are those things which are *always true* after successfully executing *any* decomposition of plans to achieve that event-goal. We also calculate *possible effects* of an event-goal which are those things that *may* result from *some* decomposition of plans to achieve the event-goal. More precisely, a *definite effect* of an event-goal corresponds to a literal that *(i)* is asserted in every successful sequence of steps starting from the event-goal, and *(ii)* holds at the end of all such successful sequences. A *possible effect* of an event-goal, on the other hand, corresponds to a literal that *(i)* is mentioned within a decomposition of the event-goal, and *(ii)* may or may not hold at the end of some successful sequence of steps starting from the event-goal.[3]

Although our approach is similar to that of [2], there are also some important extensions and differences. Most importantly, we allow variables in literals, event-goals and actions, whereas the summary algorithm of [2] does not allow variables in any of these entities. Consequently, we need to account for the possibility that values assigned at runtime to variables in literals may cause literals to conflict. For example, take the following plan-body:

$$\begin{array}{l} \vdots \\ +Colour(block1, blue); \\ ?(Block(?b) \wedge Colour(?b, blue)); \\ -Colour(?b, blue); \\ +Colour(?b, red). \end{array}$$

This plan adds a belief that $block1$ is blue, then binds the variable $?b$ to some blue block (possibly $block1$), removes the belief that $?b$ is blue and adds the belief that $?b$ is red. The literals $Colour(block1, blue)$ and $Colour(?b, red)$ are both asserted in the body of this plan. However, only $Colour(?b, red)$ can be considered a definite effect, as $Colour(block1, blue)$ will be true only if $?b$ is not bound to $block1$. Therefore, $Colour(block1, blue)$ is only a possible effect. In our summarisation, we reason about which literals will conflict as a consequence of values assigned to their variables at runtime, and determine which literals will *definitely* be met, and which literals will *possibly* be met, on successful execution of the program being summarised.

The second main difference in our approach to that of [2] is that we avoid placing constraints on interactions between entities in a plan. In [2], there is a requirement that all possible traces through a goal-plan tree resulting from a plan are able to successfully execute. If this is not the case, then the plan is said to be inconsistent. However, this requirement is too strong, since it is natural for an event-goal to be used in a plan with the expectation that only certain plans of that event-goal will be applicable. This is particularly true if event-goals, and their associated plans, are to be re-usable components. In [2], if an event-goal (say e_1) in a plan (say p_1) has some plan whose pre-condition could be clobbered by a plan of some earlier event-goal in p_1, then p_1 is said to be inconsistent, even though there may always be other suitable plans for handling e_1. Thus, for a plan to be consistent, every event-goal mentioned in it must be handled *only* by plans whose preconditions are not made false by effects brought about by plans of other event-goals mentioned in the plan in question.

We avoid constraining our plans in this way, although this leads to a weaker definition of possible effects than the corresponding definition of may summary conditions in [2]. In our definition, there can be literals which are mentioned in some plan (and are therefore part of our possible effects), but in fact can never be asserted, due to interactions which ensure that the particular plan which asserts that literal can never be applied.

Another difference as compared to [2] is that our precondition is

[2]We use the CANPLAN language [12] as our formal framework, and we use JACK (www.agent-software.com.au) and Metric-FF [7] for our implementation. We impose certain language restrictions—e.g., we disallow the specification of recursive event-goals.

[3]Definite and possible effects are referred to by Clement et al. as respectively *must* and *may summary conditions* [2].

a standard classical precondition (with disjunction), whereas their precondition is (essentially) two sets of literals: those that must hold at the start of *any* successful execution of the event-goal, and those that must hold at the start of *one or more* successful executions of the event-goal.

3.2 Planning and Validation

When we wish to obtain an abstract plan, we provide the desired goal state, and the initial state, to our chosen classical planner, which contains a representation of each of our event-goals as an abstract operator.

Because abstract operators do not model the *possible* effects of event-goals, it may happen that when mapped back to event-goals and executed, such effects create a situation in which later event-goals (or abstract operators in the plan) are unable to execute successfully. For example, a possible effect may clobber a pre-condition of a later event-goal, as in the following. Consider the abstract plan $e_1 \cdot e_2$ for initial state $p \wedge r$ and goal state s, where: *(i)* the pre-condition of e_1 is p, the definite effects are q, and the possible effects are $\neg r$; and the pre-condition of e_2 is $q \wedge r$, and the definite effects are s. Observe that it is possible for the state after the completion of e_1 to be $p \wedge \neg r$, because the decomposition of e_1 may bring about the literal $\neg r$. If this happens, it will not be possible to execute the plan for e_2. For this reason, we say that abstract plan $e_1 \cdot e_2$ is *potentially incorrect*.

Because of this potential complication due to possible effects, it is necessary to validate the abstract plan that is obtained, to ensure that it is viable. In order to determine whether an abstract plan is potentially incorrect, we do a simple check to ascertain whether there is any literal in the precondition of some action a_i such that this literal is possibly clobbered in $a_1 \ldots a_{i-1}$. If an abstract plan is detected as being potentially incorrect, we use HTN planning, as described in [12], to further investigate to determine whether the plan is *definitely* incorrect (i.e., no plan decomposition exists). In the case that the HTN planner is unable to find a successful decomposition for the abstract plan, this plan is rejected, and a new plan is requested from the classical planner, which is similarly tested for validity. Maintaining information about possible effects allows us to do a simple (and fast) check for potential incorrectness, thus eliminating the need to check viable decomposition in cases which are not potentially incorrect.

4. IDEAL HYBRID-SOLUTIONS

So far, we have shown how to obtain (correct) hybrid-plans for a goal by planning with abstract operators, i.e., BDI event-goals. However, as illustrated in the introduction, such plans, while being correct, may contain redundancy. It is also the case that they may contain a collection of abstract operators which could potentially be combined into a single (more) abstract operator, thus improving the abstraction level of the hybrid-plan. As discussed in the introduction, we prefer more abstract hybrid-plans as they capture more of what has been called *user-intent*, and importantly, they support flexibility and robustness in execution. However, this preference must be balanced against non-redundancy, both at the level of primitive actions, and of unjustified abstract operators or event-goals.

In this section, we consider three inter-related concepts, and define these precisely in order to obtain an unambiguous description of an ideal hybrid-plan. These concepts we call *maximal-abstractness*, *minimality* and *non-redundancy*. Intuitively, a hybrid-plan is maximally-abstract if it cannot be obtained by a decomposition of some other hybrid-plan. Minimality is the requirement that one cannot remove (usually abstract) tasks from a non-redundant hybrid-solution to get another non-redundant hybrid-solution. A primitive plan σ is said to be *redundant* if one or more actions in σ can be removed to obtain a plan σ' that is still a solution for the planning problem in question [6, 9]. We say that a hybrid-solution h is non-redundant if some primitive solution produced by it, for the given initial state and goal state, is non-redundant.[4] We note that although non-redundancy is the more standard concept, minimality is a stronger notion, and, as we will illustrate shortly, it is possible to have a non-redundant hybrid-solution that is not minimal.

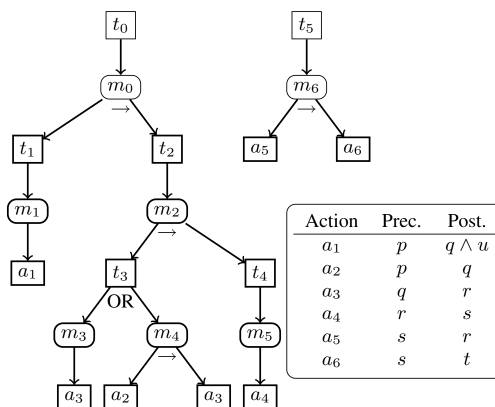

Figure 2: A simple totally-ordered HTN domain. Tasks are shown as rectangles and methods as rounded rectangles.

HYB-SOL	NON-RED	MIN	MAX-ABS	MNRMA
$h_0 = t_2 \cdot a_6$	√	√	√	√
$h_1 = t_0 \cdot t_5$	×	×	√	×
$h_2 = t_0 \cdot a_6$	√	√	√	√
$h_3 = t_1 \cdot t_2 \cdot a_6$	√	×	×	×
$h_4 = t_0 \cdot a_5 \cdot a_6$	×	×	×	×

Figure 3: Different hybrid-solutions and their properties.

Before defining these notions formally, we illustrate them using the HTN method library and the possible hybrid-solutions depicted in Figures 2 and 3, respectively. Take the initial state to be $\{p\}$ and the goal state to be $\{t\}$. Hybrid-solution h_3 is not maximally-abstract because it is a refinement of hybrid-solution h_2. Moreover, h_3 is not minimal either, as a proper subsequence of it, namely h_0, is a non-redundant hybrid-solution. This is despite the fact that h_3 is (weakly) non-redundant. Hybrid-solution $t_0 \cdot t_5$ is redundant because all of its primitive solutions, namely $a_1 \cdot a_3 \cdot a_4 \cdot a_5 \cdot a_6$ and $a_1 \cdot a_2 \cdot a_3 \cdot a_4 \cdot a_5 \cdot a_6$, are indeed redundant: one can remove action a_5 without invalidating the plan. Finally, h_2 is maximally-abstract, since it is not a refinement of any other hybrid-plan—hybrid-plan $t_0 \cdot t_5$ cannot be refined in any way to get exactly $t_0 \cdot a_6$. It is not hard to check that h_2 is also non-redundant and minimal. That is, hybrid-plan h_2 conforms to all three basic notions. We call such solutions *minimal non-redundant maximally-abstract* hybrid-plans (MNRMA).

In what follows, we shall make these three notions precise. Before doing that, we note that it is generally convenient in HTN planning to make use of a distinguished *dummy* primitive task (which we shall refer to as the ϵ task), that is, a primitive task whose precondition is always true and postcondition is always empty—an ϵ task basically amounts to "doing nothing." Without loss of general-

[4]It would also be possible to define a stronger version of non-redundancy on hybrid-solutions, where *every* primitive solution produced by it is required to be non-redundant. However, we choose to use the weaker notion.

ity, we assume from now on, that all ϵ tasks mentioned in primitive plans in set $sol(d, \mathcal{I}, \mathcal{D})$ have been removed and that hybrid-plans do not mention any ϵ tasks.

To define non-redundancy, we first extend the notion of *perfect justification* for primitive solutions from [6]. A primitive solution σ for planning problem $\langle \mathcal{I}, \mathcal{G}, Op \rangle$ is a *perfect justification* if there does not exist a proper subsequence σ' of σ such that σ' is a primitive solution for $\langle \mathcal{I}, \mathcal{G}, Op \rangle$. Basically, a hybrid-solution is non-redundant if it can produce at least one perfect justification.

DEFINITION 1. (NON-REDUNDANT HYBRID-SOLUTIONS) Let $\mathcal{H} = \langle \mathcal{I}, \mathcal{G}, \mathcal{D} \rangle$, with $\mathcal{D} = \langle Op, Me \rangle$, be a hybrid planning problem. Then, h is a *non-redundant* hybrid-solution for $\mathcal{H}$ if there exists $\sigma \in sol(h, \mathcal{I}, \mathcal{D}) \cap sol(\mathcal{I}, \mathcal{G}, Op)$ such that σ is a perfect justification for problem $\langle \mathcal{I}, \mathcal{G}, Op \rangle$. ■

Let us next turn to the notion of *minimality*. Intuitively, we say that a non-redundant hybrid-solution h is minimal, if there is no substructure of h which gives the same result. More formally, $h = [s, \phi]$ is a *minimal* non-redundant hybrid-solution for a hybrid planning problem $\mathcal{H}$ if there does not exist a non-redundant hybrid-solution $h' = [s', \phi']$ for $\mathcal{H}$ such that $s' \subset s$, where ϕ' is obtained from ϕ by replacing with $true$ every (ordering) constraint that mentions some task label occurring in the set $s \setminus s'$. We do not define minimality and non-redundancy as independent concepts, as this can lead to a situation where there is no hybrid-solution that is both minimal and non-redundant at the same time, which is then problematic for our overall ideal definition. Instead we define minimality as a strengthening of non-redundancy.

Next, we focus on the third, and most involved, property of hybrid-solutions, namely, *maximal-abstractness*. Roughly speaking, a hybrid-solution is considered maximally-abstract if it cannot be obtained by a "refinement" of any other hybrid-solution. The *refinements* of a task network d is the set of all task networks that may be obtained by *reducing* d zero or more times.

DEFINITION 2. (REFINEMENTS) Let d and $\mathcal{D}$ be a task network and an HTN domain, respectively. The set of *refinements* of d relative to $\mathcal{D}$, denoted by $refn(d, \mathcal{D})$, is defined as $refn(d, \mathcal{D}) = refn^{\omega}(d, \mathcal{D}) = \bigcup_{n \in \mathbb{N}_0} refn^n(d, \mathcal{D})$, where

$$refn^0(d, \mathcal{D}) = \{d\}; \qquad refn^{n+1}(d, \mathcal{D}) = \bigcup_{d' \in refn^n(d, \mathcal{D})} red(d', \mathcal{D}).$$

■

It is important to note that, since a refinement of a task network is *any* intermediate task network encountered along a sequence of HTN reductions, there is no guarantee that a refinement will in the end produce a primitive plan solution. Furthermore, refinements in general do not account for hybrid-plans, as the former can contain variables and state constraints—a hybrid-plan is ground, and only contains *ordering constraints* among its tasks.

So, we say that a hybrid-plan h' is more abstract than a hybrid-plan h, if one can reach h from h' by refinements, and h' produces any of the solutions of interest that are produced by h.[5] The notion of "reaching h from h'" requires some technical care, as h is a ground hybrid-plan, whereas refinements may have variables. Therefore, we need to consider ground instances of refinements (condition 2 below). Furthermore, one needs to respect those constraints obtained from the refinement of h' (condition 3 below).

[5]It would also be possible to define a stronger notion in which h' ought to produce *all* of the solutions of interest produced by h.

DEFINITION 3. (MAXIMALLY-ABSTRACT) Let $\mathcal{D}$ be an HTN domain and $\mathcal{I}$ be a state. Let Δ be a set of hybrid-plans and $h = [s_h, \phi_h] \in \Delta$ be a hybrid-plan in it.

Hybrid-plan h is *maximally-abstract* among set Δ for a set of primitive plan solutions $\Sigma_h \subseteq sol(h, \mathcal{I}, \mathcal{D})$, if there is no hybrid-plan $h' = [s_{h'}, \phi_{h'}] \in \Delta$ with $|s_{h'}| < |s_h|$ such that:

1. $d_1 \in refn(h', \mathcal{D})$;
2. $d_2 = [s_{d_2}, \phi_{d_2}]$ is a ground instance and task label renaming of d_1 such that $s_{d_2} \supseteq s_h$;
3. $d_3 = [s_{d_2}, \phi_{d_2} \wedge \phi_h]$; and
4. $\Sigma_h \cap sol(d_3, \mathcal{I}, \mathcal{D}) \neq \emptyset$. ■

That is, a hybrid-plan h is maximally-abstract among hybrid-plans in set Δ for a set of primitive plan solutions Σ_h, if there is no shorter hybrid-plan h' in Δ that can produce h by a sequence of reductions without losing all of the primitive plan solutions in Σ_h.

By putting together the notions of minimality, non-redundancy, and that of maximal-abstraction for hybrid-plans,[6] we can now define what "ideal" hybrid-solutions are.

DEFINITION 4. (MNRMA HYBRID-PLANS) A hybrid-plan h is a *minimal non-redundant maximal-abstraction* (MNRMA) for a hybrid planning problem $\mathcal{H} = \langle \mathcal{I}, \mathcal{G}, \mathcal{D} \rangle$ if and only if

1. h is a minimal non-redundant hybrid-solution for $\mathcal{H}$; and
2. h is a maximally-abstract hybrid-plan among *all* possible hybrid-plans for the set $\Sigma \cap sol(h, \mathcal{I}, \mathcal{D})$, where Σ is the set of all perfect justifications for $\langle \mathcal{I}, \mathcal{G}, Op \rangle$.

The set of all MNRMA plans for $\mathcal{H}$ is denoted MNRMA($\mathcal{H}$). ■

That is, a hybrid-plan h is considered an MNRMA hybrid-plan if it is minimal, and hence non-redundant, and moreover, a maximally-abstract hybrid-plan within *(i)* the set of all possible hybrid-plans, and *(ii)* the set of all perfect justifications it is able to produce.

The following theorem states that, whenever a hybrid planning problem can be solved, there is, at least, one (optimal) MNRMA hybrid-plan.

THEOREM 1. *Let $\mathcal{H} = \langle \mathcal{I}, \mathcal{G}, \mathcal{D} \rangle$ be a hybrid planning problem. If $sol(\mathcal{I}, \mathcal{G}, \mathcal{D}) \neq \emptyset$, then there exists an* MNRMA *for $\mathcal{H}$.*

PROOF. (Sketch) Let $h \in sol(\mathcal{I}, \mathcal{G}, \mathcal{D})$ be a hybrid-solution, and let $\sigma \in sol(h, \mathcal{I}, \mathcal{D}) \cap sol(\mathcal{I}, \mathcal{G}, Op)$ be a primitive solution. If σ is not a perfect justification for $\langle \mathcal{I}, \mathcal{G}, \mathcal{D} \rangle$, then there must exist a subsequence σ' of σ that is a perfect justification for $\langle \mathcal{I}, \mathcal{G}, \mathcal{D} \rangle$. Observe that, since σ' contains only primitives, it will also trivially be minimal for $\langle \mathcal{I}, \mathcal{G}, \mathcal{D} \rangle$. If σ' is maximally-abstract (among all possible hybrid-plans, for $\{\sigma'\}$), then σ' is an MNRMA for $\mathcal{H}$; otherwise, there must exist another hybrid-plan h' that is a MNRMA for $\mathcal{H}$. □

Unfortunately, it is not clear how one could compute an MNRMA hybrid-plan for a hybrid planning problem, without considering all possible hybrid-plans.[7] Hence, in the next section, we shall develop, and show how to implement, a weaker notion than MNRMA that looks for the most "preferred" specialization of a fixed hybrid-solution.

[6]At this stage, we are only interested in these three properties, though, there may be other desirable properties one may look for.

[7]Technically, one would consider only shorter plans than the one at hand.

5. SPECIALIZING HYBRID-SOLUTIONS

Instead of searching for the global "best" hybrid-solution, we focus here on "*improving*" a given hybrid-solution, by extracting its non-redundant and most abstract part. As was shown in Section 3, the particular hybrid-solution that we start from may have been produced by a first principles planner operating in the BDI domain.

Concretely, the task we are interested in is as follows: *given a, possibly redundant, hybrid-solution h for hybrid planning problem $\mathcal{H}$, together with one of its full successful decompositions, obtain a most abstract specialization of h within the confines of the decomposition at hand so that a perfect justification may be obtained.*

Suppose, then, that we start with a particular hybrid-solution h for hybrid planning problem $\mathcal{H} = \langle \mathcal{I}, \mathcal{G}, \mathcal{D} \rangle$. Suppose, further, that $\lambda_h = d_0 \cdot d_1 \cdots d_n$ is a (complete) decomposition trace of h, where $d_0 = h$, and that $\sigma \in comp(d_n, \mathcal{I}, \mathcal{D}) \cap sol(\mathcal{I}, \mathcal{G}, Op)$ is a solution compatible with λ_h that achieves goal $\mathcal{G}$. In what follows, we aim to "specialize" hybrid-plan h *within its decomposition* λ_h, so as to extract a preferred hybrid-plan h', that is, a most abstract, non-redundant specialization within the decomposition in question.

In order to develop an account of what a preferred specialization for a hybrid-plan is, we will define two basic notions. The first is what we call a *decomposition tree*, and the second is a *cut* in this tree. Intuitively, a decomposition tree is a tree induced by reductions on a task network, as shown in Figure 4, and a *cut* is a set of nodes in this tree. We are interested in finding a most abstract cut whose decomposition yields a non-redundant primitive solution.

In general, our preferred hybrid-plan, given an initial hybrid-plan and a decomposition tree, will not be an (ideal) MNRMA hybrid-plan as defined in the previous section. However, if an MNRMA hybrid-plan is mentioned in the decomposition tree given, then the plan will also be a preferred hybrid-plan.

It is straightforward to see that a sequence of reductions on a task network implicitly induces a decomposition tree of task nodes, as in Figure 4. A decomposition tree shows the structure of a particular refinement, by depicting how each compound task node is decomposed into lower-level tasks (the children) by applying a particular method. Observe that the constraints used for a node reduction are kept in the node itself. Technically, a <u>*decomposition tree*</u> $\mathcal{T}$ of task network $d = [s_d, \phi_d]$ relative to domain $\mathcal{D}$ is composed of AND-branches, where

1. each node u in $\mathcal{T}$ is of the form $\langle n, t, \phi \rangle$, where n is a unique label in the tree, t is a ground domain task or ϵ, and ϕ is a ground constraint formula;
2. the root of $\mathcal{T}$ is node $u = \langle 0, \epsilon, \phi_d\theta \rangle$ and $children(u, \mathcal{T}) = \{\langle n, t, \phi \rangle \mid (n : t) \in s_d\theta\}$;
3. if $u = \langle n, t, \phi\theta' \rangle$ is an internal non-root node in $\mathcal{T}$ such that $children(u, \mathcal{T}) = \{u_1\theta, \ldots, u_n\theta\}$, with $u_i = \langle n_i, t_i, \phi_i \rangle$, then there exists a reduction $[s, \phi] \in red([\{(n : t)\}, true], \mathcal{D})$ of t where $s = \bigcup_{i \leq n} \{(n_i : t_i)\}$; and
4. if $u = \langle n, t, \phi \rangle$ is a leaf node, then $\phi = true$.

Observe that nodes represent compound or primitive tasks, or are ϵ-nodes representing the root or dummy primitive tasks. It is not difficult to see that different decomposition traces may induce the same decomposition tree, up to task label renaming—decomposition trees are agnostic on *when* tasks are reduced. Lastly, note that trees are always *ground*, and that they can always be constructed with arbitrary labels for the tasks (except for the root's children, specified in d).

We say that a decomposition tree is *complete* if there is no leaf node representing a compound task. Notice that in a complete decomposition tree, a leaf node can correspond to a "normal" primitive task (i.e., action), or to a *dummy* primitive task of the form $\langle n, \epsilon, true \rangle$. We extract the set of actions in a decomposition tree as follows:

$$actions(\mathcal{T}) = \{(n : t) \mid \langle n, t, \phi \rangle \in leaves(\mathcal{T}), t \neq \epsilon\}.$$

A decomposition tree per se does not specify any ordering among tasks, and in particular, any ordering among primitive task leaf nodes. To this end, we define a *full decomposition tree*. But first, we define a <u>*completion*</u> of a complete decomposition tree $\mathcal{T}$ as a linearisation of the set $leaves(\mathcal{T})$, that is, a linear labeled primitive plan τ built from exactly the leaves in $\mathcal{T}$. For example, a completion of the tree in Figure 4 is $(2 : a_1) \cdot (3 : a_2) \cdot (6 : a_3) \cdot (7 : a_4) \cdot (9 : a_5) \cdot (12 : a_7) \cdot (10 : a_6) \cdot (13 : a_8) \cdot (15 : a_9) \cdot (17 : \epsilon)$. A <u>*full decomposition tree*</u>, then, is obtained by putting together a complete decomposition tree $\mathcal{T}$ and one of its completions τ, and is denoted $\mathcal{T}_\tau$. A full decomposition tree encodes not only how tasks are fully reduced to primitive tasks, but also how these are ordered to form a final linear plan.

Full decomposition trees, however, are still merely *syntactic* objects, and therefore independent of any (initial) state—they just describe legal syntactic ways of transforming tasks into other tasks with respect to the method library. Given a full decomposition tree $\mathcal{T}_\tau$ and an initial state $\mathcal{I}$, we say that $\mathcal{T}_\tau$ is <u>*executable*</u> in $\mathcal{I}$ if *(i)* the decomposition tree is legal in $\mathcal{I}$, that is, all the constraints along the decomposition tree are satisfied relative to τ; and *(ii)* the labeled primitive plan τ is executable in $\mathcal{I}$ (relative to the action domain Op). We write $\langle \mathcal{T}_\tau, \mathcal{I} \rangle \models \phi$ (or just $\mathcal{T}_\tau \models \phi$ if there are no state constraints in ϕ and therefore $\mathcal{I}$ is irrelevant) to denote that the formula ϕ of constraints holds under the full decomposition tree $\mathcal{T}_\tau$.[8]

A *cut* in a (full) decomposition tree, is then a set of nodes which (together with the necessary constraints) can form a hybrid-plan. Formally, a <u>*cut*</u> in a decomposition tree $\mathcal{T}$ is a set of task nodes π in $\mathcal{T}$ such that for all $u, u' \in \pi$, with $u \neq u'$, it is the case that $(descendants(u, \mathcal{T}) \cup \{u\}) \cap (descendants(u', \mathcal{T}) \cup \{u'\}) = \emptyset$. For example, $\{\langle 1, t_1, 2 \prec 3 \rangle, \langle 4, t_2, 5 \prec 8 \wedge 5 \prec 11 \rangle\}$ in Figure 4 is a legal cut, but $\{\langle 4, t_2, 5 \prec 8 \wedge 5 \prec 11 \rangle, \langle 5, t_3, 6 \prec 7 \rangle\}$ is not. In turn, a cut π in a decomposition tree $\mathcal{T}$ *induces* a new decomposition tree, denoted $\mathcal{T}|_\pi$, by projecting only on those nodes in π, and trivially adding a node $\langle root, \epsilon, true \rangle$ as root with π as its children. The projection operation trivially generalizes to full decomposition trees, denoted $\mathcal{T}_\tau|_\pi$, by projecting in τ only the primitive tasks that are leaf nodes in $\mathcal{T}|_\pi$, that is, $\mathcal{T}_\tau|_\pi = \mathcal{T}'_{\tau'}$ where $\mathcal{T}' = \mathcal{T}|_\pi$ and $\tau' = \tau|_{leaves(\mathcal{T}')}$.

We are now in a position to start defining our *preferred* hybrid-plan, relative to a given hybrid-plan and a full decomposition tree. Our preferred hybrid-plans are non-redundant while preserving their abstraction as much as possible. Guided by the notion of maximal-abstraction, some cuts are more abstract than others. Given two cuts π' and π in a decomposition tree $\mathcal{T}$, we say that π' <u>*dominates*</u> π if $\pi \subseteq \bigcup_{u \in \pi'} descendants(u, \mathcal{T}) \cup \pi'$, and $actions(\mathcal{T}|_{\pi'}) = actions(\mathcal{T}|_\pi)$. That is, π' produces, at least, π without introducing new primitive actions in the end—whatever is produced besides π does not lead to any actions. For instance, cut $\pi_1 = \{\langle 4, t_2, 5 \prec 8 \wedge 5 \prec 11 \rangle\}$ dominates cut $\pi_2 = \{\langle 5, t_3, 6 \prec 7 \rangle, \langle 8, t_4, 9 \prec 12 \rangle, \langle 11, t_5, 10 \prec 13 \rangle\}$ in Figure 4.

Finally, we can use cuts to define what the preferred specializations of hybrid-plans are. Let $decsol(h, \mathcal{H})$ be the set of full decomposition trees $\mathcal{T}_\tau$ of hybrid-plan h, that are executable in $\mathcal{I}$,

[8] Notice each constraint in a full decomposition tree is either true or false given the initial situation, while for decomposition trees alone this is not necessarily the case, as the satisfaction of the constraints generally depends on the final ordering of the leaf primitive tasks.

and such that $\tau \in sol(\mathcal{I}, \mathcal{G}, Op)$ (i.e., the completion achieves the goal). Also, when π is a cut in a full decomposition tree $\mathcal{T}_\tau$, we define the set of labeled tasks in π, and the ordering constraints on π that are implied by the full decomposition tree as follows:

$$\hat{\pi} = \{(n : t) \mid \langle n, t, \psi \rangle \in \pi, \text{ for some constraint formula } \psi\};$$

$$\Phi[\mathcal{T}_\tau, \pi] = \bigwedge \{n_1 \prec n_2 \mid \langle n_1, t_1, \phi_1 \rangle, \langle n_2, t_2, \phi_2 \rangle \in \pi, \mathcal{T}_\tau \models n_1 \prec n_2\}.$$

A preferred specialization of a hybrid-plan h is a most abstract non-redundant hybrid-plan that can be extracted from a given decomposition tree of h.

DEFINITION 5. (PREFERRED SPECIALIZATION) Let h be a hybrid-solution for a hybrid planning problem $\mathcal{H}$, and let $\mathcal{T}_\tau \in$ *decsol*$(h, \mathcal{H})$. Then, h' is a *preferred specialization* of h within $\mathcal{T}_\tau$ for $\mathcal{H}$ if $h' = [\hat{\pi}, \Phi[\mathcal{T}_\tau, \pi]]$ for some cut π in $\mathcal{T}$ such that:

1. $\tau|_{actions(\mathcal{T}|_\pi)}$ is a perfect justification for $\langle \mathcal{I}, \mathcal{G}, Op \rangle$;
2. the projected full decomposition tree $\mathcal{T}_\tau|_\pi$ is executable in $\mathcal{I}$;
3. there is no cut π', with $|\pi'| < |\pi|$, that dominates π and such that $\mathcal{T}_\tau|_{\pi'}$ is executable in $\mathcal{I}$. ■

In words, cut π is a selection of, possibly compound, non-ϵ tasks legally yielding a non-redundant primitive solution for the desired goal within the decomposition at hand, and in addition, is as abstract as possible within the decomposition. A preferred specialization is then built by taking such a cut and adding the ordering constraints implied on the cut by the decomposition.

It is clear that, by definition, the preferred specialization h is non-redundant for the goal, and that there is no other *more abstract* hybrid-plan h' within the same decomposition that achieves the goal without redundancy. Notice, however, that preferred specializations may not be minimal within the decomposition. A *minimal* preferred specialization, thus, is one for which no subset is also a preferred specialization.

The following result guarantees that there is always a preferred specialization of a hybrid-solution.

THEOREM 2. *Let $\mathcal{H}$ be a hybrid planning problem, and let h be a hybrid-plan. If $\mathcal{T}_\tau \in$ decsol$(h, \mathcal{H})$, then there exists at least one preferred specialization of h within $\mathcal{T}_\tau$ for $\mathcal{H}$.*

PROOF. (Sketch) Since $\mathcal{T}_\tau \in$ *decsol*$(h, \mathcal{H})$, there must exist a subsequence τ' of τ that is a perfect justification for $\mathcal{C} = \langle \mathcal{I}, \mathcal{G}, Op \rangle$. Take the cut π as the set of labeled tasks occuring in τ'. Clearly, $\tau|_{actions(\mathcal{T}|_\pi)} = \tau'$ is a perfect justification for $\mathcal{C}$. Also, since $\mathcal{T}_\tau|_\pi$ is actually τ' and τ' solves $\mathcal{C}$, tree $\mathcal{T}_\tau|_\pi$ ought to be executable in $\mathcal{I}$. So, if there is no cut in $\mathcal{T}$ that dominates π while inducing an executable projected tree, then primitive plan $[\hat{\pi}, \Phi[\mathcal{T}_\tau, \pi]]$ is indeed a preferred specialization. Otherwise, if such a cut π' does exist, then it can be proved that the first two conditions above hold for π', which means that either $[\hat{\pi'}, \Phi[\mathcal{T}_\tau, \pi']]$ is a preferred specialization, or there is another cut π'' that dominates π while inducing an executable projected tree. This reasoning can be followed all the way to the top of tree $\mathcal{T}$, where no dominating cut may exist and plan h itself would be the preferred specialization. □

Of course, since the dominance relation among cuts is not total, and since there may exist more than one perfect justification that can be extracted from a completion, there may actually be more than one preferred specialization.

Recall that our ideal MNRMA hybrid-plan (Definition 4) essentially defined a most abstract non-redundant hybrid-solution *among*

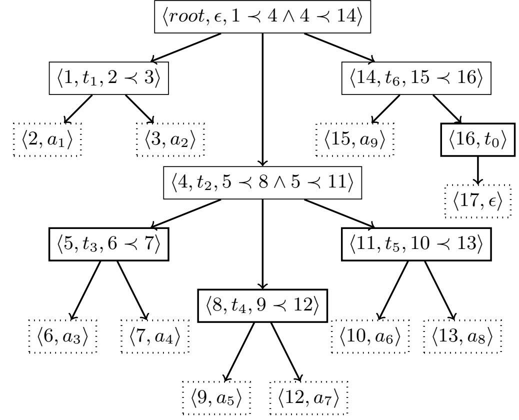

Figure 4: A complete decomposition tree $\mathcal{T}$ of task network $d = [\{(1 : t_1), (4 : t_2), (14 : t_6)\}, (1 \prec 4) \wedge (4 \prec 14)]$. Dotted rectangles stand for primitive actions or ϵ tasks (node $\langle 17, \epsilon \rangle$); missing constraints stand for $true$.

all conceivable hybrid-plans. Our preferred specialization, however, is limited to what can be extracted from a given decomposition of an existing hybrid-plan. Nonetheless, the following theorem states that if a hybrid-plan's full decomposition being considered does contain an MNRMA hybrid-plan that leads to a non-redundant primitive solution within the confines of the decomposition, then the MNRMA hybrid-plan is guaranteed to be a preferred specialization. In the definition below, set *decsol*$^{nr}(h, \mathcal{H})$ stands for the set of full decomposition trees $\mathcal{T}_\tau$ in *decsol*$(h, \mathcal{H})$ such that $\tau|_{actions(\mathcal{T})}$ is a perfect justification for $\mathcal{H}$. (Note that whenever a tree is in this set, so is any variation of it obtained by a consistent task label renaming.)

THEOREM 3. *Let h be a hybrid-solution for hybrid planning problem $\mathcal{H}$, and let $\mathcal{T}_\tau \in$ decsol$(h, \mathcal{H})$. Suppose that there exists a cut π in $\mathcal{T}_\tau$ such that $h_\pi = [\hat{\pi}, \Phi[\mathcal{T}_\tau, \pi]] \in$ MNRMA$(\mathcal{H})$ (that is, π represents an* MNRMA *solution for $\mathcal{H}$) and such that there is a decomposition in decsol$^{nr}(h_\pi, \mathcal{H})$ that is equivalent to $\mathcal{T}_\tau|_\pi$, modulo their root nodes. Then, hybrid-plan h_π is a preferred specialization of h within $\mathcal{T}_\tau$ for $\mathcal{H}$.*

PROOF. (Sketch) Since $h_\pi \in$ MNRMA$(\mathcal{H})$, there cannot be a cut π' in $\mathcal{T}_\tau$, with $|\pi'| < |\pi|$, such that π' dominates π while inducing an executable decomposition tree; otherwise, π' (or some other cut π'' in $\mathcal{T}_\tau$, with $|\pi''| < |\pi'|$, that induces an executable decomposition tree) will represent an MNRMA solution for $\mathcal{H}$, and π will not. Also, since there is a decomposition tree in *decsol*$^{nr}(h_\pi, \mathcal{H})$ that is equivalent to $\mathcal{T}_\tau|_\pi$ (modulo their root nodes), it is easy to see that *(i)* $\mathcal{T}_\tau|_\pi$ is executable in $\mathcal{I}$; and *(ii)* $\tau|_{actions(\mathcal{T}|_\pi)}$ is a perfect justification for $\mathcal{H}$. From this it follows that h_π is a preferred specialization of h within $\mathcal{T}_\tau$ for $\mathcal{H}$. □

Let us now explain how preferred specializations can be extracted from a given decomposition of a hybrid-solution.

5.1 Computing Preferred Specializations

Algorithm 1 computes a preferred specialization of a hybrid-solution h for a hybrid planning problem $\mathcal{H}$. In a nutshell, the algorithm works bottom-up, starting at the leaf-level with a labeled primitive perfect justification plan τ' (line 1), and repetitively abstracting out one or more steps into a higher-level more abstract step (lines 3-9). Once no more abstractions are possible, the corresponding constraints entailed by the decomposition tree for the final steps are calculated (step 11) and the final hybrid-plan returned.

Algorithm 1 Find-Preferred-Specialization$(h, \mathcal{H}, \mathcal{T}_\tau)$

Require: Hybrid-sol. h, hybrid problem $\mathcal{H}$, $\mathcal{T}_\tau \in decsol(h, \mathcal{H})$.
Ensure: A preferred specialization of h within $\mathcal{T}_\tau$ for $\mathcal{H}$.

1: $\tau' \Leftarrow$ Get-Perfect-Justification$(\tau, \mathcal{H})$ // As in [6]; ignore ϵ tasks
2: $\pi = \{\langle n, t, \phi\rangle \mid (n : t) \in \tau', \langle n, t, \phi\rangle \in \mathcal{T}_\tau\}$
3: **for** $\ell \Leftarrow 1$ **to** *height*$(\mathcal{T}_\tau) - 1$ **do** // Leafs at level 0
4: **for** each node u at level ℓ in tree $\mathcal{T}$ **do**
5: **if** *children*$(u, \mathcal{T}) \subseteq \pi$ and $\langle \mathcal{T}_\tau|_\pi, \mathcal{I}\rangle \models u_\phi$ **then**
6: $\pi \Leftarrow (\pi \setminus children(u, \mathcal{T})) \cup \{u\}$
7: **end if**
8: **end for**
9: **end for**
10: $\pi \Leftarrow \pi \setminus \Delta \Leftarrow \{u \mid u \in \pi$, all leafs of $\mathcal{T}|_{\{u\}}$ are ϵ nodes$\}$
11: $\phi \Leftarrow \Phi[\mathcal{T}_\tau, \pi]$ // As defined just before Definition 5
12: Return $[\hat{\pi}, \phi]$

At any point in time, the algorithm maintains a "current" *cut* π, which, initially, is a perfect justification without ϵ tasks removed. In line 4, a node u in the tree is selected for abstraction. If all the children of u are part of the current cut and the constraints required to decompose u into its children are indeed satisfied (line 5), then all the children of u are abstracted out into node u itself (line 6). It is not hard to see that this abstraction process can be carried on bottom-up, by performing the abstraction of all nodes at level k before abstracting to nodes at level $k + 1$.

Finally, the rationale behind line 10 is that we do not want the final hybrid-plan to include compound tasks not contributing at all to the perfect justification. An example of such a compound task is node $\langle 16, t_0\rangle$ in Figure 4, which produces the ϵ node $\langle 17, \epsilon\rangle$.

To illustrate how the algorithm works, suppose $\tau' = (2 : a_1) \cdot (9 : a_5) \cdot (10 : a_6) \cdot (12 : a_7) \cdot (13 : a_8) \cdot (15 : a_9) \cdot (17 : \epsilon)$—that is, actions $(3 : a_2)$, $(6 : a_3)$, and $(7 : a_4)$ are redundant for achieving the goal. In such a case, the computed (preferred) hybrid-plan specialization would be $h = [s, \phi]$, where:

$$s = \{(2 : a_1), (8 : t_4), (11 : t5), (14 : t_6)\};$$
$$\phi = 2 \prec 8 \wedge 2 \prec 11 \wedge 8 \prec 14 \wedge 11 \prec 14 \wedge 2 \prec 14.$$

Note that this is a partial-order plan, as the execution of compound tasks 8 and 11 may be interleaved (and, in fact, they are in $\mathcal{T}_\tau$).

It is not hard to see that Algorithm 1, once a perfect primitive plan justification is obtained (see [6]), runs in polynomial time on the size of the decomposition tree $\mathcal{T}$. More importantly, the algorithm can be proved correct with respect to Definition 5. In fact, it computes not just any preferred specializations, but *minimal* ones.

THEOREM 4. *Under the assumptions on the input, Algorithm 1 always terminates and returns a minimal preferred specialization of hybrid-solution h within $\mathcal{T}_\tau$ for $\mathcal{H}$.*

PROOF. (Sketch) Termination (of loops in lines 3 and 4) follows trivially by the fact that the tree (and its height) is finite. Minimality is due to line 10 in the algorithm, in which tasks that do not contribute to any action in the perfect justification are removed. □

6. DISCUSSION

In this paper, we have presented an approach to obtaining new abstract plans in a BDI system, that is, plans that were not given as initial procedural knowledge. This ability substantially increases the autonomy of a BDI agent. If, for instance, there is no plan applicable to achieve a high priority goal, then our agents may consider synthesizing a new plan to achieve the goal, or the context conditions of some relevant plans, thus, producing an applicable option. Unlike previous work on classical planning in BDI agents, we focus on producing *abstract* plans for *hybrid* planning problems, which considers both goal states to be achieved and the high-level plans already programmed. This has the advantage that, like the usual hierarchical plans of a BDI agent, these new abstract plans will be more robust for execution in a dynamic environment. It also has the advantage of ensuring that, to the extent possible, things are done in the way that has been programmed for the application — a notion which Kambhampati et al. [8] refers to as "*user-intent*."

The work of Kambhampati et al. [8] is the closest to ours and is indeed motivated by the desire to combine HTN and classical planning. Our work is different in that we extract the abstract operators from a BDI system, and then also execute the resulting hybrid-plan within this framework. There are also differences in the details of the approach; for example, they require the programmer to provide effects, whereas we compute these automatically. Most importantly, however, they do not address the issue of the balance between abstraction and redundancy, which we explore in depth.

In bringing classical planning into BDI hierarchical structures, we explored the tension between *abstraction* and a desire for *non-redundancy*. We first defined an "ideal" account of abstract plans which we called *minimal non-redundant maximally-abstract* (MNRMA). As searching for such plans would require, in principle, comparison among all possible plans, we developed a less ambitious approach, and an algorithm for it, in which a given candidate solution—maybe obtained using a classical planner—can be "improved" giving then what we call a *preferred* hybrid-plan.

7. REFERENCES

[1] S. S. Benfield, J. Hendrickson, and D. Galanti. Making a strong business case for multiagent technology. In *Proc. of AAMAS'06*, pages 10–15, 2006.

[2] B. J. Clement, E. H. Durfee, and A. C. Barrett. Abstract reasoning for planning and coordination. *Journal of Artificial Intelligence Research*, 28:453–515, 2007.

[3] O. Despouys and F. F. Ingrand. Propice-Plan: Toward a unified framework for planning and execution. In *Proc. of European Conference on Planning*, pages 278–293, 1999.

[4] J. Dix, H. Muñoz-Avila, D. S. Nau, and L. Zhang. IMPACTing SHOP: Putting an AI planner into a multi-agent environment. *Annals of Mathematics and Artificial Intelligence*, 37(4):381–407, 2003.

[5] K. Erol, J. A. Hendler, and D. S. Nau. Complexity results for HTN planning. *Annals of Mathematics and Artificial Intelligence*, 18(1):69–93, 1996.

[6] E. Fink and Q. Yang. Formalizing plan justifications. In *Proc. of the Ninth Conference of the Canadian Society for Computational Studies of Intelligence*, pages 9–14, 1992.

[7] J. Hoffmann. The Metric-FF planning system: Translating "ignoring delete lists" to numeric state variables. *Journal of Artificial Intelligence Research*, 20:291–341, 2003.

[8] S. Kambhampati, A. D. Mali, and B. Srivastava. Hybrid planning for partially hierarchical domains. In *Proc. of AAAI*, pages 882-888,1998.

[9] C. A. Knoblock. Automatically generating abstractions for planning. *Artificial Intelligence*, 68(2):243–302, 1994.

[10] F. R. Meneguzzi, A. F. Zorzo, and M. D. C. Móra. Propositional planning in BDI agents. In *Proc. of the ACM Symposium on Applied Computing*, pages 58–63, 2004.

[11] S. Minton, J. Bresina, and M. Drummond. Total-order and partial-order planning: A comparative analysis. *Journal of Artificial Intelligence Research*, 2:227–262, 1994.

[12] S. Sardina, L. de Silva, and L. Padgham. Hierarchical planning in BDI agent programming languages: a formal approach. In *Proc. of AAMAS'06*, pages 1001–1008, 2006.

[13] D. E. Wilkins, K. L. Myers, J. D. Lowrance, and L. P. Wesley. Planning and reacting in uncertain and dynamic environments. *Journal of Experimental and Theoretical AI*, 7(1):197–227, 1995.