et al. (2010) where "excuses" are being investigated as potential explanations for when a problem has no solutions.

Our approach tries to handle the case of when the problem is too difficult to solve within tight time constraints. For example, in robotics, algorithms often need to be run within refresh rates of several Hertz giving the algorithm only fractions of a second to give an updated result. Since planning is at least EXPSPACE-hard for HTN using complex representation (Erol *et al.* 1994), computing only the first plan level of a hierarchical domain is much easier in relation to the complete problem.

While abstract plans are not complete solutions, they still display a useful set of properties for various applications. The most immediate application is for explainable planning (Fox *et al.* 2017; Seegebarth *et al.* 2012). Indeed a high-level plan is more concise and does not contain unnecessary implementation details that would confuse a non-expert.

Another potential application for such plans are relative to domains that works with approximative data. Our main example here is intent recognition which was the original motivation for this work. Planners are not meant to solve intent recognition problems. However, several works extended what is called in psychology the *theory of mind*. That theory is the equivalent of asking "*what would I do if I was them?*" when observing the behavior of other agents. This leads to new ways to use *inverted planning* as an inference tool. One of the first to propose that idea was Baker *et al.* (2007) that use Bayesian planning to infer intentions. Ramirez and Geffner (2009) found an elegant way to transform a plan recognition problem into classical planning. This is done simply by encoding temporal constraints in the planning domain in a similar way as Baioletti *et al.* (1998) describe it to match the observed action sequence. A cost comparison will then give a probability of the goal to be pursued given the observations. Chen *et al.* (2013) extended this with multi-goal recognition. A new method, proposed by Sohrabi *et al.* (2016), makes the recognition fluent centric. It assigns costs to missing or noisy observed fluents, which allows finer details and less preprocessing work than action-based recognition. This method also uses a meta-goal that combines each possible goal and is realized when at least one of these goals is satisfied. Sohrabi *et al.* state that the quality of the recognition is directly linked to the quality and domain coverage of the generated plans. Thus guided diverse planning[1] was preferred along with the ability to infer several probable goals at once.

Related Works

HTN is often combined with classical approaches since it allows for a more natural expression of domains making expert knowledge easier to encode. These kinds of planners are called **decompositional planners** when no initial plan is provided (Fox 1997). Most of the time the integration of HTN simply consists in calling another algorithm when introducing a composite operator during the planning process. In the case of the DUET planner by Gerevini *et al.* (2008), it is done by calling an instance of an HTN planner based on task insertion called SHOP2 (Nau *et al.* 2003) to deal with composite actions. Some planners take the integration further by integrating the decomposition of composite action as a special step in their refinement process. Such works include the discourse generation oriented DPOCL (Young and Moore 1994) and the paper by Kambhampati *et al.* (1998) generalizing the practice for decompositional planners.

In our case, we chose a class of hierarchical planners based on Plan Space Planning (PSP) algorithms (Bechon *et al.* 2014; Dvorak *et al.* 2014; Bercher *et al.* 2014) as our reference approach. The main difference here is that the decomposition is integrated into the classical POCL algorithm by only adding new types of flaws. This allows to keep all the flexibility and properties of POCL while adding the expressivity and abstraction capabilities of HTN. We also made an improved planning framework based on the one used by HiPOP to reduce further the number of changes needed to handle composite actions and to increase the efficiency of the resulting implementation.

As stated previously, our goal is to obtain intermediary abstract plans and to evaluate their properties. Another work has already been done on another aspect of those types of plans. The Angelic algorithm by Marthi *et al.* (2007) exploited the usefulness of such plans in the planning process itself and used them as a heuristic guide. They also proved that, for a given fluent semantics, it is guaranteed that such abstract solutions can be refined into actual solutions. However, the Angelic planner does not address the inherent properties of such abstract plans as approximate solutions and uses a more restrictive totally ordered framework.

Definitions

In order to make the notations used in the paper more understandable we gathered them in table 1. For domain and problem representation, we use a custom knowledge description language that is inspired from RDF Turtle (Beckett and Berners-Lee 2011) and is based on triples and propositional logic. In that language quantifiers are used to quantify variables `*(x)` (forall x) but can also be simplified with an implicit form : `lost(~)` meaning *"nothing is lost"*. For reference the *exclusive quantifier* we introduced (noted ~) is used for the negation (e.g. `~(lost(_))` for *"something is not lost"*) as well as the symbol for nil. All symbols are defined as they are first used. If a symbol is used as a parameter and is referenced again in the same statement, it becomes a variable.

Domain

The domain specifies the allowed operators that can be used to plan and all the fluents they use as preconditions and effects.

Definition 1 (Domain)**.** A domain is a triplet $\mathcal{D} = \langle E_{\mathcal{D}}, R, A_{\mathcal{D}} \rangle$ where: * $E_{\mathcal{D}}$ is the set of **domain entities**. * R is the set of **relations** over $E_{\mathcal{D}}^n$. These relations are akin to n-ary predicates in first order logic. * $A_{\mathcal{D}}$ is the set of **operators** which are fully lifted *actions*.

[1] Diverse planning is set to find a set of m plans that are distant of d from one another.

Table 1: Our notations are adapted from Ghallab *et al.* (2004). The symbol ± demonstrates when the notation has signed variants.

Symbol	Description
$\mathcal{D}, \mathcal{P}$	Planning domain and problem.
$pre(a)$, $eff(a)$	Preconditions and effects of the action a.
$methods(a)$	Methods of the action a.
$\phi^{\pm}(l)$	Signed incidence function for partial order plans. ϕ^{-} gives the source and ϕ^{+} the target step of l. No sign gives a pair corresponding to link l.
$L^{\pm}(a)$	Set of incoming (L^{-}) and outgoing (L^{+}) links of step a. No sign gives all adjacent links.
$a_s \xrightarrow{c} a_t$	Link with source a_s, target a_t and cause c.
$causes(l)$	Gives the causes of a causal link l.
$a_a \succ a_s$	A step a_a is anterior to the step a_s.
A_x^n	Proper actions set of x down n levels.
$lv(x)$	Abstraction level of the entity x.
$a \rhd^{\pm} a'$	Transpose the links of action a onto a'.
$l \downarrow a$	Link l participates in the partial support of step a.
$\pi \Downarrow a$	Plan π fully supports a.
$\ddagger_f a$	Subgoal : Fluent f is not supported in step a.
$a_b \otimes l$	Threat : Breaker action a_b threatens causal link l.
$a \oplus^m$	Decomposition of composite action a using method m.
$var : exp$	The colon is a separator to be read as "such that".
$[exp]$	Iverson's brackets: 0 if $exp = false$, 1 otherwise.

Example: The example domain in listing 1 is inspired from the kitchen domain of Ramirez and Geffner (2010).

```
1 take(item) pre (taken(~), ?(item));
      //?(item) is used to make item into
      a variable.
2 take(item) eff (taken(item));
3 heat(thing) pre (~(hot(thing)),
      taken(thing));
4 heat(thing) eff (hot(thing));
5 pour(thing, into) pre (thing ~(in) into,
      taken(thing));
6 pour(thing, into) eff (thing in into);
7 put(utensil) pre (~(placed(utensil)),
      taken(utensil));
8 put(utensil) eff (placed(utensil),
      ~(taken(utensil)));
9 infuse(extract, liquid, container) ::
      Action; //Composite action of level 1
10 make(drink) :: Action; // Level 2
      containing infuse
```

Listing 1: Domain file used in our planner. In order to be concise, the methods are omitted.

Definition 2 (Fluent)**.** A fluent f is a parameterized statement $r(arg_1, arg_2, ..., arg_n)$ where:

- $r \in R$ is a relation/function holding a property of the world.
- $arg_{i \in [1,n]} \in E_{\mathcal{D}}$ are the arguments (possibly quantified).
- $n = |r|$ is the arity of r.

Fluents are signed. Negative fluents are noted $\neg f$ and behave as a logical complement. The quantifiers are affected by the sign of the fluents. We do not use the closed world hypothesis: fluents are only satisfied when another compatible fluent is provided. Sets of fluents have a boolean value that equals the conjunction of all its fluents.

Example: To describe an item not being held we use the fluent $\neg taken(item)$. If the cup contains water, $in(water, cup)$ is true.

Plan and hierarchical representation

Definition 3 (Partial Plan / Method)**.** A partially ordered plan is an *acyclic* directed graph $\pi = (S, L)$, with:

- S the set of **steps** of the plan as vertices. A step is an action belonging in the plan. S must contain an initial step I_π and goal step G_π.
- L the set of **causal links** of the plan as edges. We note $l = a_s \xrightarrow{c} a_t$ the link between its source a_s and its target a_t caused by the set of fluents c. If $c = \varnothing$ then the link is used as an ordering constraint.

In our framework, *ordering constraints* are defined as the transitive cover of causal links over the set of steps. We note ordering constraints: $a_a \succ a_s$, with a_a being *anterior* to its *successor* a_s. Ordering constraints cannot form cycles, meaning that the steps must be different and that the successor cannot also be anterior to its anterior steps: $a_a \neq a_s \wedge a_s \not\succ a_a$. In all plans, the initial and goal steps have their order guaranteed: $I_\pi \succ G\pi \wedge \nexists a_x \in S_\pi : a_x \succ I_\pi \vee G_\pi \succ a_x$. If we need to enforce order, we simply add a link without specifying a cause. The use of graphs and implicit order constraints help to simplify the model while maintaining its properties.

The central notion of planning is operators. Instantiated operators are usually called *actions*. In our framework, actions can be partially instantiated. We use the term action for both lifted and grounded operators.

Definition 4 (Action)**.** An action is a parametrized tuple $a(args) = \langle name, pre, eff, methods \rangle$ where:

- *name* is the **name** of the action.
- *pre* and *eff* are sets of fluents that are respectively the **preconditions and the effects** of the action.
- *methods* is a set of **methods** (*partial order plans*) that decompose the action into smaller ones. Methods, and the methods of their enclosed actions, cannot contain the parent action.

Example: The precondition of the operator $take(item)$ is simply a single negative fluent noted $\neg taken(item)$ ensuring the variable $item$ is not already taken.

Composite actions are represented using methods. An action without methods is called *atomic*. It is of interest to note the divergence with classical HTN representation here since normally composite actions do not have preconditions nor effects. This is because in our case we will need to insert them into abstract plans.