Scrutable Robot Actions Using a Hierarchical Ontological Model

Martin Jedwabny, Pierre Bisquert, Madalina Croitoru

Scrutable robot actions using a hierarchical ontological model

Martin Jedwabny,[1] Pierre Bisquert,[1, 2] Madalina Croitoru[1]
[1]LIRMM, Inria, Univ Montpellier, CNRS, Montpellier, France
[2]IATE, INRAE, Institut Agro, Montpellier, France
martin.jedwabny@lirmm.fr, pierre.bisquert@inrae.fr,
madalina.croitoru@lirmm.fr

Abstract

We place ourselves in the context of representing knowledge inside the cognitive model of a robot that needs to reason about its actions. We propose a new ontological transformation system able to model different levels of knowledge granularity. This model will allow to unfold the sequences of actions the robot performs for better scrutability.

1 Introduction

In this paper, we present a formal approach to knowledge representation and reasoning in the setting of a robot making decisions about which action to perform. We propose a hierarchical structure to examine information about actions in depth, by unfolding actions into several layers of complexity. The intuition of this work relies on ideas from episodic memory theory [1], that organizes information in layers of detail retrievable at the right time, in the right granularity.

We propose a hierarchical information structure composed by two combinatorial structures that can be applied in an Artificial Intelligence (AI) planning [2] setting: (i) an ontology of fluent and action symbols, and (ii) a hierarchical structure encoding preconditions and effects for the various actions.

The proposed layered catalog of actions is influenced by work on conceptual graph-based ontological knowledge representation [3]. This work can be seen as a continuation of previous work on hierarchical conceptual graphs [4], with a focus on reasoning about action.

The salient points of the paper are the following:

- A knowledge representation model relevant for ontologies of fluents and actions, and
- A formal method for translating between the different layers of details corresponding to these ontologies.

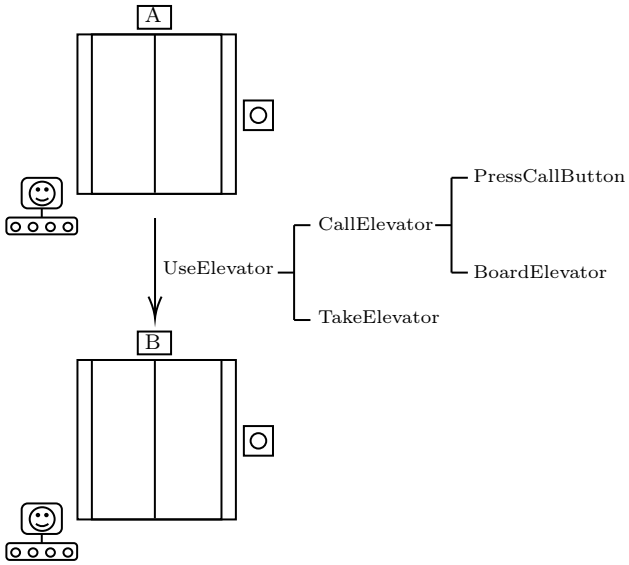

Figure 1: Elevator example illustration

After providing a motivating example in Section 2, we recall in Section 3 the necessary notions for graphical knowledge representation. In Section 4 we introduce the structures that allow to represent the robot's knowledge about the world and its possible actions. Section 5 analyses the related work and discusses ways in which our framework can be used to make a planner more scrutable for an user. Finally, we will conclude in Section 6.

2 Motivating Example

Consider a situation in which a robot has to board an elevator in a building to get from floor f_A to f_B. An action model of the situation might represent this as a single action $UseElevator$ that, upon execution, takes a situation in which the robot is in f_A and transforms it in such a way the robot goes to f_B. Modeling the action of using the elevator as a single atomic step comes with the immediate advantage of compactness.

However, in certain cases the elevator could malfunction due to electricity shortages, a certain button not working, or any other internal or external factor that disturbs its normal functioning. In this case, the robot could perceive that the action $UseElevator$ is no longer producing its intended result, but would have a hard time realizing the cause without modeling the state of the electricity in the building. On the other hand, keeping track of this property might be irrelevant to actions other than $UseElevator$ and make it harder to a user to understand the model and sequence of actions chosen by the robot if many other properties are introduced.

Upon closer inspection, we realize that the action $UseElevator$ depends on many factors like the state of electricity, the buttons used to operate the elevator and the electronic circuit that receives the button's signal. Furthermore, the action can be broken down to several steps, as depicted in Figure 1: (i) $CallElevator$ using button b_A from whichever floor it is currently at to f_A, and (ii) $TakeElevator$ by selecting the destination floor f_B with selector button b_B. Then, (i) can also be separated into two actions: $PressCallButton$, and $BoardElevator$. In the same way, these actions can be iteratively broken down into several layers of increasing complexity.

Nevertheless, from a user point of view, all of these factors and step breakdown could be irrelevant in correct operation circumstances. Both for a monitoring user and a automatic planning robot, it can be useful to take these multiple levels of model complexity into account and use them only when they are needed.

3 Background notions

3.1 Bipartite graphs

A *bipartite graph* is a graph $G = (V_G, E_G)$ with the nodes set $V_G = V_F \cup V_A$, where V_F and V_A are finite disjoint sets, and each edge $e \in E_G$ is a two-element set $e = \{v_F, v_A\}$, where $v_F \in V_F$ and $v_A \in V_A$. Usually, a bipartite graph G is denoted as $G = (V_F, V_A, E_G)$. We call $G^{\emptyset}$ the empty bipartite graph without nodes and edges.

We make the informal observation that, later on in the paper, the node set V_F and V_A will be used to represent, respectively, the uninstanced fluent (i.e., the description of a scene) and action symbols of the planning setting. Edges in the bipartite graph will thus capture the preconditions and effects, in terms of fluents that compose an action.

Let $G = (V_F, V_A, E_G)$ be a bipartite graph. The number of edges incident to a node $v \in V(G)$ is the degree, $d_G(v)$, of the node v. If, for each $v_A \in V_A$ there is a linear order $e_1 = \{v_A, v_1\}, \ldots, e_k = \{v_A, v_k\}$ on the set of edges incident to v_A (where $k = d_g(v)$), then G is called an *ordered bipartite graph*. A simple way to express that G is ordered is to provide a labelling $l : E_G \mapsto \{1, \ldots, |V_F|\}$ with $l(\{v_A, w\})$ the index of the edge $\{v_A, w\}$ in the above ordering of the edges incident in G to v_A. l is called a *order labelling* of the edges of G. We denote an ordered bipartite graph by $G = (V_F, V_A, E_G, l)$. It will be useful for the next sections to note that the labelling l can be generalized into a collection of labellings of different kinds by extending its definition to $l : E_G \mapsto 2^{K \times \{1,\ldots,|V_F|\}} - \{\emptyset\}$ with K a predefined set of kinds, while still being trivial to construct an overall labelling $l' : E_G \mapsto \{1, \ldots, |V_F|\}$ from it.

For a vertex $v \in V_F \cup V_A$, the symbol $N_G(v)$ denotes its neighbors set, i.e. $N_G(v) = \{w \in V_F \cup V_A | \{v, w\} \in E_G\}$. Similarly, if $A \subseteq V_A \cup V_F$, the set of its neighbors is $N_G(A) = \cup_{v \in A} N_G(v) - A$. If G is an ordered bipartite graph, then for each $r \in V_A$, the symbol $N^i_G(r)$ denotes the i-th neighbor of r, i.e.

$v = N_G^i(r)$ if and only if $\{r, v\} \in E_G$ and $l(\{r, v\}) = i$.

Throughout this paper we use a particular type of subgraph of a bipartite graph: $G^1 = (V_F^1, V_A^1, E_G^1)$ is a subgraph of $G = (V_F, V_A, E_G)$ if $V_F^1 \subseteq V_F$, $V_A^1 \subseteq V_A$, $N_G(V_A^1) \subseteq V_F^1$ and $E_G^1 = \{ \{v, w\} \in E_G | v \in V_F^1, w \in V_A^1\}$. In other words, we require that the (ordered) set of all edges incident in G to a vertex from V_A^1 must appear in G^1. Therefore, a subgraph is completely specified by its vertex set. In particular, if $A \subseteq V_F$:

- The *subgraph spanned by A in G*, denoted as $\lceil A \rceil^G$, has $V_F(\lceil A \rceil^G) = A \cup N_G(N_G(A))$ and $V_A(\lceil A \rceil^G) = N_G(A)$.

- The *subgraph generated by A in G*, denoted as $\lfloor A \rfloor_G$, has $V_F(\lfloor A \rfloor_G) = A$ and $V_A(\lfloor A \rfloor_G) = \{v \in N_G(A) | N_G(v) \subseteq A\}$.

- For $A \subseteq V_A$, *the subgraph induced by A in G*, denoted $[A]_G$, has $V_F([A]_G) = N_G(A)$ and $V_A([A]_G) = A$.

Now that we can represent the links between actions and fluents as a graph, we will see in the following section how we can define the types of their arguments.

3.2 Planning

We will use an abstract representation inspired by the one in [5] to model belief state planning. This representation allows to model non-deterministic/belief states and deterministic actions with conditional effects, i.e., conformant planning.

It is assumed that one is given a set of *fluents* $F_\mathcal{P}$ and *actions* $A_\mathcal{P}$, denoting the properties of a problem and the actions the agent can perform, respectively. A *literal* is either $f \in F_\mathcal{P}$ a fluent, or $\neg f$ its negation. A *state* $s \subseteq F_\mathcal{P}$ is a set of fluents denoting what properties hold in a state. This takes into account the closed-world assumption, in that all fluents that are not in a state are assumed to be false. A *belief state* $B \subseteq 2^{F_\mathcal{P}}$ is a set of states.

An action $a \in A_\mathcal{P}$ is of the form $a = (pre(a), eff(a))$, where $pre(a)$ is a set of literals over $F_\mathcal{P}$ called the *preconditions*. Then, $eff(a) = \{eff_1(a), \ldots, eff_n(a)\}$ is a set of *effects* of the form $eff_i(a) = cond_i(a) \rightarrow post_i(a)$, where $cond_i(a)$ and $post_i(a)$ are sets of literals called *condition* and *postcondition*, respectively. We say that a set of literals L over $F_\mathcal{P}$ is compatible with a state $s \subseteq F_\mathcal{P}$, denoted $compat(L, s)$, when $\forall f \in F_\mathcal{P}$ it holds that $f \in L \Rightarrow f \in s$ and $\neg f \in L \Rightarrow f \notin s$. An action $a \in A_\mathcal{P}$ is applicable in state $s \subseteq F_\mathcal{P}$, denoted $applicable(a, s)$, if and only if $compat(pre(a), s)$ holds. Similarly, given a belief state $B \subseteq 2^{F_\mathcal{P}}$, $applicable(a, B)$ holds if and only if $\forall s \in B$ it is the case that $applicable(a, s)$ is true.

If an action a is applicable in state s, the successor state, i.e, the state that results from performing a in s, is defined as $succ(a, s) = (s - \{f \in F_\mathcal{P} : \exists i \text{ s.t. } \neg f \in post_i(a) \text{ and } compat(cond_i(a), s)\}) \cup \{f \in F_\mathcal{P} : \exists i \text{ s.t. } f \in post_i(a)$

and $compat(cond_i(a), s)\}$. The successor to a belief state $B \subseteq 2^{F_\mathcal{P}}$ is defined as $succ(a, B) = \bigcup_{s \in B} succ(a, s)$.

A sequence of actions $\pi = [a_0, a_1, \ldots, a_n]$ where $n \in \mathbb{N}_0$ and $a_i \in A_\mathcal{P}$ is applicable to belief state $B \subseteq 2^{F_\mathcal{P}}$, denoted $applicable(\pi, B)$ if and only if either $n = 0$, or otherwise $applicable(a_0, B)$ and $applicable([a_1, \ldots, a_n], succ(a_0, B))$ hold. If $applicable(\pi, B)$ does indeed hold, then $succ(\pi, B) = B$ if $n = 0$, otherwise $succ(\pi, B) = succ([a_1, \ldots, a_n], succ(a_0, B))$.

A *planning problem* is a tuple $\mathcal{P} = (F_\mathcal{P}, A_\mathcal{P}, I_\mathcal{P}, G_\mathcal{P})$:

- $F_\mathcal{P}$ is a set of fluents,
- $A_\mathcal{P}$ is a set of actions over $F_\mathcal{P}$,
- $I_\mathcal{P} \subseteq 2^{F_\mathcal{P}}$ is a non-empty set of states called the initial belief state, and
- $G_\mathcal{P} \subseteq 2^{F_\mathcal{P}}$ is a non-empty set of states called the goal.

A sequence π of actions is a *strong plan*, i.e., a solution that always reaches the goal, when $succ(\pi, I_\mathcal{P}) \subseteq G_\mathcal{P}$. Less restrictive, π is a *weak plan*, i.e., a sequence that sometimes reaches the goal, when $\exists s \in succ(\pi, I_\mathcal{P})$ such that $s \in G_\mathcal{P}$.

We refer the reader to [5, 6, 7] for results on the complexity and heuristics methods to generate strong and weak plans for a planning model such as the one described here.

4 Layered Catalog Graphs

Influenced by episodic memory theory [1], the knowledge of an agent is defined by two combinatorial structures.

- On one hand, we will consider an ontology of fluent and action symbols. The fluent symbols (represented by unary concepts) are organized in a type hierarchy. Similarly, the action symbols, represented by binary or n-ary relations are also organized in a relation hierarchy. The action symbols have a signature of their arguments, represented as fluent concept types.
- On the other hand, the agent will dispose of a hierarchical structure encoding preconditions and effects for various actions.

We only consider unary fluents for simplification purposes, however it is immediate to see how we could extend this work in order to consider fluents of any arity.

Let us note that the following definitions 1-7 are based on previous work on conceptual graphs [4], although adapted to represent fluents and actions instead of abstract concepts. Moreover, the framework presented in this paper could not be modeled as-is with the mentioned literature due to the way in which multiple fluents can relate to actions as preconditions and effects at the same time.

The ontology is defined as follows:

Definition 1. *An ontology is a 4-tuple* $S = (T_F, T_A, \mathcal{I}, *)$ *where:*

- T_F *is a finite partially ordered set (poset)* $(T_F, \leq)$ *of fluent types, defining a type hierarchy which has a greatest element* $\top_C$*, namely the universal type. In this specialization hierarchy,* $\forall x, y \in T_F$ *the symbolism* $x \leq y$ *is used to denote that* x *is a subtype of* y*.*

- T_A *is a finite set of action types partitioned into* k *posets* $(T_A^i, \leq)_{i=1,k}$ *of relation types of arity* i *(*$1 \leq i \leq k$*), where* k *is the maximum arity of an action type in* T_A*. Moreover, each action type of arity* i*,* $r \in T_A^i$*, has an associated signature* $\sigma(r) \in \underbrace{T_F \times \ldots \times T_F}_{i \text{ times}}$*, which specifies the maximum fluent type of each of its arguments. This means that if we use* $r(x_1, \ldots, x_i)$*, then* x_j *is a fluent with* $type(x_j) \leq \sigma(r)_j$ *(*$1 \leq j \leq i$*), where type is the function that returns the type of a fluent. The partial orders on relation types of the same arity must be signature-compatible, i.e., it must be such that* $\forall r_1, r_2 \in T_A^i \; r_1 \leq r_2 \Rightarrow \sigma(r_1) \leq \sigma(r_2)$*.*

- $\mathcal{I}$ *is a countable set of individual markers, used represent to given constants.*

- $*$ *is the generic marker to denote an unspecified fluent (with a known type).*

- *The sets* T_F*,* T_A*,* $\mathcal{I}$ *and* $\{*\}$ *are mutually disjoint and* $\mathcal{I} \cup \{*\}$ *is partially ordered by* $x \leq y$ *if and only if* $x = y$ *or* $y = *$*, for any given* $x, y \in \mathcal{I} \cup \{*\}$*.*

It is worth to notice that the generic marker '$*$' will be used in this framework to represent variables/un-instanced concepts in the arguments of fluents and actions later.

The following depicts an ontology for the situation described before:

Example 1 (Elevator continued)**.** We specify the types of fluents and actions as:

- $T_F = \{Button, Floor, \top_C\}$, the fluent types, where $Button, Floor \leq \top_C$ are the only subtypes,

- $T_A = \{UseElevator^T\}$, the action types, where $\sigma(UseElevator^T) = (Button, Button, Floor, Floor)$.

- $\mathcal{I} = \{b_A, b_B, f_A, f_B\}$, individuals.

As we can see, the ontology provides a support of types for fluents and actions. In this case, we represent a basic action type for pressing the elevator button that has a signature composed of elements of type $Button$ and of type $Floor$, corresponding to the arguments of such an action.

For the representation of the hierarchical knowledge of the agent we define a simple graphical catalog of actions (SC) with their preconditions and effects. The simple graph catalog is rendered hierarchical by a transformation called transitional description defined further in the paper. The result will be a layered graphical catalog of actions (LC) that the robot can access according to the need at hand.

A SC provides a semantic set of pointers to the ontology of fluents and action symbols.

Definition 2. *A simple graphical catalog of actions is a 3-tuple* $SC = [S, G, \lambda]$, *where:*

- $S = (T_F, T_A, \mathcal{I}, *)$ *is the ontological support,*
- $G = (V_F, V_A, E_G, l)$ *is an ordered bipartite graph, where* V_F *are called fluent symbols and* V_A *action symbols,*
- l *is a labelling that maps each edge* $\{v_A, v_F\} \in E_G$ *to a non-empty set of pairs* $(k, i) \in l(\{v_A, v_F\})$ *where* $i \in \{1, \ldots, d_G(v_A)\}$ *and* k *is either* pre^+, pre^-, $cond_j^+$, $cond_j^-$, $post_j^+$, *or* $post_j^-$, *with* $j \in \mathbb{N}_{>0}$, *denoting whether* v_F *is a positive or negative literal in the precondition,* j*th effect condition, or* j*th effect postcondition of* v_A.
- λ *is a labelling of the nodes of* G *with elements from the support* S*:*
  $\forall r \in V_A,\ \lambda(r) \in T_A^{n_A}$ *such that* $n_A \in \mathbb{N}_{>0}$*;*
  $\forall c \in V_F,\ \lambda(c) \in T_F \times \big(\mathcal{I} \cup \{*\}\big)$ *such that*
  if $c = N_G^i(r)$, $\lambda(r) = t_r$ *and* $\lambda(c) = (t_c, ref_c)$, *then* $t_c \leq \sigma(t_r)_i$.

Intuitively, λ allows to make the link between nodes in G that represent fluent and action symbols, and their respective types defined in S. Moreover, the labelling allows to denote that a fluent is a precondition and/or effect of an action. Let us depict the meaning of λ and l with the following example.

Example 2 (Elevator continued)**.** Having defined an ontological support $S = (T_F, T_A, \mathcal{I}, *)$, we can instantiate a simple graphical catalog of actions $SC = [S, G, \lambda]$ by specifying $G = (V_F, V_A, E_G, l)$ and λ as follows:

- $V_F = \{InFloorElevator,\ InFloorAgent,\ CanUseCallButton,$
  $CanUseSelectButton\}$, the set of fluent symbols, where
  $\lambda(InFloorElevator) = (*, Floor)$,
  $\lambda(InFloorAgent) = (*, Floor)$,
  $\lambda(CanUseCallButton) = (*, Button)$,
  $\lambda(CanUseSelectButton) = (*, Button)$.
- $V_A = \{UseElevator\}$, the set of action symbols, where $\lambda(UseElevator) = UseElevator^T$.
- $E_G = \{\{UseElevator, X\} : X \in \{InFloorElevator, InFloorAgent,$
  $CanUseCallButton, CanUseSelectButton\}\}$,

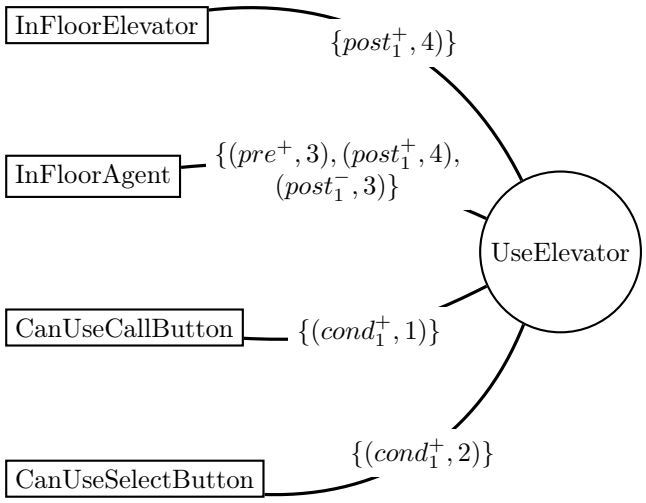

Figure 2: Bipartite graph for the action symbol $UseElevator$

- l the labelling assigns
  $l(\{UseElevator, CanUseCallButton\}) = \{(cond_1^+, 1)\}$,
  $l(\{UseElevator, CanUseSelectButton\}) = \{(cond_1^+, 2)\}$,
  $l(\{UseElevator, InFloorAgent\}) = \{(pre^+, 3), (post_1^+, 4), (post_1^-, 3)$,
  $l(\{UseElevator, InFloorElevator\}) = (post_1^+, 4)$.

We can see the SC $[S, G, \lambda]$ in Figure 2. The fluent symbols in V_F are mapped by λ to the generic marker '*' to denote that they refer to a variable and not a specific constant. In particular, the symbol $InFloorElevator$ represents the current floor of the elevator, $InFloorAgent$ the floor of the agent, $CanUseCallButton$ whether the agent can use the button to call the elevator, and $CanUseSelectButton$ the same for the button inside the elevator. The action symbol $UseElevator$ has type $UseElevator^T$ with signature $\sigma(UseElevator) = (Button, Button, Floor, Floor)$. This will be used to characterize the type of the arguments of the (instanced) action. Then, a label $l(\{UseElevator, v_F\}) = (k, i)$ denotes that the ith argument of $UseElevator$ corresponds to the (unique) argument of the fluent v_F, while k represents the role of the fluent in the action. As we will see later, such an action will get translated in our planning framework to something similar to an action $UseElevator(x, y, z, w) = (\{InFloorAgent(z)\}, \{cond_1 \rightarrow eff_1\})$, where $cond_1 = \{CanUseCallButton(x), CanUseSelectButton(y)\}$ and $eff_1 = \{InFloorAgent(w), \neg InFloorAgent(z), InFloorElevator(w)\}$.

Transitional descriptions will allow for the definition of layered catalogs, which represent the different levels of complexity in which we can break down the model. Briefly, they represent a single step of expansion of action and fluents into a new layer of complexity. A transitional description $\mathcal{TD}$ contains

a collection of bipartite graphs for each node of the original bipartite graph G that one wants to expand, which represent the new added knowledge. Let us now define what is a transitional description for a graph.

Definition 3. *Let $G = (V_F, V_A, E_G)$ be a bipartite graph. A transitional description associated to G is a pair $\mathcal{TD} = (D, (G.d)_{d \in D \cup N_G(D)})$ where*

- *$D \subseteq V_F$ is a set of complex fluent symbols, i.e: a subset of the original fluents symbols in V_F that can be expanded into a more complex representation.*
- *For each $d \in D \cup N_G(D)$ $G.d$ is a bipartite graph.*
- *If $d \in D$ then $G.d$ is the non-empty ($G.d \neq G^{\emptyset}$) description of the complex fluent symbol d. Distinct complex fluent symbols $d, d' \in D$ have disjoint descriptions $G.d \cap G.d' = G^{\emptyset}$.*
- *If $d \in N_G(D)$ then $G.d \neq G^{\emptyset}$, $N_G(d) - D \subseteq V_F(G.d)$ and $V_F(G.d) \cap V_F(G.d') \neq \emptyset$ if and only if $d' \in N_G(d) \cap D$.*

Note that $(G.d)_{d \in D \cup N_G(D)}$ is the collection of bipartite graphs for each node d of the original bipartite graph that we want to expand. Moreover, when one expands a fluent symbol, so do the action symbols it is related to. Conversely, when an action symbol is expanded, the nodes in its expanded graph are the same as those in the expansion of its related complex fluent symbols, or its non-complex neighbors.

Let us illustrate this definition as follows.

Example 3 (Elevator continued)**.** As we explained in Figure 1, we would like to be able to separate the action symbol $UseElevator$ into two: $CallElevator$ and $TakeElevator$. We can do this by using a transitional description $\mathcal{TD} = (D, (G.d)_{d \in D \cup N_G(D)})$ as depicted in Figure 3, where $D = \{CanUseCallButton, CanUseSelectButton\}$. Here, we see three SCs, namely '$G.CanUseCallButton$', '$G.CanUseSelectButton$' and '$G.CanUseElevator$'. While the first two represent the corresponding SCs obtained by augmenting the granularity of those fluents, the third one showcases the augmented version of the action '$UseElevator$', whose new level of granularity is obtained using the updated SCs of its fluents, all of which is generated with the transitional description $\mathcal{TD}$.

Finally let us now define how to apply a transitional description to a simple catalog to expand the representation.

Definition 4. *If $\mathcal{TD} = (D, (G.d)_{d \in D \cup N_G(D)})$ is a transitional description associated to the bipartite graph $G = (V_F, V_A, E_G)$, then the graph $\mathcal{TD}(G)$ obtained from G by applying $\mathcal{TD}$ is constructed as follows:*

1. *Take a new copy of $\lfloor V_F - D \rfloor_G$.*
2. *For each $d \in D$, take a new copy of the graph $G.d$ and make the disjoint union of it with the current graph constructed.*

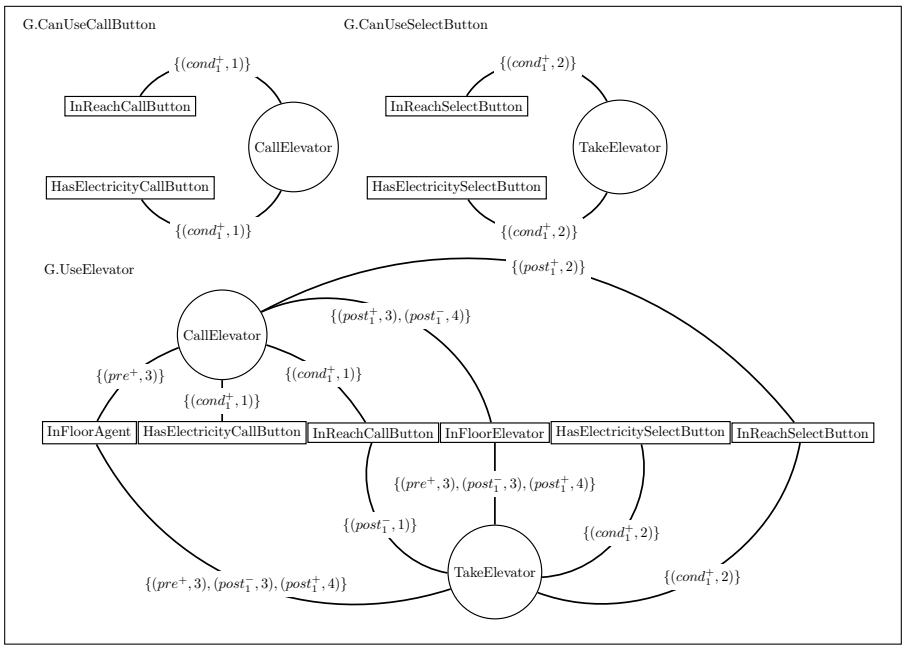

Figure 3: Transitional description for the elevator example

3. *For each $d \in N_G(D)$, identify the nodes of $G.d$ which are already added to the current graph (i.e. the atomic nodes of G that are neighbors of d and the nodes of $G.d'$ which appear in $G.d$). For each complex neighbor d' of d in G, add the remaining nodes of $G.d$ as new nodes in the current graph and link all these nodes by edges as described in $G.d$ (in order to have an isomorphic copy of $G.d$ as a subgraph in the current graph).*

In the case of the previous example, due to the fact that there is only one action symbol in the catalog $[S, G, \lambda]$ defined previously, the bipartite graphs $\mathcal{TD}(G)$ and $G.UseElevator$ are identical.

The following property for bipartite graphs follows.

Proposition 1. *If $G = (V_F, V_A, E_G)$ is a bipartite graph and $\mathcal{TD}$ is a transitional description associated to G, then the graph $\mathcal{TD}(G)$ obtained from G by applying $\mathcal{TD}$ is also a bipartite graph.*

Proof sketch. We follow the construction given in the previous definition regarding the application of a transitional description $\mathcal{TD}$ to a bipartite graph G.

It is immediate to see that the starting graph in (1) $\lfloor V_F - D \rfloor_G$ is a bipartite one because G was bipartite. Then for point (2), each expanded graph $G.d$ is a bipartite one by definition and by applying a disjoint union with another bipartite graph, the property is preserved. Finally, point (3) only links fluent nodes to actions nodes, which also preserves the bipartite property. □

This proposition ensures that the graph obtained by applying the transitional description can itself be expanded using another.

So far, our definition of transitional description only accounts for nodes and actions, but not that their expansions match their types, when considering ontological supports. The following definition extends the notion for a simple graphical catalog of actions to account for the types of fluents and actions.

Definition 5. *Let $SC = [S, G, \lambda]$ be a graphical catalog of actions, where $G = (V_F, V_A, E_G, l)$. A transitional description associated to SC is a pair $\mathcal{TD} = (D, (SC.d)_{d \in D \cup N_G(D)})$ where:*

- *$D \subseteq V_F$ is a set of complex fluent symbols.*
- *For each $d \in D \cup N_G(D)$, $SC.d = [S.d, G.d, \lambda.d]$ is a SC.*
- *If $d \in D$, then $G.d$ is the non-empty ($G.d \neq G^{\emptyset}$) description of the complex fluent symbol d. Distinct complex fluent symbols $d, d' \in D$ have disjoint descriptions $G.d \cap G.d' = G^{\emptyset}$. Moreover, for all $v \in V_F(G.d)$, it holds that $\lambda(d) = \lambda.d(v)$.*
- *If $d \in N_G(D)$, then $G.d \neq G^{\emptyset}$, $N_G(d) - D \subseteq V_F(G.d)$, and $V_F(G.d) \cap V_F(G.d') \neq \emptyset$ for each $d' \in N_G(d) \cap D$. Moreover, $S.d \supseteq S \cup_{d' \in N_G(d)} S.d'$, $\lambda.d(v) = \lambda(v)$ for $v \in N_G(d) - D$ and $\lambda.d(v) = \lambda.d'(v)$ for $v \in V_F(G.d) \cap V_F(G.d')$. Finally, for each $d' \in V_A(G.d)$, it holds that $\lambda(d) = \lambda.d(d')$, i.e., when decomposing an action through a transitional description, the corresponding actions preserve its type.*

Note that we disallow $G.d = G^{\emptyset}$, as this would erase the node when applying a transitional description. In the simplest case, the new graph can only be composed of the original node itself, thus not producing any changes.

Analogously to the extension of transitional descriptions from bipartite graphs to SCs, the following definition extends the *application* of a $\mathcal{TD}$, to take into account the types.

Definition 6. *If $\mathcal{TD} = (D, (SC.d)_{d \in D \cup N_G(D)})$ is a transitional description associated to $SC = [S, G, \lambda]$, then the simple graphical catalog of actions $\mathcal{TD}(SC)$ obtained from SC by applying $\mathcal{TD}$ is $\mathcal{TD}(SC) = [S', \mathcal{TD}(G), \lambda']$.*

Here, $\mathcal{TD}(G)$ is the bipartite graph $\mathcal{TD}(G)$ obtained from G by applying $\mathcal{TD}$, $S' = \cup_{d \in D \cup N_G(D)} S.d$ and λ' is any legal labelling function defined on $V(\mathcal{TD}(G))$ which preserves the labels given to the vertices in $V(G)$ and $V(G.d)$ for all $v \in D \cup N_G(D)$.

Note that the last condition in Definition 5 (concerning the preservation of types for the expanded action nodes) is the one that ensures that when expanding the nodes, the types coming from the ontologies will be preserved when applying a transitional description.

Moreover, using Proposition 1 we can verify that $\mathcal{TD}(SC)$ is a simple graphical catalog of actions. This means that we can nest the application of transitional description to produce several expanded catalogs of actions of increasing complexity, which we define as follows:

Definition 7. *Let n be a non-negative integer. A layered catalog graph (LC) of depth n is a family* $\mathbf{LC} = \langle\ SC^0, \mathcal{TD}^0, \ldots, \mathcal{TD}^{n-1}\ \rangle$ *where:*

- $SC^0 = [S^0, (V_F^0, V_A^0, E^0), \lambda^0)]$ *is a SC,*
- $\mathcal{TD}^0$ *is a transitional description associated to* SC^0*,*
- *for each* k*,* $1 \leq k \leq n-1$*,* $\mathcal{TD}^k$ *is a transitional description associated to* $SC^k = [S^k, (V_F^k, V_A^k, E^k), \lambda^k)] = \mathcal{TD}^{k-1}(SC^{k-1})$*.*

SC^0 *is the base simple graphical catalog of actions of the layered graph* **LC** *and* $SC^k = \mathcal{TD}^{k-1}(SC^{k-1})$ $(k = 1, \ldots, n)$*, are its layers.*

In other words, if we have an interconnected world described by a SC and if we can provide details about both some complex fluent and action symbols, then we can construct a second level of knowledge about this world, describing these new details as graphs and applying the corresponding substitutions. This process can be similarly performed with the last constructed level, thus obtaining a coherent set of layered representations of the initial world.

5 Related work

As we mentioned in the introduction, the motivation behind our work was to provide a mechanism that improved the scrutability of a planner for a human user. Unfolding representations between multiple granularity levels has been studied in the context of AI planning optimization [8] and generic conceptual graphs [4], but not for scrutability and neither in such a way that fluents and actions can be unfolded with the support of an ontology. In recent years, the AI planning community has taken interest in what in known as explainable AI planning (XAIP) [9]. Notably, this branch has proposed several critical questions for AI planners that can be of interest to human users: (i) why did you do that?, (ii) why didn't you do something else?, (iii) why is what you propose to do more efficient/safe/cheap than something else?, (iv) why can't you do that?, (v) why do I need to replan at this point? and (vi) why do I not need to replan at this point?. With that in mind, our framework could reinforce other techniques [10, 11] to provide an appropriate level of detail for the generated plan generated tailored to a specific kind of user. At the same time, it could be applied in cases where no plan can be generated by using counterfactuals [12, 13], so that the planner can iteratively inform the reason behind its failures.

Moreover, as far as we know, there is no work coupling structured hierarchical knowledge of fluents and actions in such a way that the robot can expand

sets of fluents to assess which action could be applied using a general abstract graph-based ontology. Although our work extends previous literature for conceptual graphs [4], the mentioned literature would not allow to model action preconditions and effects, due to the fact that the same fluent node cannot be attached multiple times as one or the other, to the same action.

With regards to classical AI planning, our framework can be related to the concept of 'macro operators' [14]. Macro operators were developed as an optimization technique for PDDL planners, which consists in learning operators (actions) composed of several others before computing the plan for a complex task, based on simpler ones. While this approach indeed allows to couple actions together, this does not allow for several fluent granularity levels as we proposed here.

That being said, the proposed approach is related to the field of hierarchical task networks (HTN) [15] where dependencies among actions can be structured in the form of hierarchies linking actions to higher level compound ones. However, this framework does not allow for the structured representation of knowledge we provide. Some works have proposed ontologies for HTNs [16] domains, but not with a focus on the transformation from one level of detail to another on the fly as we do here. Similar to the macro operators, their work does not allow for multiple fluent granularity levels and neither a mechanism to unfold them.

Perhaps closer to our work, hierarchical models for STRIPS-like planners have been studied [8] as a method to improve the efficiency of plan generation by mapping tasks to different granularity levels. In contrast to out framework, this avenue of work focused on mapping fluent parameters, instead of fluent and action symbols.

6 Conclusion

In this paper, we introduced a graph-based hierarchical model of knowledge about actions and fluents that allows to represent structured knowledge about the world under the form of increasingly detailed levels. We developed a graph-based framework supported by an ontology, called layered catalog of actions, to characterize an agent's actions. We showed how the fluents and actions of the framework can be specified using this catalog. Our work not only formalized this model, but also provided a method to adjust the level of detail of the fluents and actions using the concept of *transitional description*.

The proposed approach that joins structured and layered knowledge of actions opens different avenues of research for future work. In particular, by adapting the level of granularity, an agent could adapt to the level of understanding of the user. This could pave the way for robot-user explanation by task measurement, i.e., testing if the user can perform a task where multiple actions have to be performed after being exposed to an explanation by the robot of a similar problem.

Finally, because our framework relies on an ontology to represent the fluents

of states and considers belief states, it could potentially allow for plausible reasoning [17] through the type of individuals. More precisely, when the properties of a certain individual are not fully known, it could be represented by using different beliefs about the types of that individual and then implementing plausible reasoning through the planner. In future work, this approach could also be extended to other kind of uncertainty, such as partial observability planning [18].

References

[1] Alan Baddeley. The concept of episodic memory. *Philosophical Transactions of the Royal Society of London. Series B: Biological Sciences*, 356(1413):1345–1350, 2001.

[2] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning: theory and practice*. Elsevier, 2004.

[3] Marie-Laure Mugnier and Michel Chein. Conceptual graphs: Fundamental notions. *Revue d'intelligence artificielle*, 6(4):365–406, 1992.

[4] Madalina Croitoru, Ernesto Compatangelo, and Chris Mellish. Hierarchical knowledge integration using layered conceptual graphs. In *International Conference on Conceptual Structures*, pages 267–280. Springer, 2005.

[5] Daniel Bryce, Subbarao Kambhampati, and David E Smith. Planning graph heuristics for belief space search. *Journal of Artificial Intelligence Research*, 26:35–99, 2006.

[6] Blai Bonet and Hector Geffner. Planning with incomplete information as heuristic search in belief space. In *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems*, pages 52–61, 2000.

[7] Alessandro Cimatti, Marco Pistore, Marco Roveri, and Paolo Traverso. Weak, strong, and strong cyclic planning via symbolic model checking. *Artificial Intelligence*, 147(1-2):35–84, 2003.

[8] Cipriano Galindo, J-A Fernandez-Madrigal, and Javier Gonzalez. Improving efficiency in mobile robot task planning through world abstraction. *IEEE Transactions on Robotics*, 20(4):677–690, 2004.

[9] Maria Fox, Derek Long, and Daniele Magazzeni. Explainable planning. *arXiv preprint arXiv:1709.10256*, 2017.

[10] Michael Cashmore, Anna Collins, Benjamin Krarup, Senka Krivic, Daniele Magazzeni, and David Smith. Towards explainable ai planning as a service. *arXiv preprint arXiv:1908.05059*, 2019.

[11] Tathagata Chakraborti, Sarath Sreedharan, and Subbarao Kambhampati. The emerging landscape of explainable ai planning and decision making. *arXiv preprint arXiv:2002.11697*, 2020.

[12] Ruth MJ Byrne. Counterfactuals in explainable artificial intelligence (xai): Evidence from human reasoning. In *IJCAI*, pages 6276–6282, 2019.

[13] Devleena Das, Siddhartha Banerjee, and Sonia Chernova. Explainable ai for robot failures: Generating explanations that improve user assistance in fault recovery. In *Proceedings of the 2021 ACM/IEEE International Conference on Human-Robot Interaction*, pages 351–360, 2021.

[14] Adi Botea, Markus Enzenberger, Martin Müller, and Jonathan Schaeffer. Macro-ff: Improving ai planning with automatically learned macro-operators. *Journal of Artificial Intelligence Research*, 24:581–621, 2005.

[15] Kutluhan Erol, James A Hendler, and Dana S Nau. Semantics for hierarchical task-network planning. Technical report, MARYLAND UNIV COLLEGE PARK INST FOR SYSTEMS RESEARCH, 1995.

[16] Artur Freitas, Daniela Schmidt, Felipe Meneguzzi, Renata Vieira, and Rafael H Bordini. Using ontologies as semantic representations of hierarchical task network planning domains. In *Proceedings of WWW*, page 124, 2014.

[17] Allan Collins and Ryszard Michalski. The logic of plausible reasoning: A core theory. *cognitive science*, 13(1):1–49, 1989.

[18] Blai Bonet and Hector Geffner. Planning under partial observability by classical replanning: Theory and experiments. In *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.