A Novelty-Centric Agent Architecture for Changing Worlds

Faizan Muhammad
Tufts University
Medford, USA
faizan.muham@gmail.com

Vasanth Sarathy
Tufts University
Medford, USA
vsarathy@gmail.com

Gyan Tatiya
Tufts University
Medford, USA
gyan.tatiya@tufts.edu

Shivam Goel
Tufts University
Medford, USA
shivam.goel@tufts.edu

Saurav Gyawali
Tufts University
Medford, USA
saurav.gyawali@tufts.edu

Mateo Guaman
Tufts University
Medford, USA
mateo.guaman@tufts.edu

Jivko Sinapov
Tufts University
Medford, USA
jivko.sinapov@tufts.edu

Matthias Scheutz
Tufts University
Medford, USA
matthias.scheutz@tufts.edu

ABSTRACT

Open-world AI requires artificial agents to cope with novelties that arise during task performance, i.e., they must (1) detect novelties, (2) characterize them, in order to (3) accommodate them, especially in cases where sudden changes to the environment make task accomplishment impossible without utilizing the novelty. We present a formal framework and implementation thereof in a cognitive agent for novelty handling and demonstrate the efficacy of the proposed methods for detecting and handling a large set of novelties in a crafting task in a simulated environment. We discuss the success of the proposed knowledge-based methods and propose heuristic extensions that will further improve novelty handling in open-worlds tasks.

KEYWORDS

Novelty Handling; Cognitive Architecture; Open-World AI

ACM Reference Format:
Faizan Muhammad, Vasanth Sarathy, Gyan Tatiya, Shivam Goel, Saurav Gyawali, Mateo Guaman, Jivko Sinapov, and Matthias Scheutz. 2021. A Novelty-Centric Agent Architecture for Changing Worlds. In *Proc. of the 20th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2021), Online, May 3–7, 2021*, IFAAMAS, 9 pages.

1 INTRODUCTION

Much of AI research has dealt with so-called "closed worlds" where agent designers know all task-relevant concepts ahead of time and get to develop targeted algorithms based on that knowledge. In contrast, "open worlds" allow for novelty that is not known in advance and can thus not be planned or designed for. The shift from closed to open worlds, therefore, requires artificial agents to cope with novelties as they arise during task performance.

Coping with novelty involves at least three challenges: (1) detecting it, (2) characterizing it, and (3) accommodating it. The first requires the agent to determine, through direct observation or indirect inference, that something is novel, while the second requires the agent to determine the nature of the novelty (e.g., whether it is a novel object, a novel property, a novel relation, a novel event, etc.); and the third then requires the agent to use its characterization of the novelty to determine whether it is beneficial, detrimental, or irrelevant for its task performance and accordingly use, circumvent or avoid it.

While investigations in the past have explored some aspects of open-world AI (e.g., open-world goals [11, 28, 29]), none of these approaches have attempted to systematically detect novelty, but were rather restricted to particular types of novelty that could be handled (e.g., certain forms of open-world goals or instructions). The goal of this paper is to present the first systematic architectural approach for detecting, characterizing, and accommodating novelty without making any prior assumptions about the nature of the novelty. Specifically, we present a general novelty-handling framework implemented in extensions to the Distributed Integrated Affect Reflection Cognition (DIARC) architecture [24, 25] and evaluate them in two separate evaluations, one in our own simulation and the other in Polycraft [22]. To maximally remove any evaluation bias, the second large-scale evaluation was performed by a third party, quantifying in great detail the extent to which the architectural extensions addressed the three novelty-handling challenges.

The rest of the paper is structured as follows. We start by providing our definition of novelty and motivation for why novelty handling is important in open worlds, followed by a formal framework for detecting, characterizing, and accommodating novelty. We then describe in detail the implementation of the framework and two evaluations that demonstrate its efficacy, also discussing possible improvements and finishing with a brief summary of our accomplishments.

2 MOTIVATION

We start by defining what we mean by novelty and then motivate why being able to handle novelty is an important feature for AI agents that will make them more versatile and robust, enabling

Proc. of the 20th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2021), U. Endriss, A. Nowé, F. Dignum, A. Lomuscio (eds.), May 3–7, 2021, Online.

critical features for resilience and long-term autonomy in open worlds.

2.1 What does it mean to be novel?

Novelty is not a property of entities in the world, as they do not have labels that say "novel" on them. Rather, novelty is *relative* to an agent's cognitive system and past experiences. When an agent has previously encountered an entity, being able to recall the encounter makes a subsequent encounter not novel. However, if the agent does not remember encountering the entity and can also not identify the entity or its type through inference, the encountered entity will be novel for the agent (and it will continue to be novel as long as the agent does not form memories or acquire the requisite knowledge about the entity, regardless of whether an external observer watching the agent's repeated encounters with the entity expects the agents to know the entity after the first encounter).

Since novelty is in the eye of the beholder, i.e., agents observing or contemplating an entity, action, process, etc., we need to keep in mind whose novelty we are evaluating when we design artificial agents that can detect, characterize, and accommodate novelty: the agent's or ours? Typical machine learning tasks require our agents to detect what *we* – the agent designers and evaluators – consider novel in the data, i.e., novel items in evaluation data relative to a set of given items in training data that does not contain these items. Our reasoning is that any agent familiar with the training set ought to be able to determine that the evaluation data contains novel items. Whether this is actually so, however, will depend on a variety of factors, not the least the agent's discrimination capabilities.

We introduce the concept of "original" vs. "derived" novelty to elucidate an important distinction regarding the agent's cognitive capabilities: for some agents, entities they have not *per se* experienced will not be novel as these agents can construct patterns representing or denoting such entities (e.g., by combining patterns resulting from other previously experienced entities), while others not capable of such constructions will always find unexperienced entities novel. We will thus count an item as only novel for an agent when the agent has not experienced the item before (i.e., cannot recall it) and cannot actually (or even in principle) derive it from its knowledge base, which thus leaves us with four cases:

- the agent has experienced it before and remembered it, hence it is not novel
- the agent derived it without having experienced it, so if and when it experiences it, it will not be novel
- the agent could in principle derive it based on its knowledge and derivation processes if it did it for long enough (but practically not derivable), hence practically the item is novel
- the agent could not derive it given its cognitive capabilities, hence it is novel

2.2 The utility of being able to handle novelty

Novelty then can arise in many different ways, from novel objects with novel properties, to novel relations and contexts, to novel actions and events, to novel goals and tasks, to novel constraints and rules, and so forth, even in closed worlds. For example, suppose a closed-world agent encounters an unforeseen fault during task performance (such as data corruptions, sensor failure, insufficient compute power, etc.) which causes a change in its performance. The agent then may or may not be able to detect the change, and even when it detects the change, it might not be able to characterize the fault and find a way to address it. Similarly, in open worlds, agents ought to handle novelty when it presents opportunities for improving performance, especially when it enables goal accomplishment (which would not have been possible without utilizing the novelty).

To illustrate the opportunities novelty handling presents, we will situate the exposition within the *Polycraft* simulation environment [22][1] where an artificial agent has to perform a simple "pogo stick" crafting task, which we will later also use for a thorough evaluation of our proposed agent's novelty handling capabilities. The task is performed within a walled-in region (the "environment") which contains *trees* and a *crafting table* where more complex items can be crafted. The agent can directly obtain *logs* by breaking down *trees*, which can then be crafted into *planks* at the *crafting table*. These planks can subsequently be crafted into *sticks*, and a combination of *planks* and *sticks* can be crafted into a *tree tap*. Once the *tree tap* is placed next to a *tree*, *rubber* can be extracted from the *tree tap*. Finally, a combination of *sticks*, *planks* and *rubber* can be used to craft a *pogo stick*.

Solving this task is challenging for two reasons. First, the agent must follow a long sequence of actions to craft a pogo stick. Second, because the resources available in the environment are limited, the agent must not be wasteful in collecting or crafting any items, as otherwise goal achievement might become impossible. Yet, being able to handle novelties clearly increases the versatility of the agent, as it can utilize novelties it learned about in one task for other future tasks (e.g., a newly introduced axe could be use for breaking certain materials later that would not be otherwise breakable). Moreover, handling novelties also makes the agent more robust to failure (e.g., overcoming obstacles such as an unexpected fence enclosure enable the agent to not only achieve goals in this context, but the agent might then also be able to generalize the fence to other blocking structures and the need to remove them through actions in order to open up paths to goal locations). Both versatility and robustness then provide the foundations for resilience (i.e., being able to cope with unexpected deterioration of the task environment) and thus long-term autonomy. These challenges are also why we consider Polycraft to be an "open-world" environment. The agent must be able to handle domain entities beyond what is defined because the agent does not know, apriori, the universe of objects it might see. Therefore, the very symbols of the world are open to change and not just the pre-defined properties.

3 BACKGROUND AND RELATED WORK

While there are various efforts related to handling novelties in planning and reinforcement learning, to the best of our knowledge the problem of handling novelty as laid out above still needs to be addressed as most proposed methods heavily rely on assumptions about the types of novelties being introduced in the environment.

Contingent planning, for example, focuses on allowing an agent to plan online as it senses an environment [16]. Some have proposed

[1]Polycraft is a modified version of *Minecraft*, a popular first-person, open-world, 3D block-based environment where agents collect raw materials that can be crafted, or put together, into more complex tools or items. Polycraft extends the raw materials available in the world as well as the items that can be crafted.

algorithms for planning with incomplete information, whereby conditional plans have to be developed [5, 7]. However, in both these approaches, the set of possible states or belief states are assumed to be known to the agent, which may not be the case in situation where the agent encounters novelty.

Recent work using deep neural networks has shown promise in learning planning domain models from images [3, 6]. However, even if the agent is able to learn the planning domain, the challenge remains as to how the agent can revise these representations when planning fails, as is required for accommodating novelties. Work in belief revision [12] is also related here.

Integrated approaches for growing symbolic representations through exploration of sub-symbolic spaces shares similarities with our approach to learning new constants and operators [15, 18, 23, 26]. While these approaches are not configured explicitly for novelty detection and accommodation, they nonetheless provide promising techniques to enhance our sub-symbolic search capabilities.

Reinforcement learning is another paradigm that offers a helpful way to model exploration and learning. There is a body of work associated in dealing with non-stationarity in environments through reinforcement learning [21]. Curiosity-driven or intrinsically motivated RL [9, 13] is especially relevant to our setting as the approach attempts to handle domains where reward functions (even long-range ones) are not readily available or may change over time. However, those RL methods still do not allow an agent to modify its own action parameters and input representations, which may be necessary to handle certain types of novelties. Hierarchical RL and its use of "option" appears promising for representing meta-level strategies [17, 27]. Others have proposed a combined planning and RL approach for learning low-level action policies [1]. However, this work assumes a complete high-level model, which may become inaccurate or incomplete once a novelty is introduced.

Recent interest has been developed in the context of continual reinforcement learning and meta reinforcement learning to deal with non-stationary environment dynamics. Some methods learn the underlying model dynamics to predict future states for novelty and adapts to those changes [2, 10, 20]. Other related work borrows concepts from the online learning and probabilistic inference literature to learn an off-policy RL algorithm to deal with lifelong non-stationarity [31]. Learning a representation of the environment from present and past experience has been employed in this approach. Others present a policy gradient-based algorithm which incorporates counter-factual reasoning to search for optimal policies for future MDPs [8]. While this direction seems promising, the definition of anomaly heavily assumes a-priori knowledge in the system about the novelty to be introduced later. Most of the above works also assume that the environment is subject to gradual changes, rather than dealing with sudden changes.

4 THE NOVELTY HANDLING FRAMEWORK

We start by providing a general definition for task environments. Let $\mathcal{E} = \langle E, I, G, R, \tau \rangle$ be a *(task) environment specification* where E is an environment viewed as a set of possible states (e.g., a manifold), $I \subset E$ is a set of initial states, $G \subset E$ is a set of goal states, R is the environmental evolution relation defined on E over a period of time bounded by τ, the time bound on task performance. Note that we define environments as sets of states to be both as general as possible while not committing to a particular notion of state or a particular formalism to capture all possible environmental states (e.g., a set of differential equations, an MDP, etc.) and their relations (e.g., which state is accessible from a given state or whether state transitions are deterministic or stochastic). Also note that state descriptions may include objects and their properties as well as relations among objects. In particular, they include agents, the states of their embodiment, and their computational states.

Next, we define what we mean by an "agent". Let $\mathcal{A} = \langle P, C \rangle$ be an *agent specification* where P is the hardware platform (including all sensing, actuating, and computing equipment) if the agent is embodied and C is a computational system, e.g., an algorithm (plus data) initially realized on P (and possibly self-modifying afterwards), a deep neural network, etc.. If required, we could also single out the set of sensors Sen, the set of effectors Eff, and the computational systems $Comp$ being part of P such that a computation can be defined as a relation between sensors and computational states, on the one hand, and effector and computational states, on the other

$$C : States_{Sen} \times States_{Comp} \times States_{Eff} \times States_{Comp}$$

where the sensor and effector states are the transduced and non-transduced computational interface states, respectively (as opposed to voltage or angle states, say). In case C is a function, the computation is said to be deterministic, otherwise non-deterministic. Note that all state changes (effector and computational) in the agent will result in environmental state changes given that agent states are a subset of environmental states. We can then define "open-world task variations" as different types of modifications or perturbations applied to $\mathcal{E}$. For example, we add new states to E or remove some states from E, we could change I and G (the initial conditions and the goals), or we could alter the way the environment behaves (even altering τ could force the agent to contend with unknown entities).

For the purposes of this paper, we will only consider cognitive agents, i.e., agents that can reason and plan with symbolic representations. Specifically, let $\mathcal{L}$ be a first-order language (FOL) and define atoms over $\mathcal{L}$ as $p(t_1, \ldots, t_n)$ or negations $\neg p(t_1, \ldots, t_n)$ where t_i represents terms that can be variables or constants and p represents a $n - ary$ relation/property or action primitive.[2] An atom is considered *grounded* if and only if all of its terms are constants. The agent then uses $\mathcal{L}$ to express concepts about the environment specification $\mathcal{E}$ such as observations (based on information coming from the sensors via $States_{Sen}$), actions (being effected via $States_{Eff}$), goals (FOL descriptions of the goal states in G), facts (FOL descriptions of states in E, but also other concepts), and rules (FOL formulas capturing regularities in E such as knowledge about objects and their properties, action pre- and post-conditions, etc.), all representations which are ultimately expressed in terms of states in $States_{Comp}$. The agent also has a knowledge base KB for storing facts $\phi \in \mathcal{L}$ about $\mathcal{E}$, in particular, about (what it takes to be) the current state of the world in E, and a first-order proof system $\vdash$ which it uses to derive new facts $KB \vdash \psi \in \mathcal{L}$ and to generate plans to accomplish its goals. A plan $\pi = \langle \alpha_1, \ldots, \alpha_n \rangle$ is a sequence of grounded actions

[2] We do not consider functions for simplicity.

such that when the corresponding action is executed in the environmental state corresponding to σ_1 the environment (and thus the agent) will eventually be in a state corresponding to σ_{n+1} (if all goes well, see below), where σ_i is the agent's FOL description of some world state $S_i \in E$. If σ_1 corresponds to a state in I and σ_{n+1} corresponds to a state in G and successful execution of π is within bound τ, then the agent accomplished the task specified by E.

4.1 Planning and Plan Execution

The basic operation of the agent at a time t is as follows. If the agent does not have a plan, it will compute $KB \cup \sigma_t$ to derive a plan $\pi = \langle \alpha_t, \alpha_{t+1}, \ldots, \alpha_n \rangle$ such that plan execution will get it into a state in G. If such a plan cannot be found and the agent has incomplete descriptions of an object o (e.g., missing properties or action affordances), it can attempt to infer properties of o and replan (e.g., the plan then could include an exploratory goal $\gamma(o)$ to characterize o), otherwise the agent gives up (as the goal is not achievable based on its knowledge). If the agent already has a plan $\pi = \langle \alpha_1, \ldots, \alpha_t, \alpha_{t+1}, \ldots, \alpha_n \rangle$, then it simply executes α_t and inspects its FOL rendition of sensory state σ at $t + 1$ to determine whether its predicted state $\sigma_{t+1} \subseteq \sigma$. If so, action execution was successful and the agent proceeds with the execution of the next action. And if $KB \cup \sigma \nvdash \bot$, σ can be added to the KB, otherwise the agent needs to find the smallest set Φ such that $KB - \Phi \cup \sigma \nvdash \bot$. If, on the other hand, the action failed or led to a state different than the expected state, the agent needs to correct those parts of its knowledge that led to the incorrect prediction. While in stochastic environments this might just indicate deficiencies in the agent's knowledge of the transition function, in deterministic environments it clearly indicates lack of knowledge and thus potentially the presence of novelty, which we will discuss next.

4.2 Novelty Detection

Novelties can be detected when there are discrepancies between predicted and actual states, with or without action execution of the agent. If there is an inconsistency $KB \cup \sigma$ (the new sensory state at $t + 1$) for the agent it signifies a novelty. If σ does not contain any atoms the agent has not seen before and if the agent has not made a specific prediction about σ, but only the implicit prediction that nothing changed, then such unpredicted changes without novel objects require the agent to simply update its knowledge base as described above (e.g., a new tree of a known type popped up in a new location in Polycraft). Yet, if either σ contains atoms the agent has not seen before or the agent made a prediction different from σ, then there is a different type of novelty involved, either a new object with potentially new properties in the former (e.g., an axe), or a new object affordance or action in the latter case (e.g., chopping a tree with an axe produces more logs).

4.3 Novelty Characterization

For a given observation σ that the agent deems novel, it can simply characterize the novelty by determining what follows from $KB \cup \sigma$ (if σ is consistent with KB, otherwise $KB - \Phi \cup \sigma$ with the inconsistent parts Φ removed). For example, if a novel object appears, an axe, say, the agent can determine based on rules in KB that it is a material object (because only material objects can be located in the environment) and it can make an assumption (possibly wrong) that as a material object it can be broken and picked up. These inferences can be used both to describe properties and affordances of novelty but could also be used to plan for gathering more information about the novelty.

4.4 Novelty Accommodation

Novelty accommodation then means to take a novelty characterization and add it to the agent's knowledge base, although it is important to distinguish different types of accommodation that depend on the type of novelty the agent discovered. For a novel object, for example, its type and derived properties can just be added as facts to the knowledge base (e.g., an axe is a material object). In addition, action affordances that can be derived from the object's properties can be added as well (e.g., an axe can be broken given that it is a material object). However, other action affordances (e.g., the axe can break otherwise unbreakable materials) need to be determined through explicit exploration (as they are not derivable from the agent's knowledge base) for which the agent needs to generate a separate exploration goal, and/or through communication with humans and other agents, if available.

5 IMPLEMENTATION

The novelty handling framework was implemented in the "Agent Development Environment" (ADE) [4] which allows for the development of modular components that can be used and reused to create different cognitive architectures like DIARC [25]. In addition to using the existing DIARC Goal Manager for goal management and action execution, new components were developed for the Polycraft interface (see Fig. 1). The *Knowledge Base* of the agent is defined by the *Action stack* and the *Belief stack*. *Action Execution* provides planning and plan execution to reach goals based on the agent's knowledge base and the provided task goal. The *Polycraft Interface* provides a link to the Polycraft API and produces novelty assertions based on percepts received from the environment. These novelty assertions are consumed by the *Novelty Handling* system to identify any novelties within the environment and report them as well as perform any modifications to the knowledge base as needed.

5.1 The Polycraft Interface

The Polycraft Interface component in Fig. 1 translates environment information in JSON to novelty assertions and also provides functions for action execution that automatically generate JSON commands that are sent to Polycraft. Actions in the Polycraft environment can be passive or active. Examples of passive actions include, "SENSE_RECIPE" which returns a set of recipes that can be used to craft items, and "SENSE_ALL" which returns information about the current state of the environment, including a map with locations of different items and the agent and the agent's inventory. Examples of active actions include movement ("MOVE_FORWARD") and crafting ("CRAFT <params>"). Certain actions also take in parameters such as the craft action which requires a list of ingredients.

5.2 Novelty Assertions

Novelty Assertions are central to our implementation of the novelty handling framework. A novelty assertion specifies a set of grounded

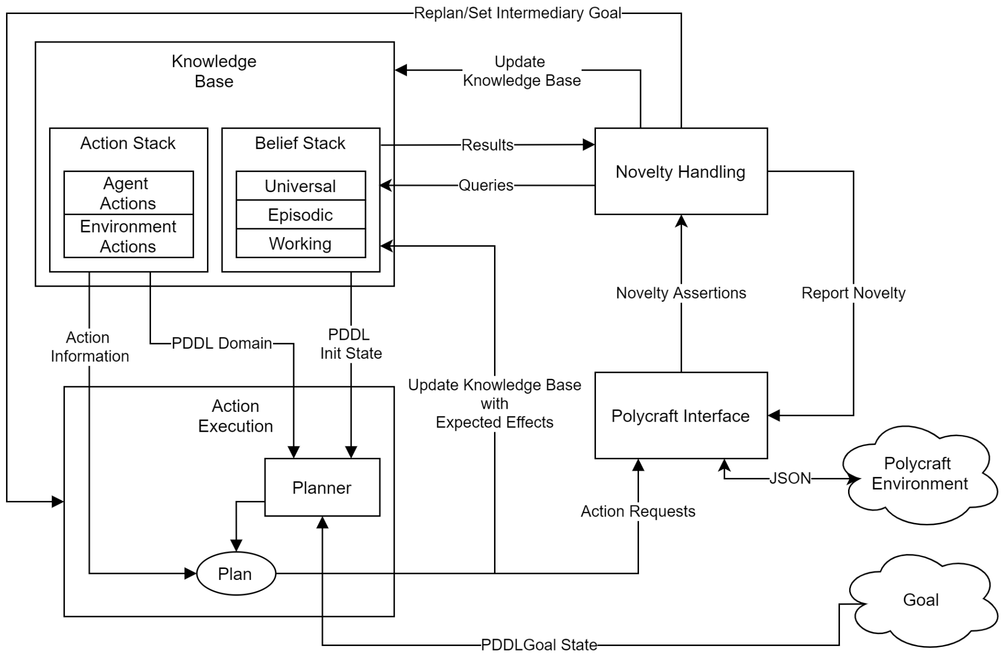


Figure 1: The overall agent architecture of the implemented novelty detection framework.

atoms to add and remove from the agent's knowledge base such that the modified knowledge base would have the information that the novelty assertion is trying to convey. The simplest form of a novelty assertion are grounded atoms which can be true in and of themselves, independently of others. For example, if a new material is seen in a particular location, we can assert that it is of type "material" in our knowledge base independently of any other fact.

Another type of novelty assertion is based on KB inference, as described in Section 4.2, where atoms inferred from the novelty assertion are compared to atoms in the knowledge base and the knowledge base is updated accordingly, adding inferred atoms not in the knowledge base and removing atoms from the knowledge based inconsistent with the novelty assertion. This type of novelty assertion is, for example, used in describing the environment map since there can be only one kind of block in a particular location in Polycraft, hence if the knowledge base contains a statement about one type of block being in a given location and the novelty assertion of another type of block being in the same location, then a KB atom needs to be replaced by the one from the novelty assertion.

Finally, another useful type of novelty assertions concern rules and exceptions, for example, in the case of conveying that all items not included in the agent's inventory percept (the items the agent holds) should have the amount 0 in the agent's knowledge base. Here, the rule specifies the amount in inventory is 0 and the exceptions contain materials that are known to have non-zero amount.

5.3 The "Belief Stack" in the Knowledge Base

Cognitive Architectures often have tiered or typed memory systems [19]: *procedural memory* contains rule-based based knowledge such as implication or condition-action rules, *declarative memory* contains semantic facts such as relationships between objects, and *episodic memory* typically contains the history of past experiences. We opted for a different memory design based on information permanence and duration of validity, identifying three levels of validity: *universal*, *episodic*, and *working*. Each of the levels can have facts, rules, and history. The universal level contains knowledge that is known to be true across different tasks (e.g., the definition and arity of primitive predicates). The *episodic level* contains knowledge that the agent has amassed over several executions of the same task (for example, recipes of secret items). The *working level* contains knowledge that is pertinent to a single specific task execution (for example, the current location of the agent).

As these levels – the "belief stack" – are nested (see Figure 2), a fact asserted at a universal level can be successfully accessed from the episodic and working levels, but not vice versa, as assertions to the working level are only retained for the duration of a task performance and are erased when the task is complete, very much like the operation of working memories (procedural and declarative) in typical cognitive architectures. To retain working-level knowledge across task performances, the agent needs to transfer it to the episodic level. The three levels are implemented via three instances of Prolog with the addition that assertions or retractions made at a higher level (e.g., universal) are also made at the lower level (e.g., episodic and working), ensuring that higher level instances cannot have facts or rules that lower levels do not. Hence, the agent's total knowledge is always accessible at the working level, but not all of that knowledge will be retained across trials or different tasks.

This compartmentalization is also critical for providing a high-level abstraction for Novelty Handling so that, if needed, aspects of the knowledge base can be modified without any long-term detriments. For example, potential modifications can first be applied

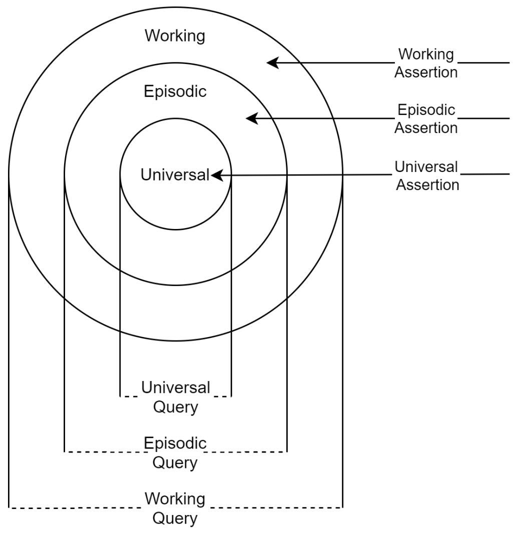


Figure 2: Belief Stack levels are nested based on permanence.

at the working level and then upgraded to the episodic level only after they have been verified. Therefore, the agent is empowered to modify its knowledge base freely and only retain the changes if they are successful.

5.4 The "Action Stack" in the Knowledge Base

The Action Stack is primarily responsible for maintaining the translation between actions as defined by the agent and actions as defined by the environment (see $States_{eff}$ in Section 4). The action descriptions based on the FOL $\mathcal{L}$ are used for planning whereas the level of $States_{eff}$ is used for interacting with the simulation environment. For example, the Polycraft component accepts only egocentric directions as parameters to the "move<direction>" action ("up", "down", "left" and "right", corresponding to key strokes if a human player were to interact with Polycraft), whereas for the FOL level we might want to use locations (rather than directions) as arguments in the move action and we might add the current and the destination locations to make it a two-place predicate (this representation also works for action representations in PDDL which the planner expects).

5.5 Action Execution

The Action Execution component, based on the DIARC Goal Manager [25] ties together the Knowledge Base and Polycraft Interface by providing goal-driven planning and action execution. When asked to generate a plan, it creates a PDDL domain and problem files from the Knowledge Base. As Figure 1 illustrates, the Goal Manager gets the information for generating PDDL files from various sources; the PDDL domain file is derived from the Action Stack, the initial state for the PDDL problem file is derived from the Knowledge Base, and the goal state for the PDDL problem file comes from the goal set for the Action Execution component. The Goal Manager uses the MetricFF planner [14] to generate a PDDL plan (MetricFF was used because it is extremely efficient and has support for numeric fluents which was critical for the Polycraft domain). If planning is successful, the PDDL plan is converted to actions the agent needs to perform. For every action, the component calls the appropriate function in the Polycraft Interface with the required parameters to perform the action in the environment. Upon a successful call to that function, the component also updates the Knowledge Base with the expected effects of the action so that it accurately reflects the agent's expectations after the action performance. Finally, the component also requests the Polycraft Interface to perform a sensory action ("SENSE_ALL") to verify that the environment does indeed match Knowledge Base's expectations and that there are no novelties. At any point the component also allows for the process to be interrupted by retraction of current goal or addition of new goals. This provides a useful abstraction for Novelty Handling which can make changes to the Knowledge Base and then request the component to run experimental goals by setting them as intermediary goals.

5.6 Novelty Handling

The Novelty Handling component implements the theoretical framework from Section 4. It does so by leveraging novelty assertions as discussed above. When non-empty sets of additions and retractions are obtained after considering the inferences for the assertion (as described in Section 4), the agent has indeed detected a novelty. Novelty characterization and novelty accommodation then simply follow from additions to and retractions from the knowledge base as discussed before.

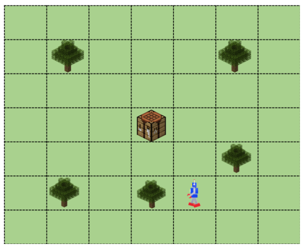
6 EXPERIMENTAL EVALUATION & RESULTS

Our approach was evaluated internally, using a combination of the actual Polycraft environment [22] and an internal "clone grid-world domain"[30] (Fig. 3a) that mimicked the Polycraft API. Internal experiments were done as regular testing of the system architecture and to verify key behavioural aspects of the agent towards novelties. External evaluations were performed by an independent team that was not informed of the agent's architecture or abilities. They devised 180 novel scenarios in the original Polycraft environment for the "pogo stick" task and evaluated the agent on being able to detect and report novelty and finish the task despite the novelty. For both evaluations, the agent's knowledge base initially populated all knowledge required for the agent to plan how to craft a pogo stick (without any novelties occurring). The *Action Stack* included information about typical preconditions and effects of known actions and the *Belief Stack* contained relevant facts and rules such as the fact that the material "bedrock" is unbreakable.

6.1 Internal Experiments

We introduced novelty into the "clone" environment to be able to show how it affects the agent's knowledge and task performance. For illustration, we here discuss one such case where a fence around the crafting table is introduced as soon as the agent has acquired all the ingredients needed to craft the pogo stick (see Fig. 3b). The "fence" is an entirely new material that the agent has never seen before, which appears in all locations adjacent to the crafting table.

After the agent initially performs the first "SENSE_ALL" action to gather information about the environment's initial state; it forms a plan and starts to execute its actions. Upon successful execution


(a) Default Environment

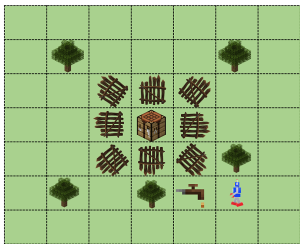

(b) With Fence Novelty

Figure 3: Illustrations of the domain used for internal evaluations. Figure (a) shows the default case of the environment with trees, crafting table and an agent facing a tree. Figure (b) shows the environment in the post-novelty scenario.

of an action, it updates its knowledge base with the known effects of the action and performs a "SENSE_ALL" action to verify that there are no novelties.

After acquiring the last ingredient needed for the pogo stick, the fence pops up and the percept is included in the results of the "SENSE_ALL" action, allowing the agent to detect the discrepancies between its prediction and the environment. Specifically, the predicate "at(fence,<loc>)" is true for each location adjacent to the crafting table which, according to the knowledge base, were supposed to be "at(air,<loc>)", and thus require an update. In addition, the predicate "material(fence)" needs to be added to the knowledge base because a fence has not been seen before. A "REPORT_NOVELTY" action is also called to report the detection of the novelty for evaluation. Using the updated knowledge base, the agent replans, assuming the fence is breakable (as every other material that is not explicitly marked as "unbreakable"). Hence, the new plan contains an action to break through the fence while navigating to the crafting table. The agent is finally able to craft the pogo stick and finish the task. Had the fence not been breakable, the agent would have detected it when trying to break it and thus reported it as another novelty.

To further illustrate the agent's novelty handling, suppose it expects a fence to surround the crafting table when rubber (one of the pogo stick ingredients) is obtained. We can implement this by modifying the known effects of the "EXTRACT_RUBBER" action. In this case, the agent's plan includes an action to break through the fence while navigating to craft the pogo stick because it expects a fence to be erected around it as soon as it acquires rubber. However, when the agent obtains rubber, and no fence is observed in the environment, novelty detection, reporting, and accommodation proceed similar as above, except this time, just the post-conditions are updated, and there is no need for adding knowledge about new objects. The agent replans once again, without an action to break through the fence, again able to complete its task. Table 1 shows performance results (number of steps and run-time) for the combinations of the configurations averaged over 50 trials. The results are as expected though it is interesting to note that it takes the agent more steps to finish the task in the fenced environment when it is expecting it than when it is not expecting it. This is most likely due to the fact that the increased PDDL model complexity results in the planner producing less optimal plans.

Agent	Environment	# Steps	Time(s)
Not Expecting Fence	Default	87.7	55.26
Not Expecting Fence	Fenced	90.4	71.85
Expecting Fence	Default	96.0	78.07
Expecting Fence	Fenced	94.8	67.07

Table 1: Results from internal experiments

Performance Metrics	Mean±SD/ [min|med|max]
M1: Novelty Detection Precision	100% ± 0%
M2: Novelty Detection Recall	84.5% ± 36.2%
M3: Novelty Detection Combined	92.3% ± 18.1%
M4: Novelty Reaction Performance	56% [0%|82%|99%]

Table 2: Overall results from external evaluations

6.2 External Evaluations

The external evaluations were performed by a third-party team that created novelties in the original Polycraft domain. According to their metrics, our agent was evaluated on two levels of novelties as described below.

- Level 1 [Class]: A new class of object is introduced in the environment. This class of object has never been seen by the agent before. An example of this novelty level can be a "silver axe" or a "wooden axe". It can appear as an entity in the agent's environment to use it, or a recipe to craft this object can be provided to the agent. In this novel scenario, the agent must learn to explore the new item to understand its importance in task completion.
- Level 2 [Attribute]: The property of an existing object changes. An example of this scenario can be a change in the material of the wall in the environment.

Each of the above levels had three variants, five types with two sub-types and three difficulty levels (easy, medium, and hard). The difficulty levels denoted the difficulty of using the novelty. For example, if the novelty involved the addition of a wooden axe, an "easy" difficulty means the axe was available as an object in the inventory, a "medium" difficulty means the axe was lying on the ground and available to be picked up, and a "hard" difficulty means only a recipe for making an axe is available to the agent. In total, there were 180 novel scenarios in which the agent was tested, and results of 100 instance trials were reported. Furthermore, in each trial, the environment was reset and the agent asked to craft a pogo stick 1000 times. Initially in the 1000 runs, the environment contains no novelties but at some random point that switches and subsequent resets of the environment present the novelty scenario. No information about the novel scenarios was revealed to our team before evaluations and similarly the evaluation team had no prior knowledge of our agent architecture.

The overall results are shown in Table 2. Our agent was evaluated on 4 performance metrics. Performance metric M1, is the precision score of agent's novelty detection. The second metric M2, is the recall score of the agent's novelty detection. The performance metric M3, provides the combined score for novelty detection (mean of precision and recall) and M4 denotes the percentage of tasks the agent was able to complete despite novelties (novelty reaction).

Novelty Level 1: Class			
Metrics	**Easy**	**Medium**	**Hard**
	Mean±SD/ [min|med|max]	Mean±SD/ [min|med|max]	Mean±SD/ [min|med|max]
M1	100% ± 0%	100% ± 0%	100% ± 0%
M2	98.4% ± 12.5%	100% ± 0%	89.8% ± 30.2%
M3	99.2% ± 6.2%	100% ± 0%	94.9% ± 15.1%
M4	91% [73%|97%|99%]	86% [24%|97%|99%]	77% [0%|97%|99%]
Novelty Level 2: Attribute			
M1	100% ± 0%	100% ± 0%	100% ± 0%
M2	68.1% ± 46.6%	64.0% ± 48.0%	65.3% ± 47.6%
M3	84.1% ± 23.3%	82.0% ± 24.0%	82.7% ± 23.8%
M4	85% [0%|97%|100%]	76% [0%|97%|100%]	76% [0%|96%|100%]

Table 3: Level-wise breakdown of Novelty detection and Novelty Reaction performance metrics from external evaluations.

A mean number with the standard deviation or minimum, median, and maximum values were reported in all the metrics. Table 3 presents a more detailed evaluation. It lists the performance on all the two levels of novelty (class and attribute) and three difficulty levels (easy, medium, and hard) individually.

The agent performed very well, achieving a perfect precision score (i.e., it never reported any false positives) and is able to overcome a lot of otherwise obstructive novelties by employing its simple inference-based novelty accommodation strategy. However, the lack of a perfect recall score was surprising – we expected the agent to at least *detect* all the novelties (given how the knowledge base is used to track objects and actions). On inspection, it turned out that the agent's knowledge about the environment and task did not include all aspects the evaluators manipulated, e.g., in one type of novelty, the material of the environment's boundary walls was changed but the agent did not have any expectations for grid walls having a particular material or even of the grid being a specific size. As a consequence, it did not consider those changes to be a novelty because it did not detect the changes in the first place (as it did not track them and did not form any expectations about them).

As can be seen from the M4 reaction scores in Table 3, the simple novelty accommodation strategy – adding inferable properties to the knowledge base, without further exploring properties of objects and action affordances – allows the agent to handle Level 1 and 2 novelties very well. For Level 1, this strategy worked just fine when the introduction of novel objects did not prevent goal accomplishment, e.g., when a novel type of tree is introduced that yields more wood (without exploring the properties of the novel tree, the agent misses the opportunity to perform the task faster, but at the same time it can still execute its original plan using the trees it knows about). Yet, if only novel trees were available, the agent would not know what to do with the tree (unless it could recognize them as "trees"). The same reasoning applies to Level 2 as well.

7 DISCUSSION

The knowledge and inference-based novelty handling mechanisms, based on our notion of novelty discussed in Section 2, demonstrated both the potential and limitation of knowledge-based approaches: when general knowledge was available that allowed the agent to make relevant inferences and predictions, detected novelties could be accommodated by adding inferences resulting from augmenting the agent's knowledge base with the observed novelty and allowed the agent to complete its task in altered environments without any further exploration (something an RL-based agent, for example, is not able to do). Yet, the evaluations also showed that active explorations of detected novelties are necessary to better and more comprehensively characterize the novelty (e.g., when a novel tool was placed in a location, the agent would not accidentally visit and thus was never picked up by the agent), especially when it comes to object affordances and actions effects that are necessary for finding plans that accomplish goals in altered environments. As such, we propose to add two heuristics to improve novelty characterization and accommodation. The first is exploration-driven: Once a novelty has been detected, we create intermediary goals associated with that novelty to discover even more potential novelties that correct any initial (possibly wrong) assumptions we made about the novelty. For example, if a new type of axe is discovered, the agent can set intermediary goals to attempt to perform different actions such as "BREAK_BLOCK" on a tree while holding this axe. If the axe causes unexpected effects, the postconditions can be corrected and properly generalized.

The second heuristic is experiment-driven. Once a novelty is detected and the agent has multiple hypothesized ways to accommodate it, the agent can carry out experiments to test the effects of each hypothesis by temporarily adding a possible accommodation to its knowledge base and then setting goals based on it that it expects to either pass or fail if the hypothesis was valid (in which case the hypothesized accommodation can be kept, otherwise discarded).

While the pursuit of additional goals for both heuristics will cause the agent to suspend its pursuit of the overall task and possibly fail at it (if it has time limitations to perform it), the investment of exploring and subsequently being able to accommodate novelties will pay off in the long run when the agent either has to perform the task repeatedly (as was the case in the external evaluation) or when the novelty is relevant in other tasks the agent needs to perform.

8 CONCLUSION

We introduced a novelty handling framework to enable agents to detect, characterize, and accommodate novelties in open-world settings and implemented the framework in a cognitive agent. The agent's novelty handling abilities were thoroughly evaluated in our own proxy simulations and by a third party in the Polycraft domain where the agent had to craft a "pogo stick". The results demonstrated that the proposed knowledge-based and inference-based novelty handling methods are effective for detecting novelties and accommodating many different types of novelty. However, they also showed that additional methods are needed for more fully characterizing and accommodating novelties, especially ones that are needed to improve task performance or enable it in the first place (when unexpected changes in the environment make goal accomplishment impossible without utilizing the novelty).

ACKNOWLEDGMENTS

This work was funded in part by DARPA grant W911NF-20-2-0006. The authors would also like to thank Eric Kildebeck, Walter Voit, and the other members of the UT Dallas evaluation team for the external evaluation results.

REFERENCES

[1] Javier Segovia Aguas, Jonathan Ferrer-Mestres, and Anders Jonsson. 2016. Planning with Partially Specified Behaviors.. In *CCIA*. 263–272.

[2] Maruan Al-Shedivat, Trapit Bansal, Yuri Burda, Ilya Sutskever, Igor Mordatch, and Pieter Abbeel. 2017. Continuous adaptation via meta-learning in nonstationary and competitive environments. *arXiv preprint arXiv:1710.03641* (2017).

[3] Leonardo Amado, Ramon Fraga Pereira, Joao Aires, Mauricio Magnaguagno, Roger Granada, and Felipe Meneguzzi. 2018. Goal recognition in latent space. In *2018 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 1–8.

[4] Virgil Andronache and Matthias Scheutz. 2006. ADE—an architecture development environment for virtual and robotic agents. *International Journal on Artificial Intelligence Tools* 15, 02 (2006), 251–285.

[5] Ankuj Arora, Humbert Fiorino, Damien Pellier, Marc Métivier, and Sylvie Pesty. 2018. A review of learning planning action models. *The Knowledge Engineering Review* 33 (2018).

[6] Masataro Asai and Alex Fukunaga. 2017. Classical planning in deep latent space: Bridging the subsymbolic-symbolic boundary. *arXiv preprint arXiv:1705.00154* (2017).

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

[8] Yash Chandak, Georgios Theocharous, Shiv Shankar, Sridhar Mahadevan, Martha White, and Philip S Thomas. 2020. Optimizing for the Future in Non-Stationary MDPs. *arXiv* (2020), arXiv–2005.

[9] Nuttapong Chentanez, Andrew G Barto, and Satinder P Singh. 2005. Intrinsically motivated reinforcement learning. In *Advances in neural information processing systems*. 1281–1288.

[10] Chelsea Finn, Aravind Rajeswaran, Sham Kakade, and Sergey Levine. 2019. Online Meta-Learning. In *ICML*.

[11] Marc Hanheide, Moritz Göbelbecker, Graham S. Horn, Andrzej Pronobis, Kristoffer Sjöö, Alper Aydemir, Patric Jensfelt, Charles Gretton, Richard Dearden, Miroslav Janícek, Hendrik Zender, Geert-Jan M. Kruijff, Nick Hawes, and Jeremy L. Wyatt. 2017. Robot task planning and explanation in open and uncertain worlds. *Artif. Intell.* 247 (2017), 119–150. https://doi.org/10.1016/j.artint.2015.08.008

[12] Andreas Herzig. 2014. Belief Change Operations: A Short History of Nearly Everything, Told in Dynamic Logic of Propositional Assignments.. In *KR*.

[13] Todd Hester and Peter Stone. 2017. Intrinsically motivated model learning for developing curious robots. *Artificial Intelligence* 247 (2017), 170–186.

[14] Jörg Hoffmann. 2003. The Metric-FF Planning System: Translating"Ignoring Delete Lists"to Numeric State Variables. *Journal of artificial intelligence research* 20 (2003), 291–341.

[15] Leslie Pack Kaelbling and Tomás Lozano-Pérez. 2013. Integrated task and motion planning in belief space. *The International Journal of Robotics Research* 32, 9-10 (2013), 1194–1227.

[16] Radimir Komarnitsky and Guy Shani. 2016. Computing contingent plans using online replanning. In *Proceedings of the thirtieth AAAI conference on artificial intelligence*. 3159–3165.

[17] George Konidaris and Andrew G Barto. 2009. Skill discovery in continuous reinforcement learning domains using skill chaining. In *Advances in neural information processing systems*. 1015–1023.

[18] George D Konidaris, Leslie P Kaelbling, and Tomas Lozano-Perez. 2014. Constructing symbolic representations for high-level planning. (2014).

[19] Iuliia Kotseruba and John K Tsotsos. 2020. 40 years of cognitive architectures: core cognitive abilities and practical applications. *Artificial Intelligence Review* 53, 1 (2020), 17–94.

[20] Anusha Nagabandi, Ignasi Clavera, Simin Liu, Ronald S Fearing, Pieter Abbeel, Sergey Levine, and Chelsea Finn. 2018. Learning to Adapt in Dynamic, Real-World Environments through Meta-Reinforcement Learning. In *International Conference on Learning Representations*.

[21] Sindhu Padakandla, KJ Prabuchandran, and Shalabh Bhatnagar. 2020. Reinforcement learning algorithm for non-stationary environments. *Applied Intelligence* (2020), 1–17.

[22] Tim Palucka. 2017. Polycraft World teaches science through an endlessly expansive universe of virtual gaming: https://polycraft.utdallas.edu. *MRS Bulletin* 42, 1 (2017), 15–17.

[23] Vasanth Sarathy, Daniel Kasenberg, Shivam Goel, Jivko Sinapov, and Matthias Scheutz. 2021. SPOTTER: Extending Symbolic Planning Operators through Targeted Reinforcement Learning. In *Proceedings of the 2021 International Conference on Autonomous Agents & Multiagent Systems*.

[24] Paul W Schermerhorn, James F Kramer, Christopher Middendorff, and Matthias Scheutz. 2006. DIARC: A Testbed for Natural Human-Robot Interaction.. In *AAAI*, Vol. 6. 1972–1973.

[25] M. Scheutz, T. Williams, E. Krause, B. Oosterveld, V. Sarathy, and T. Frasca. 2019. An Overview of the Distributed Integrated Cognition Affect and Reflection DIARC Architecture. In *Cognitive Architectures*, M. Aldinhas Ferreira, J. Silva Sequeira, and R. Ventura (Eds.). Intelligent Systems, Control and Automation: Science and Engineering, Vol. 94. Springer.

[26] Siddharth Srivastava, Eugene Fang, Lorenzo Riano, Rohan Chitnis, Stuart Russell, and Pieter Abbeel. 2014. Combined task and motion planning through an extensible planner-independent interface layer. In *2014 IEEE international conference on robotics and automation (ICRA)*. IEEE, 639–646.

[27] Richard S Sutton, Doina Precup, and Satinder Singh. 1999. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence* 112, 1-2 (1999), 181–211.

[28] Kartik Talamadupula, J. Benton, Subbarao Kambhampati, Paul Schermerhorn, and Matthias Scheutz. 2010. Planning for Human-Robot Teaming in Open Worlds. *ACM Transactions on Intelligent Systems and Technology* 1, 2 (2010), 14:1–14:24.

[29] Kartik Talamadupula, Gordon Briggs, Matthias Scheutz, and Subbarao Kambhampti. 2017. Architectural Mechanisms for Handling Human Instructions for Open-World Mixed-Initiative Team Tasks and Goals. *Advances in Cognitive Systems* 5 (2017).

[30] Gyan Tatiya. 2020. Novel Gridworlds Environment for OpenAI Gym. https://github.com/gtatiya/gym-novel-gridworlds.

[31] Annie Xie, James Harrison, and Chelsea Finn. 2020. Deep Reinforcement Learning amidst Lifelong Non-Stationarity. (2020).