A Unified Cognitive Architecture for Embodied Intelligent Agents

Pat Langley PATRICK.W.LANGLEY@GMAIL.COM
Institute for the Study of Learning and Expertise, Palo Alto, CA 94306 USA

Edward P. Katz EDPKATZ@STANFORD.EDU
Stanford Intelligent Systems Laboratory, Stanford University, Stanford, CA 94305 USA

Abstract

In this paper, we examine the PUG/U cognitive architecture for embodied intelligent agents. We describe the framework's assumptions about mental representations and processes, focusing especially on its unified approach to task planning, motion planning, continuous control, and conceptual inference. After this, we report demonstrations of the architecture's behavior on four scenarios from a simulated robotics domain. In conclusion, we discuss how our theory relates to other frameworks and consider promising directions for future research.

1. Introduction

Research on cognitive architectures (Laird, Langley, & Rogers, 2009) aims to develop unified theories of the mind. Since they were introduced in the 1970s, the field has seen substantial progress, with the community developing a variety of architectural frameworks. Some efforts have focused on fitting specific models to observed human behavior on particular tasks, but others, more relevant to the cognitive systems agenda, have developed intelligent agents that operate in complex settings. One drawback of the cognitive architecture movement has been its focus on high-level cognition, which meant that it allocated less attention to interactions with environment. In this sense, it was closely aligned with other early work in AI and cognitive psychology, which emphasized topics like problem solving, reasoning, and language rather than ones like perception and action. But it also meant that architectural research, although acknowledging the latter's import, did not incorporate them into its unified theories of mental representation and processing.

To some extent, the architecture community has addressed this issue by interfacing frameworks with existing techniques for with environmental interaction. For example, when Mininger and Laird (2019) connected Soar to a robot, their agent invoked a PID controller to handle closed-loop execution. Similarly, Trafton et al. (2013) reported linking ACT-R to modules for visual processing and motor execution that let their agent operate a robotic platform. However, such efforts at *integration* run counter to Newell's (1990) vision for *unified* theories of intelligence. In this paper, we report progress toward this original objective using PUG, an architecture for embodied agents that shares

features with classic frameworks but also has key differences. In the next section, we describe some target abilities that we hope to achieve and constraints on our approach. After this, we describe PUG's cognitive structures and the mental processes that manipulate them, using examples from a simulated rover domain as illustrations. Next we report demonstrations of the architecture's behavior in this domain. Finally, we discuss related work on both cognitive and robotic architectures, as well as promising avenues for additional research.

2. Target Abilities and Constraints

We desire a computational theory that supports not only high-level cognition like symbolic planning, but also generation of detailed motion plans and continuous control. The framework should let an intelligent agent exhibit goal-directed in an external environment, which means that it must draw inferences based on its perceptions, make predictions about the effects of its actions, and calculate utilities to let it select among options. We can clarify these target abilities with a motivating example. Consider a planetary rover that carries out missions on which it must deliver sensors to target sites, collect samples it encounters for analysis, and avoid obtacles that arise during its journeys. This setting requires not only the capabilities listed above, from high-level task planning to low-level state inference, but also their combination into a single cognitive system.

Many robotic systems support such scenarios, but few of them take the form of cognitive architectures, which make strong theoretical assumptions about the mind. Such frameworks typically borrow ideas from cognitive psychology, such as that short-term memories are distinct from long-term stores and that both contain modular symbol structures. Moreover, cognitive processing occurs in recognize-act cycles, on each pass matching long-term memories against the contents of dynamic ones. Furthermore, cognition involves the dynamic composition of these mental structures, with multi-step inference and problem-space search being important examples. Finally, a cognitive architecture usually comes with a programming language for constructing knowledge-based agents, with its syntax reflecting theoretical assumptions about representation.

In previous publications, we have reported on PUG, a cognitive architecture that supports many of these target behaviors. However, earlier versions fell short along certain dimensions. The initial incarnation (Langley et al., 2016) combined symbolic relations with numeric attributes and used mental simulation to evaluate plans, but it could carry out only one action at a time and its forward-search mechanism was not goal driven. A more recent variant, PUG/C, enabled continuous control and motion planning (Langley & Katz, 2022), but the implemented software was not fully integrated with a symbolic task planner. In the sections that follow, we describe PUG/U, a more fully unified architecture that extends these predecessors.

3. Cognitive Structures in the PUG/U Architecture

Theoretical assumptions about mental structures play a central role in a cognitive architecture, so we should examine PUG/U's distinctive representational tenets before considering the processes that interpret them. We will divide our treatment into long-term, generic knowledge structures and the dynamic elements that draw upon and instantiate them.

3.1 Generic Knowledge Structures

The architecture incorporates four distinct types of generic long-term knowledge elements that deal with different types of agent expertise. Let us examine each of them in turn.

Conceptual knowledge. The PUG/U framework encodes knowledge about classes of entities, and relations among them, as *concepts*. Each concept is defined by a separate rule, similar to a Prolog clause, that includes a head and a set of antecedents. These *ground* symbols in terms of *percepts*, each of which describes quantitative attributes of objects in the environment. Moreover, concepts are *graded* in that they match against particular situations to greater or lesser degrees, with each match having an associated *veracity* that denotes this quantity.

Table 1 (a) shows two *conceptual rules* for a two-dimensional rover domain like that outlined earlier. These define the relations *robot-at* and *robot-oblique*. Each rule has a head that specifies a predicate and a set of attribute values, including an identifier for the relation. In addition, an *:elements* field describes a set of typed objects, each with an identifier and numeric attribute values, and an optional *:tests* field with Boolean tests that must be satisfied for the concept to match. The *:veracity* field specifies how to compute the concept's degree of match as a function of bound variables, supporting the notion of graded category membership. An optional `:binds` field introduces new variables defined as arithmetic combinations of ones in the `:elements` field that can be used in the head or the `:tests` field.

We should clarify that PUG can use this syntax to encode many different conceptual relations. The concepts and beliefs in Table 1, which focus on the robot's distance and angle to an object, are only examples relevant to the rover domain. The main limit on a belief's attributes is that they be computable efficiently from values the agent can predict or perceive directly. These are not part of the architecture and a developer can use whichever attributes seem most appropriate. The framework supports the specification of three-dimensional relations (e.g., in spherical coordinates) and attributes that describe change (e.g., linear and rotational velocity). Moreover, the notation can define complex relations in terms of simpler ones, including composite objects with components whose attributes satisfy certain constraints.

Skill knowledge. In addition, PUG uses *skills* to represent knowledge about how to achieve the agent's objectives. Again, these *ground* symbols that refer to actions in terms of percepts, concepts, and their associated numeric attributes. Each skill specifies a graded *target concept* that it aims to achieve, which can match to a greater or lesser degree. It also includes equations for *control attributes* that depend on the *mismatch* between the target concept and the agent's belief state. This number serves as an error signal that supports continuous control, although it is linked to a symbolic description similar to that found in STRIPS operators.

Table 1 (b) gives two sample skills from the robot domain, one for moving toward a given object and another for turning toward an object. Each skill has a head that specifies a name and set of arguments, along with an *:elements* field that describes the arguments' types and values for a subset of their attributes. As in conceptual rules, a *:tests* field includes Boolean tests that must be satisfied for the skill to apply. Most important, a skill specifies a *:target* relation that it aims to achieve and a *:control* field with expressions for computing values for control attributes as a function of the degree to which the target concept is mismatched (i.e., one minus the target's *:veracity* score).

Table 1. (a) Two PUG concepts for the rover domain, each of which includes a head, observed elements, and a veracity function based on the entities' attributes. The *cond* statement returns different veracities depending on the bound values of variables. (b) Two skills for the robot domain, each of which includes a relational head, a set of observed entities, arithmetic tests, one or more control equations, and a target concept. The *move-to* skill influences both the attributes *move-rate* and *turn-rate*, while *turn-to-oblique* affects only *turn-rate*.

```
(a) ((robot-at ^id (?r ?o) ^distance ?d)
      :elements   ((robot ^id ?r ^radius ?rr)
                   (object ^id ?o ^distance ?d ^radius ?or))
      :binds      ((?b (- ?d (+ ?rr ?or))))
      :veracity   (cond ((> ?b 10.0) 0.0)
                        ((= ?b 0.0) 1.0)
                        (t (/ (- 10.0 ?b) 10.0))))

    ((robot-oblique ^id (?r ?o) ^angle ?a)
      :elements   ((robot ^id ?r) (object ^id ?o ^angle ?a))
      :veracity   (cond ((> ?a 45.0) (- 1.0 (/ (- ?a 45.0) 135.0)))
                        ((< ?a -45.0) (- 1.0 (/ (+ ?a 45.0) -135.0)))
                        (t 1.0)))
```

```
(b) ((move-to ?r ?o)
     :elements   ((robot ^id ?r ^turn-rate ?t) (object ^id ?o ^angle ?a))
     :conditions ((robot-oblique ^id (?r ?o)))
     :control    ((robot ^id ?r ^move-rate (* 0.5 $MISMATCH))
                  ^turn-rate (* 1.0 ?a $MISMATCH)))
     :target     ((robot-at ^id (?r ?o))))

   ((turn-to-oblique ?r ?o)
     :elements   ((robot ^id ?r)
                  (object ^id ?o ^angle ?a ^distance ?d))
     :control    ((robot ^id ?r ^turn-rate (* 10.0 (sign ?a) $MISMATCH)))
     :target     (robot-oblique ^id (?r ?o)))
```

Motivational knowledge. The PUG framework represents knowledge about the agent's goals or objectives as a collection of *motives*, which it also stores in a rule-like format. Each of these structures specifies how to compute the utility of a belief based on the agent's inferences about its situation. These beliefs need not have a high veracity and may not even have been inferred on the current cycle, so although motives are similar to concepts in their syntax, they provide support for *goal reasoning* rather than belief inference.

Table 2 (a) presents two motives from the robotics domain that specify utility functions for the relations *robot-at* and *approaching*. These functions refer to variables bound in the conditions, such as the robot's radius *?rr*, the object's radius *?or*, and the robot's distance *?od* to the object. Because some matched values will change over time, the computed value may vary in response. The first structure is an achievement motive, which generates utility only when it first matches for a particular belief above a veracity threshold. The second example is a maintenance motive, which assigns utility to a belief repeatedly on every cycle for which its conditions match. The latter type often specify negative values, which means that the agent should avoid them if possible.

Table 2. (a) Two PUG motives for the rover domain, which specify utility functions for beliefs in their heads and indicate whether they address achievement or maintenance goals. (b) Two processes that describe the effects of control attributes on the robot's *distance* and *angle* to an object. The symbols $*da$ and $*da$ denote trigonometric functions for computing linear and angular change.

```
(a) ((robot-at ^id (?r ?o))
      :conditions ((robot ^id ?r ^radius ?rr)
                   (object ^id ?o ^type target ^distance ?d ^radius ?or))
      :function   (cond ((< ?d (+ ?rr ?or 0.25)) 10.0) (t 0.0))
      :type       achievement)
    ((approaching ^id (?r ?o))
      :conditions ((robot ^id ?r ^radius ?rr)
                   (object ^id ?o ^type obstacle ^distance ?d ^radius ?or))
      :function   (cond ((< ?d (+ ?rr ?or)) -20.0) (t 0.0))
      :type       maintenance)
```

```
(b)((move-relative ?r ?o)
     :elements ((robot ^id ?r ^move-rate ?m)
                (object ^id ?o ^distance ?d ^angle ?a))
     :changes  ((object ^id ?o ^distance (*dd ?d ?a ?m) ^angle (*da ?d ?a ?m))))
   ((turn-relative ?r ?o)
     :elements ((robot ^id ?r ^turn-rate ?t)
                (object ^id ?o ^angle ?a))
     :changes  ((object ^id ?o ^angle (* -1.0 ?t))))
```

Process knowledge. Finally, the PUG framework incorporates knowledge about *processes* that it uses to predict future states. These describe how the values of control and state attributes influence the current values of state attributes, such as how turning changes the agent's angle to a given object. Processes in PUG are not the same as those in PDDL+ (Fox & Long, 2006), although there are some similarities. They specify the dynamic effects of causal attributes, reserving operator-like skills to encode details about how to achieve goals.

Table 2 (b) provides two examples of PUG processes. These include an *:elements* field that describes arguments and types, along with optional *:conditions* that must match against beliefs and *:tests* that specify Boolean expressions. The key differences from skills are that they specify no control equations and they lack a target concept, as they are not teleological in character. Instead, they incorporate a *:changes* field that specifies how values for attributes of one or more entities will change as a function of variables matched in the *:elements* field. Thus, they encode causal knowledge about the effects of actions or environmental forces.

3.2 Instantiated Mental Structures

Generic structures in PUG are static, but the architecture also needs dynamic ones that describe the agent's evolving mental state. Unlike other architectures, the framework requires that each such element be an *instance* of some generic structure. The system also organizes these elements into

Table 3. A PUG task plan for the rover domain that specifies how the robot R1 can deposit sensor S1 at target site O1. The plan involves two down subproblems, one to turn the robot to an oblique angle with respect to O1 and another to reposition it there. Each subplan specifies the initial state, goals, a motion plan M, the state before M is carried out, the state after this occurs, and a final state. Because this solution involves no right subplans, the *after* and *final* states are the same. Two motion plans include a single intention, but one has two intentions to let the robot avoid an obstacle on its path to the target.

```
Initial: ((robot-grasping ^id (R1 S1))
Goals:   ((sensor-at ^id (S1 O1)))
Down: Initial: ((robot-grasping ^id (R1 S1))
      Goals:   ((robot-grasping ^id (R1 S1))(robot-at ^id (R1 O1)))
      Down: Initial: ((robot-grasping ^id (R1 S1))
            Goals:   ((robot-oblique ^id (R1 O1)))
            Mplan:  ((turn-to-oblique R1 O1))
            After:  ((robot-oblique ^id (R1 O1))(robot-grasping ^id (R1 S1)))
            Final:  ((robot-oblique ^id (R1 O1))(robot-grasping ^id (R1 S1)))
      Before: ((robot-oblique ^id (R1 O1))(robot-grasping ^id (R1 S1)))
      Mplan:  ((move-toward R1 O1)(avoid-on-left R1 O2)
      After:  ((robot-at ^id (R1 O1))(robot-facing ^id (R1 O1))
               (robot-oblique ^id (R1 O1))(robot-grasping ^id (R1 S1)))
      Final:  ((robot-at ^id (R1 O1))(robot-facing ^id (R1 O1))
               (robot-oblique ^id (R1 O1))(robot-grasping ^id (R1 S1)))
Before: ((robot-at ^id (R1 O1))(robot-facing ^id (R1 O1))
         (robot-oblique ^id (R1 O1))(robot-grasping ^id (R1 S1)))
Mplan:  ((deposit-at R1 S1 O1))
After:  ((sensor-at ^id (S1 O1))(robot-at ^id (R1 O1))(robot-facing ^id (R1 O1))
         (robot-oblique ^id (R1 O1))(robot-grasping ^id (R1 S1)))
Final:  ((sensor-at ^id (S1 O1))(robot-at ^id (R1 O1))(robot-facing ^id (R1 O1))
         (robot-oblique ^id (R1 O1))(robot-grasping ^id (R1 S1)))
```

larger-scale structures that support high-level cognition. Because we are focused here on planning for embodied agents, we will start will start with the highest level and work downward from there.

Task plans and search trees. The architecture's topmost structures are *task plans*, which take the form of trees that decompose problems into subproblems and their associated solutions. Each element of these AND trees has four primary components:

- A *problem* statement with a belief state and goal description;
- A *motion plan* M that contains a set of agent intentions;
- An optional *down subplan* to be solved before executing M;
- An optional *right subplan* to be solved after executing M.

Each element in a task plan also describes intermediate states that would be generated by its down subplan and motion plan if the agent carried them out in the environment.

Table 3 presents a sample hierarchical task plan for the rover domain that involves a top-level problem – depositing the sensor *S1* at target site *O1* and two subplans. In this case, both subplans involve *Down* subproblems that aim to achieve the conditions of the first intention shown in the

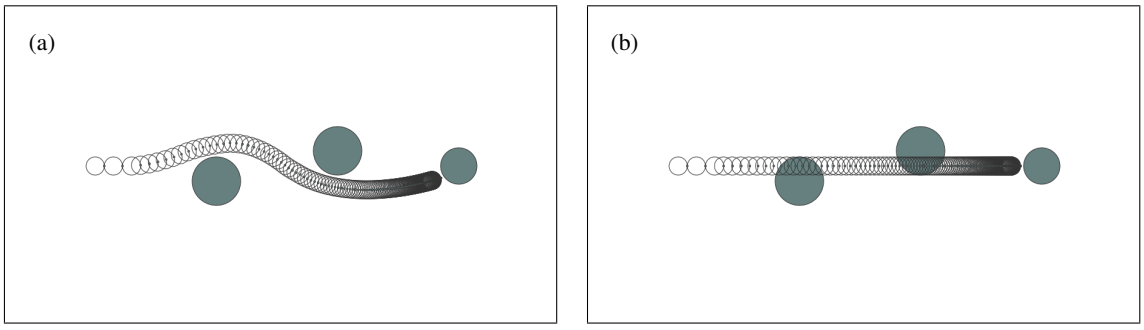

Figure 1. Trajectories for two motion plans in which a robot approaches a target object: (a) with the intentions *((move-to R1 O3) (avoid-on-left R1 O1) (avoid-on-right R1 O2))*, which leads it to swerve around obstacles *O1* and *O2*; and (b) with the single intention *((move-to R1 O3))*, which causes it to run into both obstacles.

associated motion plan (in the *Mplan* field). The *Before* field shows the belief state before carrying out a given motion plan and the *After* field shows the state that it would produce. The table omits the numeric attribute values associated with each belief, as they are not relevant at this level of description. In this example, one of the motion plans involves two intentions, as a second is required to avoid an obstacle. This decompositon is similar to ones generated by means-ends analysis (Newell et al., 1960), but it can also encode plans with only right-branching trees.

We say that a PUG plan is *complete* if it includes solutions to each down subproblem, thus enabling application of its motion plans. However, a plan can be complete even if it does not achieve all top-level goals, and thus has an unsolved right subproblem, as can occur when there are limited resources or inherent tradeoffs. Each complete plan has an associated utility that combines the utilities of its constituent motion plans. Of course, the architecture also generates *partial* plans along the way, which it organizes into an OR tree that supports its search through the space of problem decompositions, as we describe later. For instance, the root node specifies only the state and goals of the original problem, whereas its immediate children specify alternative motion plans but not how to solve their subproblems, if there are any.

Motion plans and mental simulations. As noted above, each task plan and subplan refers to a *motion plan* that, if executed, should achieve one of the elements in the associated goal description. Each such motion plan includes:

- An *initial state* from which activity begins;
- A *target belief* that serves as the plan's goal;
- A *primary intention* that aims to achieve this goal;
- *Supporting intentions* that increase plan utility; and
- Supplementary information about plan duration and utilities.

Each intention in a motion plan specifies a skill and the entities to which it applies, but it does not include the values of numeric attributes, as these will vary over the course of the trajectory.

Figure 1 (a) shows the trajectory for a motion plan with the primary intention *(move-to R1 O3)*, which has the associated goal *(robot-at ^id (R1 O3))*. There are two supporting intentions, *(avoid-on-left R1 O1)* and *(avoid-on-right R1 O2)*. The primary intention leads the robot *R1* to move toward *O3*, whereas the others cause it to veer around obstacles *O1* and *O2*. The latter intentions becomes inactive once the robot has passed the obstacles, but the first remains in play until it achieves the primary goal. A motion plan does not store its trajectory because, as explained later, the initial state and intentions suffice to generate it deterministically. However, it does retain the final state, which corresponds to the *After* state in the task plan. It also stores the sum of utilities in the trajectory's states and the number of steps, which let it compute average utility. In addition, it includes states that have negative utilities and the motives responsible, which are needed for later processing.

As with task plans, PUG/E organizes candidate motion plans for a given target belief into a search tree. The root node includes only the primary intention that, if executed, should achieve that goal. The children of this node each include an additional intention that produces a distinct trajectory, typically with a different utility. Each of the grandchildren introduces a second intention, and so forth, with one of the terminal nodes encoding the final motion plan associated with a task plan or subplan that it supports. For comparison, Figure 1 (b) shows the trajectory for a plan with *(move-to R1 O3)* as its only intention. This course of action produces some states with negative utility because the robot collides with the obstacles along its path.

Beliefs, intentions, predictions, and states. At the lowest level are beliefs and intentions, which provide the content for both task and motion plans. As noted earlier, beliefs are instances of generic concepts that include particular entities in their ^id field and specific values for other attributes. Each belief also includes a *veracity* that denotes the degree of match to its corresponding concept and a *utility* that indicates its usefulness. For instance, two percepts at the start of the trajectory in Figure 1 (a) would be *(robot ^id R1 ^radius 0.15)* and *(object ^id O1 ^distance 2.0156 ^angle –7.125 ^radius 0.4)*, both with *veracity* 1.0. Inferred beliefs would include *(robot-at ^id (R1 O1) ^distance 2.0156)* and *(robot-oblique ^id (R1 O1) ^angle –7.125)*, with veracities 0.853 and 1.0, respectively. In this scheme, a goal is simply a belief that has a utility with high absolute value.

Similarly, intentions are instances of skills that include particular entities as arguments, specific values for attributes, and activation levels that depend on the degree of mismatch to target beliefs. The architecture also encodes predictions about changes to entities' attributes, which are instances of processes, but it does not retain them for later use. Beliefs and intentions are organized into *states* that describe the environment on particular cycles. The beliefs in a state characterize the situation that holds on its associated cycle, whereas the intentions indicate what actions the agent takes to change this situation. Note that states can describe either imagined situations and actions that the agent generates through planning or actual situations and actions based on its perception. The two types of content rely on the same cognitive structures and they play similar roles in the two settings.

4. Cognitive Processes in the PUG/U Architecture

Now that we have described the framework's mental structures, we can discuss the processes that inspect and manipulate them. Like other cognitive architectures, PUG/U operates in discrete cycles that match generic knowledge against dynamic ones. An important difference is that it relies on four different levels of temporal resolution, which we will address in turn.

4.1 Task Planning

The task planning module carries out heuristic search through a space of problem decompositions seeking one or more hierachical plans that achieve its top-level goals. PUG begins with a root note that specifies an initial state and a set of achievement goals, each with an associated utility. The system uses a variety of means-ends analysis to recursively break this problem into subproblems, exploring different decompositions and adding new nodes to the search tree as it proceeds. This problem-solving activity continues until it finds enough solutions with positive utility or until it exceeds allocated resources, such as number of nodes or CPU time.

The module operates in discrete cycles, on each pass deciding whether it has found enough acceptable plans or there are no remaining options, in which case it halts and returns the solutions found. Otherwise, if the plan P for the current search node N is an acceptable solution, then it stores P, but if P is unacceptable (e.g., involves a state or goal loop), then it marks N as failed, in either case selecting a new node. The search process revolves around the *focus* problem, which is a subtask in the current node that does not yet have a decomposition. This may be a down subproblem or a right subproblem, but the latter can become the focus only after solving the former.

If the focus problem F has no candidate intentions, then PUG retrieves intentions that are relevant to F. The default strategy favors skills with an associated target concept that unifies with one of F's unsatisfied goals, which leads to means-ends problem solving. If retrieval finds a relevant but untried intention I, then the system creates a child node C that decomposes F and checks whether I's conditions match the state. If not, then it creates a new down subproblem with these conditions as goals and attempts to solve it. On success, or if the conditions already match, the module generates a motion plan M based on I that, if needed, adds intentions to increase utility. If the motion plan's final state S satisfies the problem's goals G, the system shifts its focus to its parent problem. Otherwise it creates a new right subproblem with S as initial state and G as goal description.

We should note that a successful task plan need not achieve all top-level goals, which may not be possible for a given situation. In fact, PUG will even reject a child in the search tree that achieves more goals than its parent if this would decrease utility. PUG calculates the utility of a task plan from the utilities of its constituent motion plans. Motives that involve maintenance goals (e.g., avoiding obstacles) play a key role in these scores. Achievement goals contribute to overall utility for the top-level problem, but not for subproblems, where they serve only to establish success. The current implementation uses these scores primarily for ranking task plans, not for search guidance, although future versions should support the latter.

4.2 Motion Planning

As we have seen, the task planner selects promising intentions, but it must call on a module for motion planning to ensure that a candidate is acceptable. In our framework, a single intention, such as moving to a target object, constitutes a motion plan, but it can work well in some cases yet encounter issues when other factors come into play. For instance, if there is an obstacle between the robot and the target, then moving directly to the target will produce a collision and additional intentions are needed to produce a trajectory with acceptable utility. The process of motion planning addresses such cases by searching through a space of motion plans, each of which is a set of intentions.

The module starts with a singleton set that contains an intention produced during backward chaining on a goal by the task planner. The architecture then uses mental simulation to envision the trajectory that this intention set would produce if it were executed in the environment. For a given set of intentions, this repeatedly determines which intentions have conditions that match the current belief state, uses their *:control* field to compute values for control attributes, and uses matched processes to predict the successor state produced. Mental simulation does not require search, as the trajectory follows deterministically from an initial situation and a set of intentions. In addition, on each time step the planner uses matched motives to compute each beliefs' utility and stores a running total of the values associated with them. If some beliefs are sources of negative utility, then it attempts to retrieve new intentions that should, at least at first glance, reduce or eliminate them.

In this process, PUG favors sources of negative utility that occurred earlier in the trajectory based on stored timing statistics. For example, if the robot's simulated path intersects with two obstacles, then the module would focus on the first one encountered. Once it has selected a source belief, the motion planner matches it against the target concepts associated with stored skills, chaining backward to generate a set of relevant intentions. The system adds each candidate intention to the current motion plan, simulates it mentally, and selects the best one. However, the new trajectory may introduce further sources of negative utility (e.g., new obstacles) that require additional repairs. This process continues until it finds no way to improve on the best candidate found so far, at which point it halts and returns this set of intentions, along with its duration and overall utility.

In summary, PUG/U carries out greedy search through a space of motion plans, starting with a singleton intention set that addresses a single goal and elaborating on it to improve utility. Mental simulation produces a continuous trajectory for each node in the search tree, but every candidate comprises a discrete set of intentions. This approach differs from classic methods to motion planning, which either introduce waypoints that guide continuous control or search through a fine-grained discretized space. In contrast, our architecture combines conceptual inference, feedback control, and process-driven prediction to generate deterministic trajectories, then uses motives to assign utilities and to retrieve intentions that improve on them.

4.3 State Processing

Mental simulation relies on a third module for state processing, which examines the current belief state in light of active intentions, processes, and motives. For each intention I, the module checks to see whether its conditions match the belief state. If so, then it accesses I's target belief and the associated *:veracity* score V. The interpreter substitutes $1 - V$ for the symbol *$MISMATCH* in I's control equations, along with values for any bound variables. Next it evaluates the instantiated expressions to compute a value for each control attribute. For instance, consider the skill *move-to* in Table 2 (b). Given the intention *(move-to R1 O1)* and the *:veracity* score 0.7 for target belief *(robot-at ^id (R1 O1))*, the mismatch would be $1 - 0.7 = 0.3$, so *move-rate* would be $0.5 \times 0.3 = 0.15$ and, if the angle to *O1* is 20.0, then *turn-rate* would be $0.3 \times 20.0 = 6.0$.

When multiple intentions match in the belief state, the module computes target mismatches and control values for each one. If these intentions affect the same control attribute, such as the *move-rate* and *turn-rate*, then PUG takes the sum of their computed values, although it caps them at the maximum allowed for each attribute . For instance, suppose the agent is attempting to reach

object *O2* but the obstacle *O1* lies on its path. In this case, two intentions – *(move-to R1 O2)* and *(avoid-on-left R1 O1)* – would affect the control attribute *turn-rate*, with one indicating that the robot should turn toward *O2* and the other stating that it should turn away from *O1*.

Finally, state processing uses known processes to predict changes to the environment based on variables bound in their conditions. Some of these variables may encode the values for control attributes, but others describe only external, nonvolitional influences. When more than one process affects the same attribute for a given entity, their effects are summed, producing a set of changes anticipated from the current belief state and intentions. For instance, given the value 3.5 for the control attribute *turn-rate* and the current *angle* of -30.0 to object *O1*, the instantiated process *(turn-relative R1 O1)* would predict a new angle of -26.5 to object *O1*. Different instances of the *turn-relative* process would predict similar changes to the robot's *angle* to other objects.

4.4 Belief Processing

Both task planning and motion planning, including mental simulation, refer to beliefs that, taken together, describe the environment state at a particular time. For this reason, PUG incorporates an inference module that uses the available conceptual knowledge to generate beliefs from either observed percepts or their predicted analogs. This module also operates in cycles, on each pass finding all concept definitions with elements whose types match against current percepts. For each match, it computes derived attribute values, calculates the *veracity* score, and, if this exceeds a threshold, adds an inferred relation to belief memory.

Next the system finds conceptual rules with elements whose types match against a subset of the percepts and these inferred beliefs, leading to additional beliefs. These may incorporate additional attributes with values derived from beliefs matched in the rules' antecedents. This procedure continues until no more conceptual matches occur, giving a set of beliefs and percepts that encode the current state. For instance, given the initial scenario in Figure 1 (a), the concepts in Table 1 (a) and the percepts in Table 1 (b) would infer a number of relations between *R1* and the other three objects. In summary, the inference module computes the full deductive closure of the agent's conceptual knowledge and its percepts. Human inference is much more selective, but PUG's approach suffices for environments that involve tens of entities.

5. Empirical Demonstrations

We have implemented PUG/U in Steel Bank Common Lisp. The architecture lets users specify knowledge bases in terms of the syntax described above for concepts, skills, motives, and processes, which they can load from a file along with problem descriptions that include initial states and goals. Users also specify percepts that occur in the environment and control attributes that influence it. As noted earlier, the architecture interprets its knowledge structures in discrete cycles, calling on modules for conceptual inference, state processing, motion planning, and task planning.[1] Each run produces a trace of the agent's dynamic mental structures, along with trajectories of states.

1. An earlier version of the PUG architecture also supported execution of task and motion plans in an external environment, but our current work has emphasized mental processing.

For demonstration and testing purposes, we have used a simulated environment in which a planetary rover must deliver sensors to target sites while avoiding obstacles and dangerous areas. The generic knowledge base for this domain includes six concepts, six skills, three motives, and five processes, although not all are relevant in every scenario. The PUG task planner includes a number of parameters that modulate its behavior. For the runs reported here, we set these to produce a depth-first version of means-ends analysis, to consider candidates with no more than eight motion plans, to find three or fewer alternative solutions, and to rank the latter based on average utility.

Basic navigation. The first scenario demonstrates the ability to carry out simple navigation that combines symbolic task planning with continuous control. The particular setting involves a mobile robot *R1* and a target object *O1*, with the latter initially at a distance of 3.0 units from *R1* at an angle of 60 degrees in egocentric coordinates. In this case, the architecture generates a single task plan that first carries out the motion plan *((turn-to-oblique R1 O1))* to achieve *(robot-oblique R1 O1)*, followed by the motion plan *((move-to R1 O1)* to achieve the top-level goal *(robot-at ^id (R1 O1))*. The first of these addresses a down subproblem of the top-level task, takes 33 steps in mental simulation to achieve *(robot-oblique ^id (R1 O1))*. The second motion plan, which takes 65 steps, causes the robot to turn toward the target as it moves forward. This illustrates PUG's ability to influence two control attributes – *move-rate* and *turn-rate* – based on mismatch to its target belief, *(robot-at ^id (R1 O1))*, which previous versions did not support.

Obstacle avoidance. The second scenario demonstrates PUG's ability to avoid obstacles along the path to a target object in the context of a generated task plan. This case involves a robot *R1*, a target object *O3*, and two obstacles *O1* and *O2*. Here the architecture again generates a simple task plan, similar to the one from the first scenario, except that the top-level motion plan includes three intentions, *((move-to R1 O3) (avoid-on-right R1 O1) (avoid-on-left R1 O2))*. To find this combination, the motion planner starts with the intention *(move-to R1 O3)* proposed by the task planner, but finds the trajectory crosses *O1*, giving it negative utility. In response, it considers two possible elaborations – *(avoid-on-right R1 O1)* and *(avoid-on-left R1 O1)* – and selects the first as having higher utility. However, the new trajectory crosses *O2*, leading the module to elaborate further, in this case selecting *(avoid-on-left R1 O2)* rather than *(avoid-on-right R1 O2)*, which produces an acceptable, collision-free plan. An earlier version of PUG (Langley & Katz, 2022) produced similar results, but it was not embedded in a symbolic task planner.

Object delivery. The third scenario involves the slightly more complex task in which the rover *R1* must deliver a sensor *S1* that it is already holding to the target site *O1*. In this case, the architecture generates a single task plan that involves the robot turning to an oblique angle with respect to *O1*, then going to the target by moving forward and turning at the same time, and, on reaching the destination, depositing the sensor there. The result includes three motion plans, each with a single intention: *((deposit R1 S1 O1))* for the top-level problem, *((move-to R1 O1))* for its down subproblem, and *((turn-to-oblique R1 O1))* for the latter's down subproblem. No right subproblems arise because there is only one top-level goal. This scenario is interesting because it demonstrates PUG's ability to use process knowledge to reason about intentions' side effects, in this case that *S1* moves with the robot when the latter has it grasped. An earlier version (Langley et al., 2016) lacked this ability, as it did not separate a skill's effects from its control equations.

Plan selection. The final scenario involves two sensors, *S1* and *S2*, that the rover *R1* must deliver to target sites *O1* and *O2*, respectively. Here the architecture finds two distinct task plans, each with six constituent motion plans that contain one intention each. In the first alternative, the robot travel to *O1* and deposits *S1* there, after which it goes to *O2*, where it drops *S2*. The other plan carries out the same high-level steps but delivers *S2* to *O2* before taking *S1* to *O1*. However, the robot begins 2.0 units away from *O1* and 4.0 units away from *O2* in roughly the same direction. Thus, the plan trajectories are very different, with the second one taking considerably longer and producing a lower average utility, so that, upon comparing their scores, PUG favors the first option. This scenario illustrates the role of utility calculation in ranking plans based on their respective qualities.

6. Related Research

As noted earlier, our objective is to develop a unified cognitive architecture for embidied agents that displays a wide range of capabilities, especially the capacity to combine task planning, motion planning, continuous control, and conceptual inference. These aims overlap with those pursued by other research communities, expecially ones that integrate multiple abilities and provide formalisms for stating expertise. We can divide this work into four broad categories that we discuss in turn:

- *Robotic architectures* (Kortenkamp & Simmons, 2008) support embodied agenby by integrating state inference, motion planning, and in some cases task planning (Garrett et al., 2021). Research in this paradigm often assumes a set of specialized modules that interact by passing messages to each other, leading to distributed forms of processing. Some robotics work, such as Bonasso et al.'s (1997) 3T architecture, offers formalisms for knowledge but, unlike PUG, adopts different notations for each level of abstraction. Both approaches differ from our own, which relies on a shared memory and which provides a unified formalism that the architecture uses for task planning, motion planning, state processing, and conceptual inference.
- *Cognitive architectures*, such as ACT-R (Anderson & Lebiere, 1998) and Soar (Laird, 2012), come with high-level programming languages that let users specify modular knowledge elements to produce intended agent behavior. PUG incorporates many ideas from this paradigm, including the distinction between long-term and short-term memory and a reliance on discrete cognitive cycles. However, because these frameworks developed from theories of high-level cognition, they seldom support embodied agency at the architectural level and typically call on external routines, such as PID controllers. Thus, we maintain that PUG offers a more unified theory of embodied intelligence than most of its predecessors.
- Research in *cognitive robotics* (e.g., Ferrein & Lakemeyer, 2008; Levesque et al., 1997) provides logic-based formalisms that represent expertise about agents' goal-directed activities. Despite its emphasis on logical foundations, this paradigm favors procedural notations rather than the modular structures that PUG employs. More important, like most cognitive architectures, these systems focus on high-level planning and action selection, leaving the issues of continuous control to external routines. Again, our architecture provides an alternative account of intelligent physical agents that unifies representation and processing at multiple levels of description.

- The *spatial semantic hierarchy* (Kuipers, 2000) encodes and interprets knowledge at multiple levels of spatio-temporal abstraction, from continuous control to high-level path planning. Moreover, the framework associates quantitative control laws with qualitative regions that drive a robot to distinctive states, much like PUG's skills and the associated target concepts. This architecture comes closer to our own, although it does not include separate knowledge structures for motives and processes. On the other hand, it offers a fuller account of spatial cognition, which we have only started to address in recent work (Langley & Katz, 2024).

We should also mention that PUG/U borrows directly from the ICARUS architecture (Choi & Langley, 2018), which included a similar formalism for concepts, skills, and motives, but did not provide mechanisms for feedback control or motion planning. Our framework also borrows substantially from other research paradigms, including logical inference, continuous control, decision theory, task planning, and heuristic search, but it combines them in novel ways to produce a distinctive architecture for embodied cognitive systems.

We should emphasize that none of PUG's individual features are unique and its contribution lies in combining earlier ideas in novel ways to provide a unified theory of knowledge-based embodied agents. We will not claim that the framework as it stands supports entirely new abilities or that it provides more robust operation than alternative ones. However, PUG's formalism should ease the construction of robotic systems in much the same way that Prolog (Clocksin & Mellish, 1981) aids the creation of logic programs and PDDL (McDermott et al., 1998) simplifies the development of planning systems. One can implement effective embodied agents in C or Python, but using a high-level language will be far more efficient and take much less time.

The declarative formalism for concepts, motives, skills, and processes should also support more advanced abilities. For instance, we could extend the architecture to store traces of structures generated during inference, control, mental simulation, and planning. These in turn should support *explainable agency* (Langley, 2019), enabling systems that can answer questions about their reasons for making choices and pursuing specific plans. The formalism should also provide an inductive bias that allows sample-efficient learning of new knowledge structures, with distinct mechanisms for acquiring or revising concepts, skills, processes, and even motives (Langley, 2023). Moreover, the results of learning should be interpretable because they will be encoded in a modular notation with clear implications for behavior.

7. Concluding Remarks

Despite the progress that we have reported toward a unified architecture for embodied agents, we should extend the theoretical framework along certain fronts to provide even broader coverage. In particular, future research should concentrate on six complementary issues:

- Introducing more flexible processing for conceptual inference (e.g., abduction of postulated entities and attributes in partially observable environments) and supporting more sophisticated control schemes (e.g., replacing proportional with PID calculations);
- Augmenting the task planner to fall back on forward-chaining search, as in earlier versions of PUG (Langley et al., 2016), when means-ends analysis cannot retrieve a relevant intention, and integrating it with earlier modules for plan execution and monitoring (Langley et al., 2017);

- Incorporating temporal constraints into concepts and skills about the initiation, termination, and duration of beliefs and intentions (e.g., that some are mutually exclusive) and introduce details about timing into motives' conditions and utility functions;
- Elaborating the formalism for concepts to encode spatial knowledge about the shapes of complex objects, places that are recognizable in terms of visible landmarks, and topological maps that describe large-scale relationships (e.g., Langley & Katz, 2024);
- Organizing skills into hierarchical structures, similar to those in hierarchical task networks (Nau et al., 2003), that represent extended activities, with both sequential and parallel elements, to constrain and guide task and motion planning; and
- Supporting stochastic skills and processes that encode uncertainty about the values of control attributes and causal effects, invoking repeated mental simulations to predict trajectories and taking their probability distributions into account during utility calculations.

We should also improve the system interface so that users can trace and analyze agent behaviors more easily, as well as alter parameters for different modules. Together, these changes should make PUG a more attractive and effective architecture for developing knowledge-rich embodied agents.

In addition, we should demonstrate the framework's usefulness on physical platforms. These should include classic mobile robots that combine task planning, motion planning, and obstacle avoidance, say that deliver objects in office settings, which map well onto test cases we have already used. However, to show evidence of generality, we should also apply the architecture to manipulation tasks, like equipment assembly and object sorting, that rely on multi-jointed effectors. This will require propagating constraints along joints either by calling on routines for inverse kinematics or, preferably, by adapting skills to carry out analogous stepwise calculations within the architecture. These demonstration efforts will undoubtedly suggest more extensions that are needed to support a truly unified framework for constructing embodied agents.

Acknowledgements

The research reported in this paper was supported by Grant FA9550-20-1-0130 from the US Air Force Office of Scientific Research and by Grant N00014-23-1-2525 from the Office of Naval Research, which are not responsible for its contents.

References

Anderson, J. R., & Lebiere, C. (1998). *The atomic components of thought*. Mahwah, NJ: Lawrence Erlbaum.

Bonasso, R. P., Firby, R. J., Gat, E., Kortenkamp, D., Miller, D., & Slack, M. (1997). Experiences with an architecture for intelligent, reactive agents. *Journal of Experimental and Theoretical Artificial Intelligence*, *9*, 237–256.

Choi, D., & Langley, P. (2018). Evolution of the ICARUS cognitive architecture. *Cognitive Systems Research*, *48*, 25–38.

Clocksin, W. F., & Mellish, C. S. (1981). *Programming in Prolog*. Berlin: Springer-Verlag.

Ferrein, A., & Lakemeyer, G. (2008). Logic-based robot control in highly dynamic domains. *Robotics and Autonomous Systems*, *56*, 980–991.

Fox, M., & Long, D. (2006). Modelling mixed discrete-continuous domains for planning. *Journal of Artificial Intelligence Research*, *27*, 235–297.

Garrett, C. R., Chitnis, R., Holladay, R., Kim, B., Silver, T., Kaelbling, L. P., & Lozano-Perez, T. (2021). Integrated task and motion planning. *Annual Review of Control, Robotics, and Autonomous Systems*, *4*, 265–293.

Kortenkamp, D., & Simmons, R. (2008). Robotic systems architectures and programming. In B. Siciliano & O. Khatib (Eds.), *Springer handbook of robotics*. Berlin: Springer.

Kuipers, B. (2000). The Spatial Semantic Hierarchy. *Artificial Intelligence*, *1–2*, 191–233.

Laird, J. E. (2012). *The Soar cognitive architecture*. Cambridge, MA: MIT Press.

Langley, P., Laird, J. E., & Rogers, S. (2009). Cognitive architectures: Research issues and challenges. *Cognitive Systems Research*, *10*, 141–160.

Langley, P. (2019). Explainable, normative, and justified agency. *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence* (pp. 9775–9779). Honolulu, HI: AAAI Press.

Langley, P. (2023). Cognitive systems and theories of open-world learning. *Advances in Cognitive Systems*, *10*, 3–14.

Langley, P., Barley, M., Meadows, B., Choi, D., & Katz, E. P. (2016). Goals, utilities, and mental simulation in continuous planning. *Proceedings of the Fourth Annual Conference on Cognitive Systems*. Evanston, IL.

Langley, P., Choi, D., Barley, M., Meadows, B., & Katz, E. P. (2017). Generating, executing, and monitoring plans with goal-based utilities in continuous domains. *Proceedings of the Fifth Annual Conference on Cognitive Systems*. Troy, NY.

Langley, P., & Katz, E. P. (2022). Motion planning and continuous control in a unified cognitive architecture. *Proceedings of the Tenth Annual Conference on Advances in Cognitive Systems*. Arlington, VA.

Langley, P., & Katz, E. P. (2024). Spatial representation and reasoning in an architecture for embodied agents. *Spatial Cognition & Computation*, 1–23.

Levesque, H. J., Reiter, R., Lespérance, Y., Lin, F., & Scherl, R. B. (1997). Golog: A logic programming language for dynamic domains. *The Journal of Logic Programming*, *31*, 59–83.

Mininger, A., & Laird, J. E. (2019). Using domain knowledge to correct anchoring errors in a cognitive architecture. *Advances in Cognitive Systems*, *8*, 133–148.

Newell, A. (1990). *Unified theories of cognition*. Cambridge, MA: Harvard University Press.

Newell, A., Shaw, J. C., & Simon, H. A. (1960). Report on a general problem-solving program for a computer. *Proceedings of the International Conference on Information Processing* (pp. 256–264). UNESCO House, France: UNESCO.

McDermott, D., Ghallab, M., Howe, A., Knoblock, C., Ram, A., Veloso, M., Weld, D., & Wilkins, D. (1998). *PDDL—The Planning Domain Definition Language*. Technical Report CVC TR98003, Center for Computational Vision and Control, Yale University, New Haven, CT.

Nau, D., Au, T., Hghami, O., Kuter, U., Murdock, J., Wu, D., & Yaman, F. (2003). SHOP2: An HTN planning system. *Journal of Artificial Intelligence Research*, *20*, 379–404.

Trafton, J. G., Hiatt, L. M., Harrison, A. M., Tamborello, F., Khemlani, S. S., & Schultz, A. C. (2013). ACT-R/E: An embodied cognitive architecture for human robot interaction. *Journal of Human-Robot Interaction*, *2*, 30–55.