A Unified Formalism for Constructing Embodied Agents

Pat Langley[1], **Edward P. Katz**[2], **Mohan Sridharan**[3]

[1] Institute for the Study of Learning and Expertise, 2164 Staunton Court, Palo Alto, CA 94306 USA
[2] Stanford Intelligent Systems Laboratory, Stanford University, Stanford, CA 94305 USA
[3] School of Computer Science, University of Birmingham, Birmingham B15 2TT UK

Abstract

This paper reviews PUG, a cognitive architecture for embodied agents, with a focus on its formalism for representing expertise. This includes a rule-like notation for encoding concepts that describe states, motives that compute utilities, skills that calculate control values to achieve goals, and processes that predict changes to the environment. In each case, we review the syntax for modular knowledge elements and how the architecture uses them to generate dynamic content. We also discuss how the framework integrates them to produce teleoreactive behavior over time.

Introduction and Motivation

Embodied agents that interact with some external environment – whether they are humans, robots, or virtual characters – must support a number of distinct but interconnected abilities. These include capacities for inferring the current state, setting the agent's goals, generating plans to achieve those objectives, controlling effectors to implement its plans, and predicting their impact on future states. Many existing robotic architectures incorporate these elements, but they typically use opaque routines that make it difficult to add content. Moreover, they often integrate modules in an ad hoc manner that falls short of a theory for embodied intelligence.

We maintain that the research community would benefit from well-defined formalisms that support these component abilities and their principled integration. These can serve as programming languages that let their developers construct knowledge-rich agents for physical environments. However, they would also support more effective acquisition of this content through learning by providing an inductive bias in the same way that Prolog underpins work on inductive logic programming. In addition, such computational frameworks would ensure the interpretability of learned content and aid understanding of agent behavior.

In this paper, we present a candidate formalism for embodied expertise that satisfies these constraints. We start with a brief review of PUG, an architecture for physical agents with an associated programming language. After this, we discuss the purpose, syntax, and operation of four types of knowledge elements: concepts, motives, skills, and processes. Next, we clarify how PUG integrates these structures during cognitive processing, although we focus mainly on their representation. In closing, we consider related research and directions for future work.

Overview of the PUG Architecture

PUG (Langley et al., 2016; Langley & Katz, 2022) is a cognitive architecture that, like similar frameworks, incorporates key ideas from psychological theories. These include assumptions that: dynamic short-term memories are distinct from stable long-term stores; both memories contain modular elements that are cast as discrete symbol structures; long-term elements are accessed through pattern relational matching against short-term elements; processing revolves around discrete cycles that involve retrieval, selection, and application of knowledge structures; and cognition – both performance and learning – involves the dynamic composition of these mental structures.

Like other architectures, PUG comes with a programming language whose syntax and interpreter reflects its assumptions about representation and processing. This formalism supports the construction of embodied intelligent agents, with its syntax covering both long-term knowledge structures that are stable over time and short-term elements that change during processing. However, PUG differs substantially from most earlier frameworks in that it:

- Grounds symbolic relations in quantitative descriptions of the agent's physical situation;
- Associates numeric utilities with symbolic goals that can reflect tradeoffs among competing objectives;
- Combines discrete actions with continuous parameters to specify fine-grained activities;
- Supports adaptive control that takes into account both the agent's objectives and its situation;
- Uses numeric simulation to generate motion trajectories that guide high-level planning.

These commitments elaborate on ideas from its predecessor, ICARUS (Choi & Langley, 2018), which also focused on agents that operate in physical domains. In the sections that follow, we examine the new framework's assumptions about representing and processing different types of expertise.

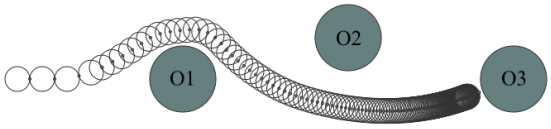

Figure 1: A scenario in which a robot must approach an object (O3) when two obstacles (O1 and O2) fall along its path, causing it to veer around them. Traces show the robot's pose at equal time intervals, indicating that position and orientation change more slowly as it approaches the target.

Figure 1 depicts a simple scenario in a two-dimensional simulated environment that we can use to illustrate PUG's formalism and its operation. The robot, denoted by the small circle, perceives its distance and angle to other objects in egocentric polar coordinates. The robot begins by facing the target object, but the direct path is blocked by two obstacles, so it skirts the first one on the left and the second one on the right before continuing to the goal. This setting requires the agent to make inferences about its current situation, generate motion plans to achieve its objective, compute the utilities of alternatives, and execute the selected plan in the environment. Langley et al. (2016) examine more complex scenarios that require task-level planning.

Concepts and Beliefs

The PUG architecture encodes knowledge about classes of entities, and relations among them, as *concepts*. Each concept is encoded by a separate rule, similar to a Prolog clause, that includes a head and a set of antecedents. These rules *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 score.

Table 1 (a) shows two *conceptual rules* for the two-dimensional robotic domain depicted in Figure 1. These define the relations *robot-at* and *robot-facing*. 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 gives 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 either the head or the *:tests* field.

Inferred *beliefs* are instances of these defined concepts that refer to perceived or imagined objects. Table 1 (b) presents examples of beliefs that describe the scenario in Figure 1 from the robot's egocentric perspective. The four percepts – which come directly from the environment – specify an object type, an object identifier, and attribute values that describe it. The latter include the object's distance and angle from the robot in agent-centered polar coordinates, along with their radii. The table also contains four inferred beliefs – *robot-at* and *robot-facing* – to the perceived objects *O1*, *O2*, and *O3*. These contain more than symbolic predicates; they include attribute values derived from constituent entities, along with *veracity* scores that reflect how well their respective concepts match. For instance, *(robot-facing R1 O2)* has the score 0.922 because the veracity expression *(– 1.0 (/ 14.032 180.0)) = 0.922*. This score, along with the belief's derived attribute values, can change over time while its symbolic aspects remain stable.

The conceptual inference module 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 belief to memory. Next the system finds rules with elements whose types match a subset of the percepts and these inferred beliefs, leading to additional beliefs. 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 scenario in Figure 1, the concepts in Table 1 (a) and the percepts in Table 1 (b) would produce the four inferred beliefs in Table 1 (b).

We should clarify that PUG can use this syntax 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 Figure 1 scenario. 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.

Motives and Utilities

The PUG framework represents knowledge about the agent's goals or objectives as a collection of *motives*, which it also encodes 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 in the current situation, so although motives are similar to concepts in their syntax, they provide support for *goal reasoning* (Aha, 2018) rather than belief inference.

Table 2 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 will vary in response. The first structure is an achievement

Table 1: (a) Two PUG concepts for a two-dimensional robot domain, each including a head, a set of observed elements, and a veracity function of the entities' attributes. The head and elements comprise a predicate, an identifier, and attributes with values. (b) Four percepts for this domain that describe perceived objects and four beliefs inferred from them. Each has a predicate, identifier, and attributes with values, the latter observed for percepts and inferred for beliefs.

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

    ((robot-facing ^id (?r ?o) ^angle ?a)
     :elements ((robot ^id ?r)
                (object ^id ?o ^angle ?a))
     :veracity  (cond ((< ?a 0.0) (- 1.0 (/ ?a –180.0)))
                      ((> ?a 0.0) (– 1.0 (/ ?a 180.0)))
                      ((= ?a 0.0) 1.0)) )
```

```
(b) (robot ^id R1 ^radius 0.15 ^move-rate 0.0 ^turn-rate 0.0)  [1.0]
    (object ^id O1 ^distance 2.0 ^angle 0.0 ^radius 0.4)  [1.0]
    (object ^id O2 ^distance 4.123 ^angle 14.032 ^radius 0.4)  [1.0]
    (object ^id O3 ^distance 6.0 ^angle 0.0 ^radius 0.4)  [1.0]
    (robot-at ^id (R1 O1) ^distance 2.0) [0.85]
    (robot-at ^id (R1 O2) ^distance 4.123)  [0.643]
    (robot-facing ^id (R1 O1) ^angle 0.0)  [1.0]
    (robot-facing ^id (R1 O2) ^angle 14.032)  [0.922]
```

motive, which generates utility only when it first matches against a particular belief with veracity above threshold. The second example is a maintenance motive, which assigns utility to a belief repeatedly each time that its conditions are satisfied. The latter type often specifies negative values, which means that the agent should avoid them if possible.

PUG interprets these structures during its state-processing cycle, comparing each motive to current beliefs. For each matched instance, it substitutes bound variables into the associated utility function, computes the result, and deposits it in the *:utility* field of the corresponding belief. If multiple motives have the same head, then it assigns the sum of their utilities to this belief. Earlier versions of the architecture stored this information in a separate goal memory, but the current one associates utilities with beliefs themselves, even when their *veracity* scores are too low to include them otherwise. In such cases, they encode desired beliefs that the agent does not yet hold and play the role of unsatisfied goals.

Skills and Intentions

In addition, PUG uses *skills* to encode knowledge about how to achieve the agent's objectives. Again, these structures *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

Table 2: (a) Two motives for the two-dimensional robot domain, each of which includes a relational head, a set of observed entities, a function for computing the utility of the relation, and a type. The first motive specifies the positive value of the robot's position being next to a target object; the second gives the negative utility of being near an obstacle. An achievement motive assigns utility to a belief only on its first match; a maintenance motive does so repeatedly.

```
((robot-at ^id (?r ?o))
 :conditions ((robot ^id ?r ^radius ?rr)
              (object ^id ?o ^type target ^distance ?od ^radius ?or))
 :utility    (cond ((< ?od (+ ?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 ?od ^radius ?or))
 :utility    (cond ((< ?od (+ ?rr ?or)) –20.0) (t 0.0))
 :type       maintenance)
```

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 3 (a) gives two sample skills from the robot domain, one for moving toward a given object and another for turning to face 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 belief's *veracity* score).

Just as beliefs in PUG are instances of general concepts, so *intentions* are instances of defined skills. Each structure has a skill name and arguments, along with an *activation* (the mismatch of the target) and values for its associated control attributes. For instance, for the scenario in Figure 1 (a), the agent might have the intention set *((move-to R1 O3) (turn-to R1 O3) (avoid-on-left R1 O1) (avoid-on-right R1 O2))*, which the agent could execute in parallel when they are applicable. Each intention's mismatch to the target and its control values would vary over time, while its symbolic description would remain the same.

The architecture examines each current intention I during its state-processing cycle to see whether its conditions are satisfied by the agent's current belief state. If so, then the module accesses I's target belief and its associated *veracity* score. The interpreter substitutes one minus this score for the symbol *$mismatch* in I's control equations, along with the values for any bound variables. Next the system evaluates the instantiated expression to compute the value for each

Table 3: (a) Two skills for the robot domain, each specifying a relational head, a set of observed entities, control equations, and a target concept. Each entity is described by a predicate and a set of attribute-value pairs, the latter referring to variables that may be shared across elements. The first skill influences the control attribute *move-rate*, whereas the second affects *turn-rate*. (b) Four intentions for approaching the target object $O3$ and for avoiding obstacles $O1$ and $O2$, the first two being instances of the skills in (a). A bracketed number denotes an intention's activation, which is one minus the veracity of its target concept.

(a) ((move-to ?r ?o)
:elements ((robot ^id ?r ^turn-rate ?t) (object ^id ?o ^angle ?a))
:tests ((> ?a –90) (< ?a 90))
:control ((robot ^id ?r ^move-rate (∗ 0.3 $mismatch)))
:target ((robot-at ^id (?r ?o))))

((turn-to ?r ?o)
:elements ((robot ^id ?r) (object ^id ?o ^angle ?a))
:control ((robot ^id ?r ^turn-rate (∗ 5.0 (sign ?a) $mismatch)))
:target ((robot-facing ^id (?r ?o))))

(b) (move-to R1 O3) [0.675]
(turn-to R1 O3) [0.812]
(avoid-on-left R1 O1) [0.793]
(avoid-on-right R1 O2) [0.0]

control attribute. For example, given the intention *(move-to R1 O1)* and the *veracity* score 0.518 for target belief *(robot-at ^id (R1 O1))*, the mismatch would be $1 - 0.518 = 0.482$ and the *move-rate* would be $0.3 \times 0.482 = 0.145$. When multiple intentions apply on a given cycle, PUG computes the target mismatches and control values for each one. If different intentions affect a control attribute, then the module takes the sum of their results.

The architecture also uses skills at the higher level of task planning. Briefly, the module responsible for this ability matches the conditions of skills against the agent's current belief state to generate candidate intentions that could achieve its goals (high-utility, low-veracity beliefs). This supports forward-chaining search through the space of task plans, which the system guides by using the results of mental simulation to calculate the utility of alternative plan trajectories over time. We discuss this module's operation, and its interaction with others, later in the paper.

Processes and Predictions

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 similarities. They specify the dynamic effects of causal attributes, reserving operator-like skills to encode details about how to achieve goals.

Table 4: Two processes for the two-dimensional robot domain, each of which includes a relational head, a set of observed elements, and a set of changes to the values of these entities. The first process predicts the change in the agent's distance and angle to an object when it moves forward. The second describes the change in the agent's angle with respect to an object when it turns. The functions $\star dd$ and $\star da$ invoke trigonometric equations to make calculations.

((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 (– ?t))))

Table 4 provides two examples of PUG processes. These include an *:elements* field that describes arguments and their types, along with an optional *:tests* field with 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, environmental forces, or their combination. *Predictions* are instances of processes that specify a predicate, its arguments, and the derivatives it predicts for the current situation.

On a given state-processing cycle, once PUG has computed the summed values for its control attributes, it uses matched instances of processes to predict changes to the environment. Each process instance predicts a change to one or more environmental attributes, with their individual effects being summed to determine overall results. For instance, given the value 0.145 for the control attribute *move-rate* and given the current *distance* of 5.105 and *angle* of 33.189 to *O1*, the process instance *(move-relative R1 O1)* predicts a change of -0.121 for *distance* and 0.913 for *angle*. Different instances of *move-relative* also predict shifts of the robot's *distance* and *angle* to other objects in the environment.

Cascaded Integration in PUG

Like other cognitive architectures, PUG operates in discrete cycles that match long-term knowledge structures against short-term elements to produce instances of the latter, as well as generate behavior in the environment. An important difference is that the framework relies on five distinct levels of temporal resolution. These include:

- *Belief processing*, which matches conceptual knowledge against the agent's perceptions and beliefs to draw inferences and thus generate new or updated beliefs;

- *State processing*, which matches skills against perceptions and beliefs to calculate values for intentions' control attributes and matches processes against perceptions and control values to predict their effects;
- *Reactive execution*, which generates motion trajectories either in the environment, using feedback control, or in the agent's mind, combining prediction with the mental simulation of such reactive control;
- *Motion planning*, which carries out greedy search in a space of possible motion trajectories that are generated by alternative sets of intentions and guided by utilities associated with their states; and
- *Task planning*, which uses forward search for candidate sequences of motion plans, again guided by their utilities, that achieve some or all of the agent's top-level goals.

PUG applies these modules in a cascaded manner, with each one using mental structures produced by those below it. For instance, state processing draws on beliefs generated by conceptual inference, whereas execution uses results from state processing to generate trajectories for a sequence of state cycles. These operate at different time scales, in that multiple cycles of belief processing occur for each step of state processing and multiple cycles of the latter occur for each execution or mental simulation run.

We have already discussed the two lowest levels, belief and state processing, in previous sections. The next two layers are responsible for mental simulation and motion planning. The first relies on a simple iterative mechanism. Recall that, combined with the calculation of control attributes by skill execution, processes let PUG predict the next state in terms of what the agent can expect to perceive. Conceptual inference creates a new belief state, which in turn lets it invoke skill execution, process prediction, and utility calculation. When applied repeatedly, this mechanism of mental simulation produces a trajectory of belief states over time. Each simulated trajectory follows deterministically from an initial situation and a set of intentions.

PUG users can provide a motion plan, but it can also find such structures using mental simulation and utilities to guide heuristic search. Motion planning starts with goals, which the system uses to retrieve candidate intentions by matching skills' target concepts. The architecture uses simulation to envision each plan's trajectory if it were executed in the environment. On each time step, it computes each beliefs' utility and stores a running total of values for them. If some beliefs have negative utility, the system retrieves new intentions that should eliminate them, favoring ones that occur earlier in the trajectory. PUG adds each intention to the current motion plan, simulates these elaborations, and selects the best one. However, the new trajectory may introduce further sources of negative utility (e.g., obstacles) that require additional repairs. This process continues until it finds no way to improve on the best candidate, at which point it halts.

The highest level of processing involves heuristic search through a space of task plans. Given the current belief state, the architecture finds which skills are applicable, simulates them mentally, and evaluates their trajectories in terms of motive-computed utilities. It selects the highest-scoring intention (or intention set) not yet been tried, infers the belief state that it produces, and extends the plan by selecting additional intentions if needed. In this manner, PUG carries heuristic depth-first search through the space of task plans, backtracking when plan length exceeds a user-specified limit. The system continues search until it finds N plans with high-enough utility, where N is another user-set parameter.

In this manner, PUG integrates its knowledge about concepts, motives, skills, and processes to produce reactive but goal-driven behavior. The reliance on discrete cycles is akin to that in other cognitive architectures, but its cascaded character makes its approach to integration unique. Each level of processing draws on results produced by lower ones that operate on a more rapid time scale, running the gamut from the generation of beliefs at one extreme to the creation of sequential task plans at the other.

Related Research and Impact

The robotics literature is sizable and our architecture for embodied agents incorporates many ideas that first appeared elsewhere. We have been influenced especially by frameworks that integrate multiple abilities and that provide formalisms for stating expertise. We can divide this work into four broad categories:

- *Robotic architectures* (Kortenkamp & Simmons, 2008), which integrate 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 rather than adopting a unified framework with shared memories. Other work, such as Bonasso et al.'s (1997) 3T architecture, offers formalisms for knowledge but, unlike PUG, adopts different notations for each level of processing.
- *Cognitive architectures*, such as ACT-R (Anderson & Lebiere, 1998) and Soar (Laird, 2012), which 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 do not support embodied agency at the architectural level and instead call on external routines (e.g., PID controllers).
- Research in *cognitive robotics* (e.g., Ferrein & Lakemeyer, 2008; Grosskreutz & Lakemeyer, 2000; Levesque et al., 1997), which 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.
- The *spatial semantic hierarchy* (Kuipers, 2000), which 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 structures for motives and processes.

As noted earlier, PUG builds directly on 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 intelligent systems.

We should reiterate that none of PUG's individual features are unique and its contribution lies in combining earlier ideas in novel ways to provide a unified formalism for 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. Thus, although the current framework offers no new capabilities, it holds clear pathways for their development.

Directions for Future Research

We have implemented the PUG architecture in Steel Bank Common Lisp. This lets users specify knowledge bases for embodied agents using the syntax outlined above for concepts, motives, skills, and processes. They can load this content from a file along with the description of an initial situation in terms of object poses and other attributes. Users must also specify a set of percepts produced by the environment and a set of control attributes that can affect it. As described earlier, the architecture interprets its knowledge structures in discrete cycles, calling on modules for conceptual inference, utility calculation, reactive control, state prediction, mental simulation, motion planning, and task planning. Each run produces a trace of the agent's dynamic mental structures, along with the trajectory of physical states produced when it interacts with the environment.

Despite PUG's unification of these abilities, we can still extend the theoretical framework along various fronts to provide even broader coverage. In particular, future research on the cognitive architecture should:

- Introduce more flexible processing for conceptual inference (e.g., abduction of postulated entities and attributes in partially observable environments) and support more sophisticated control schemes (e.g., replacing proportional with PID methods);
- Integrate the original PUG task planner (Langley et al., 2016), which carried out backtracking search guided by utilities, with the newer modules for control and motion planning, as well as with earlier software for plan execution and monitoring (Langley et al., 2017);
- Incorporate 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;
- Elaborate 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, in press);
- Organize 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 agent planning and execution; and
- Support 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 (e.g., Zita Haigh & Veloso, 1996), 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 additional extensions that are needed to support a truly unified framework for constructing embodied agents.

Acknowledgements

The research reported here 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

Aha, D. W. (2018). Goal reasoning: Foundations, emerging applications, and prospects. *AI Magazine*, *39*, 3–24.

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.

Grosskreutz, H., & Lakemeyer, G. (2000). cc-Golog: Towards more realistic logic-based robot controllers. *Proceedings of the Seventeenth National Conference on Artificial Intelligence* (pp. 476–482). Austin, TX: AAAI Press.

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. (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.

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.

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.

Zita Haigh, K., & Veloso, M. M. (1996). Interleaving planning and robot execution for asynchronous user requests. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems* (pp. 148–155). Osaka, Japan: IEEE Press.