Creating and Using Tools in a Hybrid Cognitive Architecture

Dongkyu Choi
Department of Aerospace Engineering
University of Kansas
Lawrence, KS 66045 USA
dongkyuc@ku.edu

Pat Langley and **Son Thanh To**
Institute for the Study of Learning and Expertise
2164 Staunton Court, Palo Alto, CA 94306 USA
patrick.w.langley@gmail.com
son.to@knexusresearch.com

Abstract

People regularly use objects in the environment as tools to achieve their goals. In this paper we report extensions to the ICARUS cognitive architecture that let it create and use combinations of objects in this manner. These extensions include the ability to represent virtual objects composed of simpler ones and to reason about their quantitative features. They also include revised modules for planning and execution that operate over this hybrid representation, taking into account both relational structures and numeric attributes. We demonstrate the extended architecture's behavior on a number of tasks that involve tool construction and use, after which we discuss related research and plans for future work.

Introduction

The ability to create and use complex tools is a distinctive feature of human cognition. People use objects in their surroundings to help achieve goals, sometimes combining multiple objects into a new tool that fits their need. This involves planning but focuses on constructing physical artifacts to achieve other ends, rather than generating isolated action sequences. For example, scenes from a popular television series, MacGyver, often depicts the protagonist creating tools from materials that seem unrelated to his objectives. The character ingeniously uses objects in ways for which they were not intended, often combining them into a tool for his purpose. Current AI systems, including our current work, do not demonstrate such creative abilities.

In this paper, we report progress toward intelligent agents that exhibit the ability to create and use physical tools. Our approach extends an existing cognitive architecture to support this capacity. In the next section, we present a scenario that illustrates how tool construction and use can help an agent achieve its goals. Next we briefly review ICARUS, an architecture for physical agents, and we describe extensions to its representation and processes that let it create and use tools. After this, we report runs in a simulated environment, drawing on the scenario presented earlier, that demonstrate the revised system's abilities. We conclude by discussing related work and plans for future research.

A Motivating Scenario

We can clarify the challenge of tool creation and use with a scenario. Consider a robot that wants to escape from inside a crumbled building. Its goal is to move from a location inside the building to another one outside, but between them is a wide gap in the floor that the robot cannot traverse and an opening in the wall that is too high for it to reach without other support. The robot observes some wooden planks of different lengths and thicknesses. Knowing its own weight and the maximum height it can climb, it stacks planks across the gap to build a bridge that will support its weight. The robot then crosses the bridge and thus traverses the gap. In a similar fashion, it builds a staircase to the opening on the wall, goes up the staircase, and escapes from the building.

In this scenario, the robot manipulates objects in its environment and assembles them into tools which it then uses to achieve its goal. To create the right tool, it considers both structural and numeric factors. Wooden planks laid over the gap can serve as a basic bridge, but they must be long enough to cross the gap and strong enough to hold the robot's weight. A single plank may appear qualitatively sufficient, but a second plank may be needed to make the bridge strong enough. For an effective staircase, the building blocks must be arranged to give enough footing on each step and the steps should be no higher than the robot can climb.

We can view bridges and staircases as tools that are constructed from available components. Computing the load a bridge must hold or the height of a step requires quantitative reasoning about objects' positions and dimensions, but the agent must first devise some qualitative structure that its numbers describe. We believe the scenario provides a reasonable challenge for testing an intelligent agents' ability to create and use tools, as it requires a combination of qualitative and quantitative reasoning.

A Brief Review of ICARUS

ICARUS (Langley, Choi, and Rogers 2009) is a cognitive architecture that provides an infrastructure for building intelligent agents that operate in physical settings, simulated or actual. As with other architectures like Soar (Laird et al. 1986) and ACT-R (Anderson and Lebiere 1998), it makes commitments about the representation of content, the memories that store that information, and the processes that manipulate it. ICARUS incorporates many ideas from cognitive psy-

Table 1: Sample ICARUS concepts for the staircase problem.

```
((on ?o1 ?o2)
 :elements ((block ?o1 ^x ?x1 ^y ?y1 ^length ?length1)
            (block ?o2 ^x ?x2 ^y ?y2 ^length ?length2
                       ^height ?height2))
 :tests    ((*overlapping ?x1 ?length1 ?x2 ?length2)
            (= ?y1 (+ ?y2 ?height2))))

((staircase ?o ?o1)
 :elements   ((block ?o ^height ?h))
 :conditions ((on ?o ?o1)
              (staircase ?o1 ?o2)
              (step-size ?step))
 :tests      ((<= ?h ?step)))
```

Table 2: Sample ICARUS skills for the bridge problem.

```
((pick-up ?o)
 :elements   ((robot ?robot)
              (block ?o))
 :conditions ((clear ?o) (not (holding ?robot ?any)))
 :actions    ((*pick-up ?robot ?o)))
 :effects    ((holding ?robot ?o))

((build-bridge ?block ?bottom)
 :elements   ((block ?block))
 :conditions ((bridge ?top ?bottom))
 :subskills  ((stack ?block ?top))
 :effects    ((bridge ?block ?bottom))
```

chology, but it emphasizes construction of intelligent systems that carry out complex activities rather than fitting the results of psychological experiments. In this section, we review the architecture, starting with assumptions for representation and memories and then describing its mechanisms for inference, reactive execution, and problem solving.

Representation and Memories

ICARUS distinguishes between two forms of long-term knowledge: *concepts* that underlie inference and procedural *skills* that support activity. The framework also separates *percepts* from the environment from *beliefs* inferred about them. The former describe observed objects in terms of their attributes, typically numeric, while the latter take the form of relational literals like *(on A B)*. This distinction will figure centrally later in the paper. The conceptual knowledge base links percepts to beliefs through a set of defined *concepts*. Each conceptual rule specifies the conditions that must match to infer a belief of a given type. The conditions of a *primitive* concept refer only to percepts and their attribute values, whereas the conditions of a *nonprimitive* concept also refer to more basic conceptual predicates.

Table 1 shows some ICARUS concepts that describe relations and situations for the staircase scenario. The first conceptual rule, for the predicate *on*, is primitive, as it has only an *:elements* field, which describes perceived objects and their attributes, along with a *:tests* field that constrains the matched variables. This concept refers to two *block* objects and checks numeric relations between their positions, lengths, and heights. The second concept, for the predicate *staircase*, is nonprimitive, as it refers to other concepts in its *:conditions* field. These include the concepts like *on*, *step-size*, and *staircase*, so the definition is recursive. Thus, concepts are organized into a hierarchy, with more complex predicates defined in terms of simpler ones.

ICARUS skills build on its conceptual knowledge. Each skill clause includes generalized percepts, conditions that must hold for application, and effects that its application produces. A *primitive* skill clause refers to some action that the agent can execute directly in the environment, whereas a *nonprimitive* skill clause refers to other, more basic, skills.

Table 2 shows examples of ICARUS skills relevant to the bridge problem in our scenario. The first skill clause, *pick-up*, refers to two perceived objects, a *robot* and a *block*, and has two conditions, one positive (for *clear*) and the other negative (for *holding*). This clause is primitive because it includes the executable action **pick-up*. The second skill, *build-bridge*, mentions one percept and one conceptual condition, but it is nonprimitive because it includes the subskill *stack*. Such references organize skills into a hierarchy in which primitive clauses serve as terminal nodes, much as in a hierarchical task network (e.g., Nau et al. 2003).

Cognitive Processes in ICARUS

The architecture utilizes its concepts and skills during processing, which operates in four-step cycles. First, ICARUS deposits percepts from the environment in a perceptual buffer. The system does not model the extraction of percepts from sensors, but they serve as plausible outputs of sensory processing. Second, the architecture combines its conceptual knowledge with these percepts to infer beliefs that hold for the current situation. ICARUS matches primitive conceptual clauses against perceived objects to generate low-level beliefs, then matches nonprimitive concepts against them to produce higher-level beliefs. For example, the first clause in Table 1 generates a belief about the *on* relation when a block's *y* position equals that of another block plus its height and when the **overlapping* test is true.

Once ICARUS has inferred beliefs about the current situation, an execution stage attempts to find a path downward through the skill hierarchy that it can carry out in the environment. This module starts with a top-level goal, retrieves a skill clause that should achieve it and has conditions satisfied by current beliefs. If this skill instance is primitive, the architecture executes its associated action; if not, then it considers matched subskills. This recursive process returns a path through the skill hierarchy whose execution should bring the agent closer to its goal(s). When ICARUS cannot find such an applicable path, it invokes a problem-solving module that carries out search for sequences of skills which achieve the current goals. Execution and problem solving are tightly interleaved, with the system carrying out selected skill instances when applicable and resorting to problem solving when it encounters an impasse.

Table 3: An ICARUS concept that illustrates the extended numeric representation.

```
((bridge ?b ?g ?leftend ?rightend)
 :elements   ((block ?b ^x ?leftend ^length ?ln)
              (gap ?g ^left ?gl ^right ?gr))
 :attributes (?left is (- ?gl 1)
              ?right is (+ ?gr 1)
              ?rightend is (+ ?leftend ?ln))
 :tests      ((<= ?leftend ?left)
              (>= ?rightend ?right)))
```

We should note that, although ICARUS grounds its concepts and skills in quantitative percepts and actions, the inference, execution, and problem-solving modules primarily produce qualitative and relational structures. This does not keep the architecture from operating in continuous domains like simulated urban driving (Langley et al. 2009; Choi 2011), but we will see that it raises challenges for the construction and use of complex tools.

Numeric Representation and Processing

As noted earlier, reasoning about tools often requires that an agent operate over not only qualitative aspects of the environment, but also its quantitative properties. In this section, we discuss two extensions to ICARUS that let the architecture support numeric processing, the first involving representation and the second concerning planning.

Representational Extensions

ICARUS receives and processes perceptual elements that include types, names, and attribute-value pairs for objects in the world. The original system can represent symbolic relations among objects and concepts can include simple tests on numeric attributes. But it cannot reason about numeric relations or specify arithmetic computations and associate their results with a new variable. In previous research, this limitation has caused problems when using ICARUS to control physical robots, where the continuous domain requires encoding of numeric constraints. Naturally, this issue also arises in tool creation and use. To address the problem, we extended the conceptual formalism to specify arithmetic combinations of numeric attributes and associate them with new variables that can appear elsewhere in the concept.

Table 3 shows a sample concept that uses this extended notation. The clause includes a new field, *:attributes*, that specifies desired numeric calculations and their variable assignments. This specific clause states that the position of a block's right side (denoted by the variable *?rightend*) can be computed from its left side position, *?leftend*, and its length, *?ln*. The concept also specifies how to compute the left and right positions, *?left* and *?right*, for a spatial gap with one unit margins at both ends. These values are also used, along with the left and right ends, in two inequality tests.

This extension lets ICARUS specify numeric calculations and how to reuse their results elsewhere in a conceptual clause, complementing the qualitative structures it could already express. However, this only describes the environmental situation, not how agent's actions will alter it. In response, we also extended the notation for skills to incorporate details about quantitative effects of their execution.

Table 4: An ICARUS skill for creating a bridge that illustrates the extended numeric formalism.

```
((fill-gap-center ?b ?g)
 :elements   ((block ?b ^x ?x0 ^length ?l ^weight ?w)
              (robot ?robot ^weight ?weight
                            ^status ?status ^holding ?b)
              (gap ?g ^left ?gl ^right ?gr))
 :actions    ((*fill-gap-center ?robot ?b ?gl ?gr))
 :effects    ((bridge ?b ?g ?x0 (+ ?x0 ?l))
              (block ?b ^x (/ (- (+ ?gl ?gr) ?l) 2) ^y 0
                        ^len ?l ^weight ?w)
              (robot ?robot ^weight (- ?weight ?w)
                     ^status ?status ^holding nothing)))
```

Table 4 shows a skill that takes advantage of this extension. The main change is in the *:effects* field, which describes the outcome of a skill's successful execution. Previously, this field could only include symbolic effects about relational beliefs that would become true or false after application. In the new notation, the field can describe expected changes not only in symbol structures, but also in the numeric attributes of objects. The skill will not only cause the symbolic relation *(bridge ...)* to become true, but also change the block's *x* position to the value of the expression, *(/ (- (+ ?gl ?gr) ?l) 2)*, and reduce the robot's *weight* by *?w*.

Extensions to Processing

The original architecture could match against numeric attributes of perceived objects, but it could neither perform mathematical calculations over these numbers nor allow the results in concept heads. The representational changes to concepts and skills remedies these limitations, but taking advantage of them also required us to augment ICARUS's information processing along two fronts. The first deals with inference, which now calculates the values of arithmetic expressions in concepts and binds them to specified variables that may appear in the heads. These numeric values, in turn, can influence inference of symbolic beliefs at higher levels, as they are carried upward through the hierarchy during the conceptual inference process.

These changes to the formalism require no alteration of the execution module, but they do necessitate changes to problem solving. In response, the revised module computes not only symbolic relations during its mental execution of skills but also numeric values associated with them. The new problem solver utilizes forward chaining, which lets the system update numeric attributes of an object, add new literals, or delete existing literals from the state using information encoded in skills' *:effects* field. Such mental execution has direct effects on the projected state, but indirect changes can also occur, which the architecture determines by invoking the inference module. As a result, the problem solver can generate plans that satisfy both symbolic and numeric requirements specified in the agent's goals.

Encoding and Processing Virtual Objects

Despite its new ability to reason about quantitative aspects of the environment, the extended ICARUS still cannot recognize an existing object as a potential tool or reason about how to create one from available elements. This is because the architecture only recognizes primitive objects as distinct entities, not combinations of them. In contrast, people readily view composite structures as objects themselves, describe numeric features associated with them, and reason about them as unified entities. To create and utilize complex tools, ICARUS needs the ability to reify and process such *virtual* objects in its environment.

Representational Extensions

The ability to include numeric attributes in concept heads paves the way to handling virtual objects. Without this extension, the architecture can infer beliefs only as symbolic literals, which makes them different from perceived objects in that they lack numeric attributes. Previously, for example, a *bridge* concept that describes a composite object could only produce a symbolic belief that informs the agent about its existence. In contrast, the new version can calculate the values for numeric attribute associated with the *bridge* entity, such as its thickness and weight limit.

However, computing such numeric attributes is not enough. We also need some way to associate them with the virtual object, which requires giving it a symbolic identifier in the same manner as percepts. This extension effectively eliminates the distinction in the original ICARUS between beliefs and percepts, so the new architecture stores them in a single working memory. The only remaining differences are that percepts come directly from an external environment, while beliefs are inferred, and that beliefs include a symbolic relation, while percepts lack them. Of course, we can apply this idea recursively to specify higher-level virtual objects in terms of lower-level ones.

For example, the two conceptual clauses for *bridge* that appear in Table 5 not only describe the class of *situations* in which one or more blocks cover a gap, but also specify a new *virtual object* that denotes the bridge. This composite object has its own attributes, such as its left position, right position, and weight, the values of which are calculated from the attribute values of its component objects.

Implications for Processing

Once the extended ICARUS has created virtual objects, it can use them as if they were objects perceived directly in the environment. The second, recursive, clause for *bridge* concept shown in Table 5 lets the system recognize situations in which a block is stacked on a bridge and generate another virtual object that is also a *bridge*, but one with a higher weight limit than the original one.

As the table shows, the new notation also changes the syntax for the `:elements` field. Here the expression *A is B* states that one should associate an identifier *A* with *B*, which may be a percept or a relational belief. Recall that percepts enter the perceptual buffer with such identifiers, but that ICARUS must name its beliefs before it can associate numeric attributes with them. The extended architecture retains the identifiers for these virtual objects in working memory, so they can appear as arguments in higher-level beliefs that result from conceptual inference.

Table 5: Some ICARUS concepts that specify virtual objects.

```
((bridge ?b ^gap ?gl ^left ?l ^top-left ?tl
           ^top-right ?tr ^right ?r ^weight ?weight)
 :elements (?b is (block ?b ^x ?tl ^y 0 ^len ?len
                         ^weight ?weight)
            ?gl is (gap ?gl ?gr))
 :tests    ((<= ?tl (- ?gl 1))
            (>= (+ ?tl ?len) (+ ?gr 1)))
 :attributes (?l is ?tl
              ?tr is (+ ?tl ?len)
              ?r is (+ ?tl ?len)))

((bridge ?b ^gap ?gl ^left ?l ^top-left ?tl
           ^top-right ?tr ^right ?r ^weight ?weight)
 :elements (?b is (block ?b ^x ?tl ^y ?y ^len ?len
                         ^weight ?w)
            ?b1 is (bridge ?b1 ^gap ?gl ^left ?l
                           ^top-left ?tl1 ^top-right ?tr1
                           ^right ?r ^weight ?w1)
            ?b1 is (block ?b1 ^x ?tl1 ^y ?y1
                           ^len ?len1 ^weight ?w2)
            ?gl is (gap ?gl ?gr))
 :tests    ((<= ?tl (- ?gl 1))
            (>= (+ ?tl ?len) (+ ?gr 1))
            (= (+ ?y1 1) ?y)
            (<= (+ ?tl1 1) ?tl)
            (<= (+ ?tl ?len 1) ?tr1))
 :attributes (?weight is (+ ?w ?w1)
              ?tr is (+ ?tl ?len)))
```

What we have described suffices for ICARUS to draw inferences about composite objects, but not to use them for driving agent activity. Of course, virtual objects can also appear in the *effects* field of skills, which means that the problem solver can form expectations about their creation or destruction upon execution. This means, for example, that the agent can use its hierarchical skills to form plans that involve constructing composite objects which enable later steps that achieve its goals. But it can also use search to generate plans entirely from primitive skills and, by invoking the inference process, deduce that an action sequence has the side effect of creating a complex virtual object that it can use as a tool.

Demonstrations of the Extended Architecture

To confirm that the extended system behaves as intended, we carried out demonstration runs on the scenario described earlier. Here an ICARUS agent controls a simulated mobile robot to reach its destination. In one case, there is a chasm between the initial and the goal location; in another problem, the goal is at a higher location than the robot can reach directly. In both cases, the agent can use blocks of different sizes to build a bridge or staircase, which it can then use.

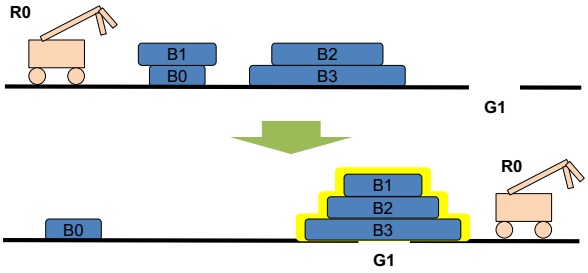

Figure 1: Initial and final states for one version of the bridge problem. The robot, *R0*, starts on the lefthand side and must use blocks to build a bridge over the gap, *G1*, to reach its goal on the righthand side.

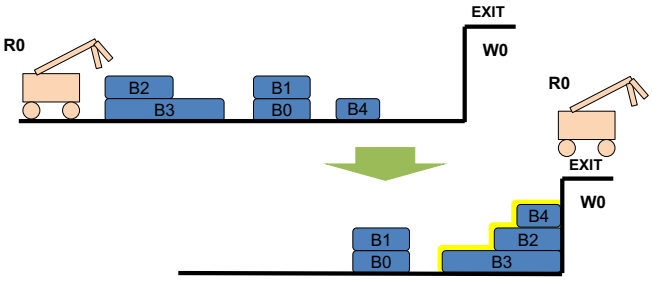

Figure 2: Initial and final states for one version of the staircase construction and climbing problem.

Simplifying Assumptions

The primary aim of these demonstration runs was to show that the extensions to ICARUS, described earlier, support the creation and use of tools. For this reason, we introduced four simplifying assumptions that made the planning and execution tasks somewhat easier than they would be in a realistic simulation:

- Although ICARUS allows durative skills that require repeated application to achieve their effects, in the runs all skills produce results in one step;
- The 2D simulated environments let agents pick up and stack objects without first needing to approach them or to move around obstacles;
- Agents must use planning to find a sequence of skills that construct composite objects that can serve as tools, but skills for using them operate in one step; and
- We provided agents with hierachical concepts for tools that appear as conditions on these tool-using skills, effectively serving as affordances (Zech et al. 2017).

Ideally, future demonstrations should use more realistic simlated environments that eliminate these assumptions. Nevertheless, the reported runs offer clear proof of concept that the extended architecture can represent, reason about, construct, and use tools to achieve goals in continuous settings.

Creating and Traversing a Bridge

In the first setting, the robot must build a bridge to cross the chasm, using long wooden blocks of different lengths and strengths. The agent knows that, for the robot to traverse the bridge safely, it must: (1) cover the chasm by a margin of at least a foot at each end; (2) withstand the robot's weight and any payloads; and (3) if it is made from stacked blocks, include a staircase at each end with steps no higher than a foot and at least a foot wide. The agent has no skill that directly creates a bridge, so it must use problem solving to find some plan to build one that satisfies these requirements. The system must then execute this plan, building the bridge in the environment and crossing the chasm to reach its destination.

For this problem, we gave ICARUS four concepts and four primitive skills, including the ones shown in Tables 4 and 5. Using this knowledge, the agent can recognize situations in which a block is stacked on another, detect a bridge composed of blocks, pick up a block to either stack it on another or cover a chasm, and finally cross the bridge when it is complete. Figure 1 shows an initial state in which the robot perceives itself, a chasm, and four blocks that are two, four, six, and eight units in length and that have weight limits of one, five, one, and two, respectively. Block *B1* is on block *B0* and block *B2* is on block *B3*.

Given these initial and goal states, the problem solver uses forward-chaining search to find a plan that achieve its goal in nine steps. During this process, ICARUS first considers a bridge that only withstands a weight of two units, which is insufficient for the robot to cross. Next the system considers stacking a second block on the first to create a bridge with the maximum load of three units. This is still not sufficient, so it stacks yet another block, making a bridge that is strong enough for it to cross the chasm safely.

Once it has found this plan, the ICARUS agent executes it in the simulated environment over 29 cycles, first picking up *B2* to clear *B3* and stacking *B2* on *B1*. Next the system picks up the longest block *B3* and covers the gap with it. Then the robot picks up another block, *B2*, and stacks it on *B3* to create a stronger bridge, after which it stacks *B1* on the result to make it even stronger. At this point, the robot traverses the reinforced bridge to reach its goal.

We ran the extended architecture on 20 similar problems that involved four blocks of random lengths and weight limits. The system executed plans that had the average duration of 29.6 cycles with a standard deviation of 10.7 cycles. We also ran it, with the same knowledge, on a slightly different goal description in which the robot must carry a certain block as its payload across the chasm. In this altered scenario, the ICARUS agent generated a similar plan, this time requiring that it construct an even stronger bridge, then pick up the payload for delivery. Again, the robot executed this plan in the simulated world to achieve its goal.

Constructing and Climbing a Staircase

In the second scenario, the robot must escape from a room in which the exit is higher on the wall than it can reach without assistance. The environment contains long wooden blocks of

different lengths that the agent can use to build a staircase for reaching the exit. The system knows that a staircase must: (1) have steps that are no taller than a foot for the robot to climb successfully and at least a foot wide so it can step on them safely; (2) be no further than a foot from the wall at its highest point; and (3) have a height that is within a foot of the exit's height. The robot must build a staircase that satisfies all these requirements before it can ascend and exit the room.

For this problem, we provided ICARUS with seven concepts and four primitive skills. The robot could use this knowledge to recognize situations in which one block is on top of another, categorize a virtual object as a staircase, pick up a block to either stack it on another or place it on the ground, and leave the room when it reaches the exit. Figure 2 shows one example of this scenario in which the robot perceives itself, the wall, and five blocks with lengths of 1.5, 1.5, 3, 4.5, and 1, respectively, and with heights of one unit.

The problem solver uses forward search to generate a plan that, in 13 steps, achieves the exit goal. During planning, ICARUS mentally constructs a staircase from three blocks that will let it leave the room, but only after considering shorter stairways. Once it has found this plan, the robotic agent executes it in the simulated environment, which takes 41 cognitive cycles. This involves picking up block *B4* to clear the area around the wall and stacking it on block *B1*. The agent then picks up block *B2* to clear *B3* and stacks *B2* on *B4*. The robot continues stacking the blocks *B3*, *B2*, and *B4*, in that order. At this point, it recognizes that it has built an acceptable staircase, so the robot climbs the stairs and exits the room, achieving its goal.

As another demonstration run, we used a variation on this problem that required the system to combine a number of shorter blocks to form steps for the staircase. This involved generating a more complex plan with additional steps that led to more virtual objects, greater search during planning, and longer execution times than in the first run, but the system handled them without any special difficulty.

In summary, the runs have demonstrated that the extended architecture can represent and reason about numeric attributes and virtual objects during inference, problem solving, and execution. This lets the revised ICARUS infer beliefs that incorporate numeric attributes, associate them with composite entities that its actions produce, and use this content to generate and carry out plans that achieve symbolic goals subject to numeric constraints. Together, these abilities support the construction of tools, such as bridges and staircases, from available components and their use once built.

Related Research

The extensions to ICARUS that let it create and use tools have clear precedents that merit discussion. We focus here on two contributions that we consider most important – reasoning over numeric attributes and using virtual objects. We have discussed the architecture's forward-chaining planning module elsewhere (To et al. 2015). We will not repeat our observations here except to note that it can use primitive skills, hierarchical ones, and their combination to generate plans, although the first option requires more search.

Research in cognitive architectures (Langley, Laird, and Rogers 2009) has emphasized symbolic representation and processing, due to their focus on high-level cognitive tasks. Nevertheless, well-established frameworks like Soar and ACT-R adopt an attribute-value notation that can easily encode the types of numeric object-based inputs we assume in both working memory and production rules. Both architectures have been used to control robotic agents, which certainly requires quantitative processing. However, they treat numeric manipulation as a special case of symbol processing, rather than giving them equal status, at the architecture level, as does the extended version of ICARUS.

Other paradigms also support a combination of symbolic and numeric processing. For example, logic programming emphasizes symbolic notations but can incorporate quantitative values and constraints, although they do not typically operate over time, as do ICARUS agents. AI planning systems also focus on symbolic tasks but have been adapted to include numeric content (e.g., Coles et al. 2012). These describe activity over time, but work in this tradition seldom supports the storage and use of hierarchical skills. Most robotic systems emphasize low-level numeric processing to the exclusion on high-level cognition. Hybrids like the 3T architecture (Bonasso et al. 1997) support both, but they adopt separate, specialized notations rather than offering a unified framework for cognition and action. Perhaps the closest robotics work (Levihn and Stilman 2014; Erdogan and Stilman 2014), also concerned with tool creation, propagates physical constraints to ensure a symbolic planner considers only acceptable configurations of objects.[1]

As for the virtual objects, most production-system architectures (Klahr, Langley, and Neches 1987) support rules that introduce new symbols, with associated attribute values, in elements they add to working memory. However, they do not elevate their creation to the architectural level or make theoretical claims about the way such objects are defined, processed, and used by other mechanisms. Our extended framework associates virtual objects with concept instances that reside in belief memory, so that any conceptual rule in long-term memory can generate them during the inference process. This allows a tight integration with other components of the ICARUS architecture.

Otherwise, the paradigm most relevant to our use of virtual objects is scene understanding (e.g., Antanas et al. 2012), which attempts to infer models of the environment from images or videos. Classical approaches construct a hierarchy of entities, from edges to angles to surfaces to 3D object models (Binford 1982). ICARUS' virtual objects are directly analogous to these intermediate entities, and its calculation of derived attribute values maps directly on computations of angles and volumes in vision systems. However, work in this paradigm has focused on scene interpretation, not with goal-directed activity. Thus, although such systems might be able to describe and recognize tools like bridges and stairs, they cannot use them to achieve objectives.

[1] Brown and Sammut (2012) report a novel approach to learning tool usage by the analysis of training cases, but their research has different aims than our own.

Plans for Future Work

We have shown that the extended ICARUS can represent and reason about tools, it can construct such tools from available objects, and it can then use them to achieve its goals. Nevertheless, we must still address a number of challenges that our work to date has left unexamined. The most obvious limitations involve the system's dependence on handcrafted knowledge about composite tools.

ICARUS already includes mechanisms for learning hierarchical skills from successful problem solving (Langley et al. 2009), and we can use this ability to acquire structures for constructing bridges, staircases, and similar artifacts, as well as ones for using them after they have been created. The latter will be useful in more realistic environments that require sequences of actions for tool use, such as taking repeated steps up a staircase. These mechanisms acquire new skills from individual solutions obtained through search, so learning can be very rapid.

A more challenging hurdle involves the acquisition of concepts that recognize composite tools. Here we plan to draw on another extension to ICARUS (Li et al. 2012) that, when it uses a problem solution to create a new skill, also defines a new conceptual predicate that describes the conditions under which that skill will achieve the relevant goals. These conceptual rules may be disjunctive or even recursive, so the mechanism should be able to produce concepts for recognizing bridges, staircases, and other tools that may have arbitrary numbers of components.

However, we can best take advantage of this ability by separating the issues of tool construction and tool use. If we present an ICARUS agent with a problem that it can solve with an existing configuration of objects, say two blocks that cover a gap, it could learn both a hierarchical skill for using that configuration and a concept that recognizes similar 'bridge' configurations in the future. Given such knowledge, it could then solve, and learn from, new problems that require the construction of a bridge before its traversal. This decomposition is not strictly necessary, but inventing the bridge concept from scratch would require more search than determining how to build one after having used another.

These are certainly not the only challenges that remain before we have a mature account of tool construction and use. For instance, numeric simulation of durative operators, as in Langley et al.'s (2016) PUG architecture, seems relevant to determining whether an agent can use a tool to achieve its goals. The ability to interleave planning, execution, and monitoring is also important in settings where tools are not fully reliable. However, the creation and use of tools is one of the distinguishing features of human intelligence, so we should not be surprised that many open problems remain.

Concluding Remarks

In this paper, we reported extensions to the ICARUS architecture that support the creation and use of tools. These included the ability to associate numeric attributes with concepts and skills, as well as calculate their values during inference, execution, and problem solving. Another augmentation let conceptual rules refer to new, complex objects that were composed from existing ones and to derive values for their numeric attributes during the process of conceptual inference. Together, these capabilities let the extended architecture not only represent and reason about tools it creates from components available in the environment, but also use those tools to achieve its goals.

We demonstrated this new functionality in two simulated environments, one that involved creating and traversing a bridge and another that required constructing and climbing a staircase. We will not claim that other approaches, such as AI planning methods, cannot handle the same tasks, but they would not represent or recognize the fact that tools played a key role in their solutions. Humans clearly exhibit this ability, and we believe that ICARUS' approach to tool creation and use has many similarities. Nevertheless, we have taken only the first steps, and future work should include demonstrations in more realistic environments and use of learning mechanisms to acquire tool-related concepts and skills.

Acknowledgements

We dedicate this work in memory of Mike Stilman, a friend and colleague who inspired us to bridge the gap between robotics and cognitive systems. This research was supported by Grants N00014-12-1-0143 and N00014-15-1-2517 from the Office of Naval Research.

References

Anderson, J. R.; and Lebiere, C. 1998. *The atomic components of thought*. Mahwah, NJ: Erlbaum.

Antanas, L.; Frasconi, P.; Costa, F.; Tuytelaars, T.; and Raedt, L. D. 2012. A relational kernel-based framework for hierarchical image understanding. In G. Gimel'farb et al., Eds., *Structural, syntactic, and statistical pattern recognition*, 171–180. Berlin: Springer.

Binford, T. O. 1982. Survey of model-based image analysis systems. *International Journal of Robotics Research* 1:18–64.

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

Brown, S.; and Sammut, C. 2013. A relational approach to tool-use learning in robots. In F. Riguzzi and F. Elezn, Eds., *Inductive Logic Programming*, 1–15. Berlin: Springer.

Choi, D. 2011. Reactive goal management in a cognitive architecture. *Cognitive Systems Research* 12:293–308.

Coles, A. J.; Coles, A. I.; Fox, M.; and Long, D. 2012. COLIN: Planning with continuous linear numeric change. *Journal of Artificial Intelligence Research* 44:1–96.

Erdogan, C.; and Stilman, M. 2014. Incorporating kinodynamic constraints in automated design of simple machines. *Proceedings of the 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2931–2936. Chicago: IEEE Press.

Fikes, R.; and Nilsson, N. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2:189–208.

Klahr, D.; Langley, P.; and Neches, R. Eds. 1987. *Production system models of learning and development*. Cambridge, MA: MIT Press.

Laird, J. E.; Newell, A.; and Rosenbloom, P. S. 1987. Soar: An architecture for general intelligence. *Artificial Intelligence* 33:1–64.

Langley, P.; Barley, M.; Meadows, B.; Choi, D.; and 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.; and Rogers, S. 2009. Acquisition of hierarchical reactive skills in a unified cognitive architecture. *Cognitive Systems Research* 10:316–332.

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

Levihn, M.; and Stilman, M. 2014. Using environment objects as tools: Unconventional door opening. *Proceedings of the 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2502–2508. Chicago: IEEE Press.

Li, N., Stracuzzi, D. J., and Langley, P. 2012. Improving acquisition of teleoreactive logic programs through representation extension. *Advances in Cognitive Systems* 1:109–126.

Nau, D. S.; Au, T.-C.; Ilghami, O.; Kuter, U.; Murdock, J. W.; Wu, D.; and Yaman, F. 2003. SHOP2: An HTN planning system. *Journal of Artificial Intelligence Research* 20:379–404.

To, S. T.; Langley, P.; and Choi, D. 2015. A unified framework for knowledge-lean and knowledge-rich planning. *Proceedings of the Third Annual Conference on Cognitive Systems*. Atlanta, GA.

Zech, P.; Haller, S.; Lakani, S. R.; Ridgeand, B.; Ugur, E.; and Piater, J. 2017. Computational models of affordance in robotics: A taxonomy and systematic classification. *Adaptive Behavior* 25:235–271.