Available online at www.sciencedirect.com

ScienceDirect

Cognitive Systems Research 48 (2018) 25–38

Cognitive Systems RESEARCH

www.elsevier.com/locate/cogsys

Evolution of the Icarus Cognitive Architecture

Dongkyu Choi[a,*], Pat Langley[b]

[a] Department of Aerospace Engineering, University of Kansas, 1530 West 15th Street, Lawrence, KS 66049
[b] Institute for the Study of Learning and Expertise, 2164 Staunton Court, Palo Alto, CA 94306

Received 16 November 2016; received in revised form 9 April 2017; accepted 9 May 2017
Available online 18 May 2017

Abstract

Cognitive architectures serve as both unified theories of the mind and as computational infrastructures for constructing intelligent agents. In this article, we review the evolution of one such framework, Icarus, over the three decades of its development. We discuss the representational and processing assumptions made by different versions of the architecture, their relation to alternative theories, and some promising directions for future research.
© 2017 Published by Elsevier B.V.

Keywords: Cognitive architectures; General intelligence; Embodied agents

1. Introduction

Research on cognitive architectures (Newell, 1990) attempts to the specify the computational infrastructure that underlies intelligent behavior. Candidate frameworks specify the facets of cognition that hold constant across different domains. This includes available memories, the representation of elements in these memories, and the processes that operate over these elements, but not the memories' contents, which can change across domains and over time. Thus, a cognitive architecture is similar to a building architecture, which describes its fixed structure and use but not its contents, like people or furniture.

These assumptions about the representations and mechanisms that underlie cognition correspond to theoretical postulates about the nature of the mind. Most cognitive architectures incorporate ideas from psychology about human information processing. They contain distinct modules, but these typically operate over the same memories and representations, rather than being 'black boxes' with communication protocols, as in most work on software engineering. The paradigm views constraints among modules as desirable, rather than something to be avoided, as they should contribute to a unified theory of cognition. Most architectures also come with a programming language that eases construction of intelligent systems; these adopt a syntax that reflects theoretical commitments about representation and processing.

In this paper, we review the development and history of Icarus, a cognitive architecture that shares important features with other frameworks like Soar (Laird, Rosenbloom, & Newell, 1986; Laird, 2012), ACT-R (Anderson & Lebiere, 1998), and Prodigy (Carbonell, Knoblock, & Minton, 1991) but that also makes some distinctive theoretical commitments. We discuss these similarities and differences in the next section, after which we summarize two early designs for the architecture that were superceded by later versions. After this, we describe the main body of Icarus research, followed by extensions and variations that branched from this central core. We conclude by noting

* Corresponding author.
E-mail addresses: dongkyuc@ku.edu (D. Choi), patrick.w.langley@gmail.com (P. Langley).

http://dx.doi.org/10.1016/j.cogsys.2017.05.005
1389-0417/© 2017 Published by Elsevier B.V.

some intellectual influences on Icarus research and directions for future work.

2. Theoretical Claims of ICARUS

As we have noted, cognitive architectures are intended as theories of intelligent behavior. Some frameworks are concerned primarily with explaining human cognition, while others emphasize the construction of intelligent agents. Icarus falls in the middle of this spectrum in that it aims for qualitative consistency with high-level findings from psychology, but it does not attempt to match detailed results from experiments with humans. As Cassimatis, Bello, and Langley (2008) have argued, focusing on such studies can delay achieving broad coverage of cognition functions, which has been our main goal.

Icarus shares several core assumptions with other candidate architectures, including production-system frameworks, that are worth stating explicitly. These include claims that:

- *Short-term memories are distinct from long-term stores*. The former store content that changes rapidly over time, like beliefs and goals; the latter contain elements that are static or change gradually through learning.
- *Memories are collections of symbolic structures*. They contain distinct elements that are typically encoded as list structures, with shared symbols among of items denoting large-scale relations.
- *Relational pattern matching accesses long-term content*. This process maps between relational patterns onto elements in short-term stores; these patterns match when their variables bind consistently with constants across elements.
- *Cognitive processing occurs in recognize-act cycles*. The core interpreter alternates between matching long-term structures against short-term ones, selecting a subset of the former to apply, and executing their associated actions to alter memory or the environment.
- *Cognition dynamically composes mental structures*. Sequential application of long-term knowledge elements alters those in short-term memories, which lead to new matches on later cycles. Similarly, structural learning composes new long-term structures from existing content.

Again, these theoretical assumptions are not novel; our framework holds them in common with Soar, ACT-R, and many other production-system architectures. Problem-space search has been a recurring theme in architectural research, with some frameworks, like Soar and Prodigy, supporting it directly. Icarus shares this feature, but, as we will see, gives the process a different status. In addition, all cognitive architectures can use expert knowledge, stated as long-term structures, in order to reduce or eliminate search completely.

Despite its similarities to alternative frameworks, Icarus makes other assumptions that differentiate it from its predecessors. These include the postulates that:

- *Cognition is grounded in perception and action*. This claim relates to an abiding concern with intelligence in embodied agents that operate in a physical environment.
- *Categories and skills are distinct types of cognitive structure*. This formalizes the intuitive difference between knowledge about situations in the environment and about activities that change them.
- *Short-term elements are instances of long-term structures*. Predicates used to describe dynamic elements, such as beliefs, goals, and intentions, must refer to defined concepts and skills.
- *Long-term knowledge is organized in a hierarchical manner*. Complex concepts and skills are specified as combinations of their simpler constituents.
- *Inference has primacy over execution, which in turn has primacy over problem solving*. Conceptual inference generates content needed for skill execution, and problem solving relies on results from both mechanisms.

Some of these tenets have also appeared elsewhere in the architectural literature, but only Icarus combines them into a unified cognitive theory. Together, they make it a distinctive contribution to theories of mental representation and processing.

3. Early Research on ICARUS

The research programme reported here grew out of a concern with embodied intelligent systems. Until the 1980s, most work on high-level processing, in both AI and cognitive science, focused on primarily mental tasks like puzzle solving, logical reasoning, game playing, and language understanding. Early research on cognitive architectures, typically associated with the production-system paradigm (e.g., Klahr, Langley, & Neches, 1987), reflected this emphasis. In response, Langley, Nicholas, Klahr, and Hood (1981) proposed a simulated three-dimensional environment that would encourage research on embodied cognitive systems, but without requiring expertise in, and the expense associated with, physical robots. The Icarus architecture was developed in direct response to this challenge.

3.1. Initial Designs for Icarus

Langley, Thompson, Iba, and Gennari (1989) presented the earliest complete design for the Icarus framework. Later versions of the architecture diverged from it along multiple fronts, but they have remained largely true to the programme's original objectives. These included: an integrated architecture whose explanatory power resides largely in interactions among its components; consistency with coarse-grained psychological phenomena, including the incremental character of human learning; interaction with an external environment through sensors and effectors, including grounding of symbolic mental structures; and organizaton of long-term memory into a hierarchy of structures that are used for indexing and retrieval.

Although the integrated architecture never saw full implementation, the research team developed three components that addressed different facets of embodied intelligence. These modules included:

- Labyrinth, which encoded a taxonomic hierarchy of concepts for simple and complex objects, classified new entities as instances of these concepts, and updated the taxonomy in response to these inputs (Thompson & Langley, 1991);
- Mæander, which stored a taxonomy of motor skills, executed and monitored selected skills in the environment, and revised its motor knowledge in light of its experiences (Iba, 1991); and
- Dædalus, which stored a hierarchy of plan elements in memory, used them to select operators during the generation of plans with means-ends analysis, and updated its memory based on successes and failures during search (Langley & Allen, 1993).

Despite their focus on quite different cognitive functions, these components[1] shared a common mechanism for storing, retrieving, and acquiring long-term content. Inspired by Fisher's (1987) Cobweb, they assumed that knowledge was stored in a hierarchy of probabilistic categories, each of which summarized specializations below it in the taxonomy. Retrieval involved sorting new items downward through the hierarchy, updating probabilities along the way, and creating new categories when needed. This mechanism made the design for Icarus consistent with key findings on human categories and incremental learning.

3.2. Intermediate Versions of Icarus

Research on the initial Icarus design ended, in part, because its mechanisms for incremental acquisition of probabilistic taxonomies were overly sensitive to training order and did not produce reliable organizations for memory. Langley (1997) rebooted the programme by focusing on knowledge-based reactive control and by introducing handcrafted symbolic content about states and activities, rather than induced probabilistic constructs. Statistical learning still played a role, but only to modulate symbolic structures already present in long-term memory.

Shapiro and Langley (1999) elaborated on these ideas further by introducing a formalism for hierarchical skills and an interpreter for executing them reactively in external environments. Each Icarus skill S included three fields: a set of requirements R that must be satisfied for S to apply, a set of objectives O that S aims to achieve, and a set of means M that specified alternative ways to achieve O when R hold. Each element in a field could be defined in terms of lower-level skills that eventually terminated in primitive sensors or actions.

An important extension to this framework added a value function to each skill that guided their selection during reactive control, along with a method for hierarchical reinforcement learning that updated these functions in response to delayed rewards (Shapiro, Langley, & Shachter, 2001; Shapiro & Langley, 2002). Experiments in a simulated driving environment showed that, when exposed to distinct reward signals, the same hierarchical skills came to generate radically different driving styles. They also revealed that availability of such hierarchical knowledge led to far more rapid learning than knowledge-lean approaches.

4. Mature Versions of ICARUS

Building on this early progress, the Icarus research programme saw substantial advances and consolidation during the ten years from 2003 to 2012, after which the architecture has remained largely stable. The version developed in this period retained the theoretical commitments noted earlier, but they took on new guises that made development more tractable. In this section, we discuss the resulting modules for conceptual inference, teleoreactive execution, problem solving and skill learning, and goal processing. Fig. 1 depicts these components and the memories through which they interact. We also discuss some agents constructed within this framework.

4.1. Conceptual Inference

The initial design for Icarus (Langley et al., 1989) included separate memories for concepts and skills, but the intermediate architecture (Shapiro & Langley, 1999) abandoned the former to focus on the latter. The new version reported by Choi, Kaufman, Langley, Nejati, and Shapiro (2004) reintroduced this distinction and described inference mechanisms that operated over conceptual structures. As Table 1 shows, these take the form of logical rules, each of which associates a conceptual predicate with a conjunction of generalized relational antecedents. The new Icarus uses these rules to define both categories of objects and relations among them.

Some rules define primitive concepts like *line-on-left* and *right-of*, the first two examples in the table. Each such rule includes a `:percepts` field that specifies one or more typed objects with associated attributes and a `:tests` field with zero or more numeric tests on variables bound in the percepts. Primitive conceptual rules let Icarus transform a set of perceived objects with numeric attribute values into

[1] A later design (Langley, McKusick, Allen, Iba, & Thompson, 1991) included a fourth module, argus, to the architecture that would be responsible for transforming continuous perceptions into discrete objects and events.

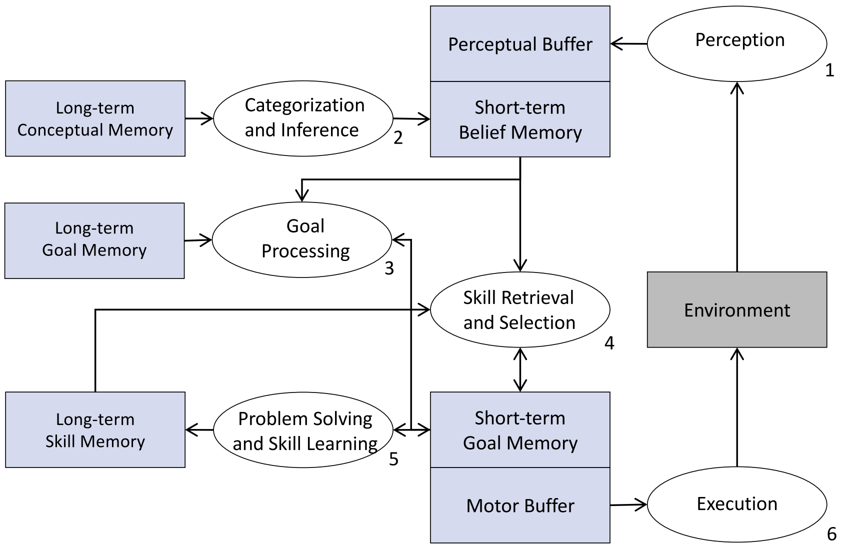

Fig. 1. Icarus' memory stores and the modules that access and alter them on each cognitive cycle.

Table 1
Some Icarus conceptual rules for urban driving adapted from Choi (2011).

```
((line-on-left ?self ?line)
 :percepts    ((self ?self segment ?sg)
               (lane-line ?line segment ?sg dist ?dist))
 :tests       ((< ?dist 0)))

((right-of ?right ?left)
 :percepts    ((lane-line ?right dist ?dist1 segment ?
              segment)
               (lane-line ?left dist ?dist2 segment ?
              segment))
 :tests       ((> ?dist1 ?dist2)))

((in-between ?between ?left ?right)
 :percepts    ((lane-line ?between dist ?dist)
               (lane-line ?left dist ?dist1)
               (lane-line ?right dist ?dist2))
 :relations   ((right-of ?between ?left)
               (right-of ?right ?between)))

((lane ?left ?right)
 :percepts    ((lane-line ?right)
               (lane-line ?left))
 :relations   ((right-of ?right ?left)
               (not (in-between ?any ?left ?right))))

((in-lane ?car ?line1 ?line2)
 :percepts    ((self ?car segment ?sg)
               (lane-line ?line1 segment ?sg)
               (lane-line ?line2 segment ?sg))
 :relations   ((lane ?line1 ?line2)
               (line-on-left ?self ?line1)
               (line-on-right ?self ?line2)))
```

discrete relations that are suitable for symbolic processing. For instance, suppose the agent perceives an object, *lane-line*, with a continuous attribute, *dist*, that describes the distance of the lane line from its perspective. The primitive concept *line-on-left* matches only when the attribute's values fall below zero. This concept discretizes the agent's perceptions into situations that satisfy its conditions and others that do not.

In contrast, nonprimitive rules like those for *in-between*, *lane*, and *in-lane*, the three latter examples in the table, impose a conceptual hierarchy like that depicted in Fig. 2 that captures multiple levels of abstraction. Each nonprimitive rule includes another field, `:relations`, that specifies conditions in terms of conceptual predicates defined in other rules or recursive references to the concept itself with different variable bindings. For instance, the nonprimitive rule for *lane* will be true when the positive condition that refers to *right-of* holds in the environment and when the negated condition that refers to *in-between* does not.[2] Another nonprimitive rule, *in-lane*, will be true when the agent perceives its three positive conditions, *lane*, *line-on-left*, and *line-on-right*, to hold in the environment. Such rules can also include numeric tests as needed. Furthermore, since two or more rules can have the same head, concepts can be disjunctive or recursive.

As shown in Fig. 2, conceptual inference in Icarus operates in a bottom-up manner. On each cognitive cycle, a perceptual process deposits a set of visible objects, each with a type (e.g., *lane-line* and *self*) and associated attributes, into the architecture's perceptual buffer. The inference mechanism first finds all ways that each primitive conceptual rule matches against these objects; for every such instantiation, it adds a primitive belief like *(right-of line2 line1)* and *(line-on-left me line1)* to the agent's belief memory. Once it has completed this step, Icarus matches nonprimitive rules against these elements to infer high-level beliefs like *(in-between line2 line1 line3)* or *(in-lane me line1 line2)*. The architecture repeats this procedure until it has gener-

[2] Unlike early production-system frameworks (e.g., Forgy & McDermott, 1978; Langley, 1983), negated conditions include only one relation. To embed one negation within another requires defining the embedded predicate in a separate conceptual rule.

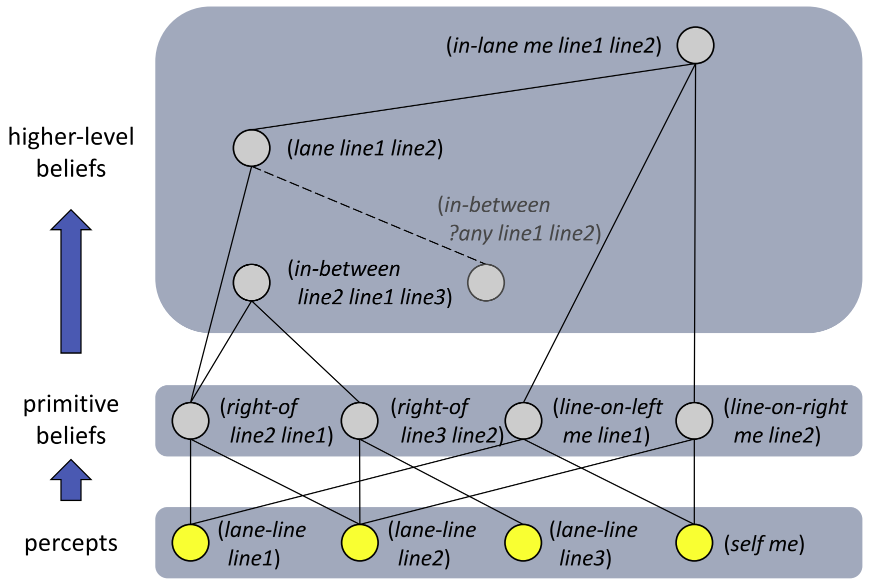

Fig. 2. Bottom-up inference of beliefs in Icarus. The dotted line connected to the pattern in gray denotes a negated relation. For the sake of clarity, we show only a subset of the observed percepts and inferred beliefs.

ated all relations that are implied deductively by its conceptual rules and observed objects.

In addition to providing Icarus with vocabulary for describing discrete situations in the environment, conceptual predicates serve another important role as a notation for agent *goals*. These take the same form as beliefs, except that some arguments may be variables rather than constants, and they may be negated to specify a relationship one wants to avoid. For instance, a driving agent might have the goal *(on-right-side-of-road ?line)*, which does not care about the name of the line that bounds the center lane. Similarly, a blocks world agent might have the goal *(not (on ?any A))*, which states that it wants to have nothing on top of block A. The architecture includes a distinct short-term memory that stores such goal elements, each of which must refer to a predicate defined in one or more conceptual rules, ensuring they are grounded in perception.

4.2. Teleoreactive Skill Execution

Early versions of the Icarus architecture incorporated reactive execution over hierarchical skills, but the introduction of conceptual knowledge lets us elaborate further on this idea. Shapiro and Langley's (1999) skills included a requirements field that specified when they were applicable and an objectives field that determined when they were relevant, but these could refer only to direct sensory tests or to subskills. The introduction of defined concepts lets agents adopt both requirements and objectives that refer to any desired level of abstraction, which extends the architecture's representational power substantially.

In some ways, skill clauses are similar to those in the previous version of Icarus. The first two examples in Table 2 are primitive clauses, each of which has an `:actions` field that specifies actions like *(∗steer 30)* or *(∗cruise)* that the agent can execute in the environment. In contrast, the other three entries are nonprimitive skills with a different field, `:subgoals`, that states how to decompose the task into ordered subgoals. For instance, the fourth skill clause breaks the task *ready-for-right-turn* down into the simpler tasks of achieving *in-rightmost-lane* and *at-turning-speed*. Similarly, the third skill decomposes *on-street* into three subgoals, *ready-for-right-turn*, *in-intersection-for-right-turn*, and *on-street*, in that order. This imposes a hierarchy on Icarus skills much like that for concepts. Also, because two or more clauses can have the same head, skills can be disjunctive or even recursive, much like a context-free grammar.

However, the extended Icarus differs from its predecessor in the ability of skills to reference concepts. Each skill clause includes a `:percepts` field that specifies a set of perceived objects, as in the body of conceptual clauses, but, in addition, the `:conditions` field specifies relational conditions that must hold for the clause to be applicable. These conditions' predicates must be defined in the conceptual knowledge base, which means they can denote primitive concepts or high-level, abstract ones. Each skill also includes a conceptual relation in its head that specifies a goal it aims to achieve. For instance, the first skill clause in Table 2 has the head *(in-lane ?self ?line1 ?line2)*, the second skill mentions *in-intersection-for-right-turn*, and the third refers to *in-rightmost-lane*. Each is defined in terms of more basic relations or percepts from the environment. We will see shortly how the architecture uses these during execution.

In summary, Icarus associates its skills with the goals they achieve and decomposes them into ordered subgoals that index lower-level skills. This distinguishes it from hierarchical task networks or HTNs (e.g., Nau et al., 2003), which comprise a set of methods that are analogous to

Table 2
Some Icarus skills for urban driving adapted from Choi (2011).

```
((in-lane ?self ?line1 ?line2)
 :percepts    ((self ?self))
 :conditions  ((lane-on-right ?self ?line1 ?line2))
 :actions     ((*steer 30)))

((in-intersection-for-right-turn ?self ?int ?c ?tg)
 :percepts    ((self ?self)
               (street ?c)
               (street ?tg)
               (intersection ?int))
 :start       ((on-street ?self ?c)
               (ready-for-right-turn ?self))
 :actions     ((*cruise)))

((in-rightmost-lane ?self ?line1 ?line2)
 :percepts    ((self ?self))
 :conditions  ((rightmost-lane ?line1 ?line2))
 :subgoals    ((in-lane ?self ?line1 ?line2)))

((ready-for-right-turn ?self)
 :percepts    ((self ?self))
 :subgoals    ((in-rightmost-lane ?self ?line1 ?line2)
               (at-turning-speed ?self)))

((on-street ?self ?tg)
 :percepts    ((self ?self)
               (street ?st)
               (street ?tg)
               (intersection ?int))
 :conditions  ((intersection-ahead ?self ?int ?tg)
               (close-to-intersection ?self ?int))
 :subgoals    ((ready-for-right-turn ?self)
               (in-intersection-for-right-turn ?self ?
              int ?st ?tg)
               (on-street ?self ?tg)))
```

skills but that include task names in their heads and subtasks in their bodies. These let one specify arbitrarily complex procedures, but they do not support goal-directed behavior. More recent efforts on hierarchical goal networks (Shivashankar, Kuter, Nau, & Alford, 2012), which grew out of the HTN paradigm, use a notation similar to the one we have described. This is somewhat ironic, as recent versions of Icarus have adopted an HTN-like formalism that uses task names in the heads of skills. However, introduction of an `:effects` field still lets the architecture index skills by the goals they achieve, which is not supported by traditional HTNs.

During execution, Icarus evaluates skills in a top-down fashion, as Fig. 3 illustrates. Each cognitive cycle begins with conceptual inference, described earlier, after which the architecture makes a single execution-related decision. The architecture starts with an unsatisfied top-level goal like *(on-street me A)* and retrieves all skills that are relevant, in that their heads unify with the goal, and that are applicable, in that elements in their `:conditions` field match consistently against current beliefs. When this process retrieves multiple skill clauses, or multiple instantiations of a single clause, a conflict-resolution method (e.g., preference for more recent skills) selects one of the candidates and makes it the agent's current intention.

If this intention is a nonprimitive skill like that shown in the figure, then on the next cycle Icarus examines its subgoals and finds the first one that is not satisfied by the agent's current belief state (e.g., *(ready-for-right-turn me)*). This becomes the agent's current intention, but it retains a link to its parent. The system repeats this process as necessary, retrieving subgoals lower in the skill hierarchy on later cycles until it selects an intention based on a primitive skill (e.g., *(in-lane me line2 line3)*). At this point, Icarus sends the actions associated with that skill to its motor buffer and executes them in the environment.

Of course, this action will take the agent only one step toward achieving its top-level goal. Because a primitive skill *S* can be durative, the architecture may need to execute an intention multiple times, on successive cycles, before it reaches a belief state that satisfies the goal *S*'s own head. Once this occurs, Icarus replaces the completed intention with its parent *P*, which becomes the current focus again. This can lead to selection of another subgoal, a skill that should achieve it, and a related intention *I* or, if actions have achieved *P*'s goal as well, to replacing *I* with its parent. This process repeats so that, across more cycles, the system carries out a sequence of external actions that eventually produce a state that matches the top-level goal. This effectively involves traversing an AND tree, as in Prolog, except that it takes place over time to produce physical change.

Even this scenario assumes that events unfold as predicted by the stored hierarchical skills, but the original reason for making execution reactive is that this will not always happen. In cases where the current intention *I* becomes inapplicable, Icarus abandons it and considers other skills that should achieve the associated goal. If no alternative are applicable, the architecture abandons *I*'s parent intention as well and considers other ways to achieve its goal. This continues until the system finds another approach that should achieve the top-level goal or it determines the environment has changed so drastically that it cannot be reached. Icarus strikes a balance between persistence, which leads it to continue executing a hierarchical skill if possible once it has started, and reactivity, which lets it respond adaptively to unexpected events. This differs from work on Markov decision processes (Puterman, 1994), which reevaluate each alternative after each action, placing them on the extreme end of the reactive spectrum.

4.3. Problem Solving and Skill Learning

Teleoreactive execution is effective when an Icarus agent has a sufficient set of hierarchical skills, and supporting concepts, to handle the situations that cross its path. However, in some cases the architecture may encounter novel goals or situations for which it lacks appropriate knowledge. This requires a mechanism that can decompose problems into smaller ones that it can handle using the skills and concepts already in long-term memory.

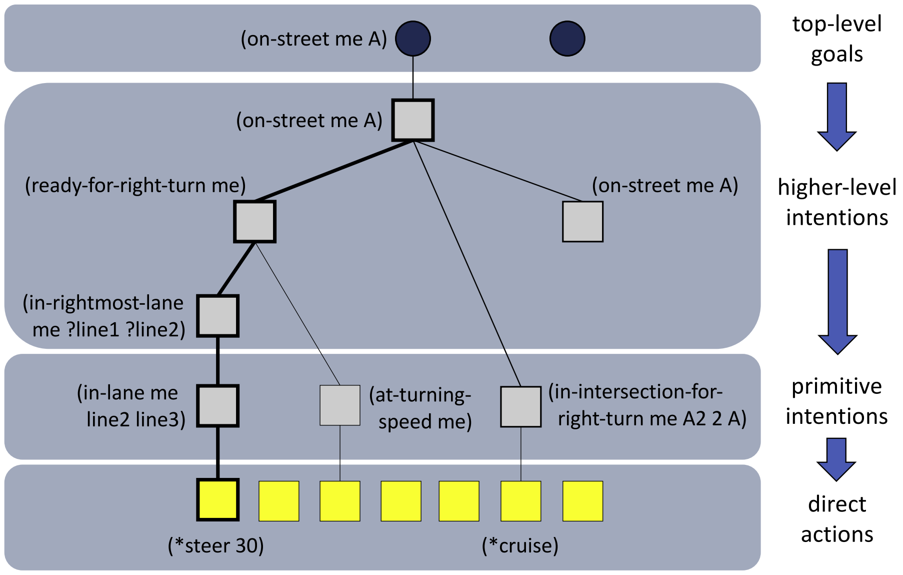

Fig. 3. Top-down selection of subgoals (boxes) and assocated intentions (not shown) by Icarus' execution module on a single cognitive cyle.

As noted earlier, the initial design for Icarus (Langley et al., 1989) included a module for means-ends problem solving, but the intermediate version (Shapiro & Langley, 1999) abandoned this ability. The core architecture incorporates a similar mechanism that we have adapted to operate over the new representations for concepts and skills. Upon encountering an unsatisfied goal or subgoal for which it lacks applicable skills, the problem solver chains backward through skills or conceptual rules to create new intentions and associated subgoals. Icarus invokes this process recursively until it finds an applicable primitive skill, which it executes in the environment,[3] or until it reaches a dead end, in which case it backtracks. This involves search through a space of possible decompositions, which the module pursues in a depth-first fashion, selecting among alternatives at random. Whenever the architecture achieves a goal in this way, it constructs a new skill that encodes the steps involved, using information stored with its intentions.

For instance, Fig. 4 shows a trace of successful means-ends problem solving in the urban driving domain, along with graphics that depict the changes in the environment. Icarus first uses its skill, *steer-for-right-turn*, to chain off the goal s_g and generate the subgoal s_2. Then it uses another skill, *in-intersection-for-right-turn*, to chain off s_2 and produce another goal, s_1. Icarus does not have a skill that achieves the subgoal, *in-rightmost-lane*, but the architecture knows the definition of this predicate, which depends on two subconcepts, *driving-in-segment* and *last-lane*. Since the latter already holds in the current state, s_0, the architecture chains off of the former, which, in turn, can be decomposed into five subconcepts. Icarus selects at random a concept that is currently unsatisfied, *in-lane*, which it can achieve directly from the current state. The system retrieves and executes a skill that should make *in-lane* true in the world. The system achieves the other two unsatisfied concepts, *centered-in-lane* and *aligned-with-lane*, in a similar manner. Together, these let the agent infer the *driving-in-segment* belief, and, in turn, one for *in-rightmost-lane*. The latter appears in the starting condition for the *in-intersection-for-right-turn* skill, which Icarus then executes. After this, the system executes another applicable skill, *steer-for-right-turn*, to achieve the top-level goal, *in-segment*.

Problem-solving traces like the one just described are stored in a distributed manner among the agent's intentions, and Icarus invokes skill learning whenever it achieves a subgoal during interleaved problem solving and execution. New skills learned from a chain that involves a concept, A, encode the achievement of A's subconcepts in the order they were satisfied during problem solving, whereas it places any subconcepts that were already true into the start conditions. New skills learned from a chain that involves a skill, B, include any achievement of B's start conditions as their first steps and then the skill B itself as the final step.

In the problem solving example above, when Icarus achieves *driving-in-segment*, it learns a new skill from this concept chain. The new skill will have as preconditions both *in-segment* and *steering-wheel-straight*, as they were already satisfied during problem solving, and it will include ordered subgoals for *in-lane*, *centered-in-lane*, and *aligned-with-lane*, as Icarus created and solved these as subgoals in that order. In contrast, when the architecture achieves *in-segment* by chaining through a skill, it learns a new structure that has *in-intersection-for-right-turn* (the precondition

[3] This eager strategy can sometimes lead the agent to carry out actions that do not lead to its top-level goal, in which case it must retrace its steps or give up if this is not possible.

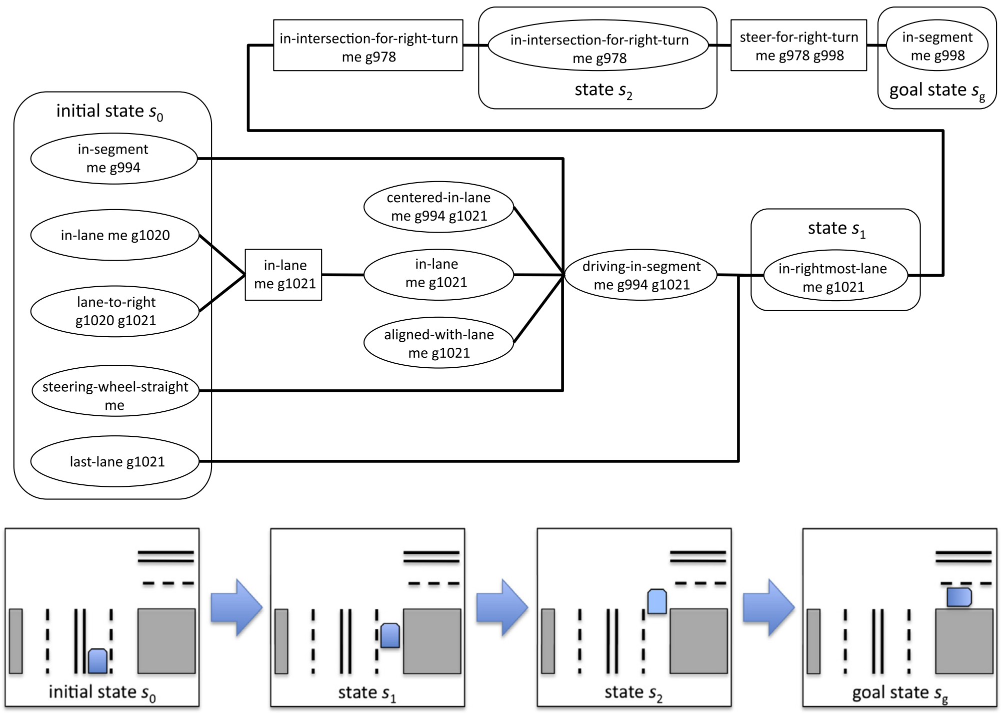
in-intersection-for-right-turn me g978
in-intersection-for-right-turn me g978
state s_2
steer-for-right-turn me g978 g998
in-segment me g998
goal state s_g
initial state s_0
in-segment me g994
in-lane me g1020
lane-to-right g1020 g1021
steering-wheel-straight me
last-lane g1021
in-lane me g1021
centered-in-lane me g994 g1021
in-lane me g1021
aligned-with-lane me g1021
driving-in-segment me g994 g1021
state s_1
in-rightmost-lane me g1021
initial state s_0
state s_1
state s_2
goal state s_g

Fig. 4. A trace of successful problem solving in the urban driving domain as reported in Langley et al. (2009). Ellipses indicate (sub)goals and rectangles denote primitive skills.

Icarus achieved) and *steer-for-right-turn* (the skill it used to chain off of the goal) as ordered subgoals with no preconditions.

4.4. Goal Selection and Generation

Icarus' modules for skill execution and problem solving both assume as inputs a top-level goal the agent wants to achieve and a perceived set of objects, but they say nothing about the selection or orgin of this goal. A more recent addition to the architecture, reported by Choi (2010, 2011), is a module that moves beyond this simple-minded view to support goal reasoning.

One function of this module is to select among the agent's top-level goals. On each cycle, it finds the highest-priority goal that is not currently satisfied by the agent's beliefs or, if there are ties, selects one at random. When priorities are static, this means that Icarus will keep working on a top-level goal until achieving it, after which it will shift attention to the next most important alternative. If changes to the environment cause a previously achieved goal to become unsatisfied, the architecture will focus on it again until the agent has reachieved it.

A more radical extension lets the architecture generate top-level goals, retract them, and change the priorities. This depends on structures stored in a third long-term memory containing rules that specify conditions for creating new goals. Table 3 shows three such rules for urban driving. The first states that, when the agent believes some pedestrian is directly ahead, then it should adopt a goal of being *stopped-and-clear* with respect to that pedestrian, where this predicate refers to a defined concept. The third rule generates a default goal for the agent to be cruising in some lane, but this has lower priority than the first goal.

The goal-processing module examines these rules on each cycle after completing conceptual inference, finding all possible ways in which their conditions match against inferred beliefs. This mechanism may instantiate a goal rule in multiple ways. For instance, if the agent believes there are three pedestrians in its path, the first rule in the table will introduce three elements to goal memory, each with a different pedestrian as its second argument. If the conditions for a rule stop holding on a later cycle, in this case because some of the pedestrians have moved away, the module will remove the corresponding top-level goals it created earlier. Rules with no conditions, like the third example, lead to active goals by default on every cycle.

After determining which goals to adopt on a given cycle, Icarus calculates their relative importance. An initial

Table 3
Sample goal-generating rules stored in Icarus' long-term goal memory, as described by Choi (2011).

```
((stopped-and-clear me ?ped)
   :nominate ((pedestrian-ahead me ?ped))
   :priority 10)

((clear me ?car)
   :nominate ((vehicle-ahead me ?car))
   :priority 5)

((cruising-in-lane me ?line1 ?line2)
   :nominate nil
   :priority 1)
```

implementation simply used the constant specfied in each rule's `:priority` field, but this did not allow values to change with the agent's perceptions. In response, we augmented the representation for concepts to allow degrees of match and used the result to modulate goal priorities. More specifically, the degree of match $0 < m < 1$ for a concept that appears as a condition in a goal rule is multiplied by the number in that rule's `:priority` field to obtain the priority for the generated the current goal instance.

Table 4 presents two concepts that illustrate this capability. These include numeric tests like the angle being equal to 10 and the speed being between 15 and 20. Including the variables *?speed* and *?angle* in the `:pivot` field tells ICARUS that it should compute how close their values come to targets, which in turn determine the degree to which the concept matches. For example, using standard Boolean matching, the second conceptual rule would be satisfied when *?angle* is 10 and false otherwise. In contrast, taking the `:pivot` field into account can produce different degrees of match, such as 0.9 when *?angle* is close to 10 and 0.1 when it is far from this target. As a result, the priorities for ICARUS' top-level goals can vary over time, say as a function of the distance between the agent's vehicle and a pedestrian. This provides a variety of adaptive response that modulates both reactive execution and problem solving.

4.5. Example ICARUS Agents

Throughout the ICARUS research programme, we have used the architecture to construct intelligent agents in a variety of domains. Most of these have involved simulated environments, for the same reasons that Langley et al. (1981) outlined, but we have on occasion used robotic testbeds. Experiments have examined classic problems like the Blocks World, the Tower of Hanoi, multicolumn subtraction, and FreeCell solitaire, although they always included an environment separate from the agent, which had to perceive entities and carry out actions. However, we have also used more challenging settings that include many objects, involve complex relations, and require agents to pursue extended activities.

One such testbed requires an agent to drive a vehicle in a simulated urban environment.[4] Choi, Morgan, Park, and Langley (2007b) developed a three-dimensional virtual city in which agents can perceive street segments, lane lines, buildings, other cars, and pedestrians, and in which they control a vehicle by accelerating, braking, and changing the steering wheel angle. Typical goals included delivering packages to target addresses, picking up passengers, and simply driving around. ICARUS agents for this testbed included concepts for being in the leftmost or rightmost lanes, going at the right speed, approaching a pedestrian, and so on, along with skills that can achieve them.

[4] Shapiro et al. (2001) used simulated highway driving to evaluate an earlier version of the architecture.

Table 4
Concepts for urban driving that let ICARUS calculate degrees of match, taken from Choi (2011).

```
((at-turning-speed ?self)
 :percepts  ((self ?self speed ?speed))
 :tests     ((>= ?speed 15)
             (<= ?speed 20))
 :pivot     (?speed))

((at-steering-angle-for-right-turn ?self)
 :percepts  ((self ?self steering ?angle))
 :tests     ((= ?angle 10))
 :pivot     (?angle))
```

Experimental runs with driving agents demonstrated not only teleoreactive execution using a knowledge base of hierchically organized concepts and skills, but also the ability generate, retract, and prioritize top-level goals in reaction to environmental changes. For example, the agent would drive an ambulance cautiously, observing all traffic signals, in normal situations, but it would also run red lights and ignore speed limits in an emergency. Other driving runs showed that ICARUS could use problem solving to handle unfamiliar tasks and learn hierarchical skills from successful search.

Another challenging testbed was Urban Combat, a simulated environment that is similar to a first-person shooter game, but in which the agent must traverse the landscape to reach targets, find objects, and manipulate them (e.g., defuse explosive devices) rather than fighting with enemies. This domain raised challenges of representing and reasoning about both interior and exterior spaces, multi-level buildings, and unexpected blocks to paths. Choi, Könik, Nejati, Park, and Langley (2007a) reported an ICARUS agent that plays Urban Combat using conceptual knowledge about object categories, pathways, and topological connections, as well as skills that let it overcome obstacles and move between connected regions. Experimental studies showed the system could use conceptual inference to reason about spatial relation, problem solving to generate novel plans, teleoreactive execution to carry them out, and learning to construct new skills. Moreover, the architecture transferred this acquired knowledge to different missions, reducing the learning needed on them.

We have also demonstrated ICARUS' ability to control a humanoid robot. Choi, Kang, Lim, and You (2009) presented results in a realistic simulated environment on a set of blocks-world tasks, such as building a tower and sorting objects based on color. Modules outside the architecture handled perception and low-level manipulation, but tasks nevertheless required substantial inference and execution of extended action sequences. Kim, Lee, Choi, Park, and You (2010) described a similar set of studies using a physical humanoid robot that manipulated real-world objects. Here the agent had to move to a table, avoid obstacles and take detours along the way, and then sort blocks on the table by their colors. Again, ICARUS called on external software to perceive the environment and handle low-level motion, but these demonstrations offered compelling evidence that the architecture supports embodied intelligent agents, although we did not test problem solving, skill learning, or goal reasoning in either testbed. Ongoing work focuses on agents that control nonhumanoid robots, including drones.

5. Extensions to the Architecture

We have described the primary path of Icarus research, starting from early designs through a mature architecture that incorporates conceptual inference, teleoreactive execution, problem solving and skill acquisition, and goal processing. We have also seen that more recent versions of the architecture have supported construction of embodied agents that operate in complex external environments. However, we should also examine, more briefly, some branches off the main trunk that have explored other important abilities. We treat them separately here because they do not build on one another, as have the central mechanisms already described.[5]

5.1. Temporal Reasoning

One of Icarus' core assumptions is that concepts, beliefs, and goals describe aspects of environment states, whereas skills and intentions describe activities that take place over time. However, Stracuzzi, Li, Cleveland, and Langley (2009) noted that this distinction does not support the recognition of activities themselves. In response, they augmented the architecture's representation to include temporal information in both short-term and long-term structures. This included adding time stamps to beliefs that indicated when they were adopted and later abandoned. The revised notation for concepts, which also referred to time stamps, included a field specifying constraints on temporal orders that matched elements must satisfy. These did not refer to actions and so remained distinct from skills, but their difference was lessened.

Stracuzzi et al. (2009) modified conceptual inference in minor ways to take advantage of the new representation. The module still processed concepts in a bottom-up manner, but it examined time stamps and temporal constraints during matching. Moreover, although Icarus updated inferred beliefs' end time stamps when they became false, it retained them indefinitely, providing a simple form of episodic memory that recorded past events. This let the system reason about temporal concepts using nearly the same deductive process as that for simpler beliefs. They demonstrated the extended architecture's ability to encode complex football plays and to recognize them from perceptual traces obtained from a game simulator.

[5] Other extensions to Icarus, which we do not have space to review here, include using learned values to guide conceptual inference (Asgharbeygi, Nejati, Langley, & Arai, 2005), goal-directed analogical mapping to support transfer (Könik et al., 2009), and learning new conceptual predicates from problem solving (Li, Stracuzzi, & Langley, 2012).

5.2. Extended Problem Solving

We have seen that Icarus' problem solver focuses on *goals*, which drive the creation of new subproblems and serve to indicate their successful solution. Langley and Trivedi (2013) observed that this keeps it from reasoning about interactions among different goals, which led them instead to organize the process around *problems*, that is, tasks for transforming one state into another that satisfies a set of goals. This let the revised framework distinguish among distinct formulations of a given problem, based on different ways the initial state partially satisfies the specified goals.

Langley and Trivedi incorporated these representational extensions into a new means-ends problem solver that took multiple goals into account when retrieving and selecting skills. The module could also generate subproblems by chaining off unmatched skills with negated conditions containing nonprimitive predicates. Both changes led this version of Icarus to search a larger space than its predecessor, which it mitigated with domain-independent heuristics that favored skill instances whose conditions matched more beliefs and whose effects would achieve more goals. The authors reported results on six domains, including puzzles like the Tower of Hanoi and common planning tasks, although their studies examined mental problem solving apart from execution in an external environment.

5.3. Learning from Failure

As described earlier, Icarus acquires new skills whenever it solves a new problem or subproblem, but humans learn not only from problem-solving successes but also from failures. Choi and Ohlsson (2010) developed an extended version of the architecture that acquires new skills when execution leads to violations of constraints. They encoded this new type of long-term knowledge as satisfaction–relevance pairs. The former specified relations that should hold when the latter conditions are satisfied. Table 5 presents two such constraints for a blocks world. The first states that if one block is on another, then they should have the same color; the other indicates that the upper block should be smaller than the lower one.

Choi and Ohlsson modified the architecture to take these new structures into account. After conceptual inference, it checked belief memory to determine which constraints were relevant and, if so, whether they were satisfied. Upon noting a constraint violation, the extended Icarus created specialized versions of the skill

Table 5
Some sample constraints for the Blocks World provided in Choi and Ohlsson (2010).

(color	:relevance	((on ?a ?b))
	:satisfaction	((same-color ?a ?b)))
(width	:relevance	((on ?a ?b))
	:satisfaction	((smaller-than ?a ?b)))

that produced the undesirable situation. This ensured that it would only apply either when the satisfaction conditions held or when the relevance conditions did not, thus avoiding constraint violations in the future. For example, if execution of a block-stacking skill led to violation of the width constraint in Table 5, the system created a variant rule that included the *smaller-than* relation, ensuring that, in later runs, the agent would sidestep similar errors.

Danielescu, Stracuzzi, Li, and Langley (2010) reported a complementary approach to learning from failure. They focused on situations in which the agent selects and executes a skill to achieve a low-priority goal but, in the process, unintentionally undoes a higher-priority one that was previously satisfied. When the extended architecture detected such a situation, it defined a new concept that combined the two interacting predicates and attempted to find some way to achieve one without undoing the other. Using counterfactual reasoning over an episodic trace, the extended architecture revisited past belief states and invoked problem solving to find a solution that would achieve the new, conjoined concept. The system then constructed a new skill that, because its head referred to this more specific concept, it would prefer over the more generic skill that caused the interaction. The authors demonstrated this extension on a lane-changing scenario from the urban driving domain.

5.4. Learning from Observation

Another of Icarus' limitations has been that it learns only from its own behavior, which requires either extensive search during problem solving or making errors during execution. Nejati, Langley, and Könik (2006) explored another alternative – learning skills from worked-out solutions – that avoids these two issues. Their approach required no changes to the representation of concepts or skills, and it relied on standard mechanisms for inference and execution. However, they introduced a new learning module that attempted to generate an explanation, in terms of background knowledge, for how an observed sequence of skill instances achieved a given goal. Explanations took a hierarchical form, with primitive skills as terminal nodes, that translated directly into a set of learned hierarchical skills, with conditions determined in the same way as earlier. The extended architecture acquired a set of skills in this manner from each solution sequence.

A later version of the module (Nejati, 2011) included augmented abilities for ignoring irrelevant actions, which it simply omitted from explanations and learned skills. A related mechanism also let the system handle traces that involved interleaved solution to different problems, from which it acquired distinct sets of skills. An alternative system, described by Li, Stracuzzi, Langley, and Nejati (2009) instead adapted Icarus' means-ends problem solver to explain observed solution traces. This had the advantage of using an existing architectural module, including its associated learning process, rather than introducing another one, but the system did not include the ability to handle interleaved solutions.

6. Intellectual Precursors

We have already noted some features that Icarus shares with alternative cognitive architectures and others that make it distinctive, but we should also discuss additional intellectual influences from which it borrows. Like Soar, ACT-R, and Prodigy, our framework draws heavily on two ideas championed by Newell and Simon (1976). The *physical symbol system* hypothesis states that the ability encode and manipulate symbol structures – organized sets of persistent patterns – provide the means for building general intelligent systems. The *heuristic search* hypothesis claims that problem solving involves search through a space of states generated by operators, both symbolic structures, and guided by heuristics. These ideas have constrained the Icarus design in major ways, although, as we have noted, problem-space search is more fundamental to Soar and Prodigy than to our framework.

Another important, historically more recent, influence comes from symbolic approaches to reactive control, often associated with robotics. Icarus' original concern with embodied agents dates back to the mid-1980s, but Nilsson's (1994) notion of *teleoreactive programs* played a central role in the architecture's design starting with Langley (1997). Research on integrating planning with reactive control, such as Bonasso et al.'s (1997) 3T framework, has also influenced our efforts over the past 20 years. However, Icarus' specific problem solver, and its method for interleaving this process with execution, comes directly from Newell, Shaw, and Simon's (1960) General Problem Solver, which introduced means-ends analysis, an approach that lends itself naturally to such integration.

Icarus also incorporates more ideas from the logic and planning communities than other cognitive architectures. We have seen that its representation for conceptual knowledge takes a form similar to the rules in Prolog (Clocksin & Mellish, 1981), although its mechanism for using these structures differs substantially. Also, its notation for skill knowledge has much in common with Nau et al.'s (2003) SHOP2 formalism for hierarchical task networks. However, they were developed independently, and the latter uses hierarchical knowledge to guide planning, whereas our architecture uses it primarily during teleoreactive execution.

One of Icarus' key abilities is acquisition of such hierarchical skills from successful problem solving. This bears similarities to the analytical learning methods in Soar and Prodigy, although they create search-control rules rather than hierarchical structures. On this dimension, Icarus is closer to Ruby and Kibler's (1991) SteppingStone, Marsella and Schmidt's (1993) PRL, and Reddy and Tadepalli's (1997) X-Learn. Each system constructed problem-decomposition rules from the results of successful search, although we believe our approach interleaves learning with problem solving in a more natural way.

The architecture's approach to goal processing also has deep roots. Simon (1967) argued the need for an interruption mechanism that shifts and focuses an agent's attention in a complex environment, whereas Sloman (1987) noted that conflicting goals require a means for resolving them, for which he proposed motivational processes. More recent efforts by Hanheide et al. (2010), Molineaux, Klenk, and Aha (2010), and Talamadupula, Benton, Schermerhorn, Kambhampati, and Scheutz (2010) have explored similar ideas, although these were developed in parallel with the Icarus approach to goal processing.

7. Limitations and Plans for Future Work

Despite Icarus' theoretical attractions and its usefulness for developing intelligent agents, the framework still has important limitations that we should address in future research. The most basic extension would introduce a hybrid representation and processes that operate over it. The current architecture distinguishes between percepts, which describe perceived objects in terms of numeric attributes, and beliefs, which denote symbolic relations among these objects. A more balanced approach would associated both symbolic and numeric information at all levels, with percepts becoming beliefs about individual objects. Langley, Barley, Meadows, Choi, and Katz (2016a) report a new architecture that adopts this idea, along with inference mechanisms that derive higher-level attributes from lower-level ones, and we should incorporate similar ideas into our framework.

Another drawback is that Icarus' conceptual inference module is both deductive and exhaustive. This means it cannot introduce plausible default assumptions to explain observations, and inference time can grow exponentially with the number of visible objects. Future versions should support an abductive form of inference, as reported by Meadows, Langley, and Emery (2014), and should produce reasonable results even when time constraints require early termination. We should also explore a return to probabilistic approaches to conceptual representation, inference, and learning, as in the earliest architectural design (Langley et al., 1989).

A third limitation is Icarus' fixed strategies for both problem solving and for interleaving it with execution. Humans often exhibit means-ends analysis, but they sometimes use other methods, such as forward search. Similarly, they may initiate execution when they solve a subproblem, but they may instead start acting before they find a complete subplan, as in game playing. This variability suggests that humans adapt their problem-solving and execution strategies to their situation, which means that future versions of Icarus should support a range of methods under its control. One approch, reported by Langley, Pearce, Bai, Barley, and Worsfold (2016b), uses modifiable parameters that determine aspects of problem solving, including methods for selecting among alternatives, criteria for success and failure, and responses to each case.

Another promising direction for future work involves adding a memory for episodic traces and mechanisms that operate over it, as Laird (2012) has reported for Soar. This would store and retrieve information not only about beliefs generated by conceptual inference, as described by Stracuzzi et al. (2009), but also about decisions made during problem solving and actions carried out during execution. Such an episodic memory could support not only question answering, but also explanation of agent behavior and analogical reasoning. Along with the extensions outlined earlier, this would bring Icarus much closer to reproducing the broad range of abiities we associate with human intelligence.

8. Concluding Remarks

In this article, we reviewed the history and developed of the Icarus cognitive architecture. The saw that it shares core theoretical tenets with other unified theories of the mind, like Soar and ACT-R, but that it also takes distinctive positions on key issues. The latter include architecture-level commitments to grounding cognition in perception and action, separating conceptual knowledge from skills, organizing memory in a hierarchical manner, and supporting problem solving with more basic mechanisms of inference and execution.

We discussed two early versions of Icarus that incorporated most, but not all, of these assumptions, and then described in more detail a third incarnation that has been stable over the past 12 years. This includes modules for perceptually-driven conceptual inference, goal-driven reactive execution, problem solving with means-ends analysis, skill acquisition from novel solutions, and generation of top-level goals. We also mentioned some variants off this main branch that support additional functionality. In closing, we examined Icarus' many intellectual precursors and some important directions for future work.

Over the past 30 years, the Icarus research programme has introduced important ideas into the literature on cognitive architectures and broadened discussion about theories of intelligence in humans and machines. We will not claim that it has produced a unified theory of the mind, as many high-level phenomena remain unexplained, but we will argue that Icarus provides an interesting and coherent account of many abilities associated with human intelligence, and that it offers a fertile framework for further research in this challenging arena.

Acknowledgments

The research presented here was supported in part by grants and contracts from the National Science Foundation (IIS-0335353), the Defense Advanced Research Projects Agency (HR0011-04-1-0008, FA8750-05-2-0283, FA8650-06-C-7606), the Office of Naval Research (N00014-09-1-0125, N00014-09-1-1029, N00014-10-1-0487, N00014-12-1-0143, N00014-15-1-2517), the Korea Institute of Science and Technology (2E20870), and the Naval Research Laboratory (N00173-17-P-0837), which are not responsible for its content. We thank Andreea Danielescu, Tolga Könik, Nan Li, Negin Nejati, Seth Rogers, Daniel Shapiro, David Stracuzzi, and Nishant Trivedi for their contributions to Icarus' development.

References

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

Asgharbeygi, N., Nejati, N., Langley, P., & Arai, S. (2005). Guiding inference through relational reinforcement learning. In *Proceedings of the Fifteenth International Conference on Inductive Logic Programming* (pp. 20–37). Bonn, Germany: Springer Verlag.

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

Carbonell, J. G., Knoblock, C. A., & Minton, S. (1991). Prodigy: An integrated architecture for planning and learning. In K. VanLehn (Ed.), *Architectures for intelligence*. Hillsdale, NJ: Lawrence Erlbaum.

Cassimatis, N. L., Bello, P., & Langley, P. (2008). Ability, breadth, and parsimony in computational models of higher-order cognition. *Cognitive Science, 32*, 1304–1322.

Choi, D. (2010). Nomination and prioritization of goals in a cognitive architecture. In *Proceedings of the Tenth International Conference on Cognitive Modeling* (pp. 25–30). Philadelphia, PA.

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

Choi, D., Kang, Y., Lim, H., & You, B.-J. (2009). Knowledge-based control of a humanoid robot. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems* (pp. 3949–3954). Louis, MO: IEEE press.

Choi, D., Kaufman, M., Langley, P., Nejati, N., & Shapiro, D. (2004). An architecture for persistent reactive behavior. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multi Agent Systems* (pp. 988–995). New York: ACM Press.

Choi, D., Könik, T., Nejati, N., Park, C., & Langley, P. (2007a). Structural transfer of cognitive skills. In *Proceedings of the Eighth International Conference on Cognitive Modeling* (pp. 115–120). Ann Arbor, MI.

Choi, D., Morgan, M., Park, C., & Langley, P. (2007b). A testbed for evaluation of architectures for physical agents. In *Proceedings of the AAAI-2007 Workshop on Evaluating Architectures for Intelligence*. Vancouver, BC: AAAI Press.

Choi, D., & Ohlsson, S. (2010). Learning from failures for cognitive flexibility. In *Proceedings of the Thirty-Second Annual Meeting of the Cognitive Science Society* (pp. 2099–2104). Portland, OR.

Clocksin, W., & Mellish, C. (1981). *Programming in Prolog*. New York: Springer-Verlag.

Danielescu, A., Stracuzzi, D.J., Li, N., & Langley, P. (2010). Learning from errors by counterfactual reasoning in a unified cognitive architecture. In *Proceedings of the Thirty-Second Annual Meeting of the Cognitive Science Society* (pp. 2566–2571). Portland, OR.

Fisher, D. H. (1987). Knowledge acquisition via incremental conceptual clustering. *Machine learning, 2*, 139–172.

Forgy, C.L., & McDermott, J. (1978). The OPS2 reference manual. Technical Report Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA.

Hanheide, M., Hawes, N., Wyatt, J., Göbelbecker, M., Brenner, M., Sjöö, K., ... Kruijff, G.-J. M. (2010). A framework for goal generation and management. In *Proceedings of the AAAI-2010 Workshop on Goal-Directed Autonomy*. Atlanta: AAAI Press.

Iba, W. (1991). Acquisition and improvement of human motor skills: learning through observation and practice. Ph.D. thesis University of California, Irvine, CA.

Kim, K., Lee, J.-Y., Choi, D., Park, J.-M., & You, B.-J. (2010). Autonomous task execution of a humanoid robot using a cognitive model. In *Proceedings of the IEEE International Conference on Robotics and Biomimetics* (pp. 405–410). Tianjin, China.

Klahr, D., Langley, P., & Neches, R. T. (1987). *Production system models of learning and development*. Cambridge, MA: MIT press.

Könik, T., O'Rorke, P., Shapiro, D., Choi, D., Nejati, N., & Langley, P. (2009). Skill transfer through goal-driven representation mapping. *Cognitive Systems Research, 10*, 270–285.

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

Laird, J. E., Rosenbloom, P. S., & Newell, A. (1986). Chunking in Soar: The anatomy of a general learning mechanism. *Machine Learning, 1*, 11–46.

Langley, P. (1983). Exploring the space of cognitive architectures. *Behavior Research Methods and Instrumentation, 15*, 289–299.

Langley, P. (1997). Learning to sense selectively in physical domains. In *Proceedings of the First International Conference on Autonomous Agents* (pp. 217–226). ACM Press.

Langley, P., & Allen, J. A. (1993). A unified framework for planning and learning. In S. Minton (Ed.), *Machine learning methods for planning*. San Mateo, CA: Morgan Kaufmann.

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

Langley, P., Choi, D., & Rogers, S. (2009). Acquisition of hierarchical reactive skills in a unified cognitive architecture. *Cognitive Systems Research, 10*, 316–332.

Langley, P., McKusick, K. B., Allen, J. A., Iba, W. F., & Thompson, K. (1991). A design for the icarus architecture. *ACM SIGART Bulletin, 2*, 104–109.

Langley, P., Nicholas, D., Klahr, D., & Hood, G. (1981). A simulated world for modeling learning and development. In *Proceedings of the Third Conference of the Cognitive Science Society* (pp. 274–276). Berkeley, CA.

Langley, P., Pearce, C., Bai, Y., Barley, M., & Worsfold, C. (2016b). Variations on a theory of problem solving. In *Proceedings of the Fourth Annual Conference on Advances in Cognitive Systems*. Evanston, IL.

Langley, P., Thompson, K., Iba, W., Gennari, J., & Allen, A.J. (1989). An integrated cognitive architecture for autonomous agents. Technical Report 89-28 Department of Information & Computer Science, University of California, Irvine, CA.

Langley, P., & Trivedi, N. (2013). Elaborations on a theory of human problem solving. In *Proceedings of the Second Annual Conference on Advances in Cognitive Systems*. Baltimore, MD.

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

Li, N., Stracuzzi, D.J., Langley, P., & Nejati, N. (2009). Learning hierarchical skills from problem solutions using means-ends analysis. In *Proceedings of the Thirty-First Annual Meeting of the Cognitive Science Society* (pp. 1858–1863). Amsterdam, Netherlands.

Marsella, S., & Schmidt, C. F. (1993). A method for biasing the learning of nonterminal reduction rules. In S. Minton (Ed.), *Machine learning methods for planning*. San Mateo, CA: Morgan Kaufmann.

Meadows, B., Langley, P., & Emery, M. (2014). An abductive approach to understanding social interactions. *Advances in Cognitive Systems, 3*, 87–106.

Molineaux, M., Klenk, M., & Aha, D. W. (2010). Goal-driven autonomy in a Navy strategy simulation. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence* (pp. 1548–1554). Atlanta: AAAI Press.

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

Nejati, N. (2011). Analytical goal-driven learning of procedural knowledge by observation. Ph.D. thesis Stanford University, Stanford, CA.

Nejati, N., Langley, P., & Könik, T. (2006). Learning hierarchical task networks by observation. In *Proceedings of the Twenty-Third International Conference on Machine Learning* (pp. 665–672). Pittsburgh, PA.

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

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

Newell, A., & Simon, H. A. (1976). Computer science as empirical enquiry: Symbols and search. *Communications of the ACM, 19*, 113–126.

Nilsson, N. (1994). Teleo-reactive programs for agent control. *Journal of Artificial Intelligence Research, 1*, 139–158.

Puterman, M. L. (1994). *Markov decision processes*. New York: John Wiley.

Reddy, C., & Tadepalli, P. (1997). Learning goal-decomposition rules using exercises. In *Proceedings of the Fourteenth International Conference on Machine Learning* (pp. 278–286). San Francisco: Morgan Kaufmann.

Ruby, D., & Kibler, D. (1991). SteppingStone: An empirical and analytical evaluation. In *Proceedings of the Tenth National Conference on Artificial Intelligence* (pp. 527–532). Menlo Park, CA: AAAI Press.

Shapiro, D., & Langley, P. (1999). Controlling physical agents through reactive logic programming. In *Proceedings of the Third Annual Conference on Autonomous Agents* (pp. 386–387). Seattle: ACM Press.

Shapiro, D., & Langley, P. (2002). Separating skills from preference: Using learning to program by reward. In *Proceedings of the Nineteenth International Conference on Machine Learning* (pp. 570–577). Sydney, Australia: Morgan Kaufmann.

Shapiro, D., Langley, P., & Shachter, R. (2001). Using background knowledge to speed reinforcement learning in physical agents. In *Proceedings of the Fifth International Conference on Autonomous Agents* (pp. 254–261). Montreal: ACM Press.

Shivashankar, V., Kuter, U., Nau, D., & Alford, R. (2012). A hierarchical goal-based formalism and algorithm for single-agent planning. In *Proceedings of the Eleventh International Conference on Autonomous Agents and Multiagent Systems* (pp. 981–988). Valencia, Spain: International Foundation for Autonomous Agents and Multiagent Systems.

Simon, H. A. (1967). Motivational and emotional controls of cognition. *Psychological Review, 74*, 29–39.

Sloman, A. (1987). Motives, mechanisms, and emotions. *Cognition & Emotion, 1*, 217–233.

Stracuzzi, D.J., Li, N., Cleveland, G., & Langley, P. (2009). Representing and reasoning over time in a symbolic cognitive architecture. In *Proceedings of the Thirty-First Annual Meeting of the Cognitive Science Society* (pp. 2986–2991). Amsterdam, Netherlands.

Talamadupula, K., Benton, J., Schermerhorn, P., Kambhampati, S., & Scheutz, M. (2010). Integrating a closed world planner with an open world robot: A case study. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence* (pp. 1561–1566). Atlanta: AAAI Press.

Thompson, K., & Langley, P. (1991). Concept formation in structured domains. In D. H. Fisher, M. J. Pazzani, & P. Langley (Eds.), *Concept formation: Knowledge and experience in unsupervised learning* (pp. 127–161). San Mateo, CA: Morgan Kaufmann.