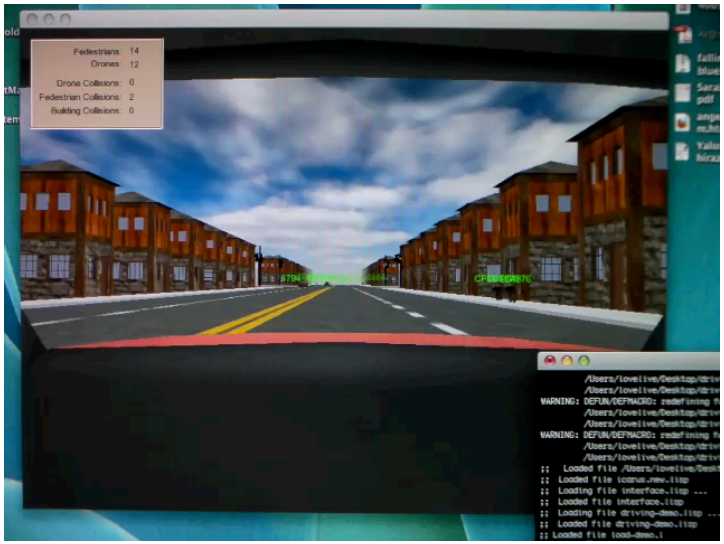
CogSci-2011 Tutorial on the ICARUS Cognitive Architecture

Pat Langley
Computer Science and Psychology
Arizona State University, Tempe

Dongkyu Choi
Psychology Department
University of Illinois, Chicago

Thanks to G. Cleveland, G. Iba, T. Konik, N. Li, N. Nejati, D. Shapiro, D. Stracuzzi, and N. Trivedi for their contributions. This talk presents research partly funded by grants from ONR and DARPA, which are not responsible for its contents.

Synthetic Agents for Urban Driving



Outline for the Tutorial

- Notion of a cognitive architecture
- Overview of the ICARUS architecture
- List processing and pattern matching
- Conceptual inference
  - Structures, processes, exercises
- Skill execution
  - Structures, processes, exercises
- Problem solving
  - Structures, processes, exercises
- Some ICARUS applications
- Open issues / Other architectures

Theory
+
Implementation

What is a Cognitive Architecture?

A cognitive architecture (Newell, 1990) is an infrastructure for intelligent systems that:

- Specifies the facets of cognition that hold constant across different domains;
- Including memories and the representations of elements in those memories;
- But not the content of those memories, which can change across domains and over time.

A cognitive architecture is analogous to a building architecture, which describes its fixed structure but not its contents.

What is a Cognitive Architecture?

A cognitive architecture provides more than an integration framework for intelligent systems, in that it:

- makes strong theoretical assumptions about the representations and mechanisms underlying cognition;
- incorporates many ideas from psychology about the nature of human cognition;
- contains distinct modules, but these access and alter the same memories and representations;
- comes with a programming language with a high-level syntax that reflects the theoretical assumptions.

A cognitive architecture is all about mutual constraints, as it aims to provide a *unified* theory of the mind.

The ICARUS Architecture

ICARUS (Langley & Choi, 2006) is a computational theory of the human cognitive architecture that posits:

1. Short-term memories are distinct from long-term stores
2. Memories contain modular elements cast as symbolic structures
3. Long-term structures are accessed through pattern matching
4. Cognitive processing occurs in retrieval/selection/action cycles
5. Cognition involves dynamic composition of mental structures

These assumptions are not novel; it shares them with architectures like Soar (Laird et al., 1987) and ACT-R (Anderson, 1993).

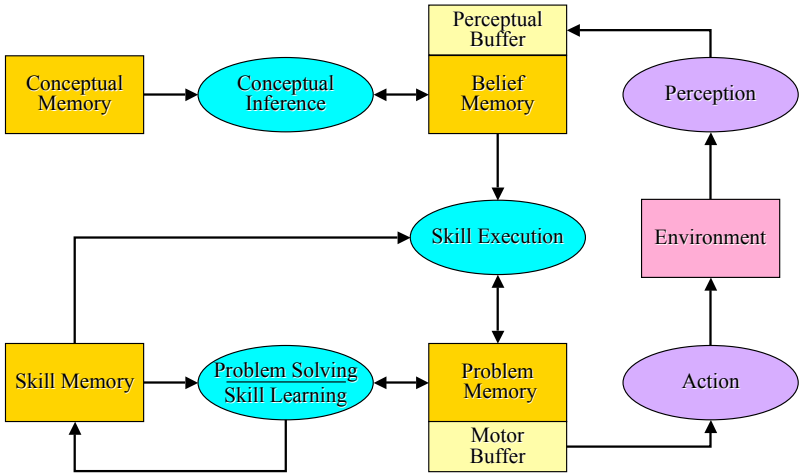
Distinctive Features of ICARUS

However, ICARUS also makes assumptions that differentiate it from other architectures:

1. Cognition is grounded in perception and action
2. Categories and skills are separate cognitive entities
3. Short-term elements are instances of long-term structures
4. Long-term knowledge is organized in a hierarchical manner
5. Inference and execution are more basic than problem solving

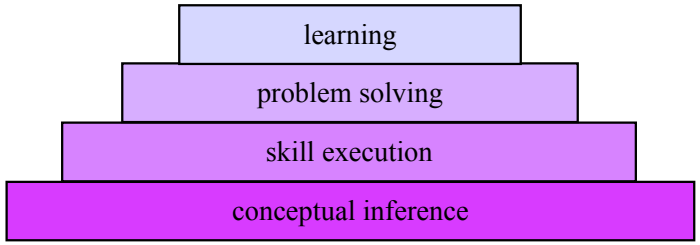
Some of these tenets also appear elsewhere, but only ICARUS combines them into a unified cognitive theory.

ICARUS' Memories and Processes



Cascaded Integration in ICARUS

Like other unified cognitive architectures, ICARUS incorporates a number of distinct modules.



ICARUS adopts a *cascaded* approach to integration in which lower-level modules produce results for higher-level ones.

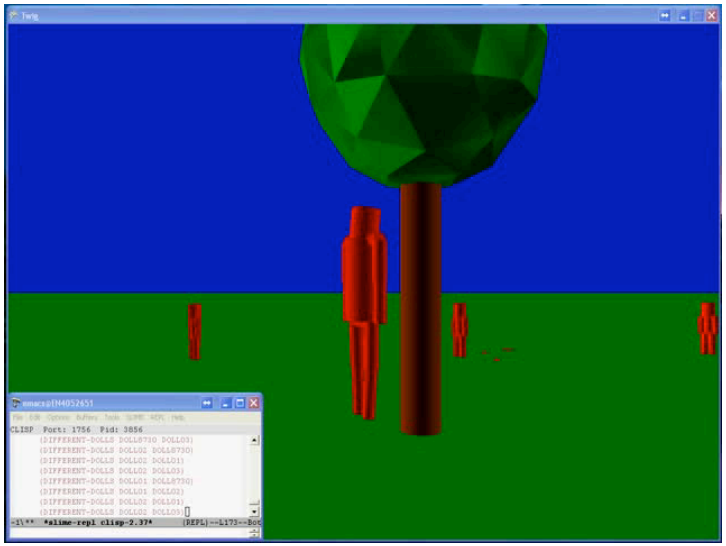
Research Goals for ICARUS

Our current objectives in developing ICARUS are to produce:

- a computational theory of high-level cognition in humans
- that is qualitatively consistent with results from psychology
- that exhibits as many distinct cognitive functions as possible
- that supports creation of intelligent agents in many settings

Modeling quantitative experimental results has its place but can delay achieving broad coverage (Cassimatis et al., 2009).

Robin Hood in the Twig Environment



Intellectual Precursors

Like many other architectures, ICARUS builds on four key ideas that are basic to cognitive science:

- Physical symbol systems
- List structures and list processing
- Symbolic patterns and pattern matching
- Relational rules and dynamic composition

These insights date back to the field's earliest days, but they remain central to accounts of high-level cognition.

List Structures and List Processing

In 1956, Newell, Shaw, and Simon introduced *list processing*.

This new framework became important to cognitive science because it could:

- Encode arbitrarily complex *structural descriptions* using
  - Symbols such as *on*, *A*, and *B*
  - Lists such as *(on A B)* and *(eats agent John object soup)*
  - List structures such as *(goal me (not (on ?any B)))*
- Create new structural descriptions *dynamically*
- Use such structures to *designate* other structures
- *Interpret* these structures to produce behavior

These abilities play crucial roles in developing computational models of high-level processing.

Symbolic Patterns and Pattern Matching

Most human knowledge is generic in that it applies to distinct instances of the same class of situations.

We can state such knowledge as symbolic *patterns* that:

- Describe the structures held in common by these instances
- Designate the class of situations with these structures
- Omit structures that instances do not hold in common
- Use variables to indicate subelements that may vary

Many symbolic systems interpret patterns by *matching* them against other symbol structures that describe a situation.

This opens the door to 'declarative' knowledge representations that support very flexible processing.

Symbolic Patterns as List Structures

We can encode symbolic patterns as lists or list structures that contain pattern-match variables. For instance,

(taller-than ?person1 ?person2) and *(on ?block1 ?block2)*

include variables *?person1*, *?person2*, *?block1*, and *?block2*.

We can encode complex patterns as lists of simple patterns that may share variables. For example,

((taller-than ?person1 ?person2) (taller-than ?person2 ?person3))

((on ?b1 ?b2) (on ?b2 ?b3) (wider ?b1 ?b2) (wider ?b2 ?b3))

are two complex patterns that have two and four subpatterns, respectively.

Simple Matches and Bindings

We can attempt to match a simple pattern against a *ground literal* (a list or list structure with no variables).

A match *succeeds* if there exists a consistent set of bindings (a substitution of constants for variables) that gives the literal.

The pattern *(on ?block1 ?block2)* matches against the ground literal *(on A B)* with the bindings

((?block1 . A) (?block2 . B)) ,

where a dotted pair denotes a variable and an associated constant.

Different variables can map to the same constant, but each given variable must map to only one constant.

Finding All Simple Matches

Symbolic patterns require content against which to match. We will refer to this set of ground literals as *working memory*.

Given a simple pattern P and the contents of working memory, one can find all matches of P against these elements.

For the simple pattern *(on ?block1 ?block2)* and the memory *((on B C) (on C D) (on A B))*, there are three matches:

(on A B) with bindings *((?block1 . A) (?block2 . B))*
(on B C) with bindings *((?block1 . B) (?block2 . C))*
(on C D) with bindings *((?block1 . C) (?block2 . D))*

Note that some other possible substitutions or bindings, such as *((?block1 . A) (?block2 . C))* are not legitimate matches.

Finding All Complex Matches

Given a *complex* pattern P and the contents of working memory, one can find all matches of P against these elements.

For example, let us consider the complex pattern

((on ?block1 ?block2) (on ?block2 ?block3))

and the working memory

((on B C) (on C D) (on A B))

In this situation, there are two distinct matches:

((on A B) (on B C)) ⇒ *((?block1 . A) (?block2 . B) (?block3 . C))*
((on B C) (on C D)) ⇒ *((?block1 . B) (?block2 . C) (?block3 . D))*

Other bindings are possible, but they do not involve consistent matches against elements in working memory.

Complex Patterns with Negations

We can also specify *negated* patterns which designate literals that should *not* be matched.

For example, let us consider the complex pattern

((on ?x ?y) (not (on ?any ?x))

and the working memory

((on B C) (on C D) (on A B))

This pattern produces only a single match

((on A B)) ⇒ *((?block1 . A) (?block2 . B))*

since *(on ?any ?x)* would match *(on A B)* if *?x* were bound to *B* and it would match *(on B C)* if *?x* were bound to *C*.

Relational Rules and Dynamic Composition

Symbolic patterns are often used to support *rule-based reasoning*.

For example, given the relational rule

If ((on ?x ?y) (on ?y z)) then (above ?x ?z)

and the working memory *((on B C) (on C D))* , one can infer the literal *(above B D)* by substituting bindings from the condition.

Moreover, relational rules can be *composed dynamically* to produce chains of reasoning. E.g., given the rule above and

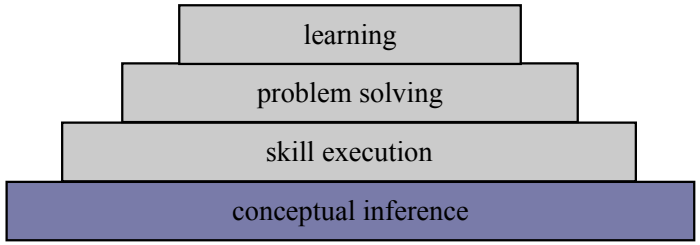
If ((above ?x ?y) (on ?y ?z)) then (above ?x ?z))

and the working memory *((on A B) (on B C) (on C D))* , one can infer the literals *(above A C), (above B D), and (above A D).*

This generative capacity gives rule-based systems great power.

Cascaded Integration in ICARUS

Like other unified cognitive architectures, ICARUS incorporates a number of distinct modules.



ICARUS adopts a *cascaded* approach to integration in which lower-level modules produce results for higher-level ones.

Theory of Conceptual Inference

Concepts are distinct cognitive entities that humans use to describe their environment:

- Most categories are *grounded* in perception, in that they refer to the physical characteristics of objects or events.
- Many concepts are *relational* in that they describe connections or interactions among objects or events.
- Concepts are organized in a *hierarchy*, with more complex categories defined in terms of simpler structures.
- Everyday conceptual inference is an automatic process that proceeds in a *bottom-up* manner.

ICARUS incorporates and instantiates these assumptions about conceptual structures and processing.

Concepts and Beliefs

ICARUS relies on three types of symbol structures to support its interpretation of the environment.

- *Concepts* are symbolic predicates that the agent can use for this cognitive purpose.
- Conceptual *rules / clauses* provide definitions for these symbols that the agent uses for recognition and inference.
- *Beliefs* are instances of concepts that, taken together, describe the agent's situation.

Together, these three types of structures support *conceptual inference,* which lets ICARUS characterize its situation.

ICARUS Concepts for Urban Driving

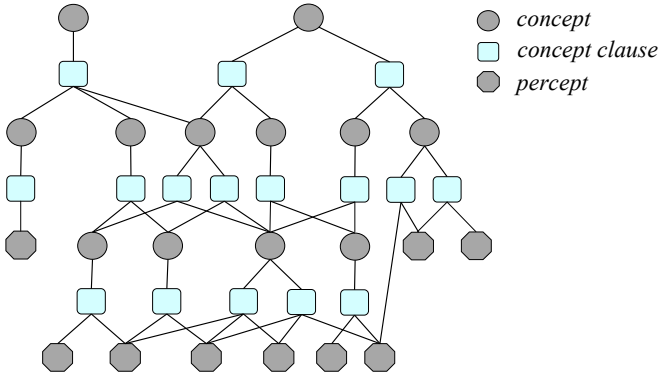
```
((driving-well-in-rightmost-lane ?self ?line1 ?line2)
 :percepts    ((self ?self) (segment ?seg) (line ?line1 segment ?seg))
              (line ?line2 segment ?seg))
 :relations   ((driving-well-in-segment ?self ?seg ?line1 ?line2)
              (not (lane-to-right ?line1 ?line2 ?anyline))))

((driving-well-in-segment ?self ?seg ?line1 ?line2)
 :percepts    ((self ?self) (segment ?seg) (line ?line1 segment ?seg))
              (line ?line2 segment ?seg))
 :relations   ((in-segment ?self ?seg)
              (aligned-and-centered-in-lane ?self ?line1 ?line2)
              (steering-wheel-straight ?self)))

((aligned-and-centered-in-lane ?self ?line1 ?line2)
 :percepts    ((self ?self segment ?seg)
              (line ?lane1 segment ?seg dist ?dist1 angle ?ang1)
              (line ?lane2 segment ?seg dist ?dist2 angle ?ang2))
 :tests       ((*nearly-equal ?dist1 ?dist2) (*nearly-equal ?ang1 ?ang2)))
```

Hierarchical Organization of Concepts

ICARUS organizes conceptual memory in a hierarchical manner.



The same conceptual predicate can appear in multiple clauses to specify disjunctive and recursive concepts.

ICARUS Beliefs / Percepts for Urban Driving

Inferred beliefs:

(current-street me A)
(lane-to-right g599 g601)
(last-lane g599)
(under-speed-limit me)
(steering-wheel-not-straight me)
(in-lane me g599)
(on-right-side-in-segment me)
(building-on-left g288)
(building-on-left g427)
(building-on-left g431)
(building-on-right g287)
(increasing-direction me)
(current-segment me g550)
(first-lane g599)
(last-lane g601)
(slow-for-right-turn me)
(centered-in-lane me g550 g599)
(in-segment me g550)
(intersection-behind g550 g522)
(building-on-left g425)
(building-on-left g429)
(building-on-left g433)
(building-on-right g279)
(near-pedestrian me g567)

Observed percepts:

(lane-line line7 segment segment23 color white dist 2.3 angle 49 …)
(building building15 address 756 dist-to-corner 21.2 angle-to-corner 49 …)
(self segment segment23 speed 33 wheel-angle 10 throttle 80 …))

Conceptual Inference in ICARUS

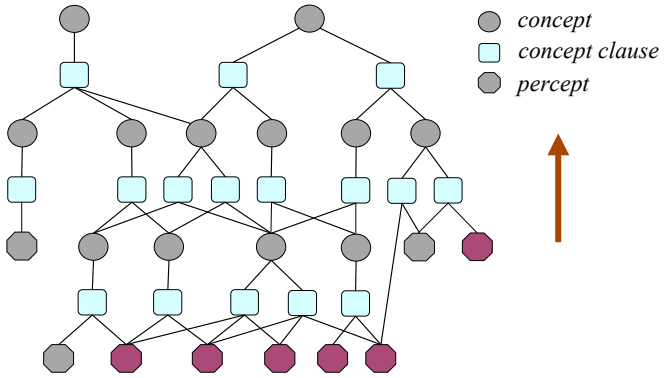
On each cognitive cycle, the architecture's conceptual inference module:

- Collects all the percepts arriving from the environment;
- Finds all consistent matches of primitive conceptual clauses against these percepts;
- Infers beliefs that correspond to these rules' instantiated heads and adds them to belief memory;
- Repeats these matching and generation steps at higher levels of the conceptual hierarchy.

In this way, ICARUS computes the full deductive closure of its conceptual knowledge and percepts.

Conceptual Inference in ICARUS

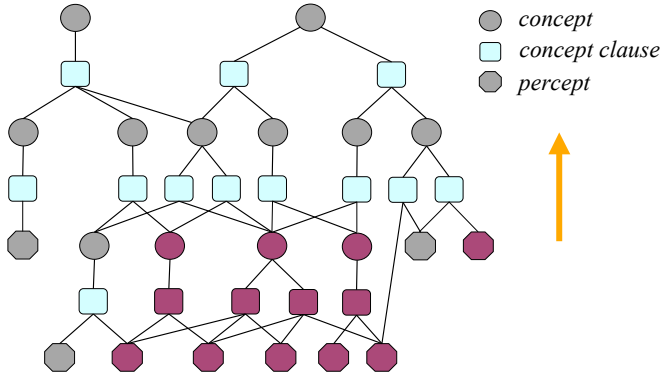
Conceptual inference in ICARUS occurs from the bottom up.



Starting with observed percepts, this process produces high-level beliefs about the current state.

Conceptual Inference in ICARUS

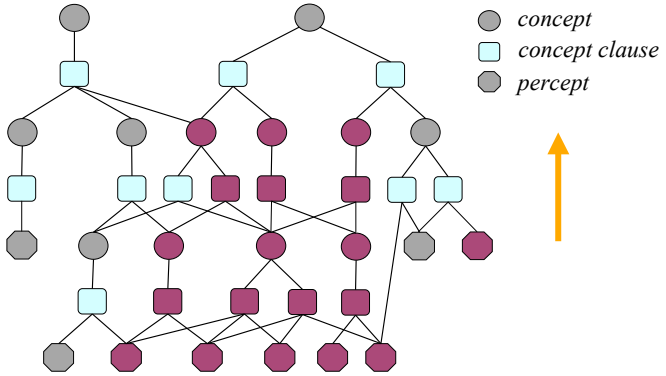
Conceptual inference in ICARUS occurs from the bottom up.



Starting with observed percepts, this process produces high-level beliefs about the current state.

Conceptual Inference in ICARUS

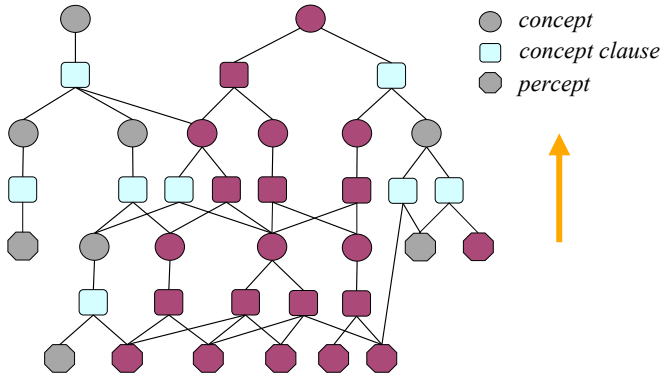
Conceptual inference in ICARUS occurs from the bottom up.



Starting with observed percepts, this process produces high-level beliefs about the current state.

Conceptual Inference in ICARUS

Conceptual inference in ICARUS occurs from the bottom up.



Starting with observed percepts, this process produces high-level beliefs about the current state.

Conceptual Inference – Limitations

ICARUS' current mechanisms for conceptual inference exhibit a number of drawbacks, in that they:

- Support only deductive forms of reasoning that rely on all-or-none matching;
- Operate in an exhaustive manner that draws all possible deductive inferences; and
- Do not retain beliefs across cognitive cycles, requiring the system to generate them anew each time.

These characteristics differ markedly from the flexible and adpative forms of inference found in humans.

Conceptual Inference – Current Research

We are developing a new and improved ICARUS inference module (Bridewell & Langley, 2011) that:

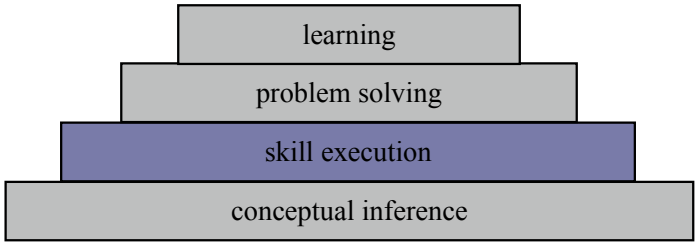
- Supports *abductive* reasoning that attempts to explain observations and creates default assumptions;
- Operates in an *anytime* manner that produces useful inferences even when interrupted; and
- Updates beliefs *incrementally* on every cycle, revising them when necessary.

The new process alternates between selecting a belief off which to chain and a rule through which to chain.

These choices are guided by heuristics based on coherence, recency, and other metrics.

Cascaded Integration in ICARUS

Like other unified cognitive architectures, ICARUS incorporates a number of distinct modules.



ICARUS adopts a *cascaded* approach to integration in which lower-level modules produce results for higher-level ones.

Theory of Skill Execution

Skills are distinct cognitive structures that describe how one interacts with the environment:

- Most human skills are *grounded* in perception (indirectly through concepts) and in action.
- Skills are *relational* in that they describe changes in conceptual structures as a result of their execution.
- Memory for skills is organized as a *hierarchy*, with more complex activities decomposed into simpler ones.
- Skills are *indexed* by the goals they achieve on their successful execution in the environment.
- Execution is *teleoreactive*, i.e., guided by the agent's goals but sensitive to environmental factors.

ICARUS incorporates and instantiates these assumptions about skill representation and processing.

Skills and Intentions

ICARUS relies on three types of symbol structures to support its interaction with the environment.

- *Skills* are symbolic predicates that the agent can use for this cognitive purpose.
- Skill *clauses* provide definitions for these symbols that the agent can use for retrieval and control.
- *Intentions* are instances of skills that, taken together, describe the agent's planned environmental activity.

Together, these three types of structures support *skill execution*, which lets ICARUS carry out extended activities.

ICARUS Skills for Urban Driving

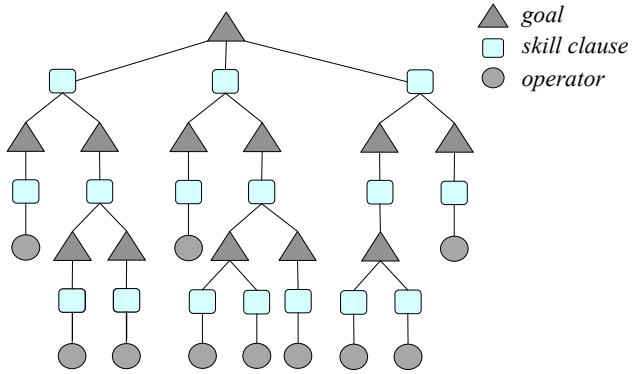
```
((driving-well-in-rightmost-lane ?self ?line1 ?line2)
 :percepts    ((self ?self) (segment ?seg) (line ?line1 segment ?seg))
              (line ?line2 segment ?seg))
 :start       ((not (lane-to-right ?line1 ?line2 ?anyline))
 :subgoals    ((driving-well-in-segment ?self ?seg ?line1 ?line2)))

((driving-well-in-segment ?self ?seg ?line1 ?line2)
 :percepts    ((self ?self) (segment ?seg) (line ?line1 segment ?seg))
              (line ?line2 segment ?seg))
 :start       ((steering-wheel-straight ?self))
 :subgoals    ((in-segment ?self ?seg)
              (aligned-and-centered-in-lane ?self ?line1 ?line2)
              (steering-wheel-straight ?self)))

((aligned-and-centered-in-lane ?self ?line1 ?line2)
 :percepts    ((self ?self))
 :start       ((misaligned-to-left-in-lane ?self ?line1 ?line2))
 :requires    ((not (steering-to-right ?self)))
 :actions     ((*steer 20)))
```

Hierarchical Organization of Skills

ICARUS organizes skills in a hierarchical manner, which each skill clause indexed by the goal it aims to achieve.



The same goal can index multiple clauses to allow disjunctive, conditional, and recursive procedures.

Skill Execution in ICARUS

When given a skill instance S, the ICARUS execution module:

- Retrieves all skill clauses that have the same predicate in their head;
- Determines whether the conditions of each clause match against the current beliefs;
- If no skill instances are applicable (do not match), then it abandons S;
- Else it selects one of the matched skill clauses at random and generates a corresponding intention I.

If S was invoked by a higher-level intention I_P, then ICARUS stores I_P as the parent of the new intention I.

Skill Execution in ICARUS

On each cognitive cycle, the ICARUS execution module works with the current intention I:

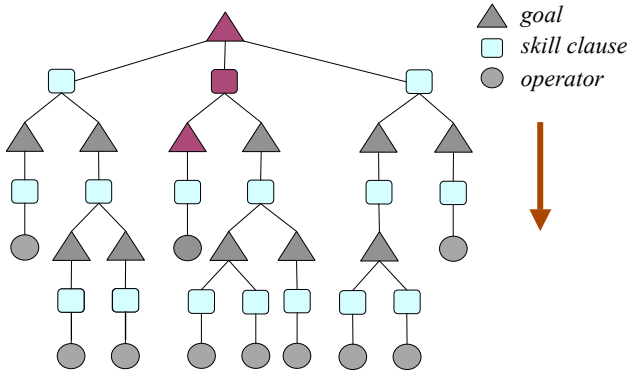
- If the effects of I match against beliefs, then it pops I and making its parent intention I_P current;
- Else if I involves a primitive skill, then it carries out the action associated with I in the environment;
- Else if I involves a nonprimitive skill, then it calls on the next one of I's ordered subskills.

This process continues on each cycle, with ICARUS moving down, right, or up through the skill hierarchy.

However, if no subskill of the current intention is applicable, all its ancestor intentions fail, including the top level.

Skill Execution in ICARUS

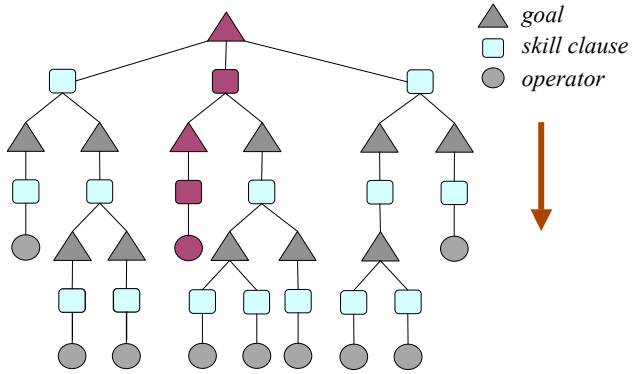
Execution operates from the top down, starting with high-level intentions, to find paths through the skill hierarchy.



A skill clause is applicable if its conditions hold and its effects are unsatisfied, given bindings from above.

Skill Execution in ICARUS

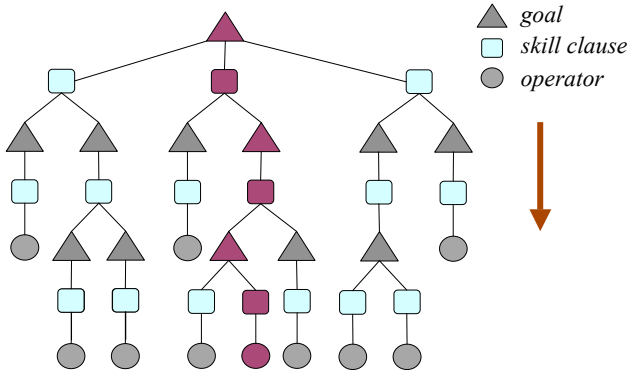
Execution operates from the top down, starting with high-level intentions, to find paths through the skill hierarchy.



A skill clause is applicable if its conditions hold and its effects are unsatisfied, given bindings from above.

Skill Execution in ICARUS

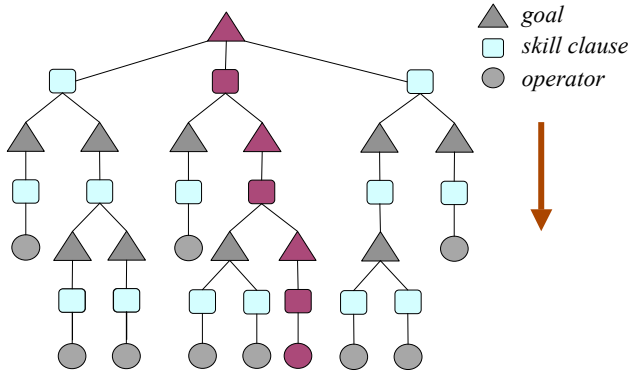
This process repeats on later cycles to produce hierarchical but reactive behavior (Nilsson, 1994).



However, ICARUS prefers to continue ongoing skills when they match, giving it a bias toward persistence over reactivity.

Skill Execution in ICARUS

If events proceed as expected, this iterative process eventually completes the agent's top-level intention.



At this point, a different intention begins to drive agent behavior, invoking different skills to pursue it.

Skill Execution – Limitations

ICARUS' processes for skill execution are limited in a number of ways, in that they:

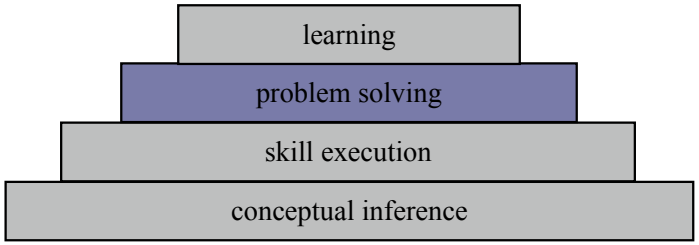
- Operate in an entirely reactive, closed-loop manner that checks conditions on each cycle;
- Incorporate no information about the expected duration of the skill being executed; and
- Support the use of only one skill at a time, rather than allowing concurrent execution.

We are developing extended versions of ICARUS that address each of these issues.

Choi (2010) reports a variant that executes skills concurrently subject to resource availability.

Cascaded Integration in ICARUS

Like other unified cognitive architectures, ICARUS incorporates a number of distinct modules.



ICARUS adopts a *cascaded* approach to integration in which lower-level modules produce results for higher-level ones.

Theory of Problem Solving

Problem solving lets humans achieve goals even on complex, unfamiliar tasks:

1. Human problem solving involves *heuristic search* through a *problem space*.
2. This search process uses *operators* (skills) to transform *states* (sets of beliefs) into ones that satisfy *goals* (desired beliefs).
3. Humans often use a mix of goal-directed backward chaining and state-driven forward chaining called *means-ends analysis*.
4. Problem solving remains *grounded* in perception and actions, yet often occurs at an *abstract* level of description.
5. Human problem solving typically interleaves mental processing and physical execution.

ICARUS adopts and utilizes these tenets about the components and operation of problem solving.

Problems, Goals, and Stacks

ICARUS relies on three types of symbol structures to respond to unfamiliar tasks that it encounters.

- *Problems* are symbolic structures that specify desired situations in the environment.
- Each problem comprises a set of *goals* that describe desired or undesired beliefs.
- *Problem stacks* are chains of problems and intentions that enable progress toward problem solution.

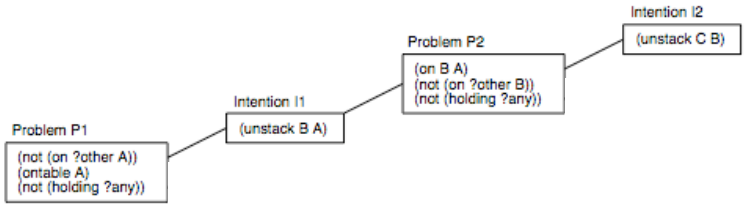
Together, these three types of structures let ICARUS engage in *problem solving* to generate novel behavior.

Problems, Goals, and Stacks

```
(problem :id P1
         :goals        ((not (on ?other A)) (ontable A) (not (holding ?any)))
         :bindings     ( )
         :differences ((not (on B A))))
(intention :id I1 :parent P1
         :head         (unstack B A)
         :bindings     ((?from . A) (?on . B))
         :conditions  ((on B A) (not (on ?other B)) (not (holding ?any)))
         :action       (*grasp-and-lift B)
         :effects      ((not (on B A)) (holding B))
(problem :id P2 :parent I1
         :goals        ((on B A) (not (on ?other B)) (not (holding ?any)))
         :bindings     ((?from . A) (?on . B))
         :differences ((not (on C B))))
(intention :id I2 :parent P1
         :head         (unstack C B)
         :bindings     ((?from . B) (?on . C))
         :conditions  ((on C B) (not (on ?other C)) (not (holding ?any)))
         :action       (*grasp-and-lift C)
         :effects      ((not (on C B)) (holding C))
```

Problems, Goals, and Stacks

Here is a graphical depiction of the list structures contained in the problem stack just shown.



This represents one possible mental state that ICARUS might generate during problem solving in the Blocks World.

Problem Solving in ICARUS

On each cognitive cycle, the ICARUS problem-solving module operates on the current problem P:

- If all of P's goals match consistently against beliefs, then it pops P and makes its parent intention I_P current;
- Else it finds all skill instances with matched conditions or with effects that match at least one unsatisfied goal;
- The module selects one of these skill instances and creates an intention I with P as its parent;
- If the conditions of intention I match beliefs, then it calls the execution module to carry it out;
- Else it creates a new problem P_N using the instantiated conditions of I.

This recursive process can produce an alternating stack of problems and intentions.

Problem Solving and Execution

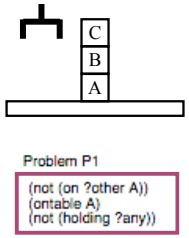
Once the ICARUS problem-solving module reaches a belief state that satisfies the conditions of intention I, it:

- Invokes the execution module to carry out I in the external environment;
- Continues execution until this achieves I's expected effects, then returns attention to I's parent problem;
- Unless execution fails, in which case the module returns to P and attempts to find another solution.

This interleaving of problem solving with execution can lead to difficulties, but this also occurs in human behavior.

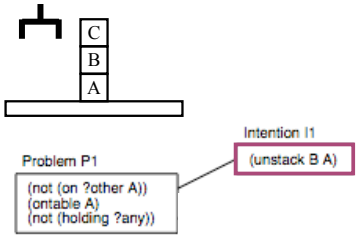
Problem Solving in ICARUS

The architecture invokes problem solving when it encounters an unsolved problem (set of goals).



Problem Solving in ICARUS

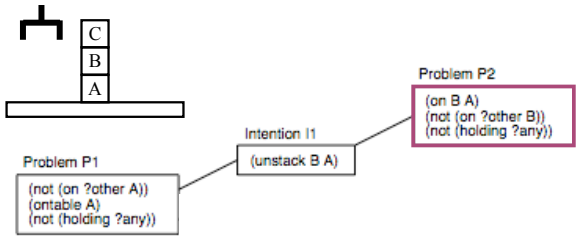
The architecture retrieves relevant skill instances and uses its heuristics to select one of them.



ICARUS creates an intention based on this skill instance with the original problem as its parent.

Problem Solving in ICARUS

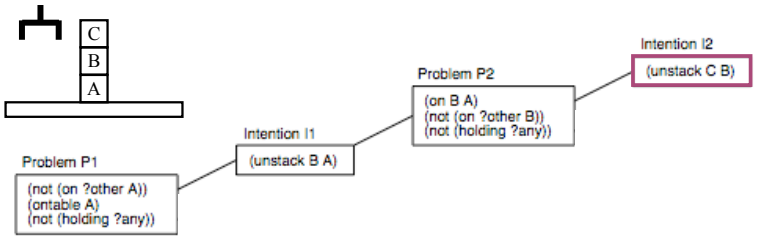
In some cases, the generated intention is not applicable because its conditions do not match current beliefs.



When such an impasse occurs, ICARUS chains backward off the intention's instantiated conditions to create a subproblem.

Problem Solving in ICARUS

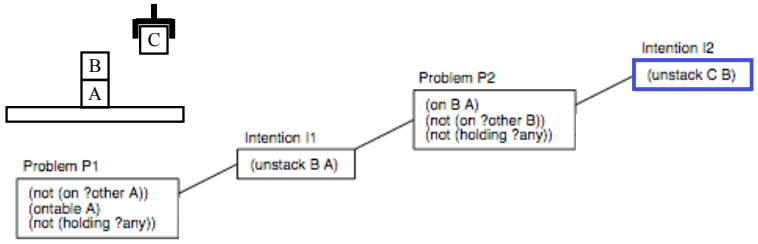
The architecture applies the same machinery recursively to the subproblem in an attempt to solve it.



This leads it to select a skill instance and generate an intention that should lead toward the subproblem's solution.

Problem Solving in ICARUS

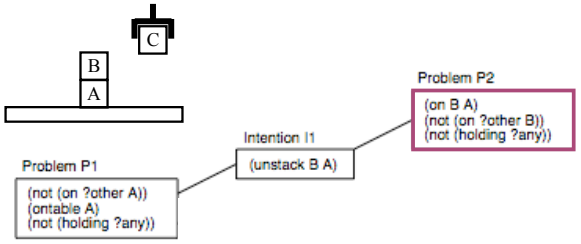
Once it has generated an applicable intention, ICARUS executes it in the environment, changing the situation.



The new situation in turn produces revised beliefs that should satisfy at least some of the problem's goals.

Problem Solving in ICARUS

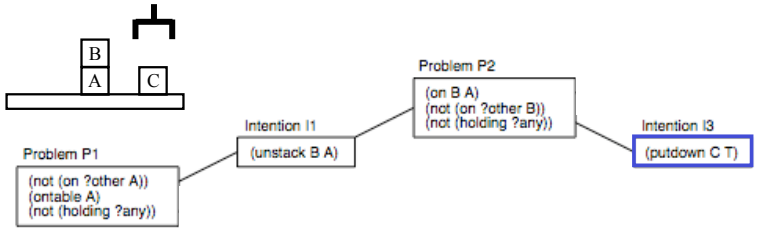
But the resulting beliefs may not yet satisfy all of the problem's goals, in which case it continues the process.



In such cases, ICARUS reactivates the parent problem and looks for a skill that would achieve additional goals.

Problem Solving in ICARUS

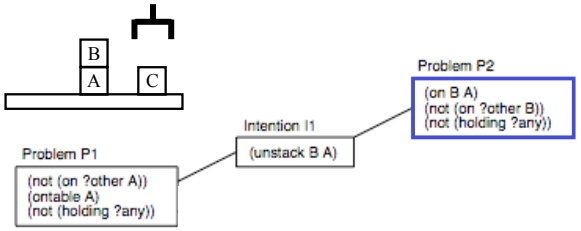
This leads the architecture to generate another intention, check its application conditions, and so on.



When this effort produces another applicable intention, ICARUS executes it, changing the situation further.

Problem Solving in ICARUS

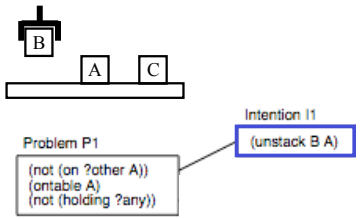
With luck, this process eventually leads to a situation and to beliefs that satisfy all of the problem's goals.



When this occurs, ICARUS realizes that it has solved the problem and pops it from the problem stack.

Problem Solving in ICARUS

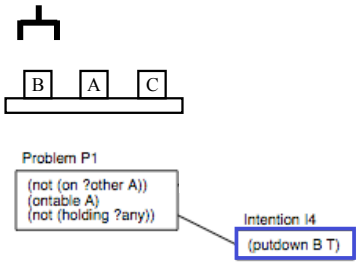
Because the subproblem was linked to the conditions of the initial intention, this means the latter is applicable.



In response, ICARUS calls on the execution module to carry out this intention in the environment.

Problem Solving in ICARUS

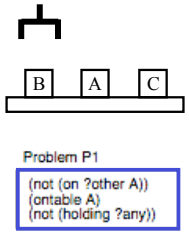
But, again, this intention may not by itself solve the original problem, which involves multiple goals.



Accordingly, ICARUS may generate another intention, which may lead to new subproblems or may be applicable.

Problem Solving in ICARUS

With luck, ICARUS will finally achieve all goals for the initial, problem, at which point it turns to other tasks.



This variant of means-ends analysis can make incorrect guesses, in which case the module backtracks and tries other choices.

Heuristics for Problem Solving

The ICARUS problem-solving module uses two heuristics to guide selection of candidate intentions:

- Prefer skill instances whose effects should achieve more of the currently unsatisfied goals.
  - Such skills can take multiple steps toward a solution.
- Prefer skill instances which have fewer conditions that are currently unsatisfied.
  - Such skills can require less effort to make applicable.

The current architecture uses only one heuristic at a time, but we plan to combine them into a single metric.

This suggests ways to unify backward-chaining and forward-chaining approaches to problem solving.

Problem Solving – Limitations

ICARUS' module for problem solving also has a number of limitations, in that they:

- Engage in *depth-first search*, rather than less systematic methods like progressive deepening (de Groot, 1946);
- Emphasize *backward chaining* from goals, rather than forward chaining from the current state; and
- Do not address the generation of new problems or their understanding;
- Cannot solve insight problems that appear to involve some type of reformulation.

In recent work, we have started to address each drawback (Choi, 2010; MacLellan, 2011; Shapiro, 2011).

Theory of Skill Acquisition

Skill learning lets humans store the results of experience to improve their future performance:

1. Skill acquisition involves monotonic addition to memory of new *symbolic structures*.
2. Learning is driven by *experience* but draws on prior *knowledge*.
3. Skill learning operates in an *incremental* and *cumulative* manner.
4. The learning process is *interleaved with problem solving*, which provides it with experience.
5. Skills are *generalized traces* of successful means-ends analysis.

Previous versions of ICARUS embodied these five claims about the character of human skill acquisition.

ICARUS Summary

ICARUS is a unified theory of the human cognitive architecture that supports:

- conceptual inference over grounded relational categories
- goal-directed but reactive execution of hierarchical skills
- means-ends problem solving when routine execution fails
- acquisition of new skills from traces of problem solving

We have used the ICARUS formalism to develop models for traditional tasks like puzzles and multi-column subtraction.

We have also used the framework to create synthetic agents for a variety of simulated physical environments.

Creating Synthetic Agents with ICARUS

Urban Combat



Urban Driving



MadRTS

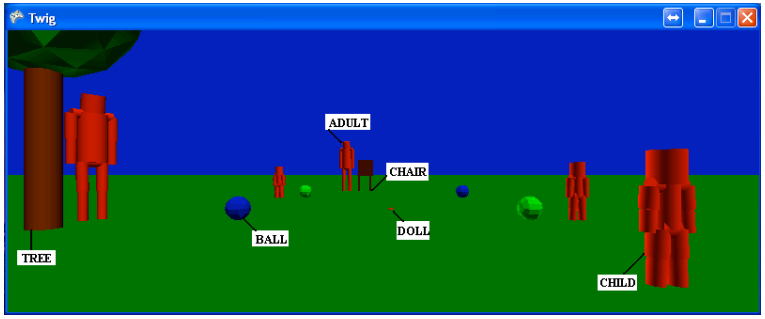


Rush 2008



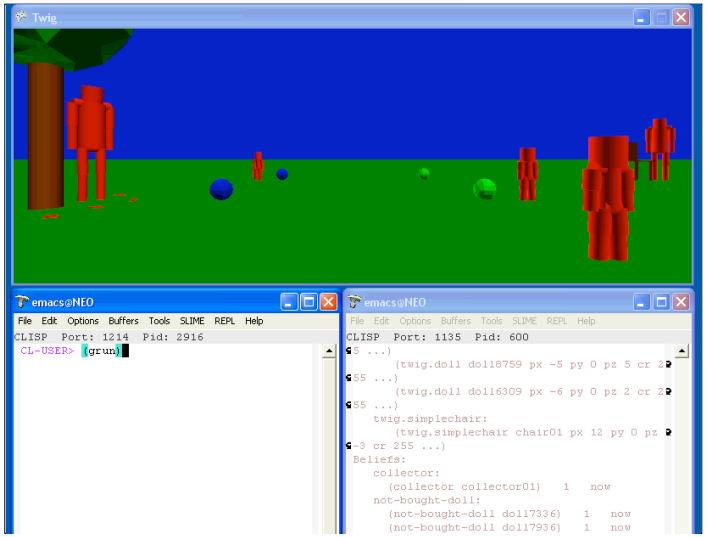
Synthetic Agents for Twig Scenarios

Most recently, we have used Horswill's (2008) Twig simulator to develop humanoid ICARUS agents.



This low-fidelity environment supports a few object types, along with simple reactive behaviors for virtual characters.

A Capitalist Twig Scenario



Open Issues – Skill Learning

Earlier versions of ICARUS acquired new skills whenever the system solved novel problems.

But this process relied on a well-structured concept hierarchy and it learned only from success.

We are extending the most recent version of the architecture to:

- Learn a new skill whenever the system makes monotonic progress toward solving a problem;
- Learn a new constraint over skills whenever heuristic search leads to loops or dead ends.

Both mechanisms will operate incrementally and be driven by single experiences to support human rates of learning.

Open Issues – Social Cognition

Humans frequently engage in social cognition, which requires encoding and processing content about others' mental states.

We are extending ICARUS in this direction so that it supports:

- Modal statements about beliefs, goals, intentions of other agents [e.g., (belief me (goal agent2 (holding agent2 doll5)))];
- Flexible inference that supports abductive explanation of others' behaviors (e.g., default assumptions about goals);
- Execution and problem solving that achieves goals for changing others' mental states (e.g., by communication);
- Learning new skills that let one be more effective at interacting with other actors.

Taken together, these should let ICARUS support intelligent agents that interact in more human-like ways.

Open Issues – Language Processing

Humans also spend substantial time interacting via language.

Because language is a social phenomenon, our extensions for social cognition should let us:

- Represent the beliefs, goals, and intentions that underlie other agents' utterances;
- Infer this content flexibly and incrementally from very partial observations and background knowledge;
- Execute hierarchical skills which achieve communicative goals that impinge on other objectives.

We are exploring these ideas in the context of task-oriented dialogues that arise in medical settings.

Relation to Other Architectures

ICARUS has much in common with architectures like Soar and ACT-R but also has key differences; e.g., our framework:

- Assumes a recognize-act cycle but is not a production-system architecture;
- Uses modular representations but has an architecture-level commitment to hierarchy;
- Relies centrally on symbolic processing but insists that symbols be grounded in perception and action;
- Adopts rule-like structures but encodes them at coarser levels of granularity; and
- Incorporates many ideas about logic-based representation and mental processing.

Together, these features give ICARUS a distinctive character that we hope many modelers will find appealing.

Concluding Remarks

In this tutorial, we covered ICARUS, a theory of the cognitive architecture that:

- Grounds high-level cognition in perception and action
- Treats categories and skills as separate cognitive entities
- Organizes skills and concepts in a hierarchical manner
- Combines teleoreactive control with means-ends analysis
- Acquires new skills from successful problem solving

ICARUS combines ideas from many different traditions in a unified account of high-level cognition.

The framework also offers a high-level programming language for developing cognitive models and intelligent agents.

Selected References

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

Choi, D., Konik, T., Nejati, N., Park, C., & Langley, P. (2007). A believable agent for first-person shooter games. *Proceedings of the Third Annual Artificial Intelligence and Interactive Digital Entertainment Conference*. Stanford, CA: AAAI Press.

Langley, P. (2006). Cognitive architectures and general intelligent systems. *AI Magazine*, 27, 33-44.

Langley, P., & Choi, D. (2006). A unified cognitive architecture for physical agents. *Proceedings of the Twenty-First National Conference on Artificial Intelligence*. Boston: AAAI Press.

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

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

For downloadable papers, documentation, and software, see *http://www.isle.org/icarus/*