A Value-Driven Architecture for Intelligent Behavior

Pat Langley
Dan Shapiro
Computational Learning Laboratory
Center for the Study of Language and Information
Stanford University, Stanford, California
http://cll.stanford.edu/

This research was supported in part by Grant NCC-2-1220 from NASA Ames Research Center. Thanks to Meg Aycinena, Michael Siliski, Stephanie Sage, and David Nicholas.

Assumptions about Cognitive Architectures

1. We should move beyond isolated phenomena and capabilities to develop complete intelligent agents.
2. Artificial intelligence and cognitive psychology are close allies with distinct but related goals.
3. A cognitive architecture specifies the infrastructure that holds constant over domains, as opposed to knowledge, which varies.
4. We should model behavior at the level of functional structures and processes, not the knowledge or implementation levels.
5. A cognitive architecture should commit to representations and organizations of knowledge and processes that operate on them.
6. An architecture should come with a programming language for encoding knowledge and constructing intelligent systems.
7. An architecture should demonstrate generality and flexibility rather than success on a single application domain.

Examples of Cognitive Architectures

Some of the cognitive architectures produced over 30 years include:

- ACTE through ACT-R (Anderson, 1976; Anderson, 1993)
- Soar (Laird, Rosenbloom, & Newell, 1984; Newell, 1990)
- Prodigy (Minton & Carbonell., 1986; Veloso et al., 1995)
- PRS (Georgeff & Lansky, 1987)
- 3T (Gat, 1991; Bonasso et al., 1997)
- EPIC (Kieras & Meyer, 1997)
- APEX (Freed et al., 1998)

However, these systems cover only a small region of the space of possible architectures.

Goals of the ICARUS Project

We are developing the ICARUS architecture to support effective construction of intelligent autonomous agents that:

- integrate perception and action with cognition
- combine symbolic structures with affective values
- unify reactive behavior with deliberative problem solving
- learn from experience but benefit from domain knowledge

In this talk, we report on our recent progress toward these goals.

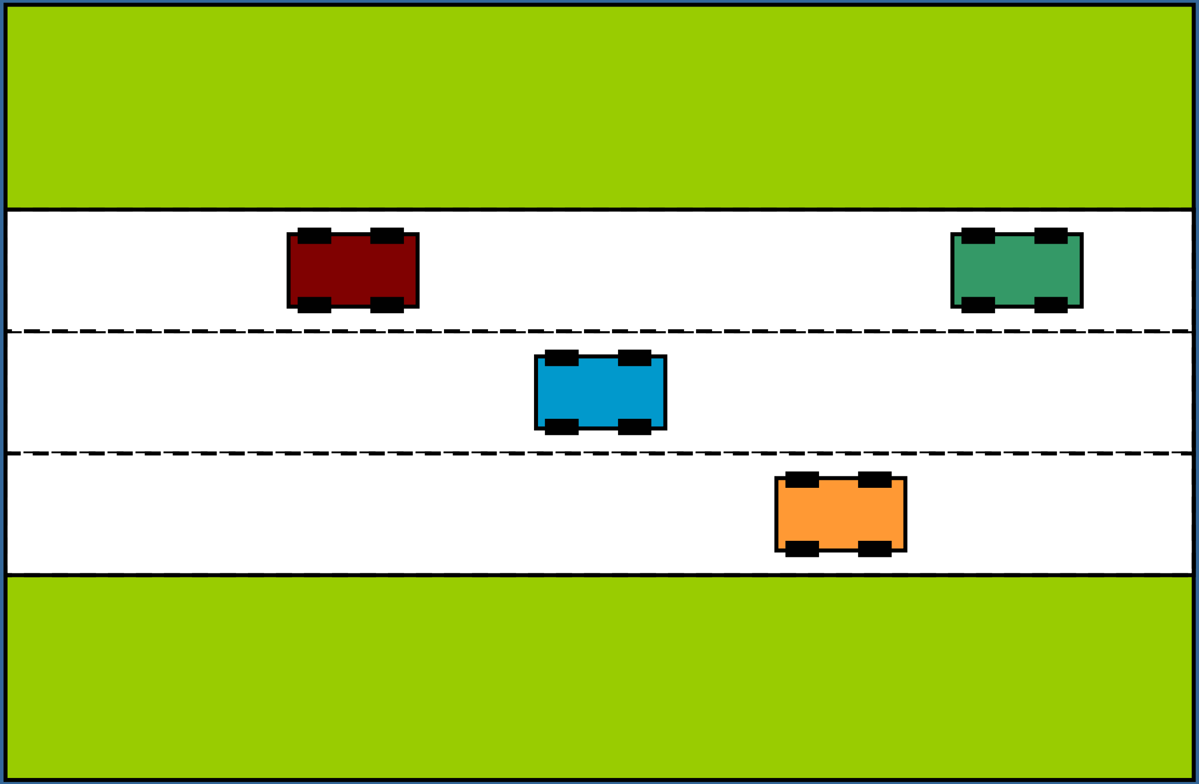
Design Principles for ICARUS

Our designs for ICARUS have been guided by five principles:

1. Affective values pervade intelligent behavior;
2. Categorization has primacy over execution;
3. Execution has primacy over problem solving;
4. Tasks and intentions have internal origins; and
5. The agent's reward is determined internally.

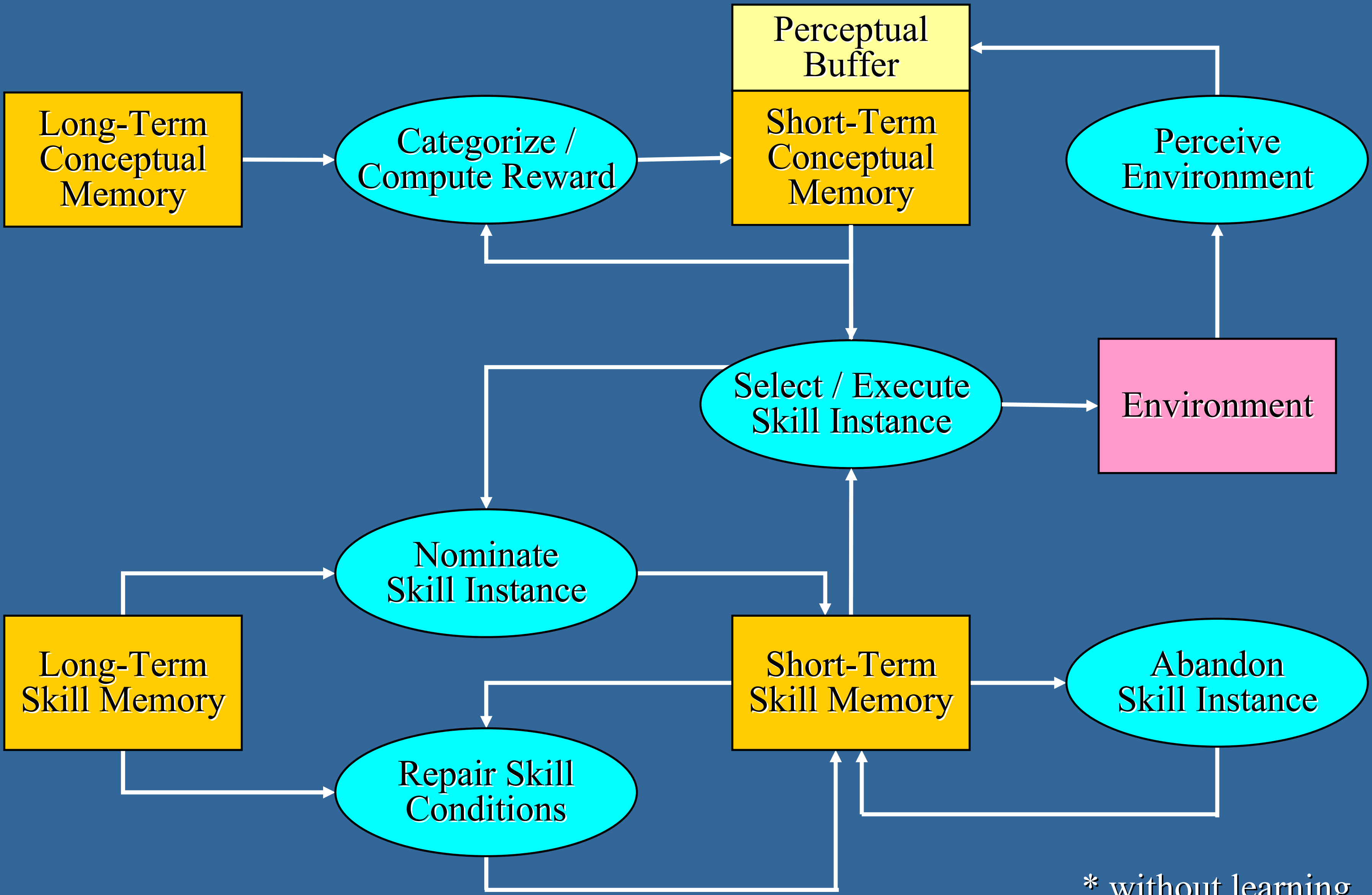
These ideas distinguish ICARUS from most agent architectures.

A Cognitive Task for Physical Agents



(car self)	(#back self 225)	(#front self 235)	(#speed self 60)
(car brown-1)	(#back brown-1 208)	(#front brown-1 218)	(#speed brown-1 65)
(car green-2)	(#back green-2 251)	(#front green-2 261)	(#speed green-2 72)
(car orange-3)	(#back orange-3 239)	(#front orange-3 249)	(#speed orange-3 56)
(in-lane self B)	(in-lane brown-1 A)	(in-lane green-2 A)	(in-lane orange-3 C)

Overview of the ICARUS Architecture*



* without learning

Some Motivational Terminology

ICARUS relies on three quantitative measures related to motivation:

- *Reward* – the affective value produced on the current cycle.
- *Past reward* – the discounted sum of previous agent reward.
- *Expected reward* – the predicted discounted future reward.

These let an ICARUS agent make decisions that take into account its past, present, and future affective responses.

Long-Term Conceptual Memory

ICARUS includes a long-term conceptual memory that contains:

- *Boolean* concepts that are either True or False;
- *numeric* concepts that have quantitative measures.

These concepts may be either:

- *primitive* (corresponding to the results of sensory actions);
- *defined* as a conjunction of other concepts and predicates.

Each Boolean concept includes an associated reward function.

* *Icarus' concept memory is distinct from, and more basic than, skill memory, and provides the ultimate source of motivation.*

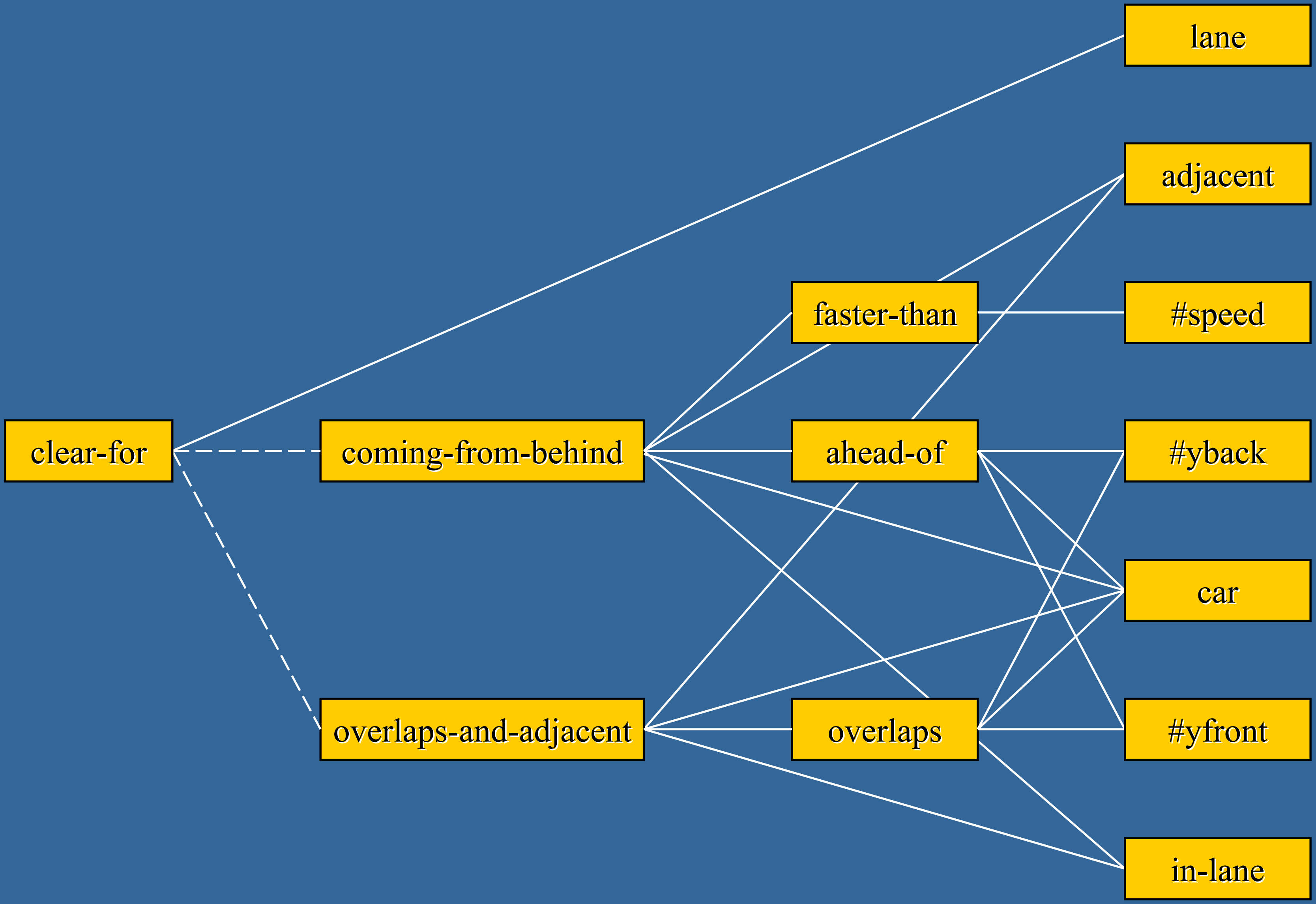
Examples of Long-Term Concepts

```
(ahead-of  (?car1 ?car2)
 :defn     (car ?car1) (car ?car2)
           (#back ?car1 ?back1)
           (#front ?car2 ?front2)
           (> ?back1 ?front2)
 :reward   (#dist-ahead ?car1 ?car2 ?d)
           (#speed ?car2 ?s)
 :weights  (5.6  3.1) )
```

```
(coming-from-behind (?car1 ?car2)
 :defn     (car ?car1) (car ?car2)
           (in-lane ?car1 ?lane1)
           (in-lane ?car2 ?lane2)
           (adjacent ?lane1 ?lane2)
           (faster-than ?car1 ?car2)
           (ahead-of ?car2 ?car1) )
 :reward   (#dist-behind ?car1 ?car2 ?d)
           (#speed ?car2 ?s)
 :weights  (4.8  –2.7) )
```

```
(clear-for  (?lane ?car)
 :defn      (lane ?lane ?left-line ?right-lane)
            (not (overlaps-and-adjacent ?car ?other))
            (not (coming-from-behind ?car ?other))
            (not (coming-from-behind ?other ?car))
 :constant  10.0 )
```

A Sample Conceptual Hierarchy



Long-Term Skill Memory

ICARUS includes a long-term skill memory in which skills contain:

- an *:objective* field that encodes the skill's desired situation;
- a *:start* field that must hold for the skill to be initiated;
- a *:requires* field that must hold throughout the skill's execution;
- an *:ordered* or *:unordered* field referring to subskills or actions;
- a *:values* field with numeric concepts to predict expected value;
- a *:weights* field indicating the weight on each numeric concept.

These fields refer to terms stored in conceptual long-term memory.

* *Icarus' skill memory encodes knowledge about how and why to act in the world, not about how to solve problems.*

Examples of Long-Term Skills

```
(pass  (?car1 ?car2 ?lane)
 :start       (ahead-of ?car2 ?car1)
              (in-same-lane ?car1 ?car2)
 :objective   (ahead-of ?car1 ?car2)
              (in-same-lane ?car1 ?car2)
 :requires    (in-lane ?car2 ?lane)
              (adjacent ?lane ?to)
 :ordered     (speed&change ?car1 ?car2 ?lane ?to)
              (overtake ?car1 ?car2 ?lane)
              (change-lanes ?car1 ?to ?lane))
 :values      (#distance-ahead ?car1 ?car2 ?d)
              (#speed ?car2 ?s)
 :weights     (0.26  0.17) )
```

```
(change-lanes  (?car ?from ?to)
 :start       (in-lane ?car ?from)
 :objective   (in-lane ?car ?to)
 :requires    (lane ?from ?shared ?right)
              (lane ?to ?left ?shared)
              (clear-for ?to ?car)
 :ordered     (*shift-left)
 :constant    0.0 )
```

ICARUS' Short-Term Memories

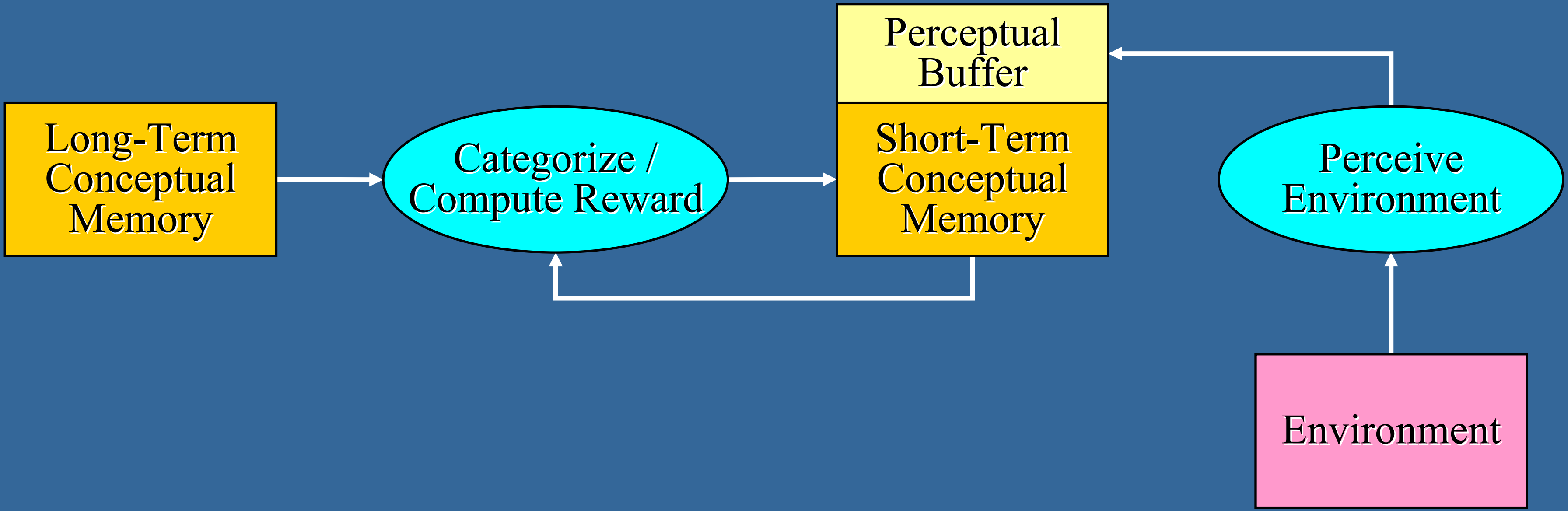
Besides long-term memories, ICARUS stores dynamic structures in:

- a perceptual buffer with primitive Boolean and numeric concepts
  - (car car-06), (in-lane car-06 lane-a), (#speed car-06 37)
- a short-term conceptual memory with matched concept instances
  - (ahead-of car-06 self), (faster-than car-06 self), (clear-for lane-a self)
- a short-term skill memory with instances of skills that the agent intends to execute
  - (speed-up-faster-than self car-06), (change-lanes lane-a lane-b)

These encode temporary beliefs, intended actions, and their values.

* *Icarus' short-term memories store specific, value-laden instances of long-term concepts and skills.*

Categorization and Reward in ICARUS


Perceptual Buffer
Long-Term Conceptual Memory
Categorize / Compute Reward
Short-Term Conceptual Memory
Perceive Environment
Environment

Categorization occurs in an automatic, bottom-up manner.

A reward is calculated for every matched Boolean concept.

This reward is a linear function of associated numeric concepts.

Total reward is the sum of rewards for all *matched* concepts.

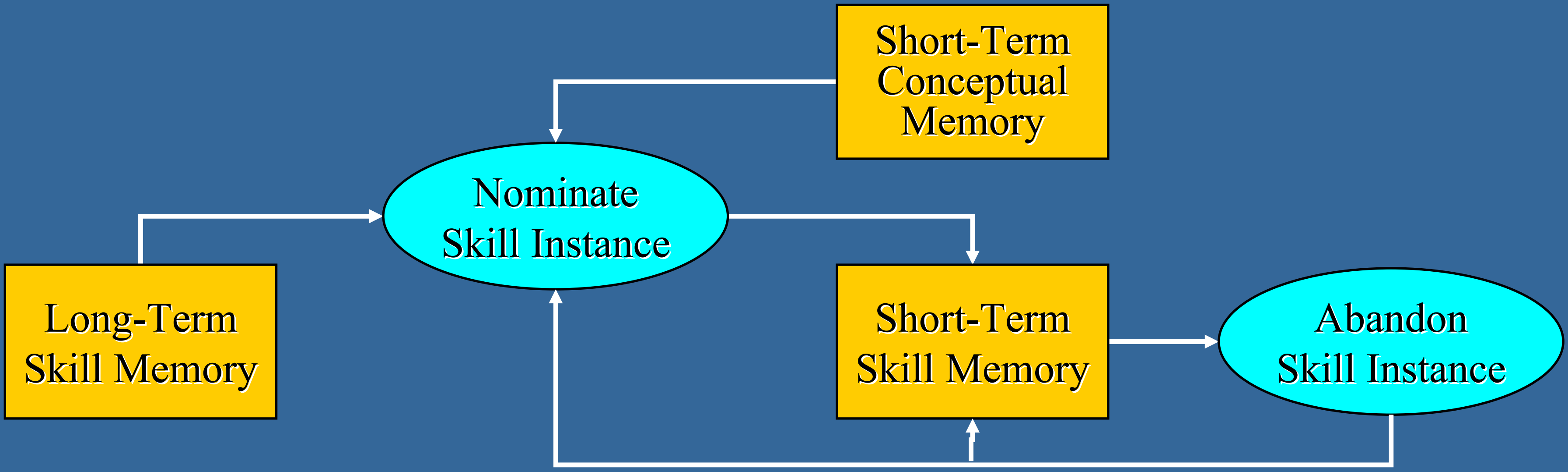
* *Categorization and reward calculation are inextricably linked.*

Skill Nomination and Abandonment

ICARUS adds skill instances to short-term skill memory that:

- refer to concept instances in short-term conceptual memory;
- have *expected reward* > agent's *discounted past reward* .

ICARUS removes a skill when its expected reward << past reward.



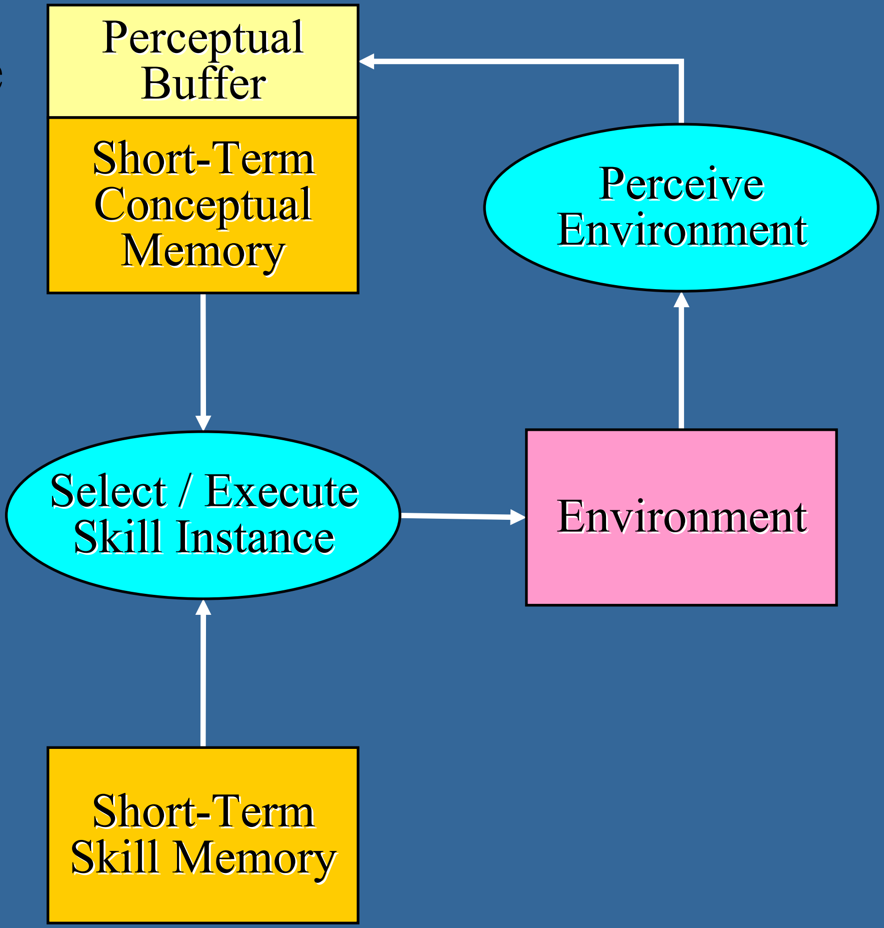
* *Nomination and abandonment create highly autonomous behavior that is motivated by the agent's internal reward.*

Skill Selection and Execution

On each cycle, ICARUS executes the skill with highest expected reward.

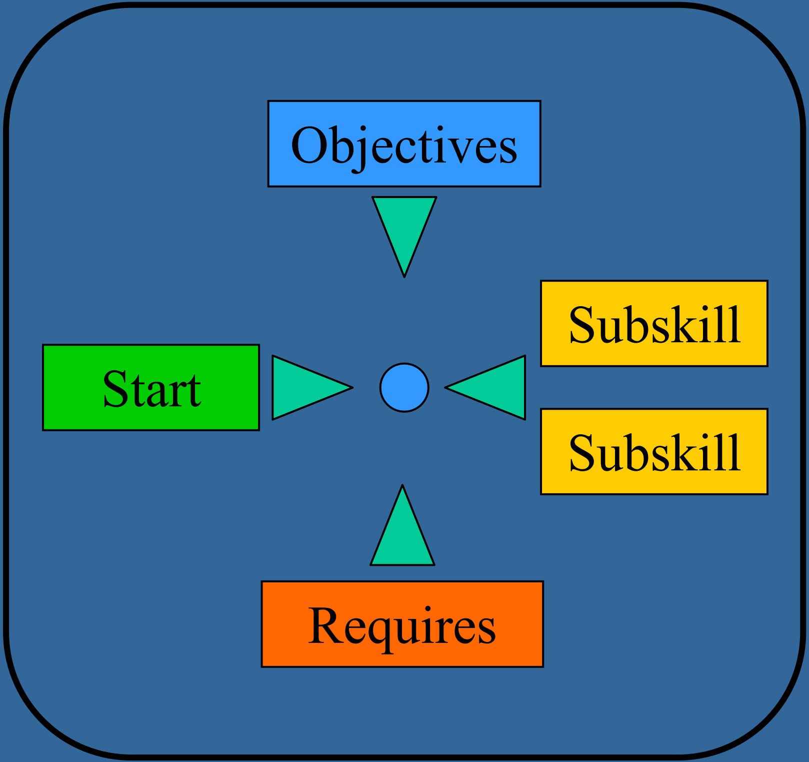
Selection invokes deep evaluation to find the action with the highest expected reward.

Execution causes action, including sensing, which alters memory.



* *ICARUS makes value-based choices among skills, and among the alternative subskills and actions in each skill.*

ICARUS' Interpreter for Skill Execution



```
(speed&change (?car1 ?car2 ?from ?to)
   :start          (ahead-of ?car1 ?car2)
                   (same-lane ?car1 ?car2)
   :objective      (faster-than ?car1 ?car2)
                   (different-lane ?car1 ?car2)
   :requires       (in-lane ?car2 ?from)
                   (adjacent ?from ?to)
   :unordered      (*accelerate)
                   (change-lanes ?car1 ?from ?to) )
```

Given Start:

If not (Objectives) and Requires, then

- choose among unordered Subskills
- consider ordered Subskills

* *ICARUS skills have hierarchical structure, and the interpreter uses a reactive control loop to identify the most valuable action.*

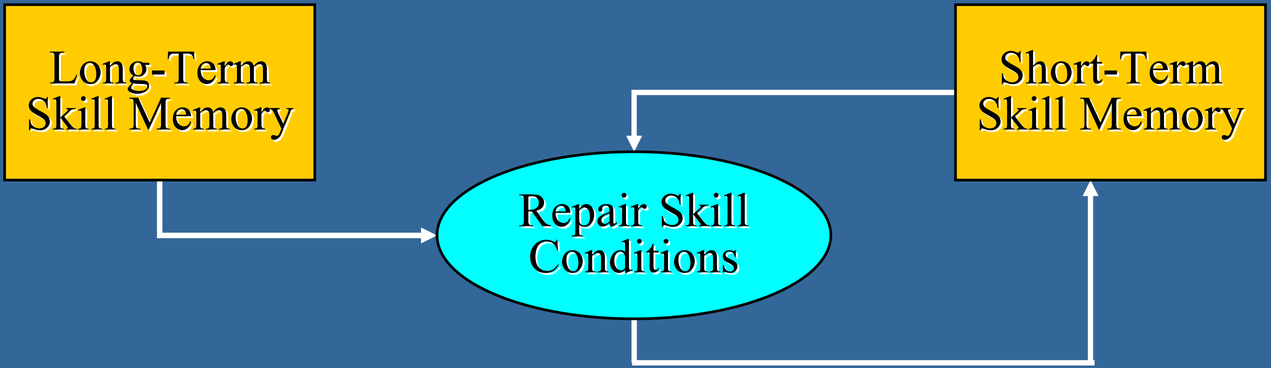
Cognitive Repair of Skill Conditions

ICARUS seeks to repair skills whose requirements do not hold by:

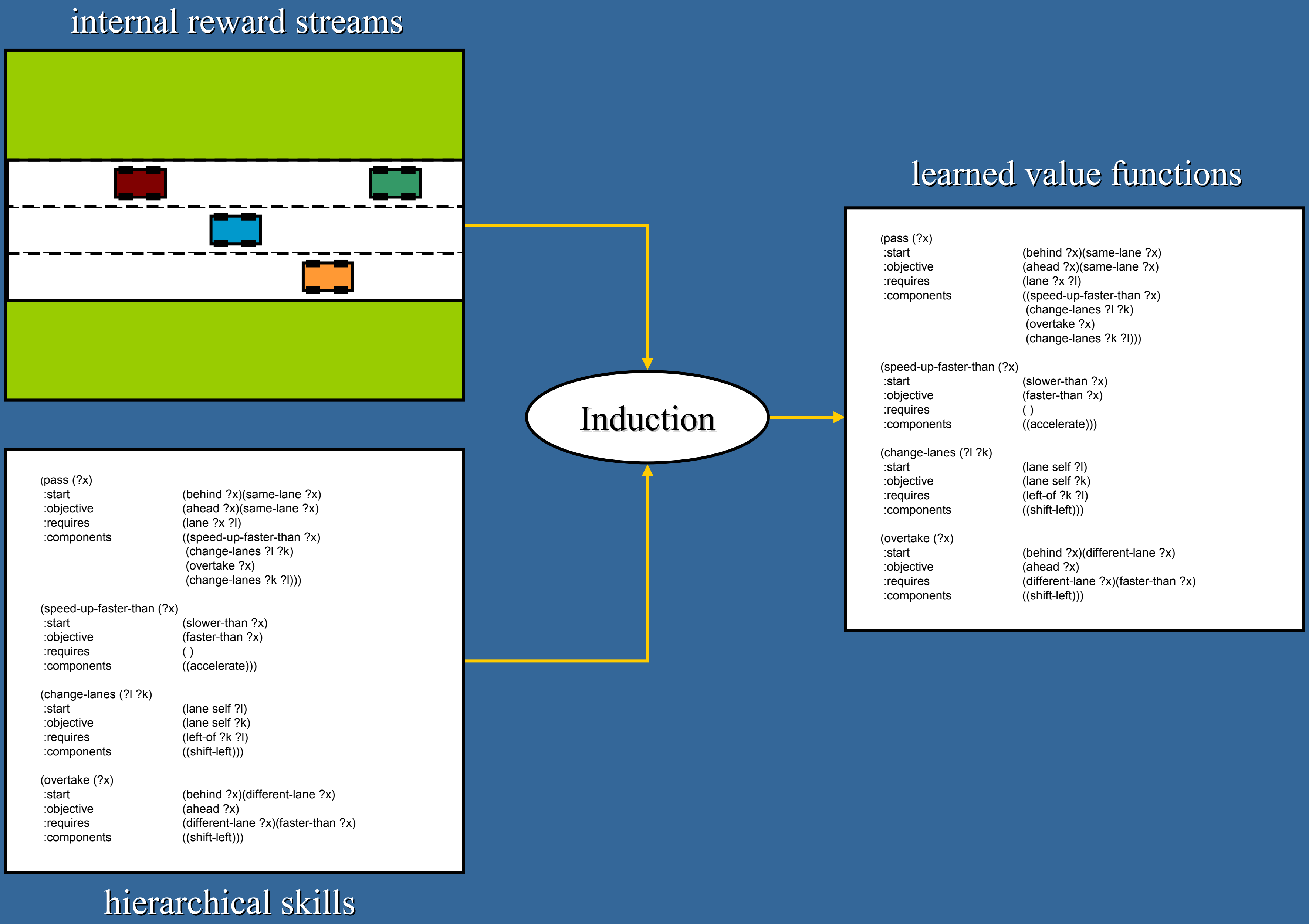
- finding concepts that, if true, would let execution continue;
- selecting the concept that is most important to repair; and
- nominating a skill with objectives that include the concept.

Repair takes one cycle and adds at most one skill instance to memory.

* *This backward chaining is similar to means-ends analysis, but it supports execution rather than planning.*



Learning Hierarchical Control Policies

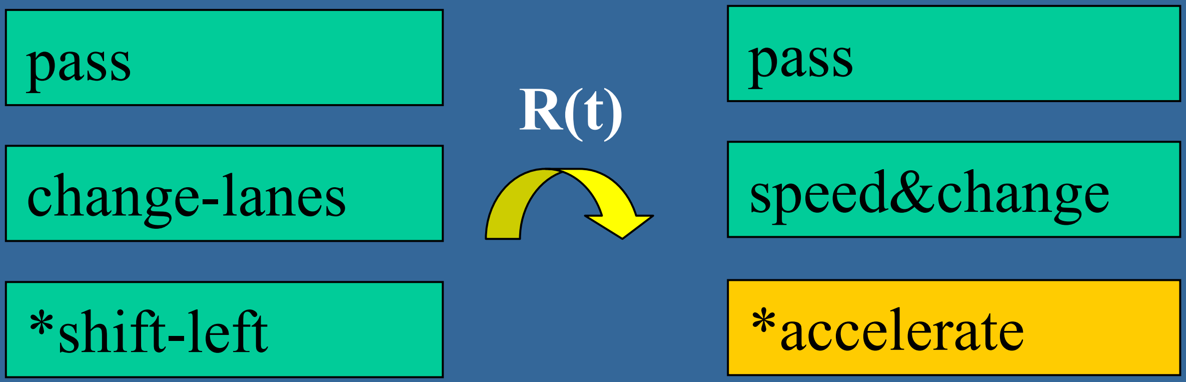


```
(pass (?x)
 :start                    (behind ?x)(same-lane ?x)
 :objective                (ahead ?x)(same-lane ?x)
 :requires                 (lane ?x ?l)
 :components               ((speed-up-faster-than ?x)
                            (change-lanes ?l ?k)
                            (overtake ?x)
                            (change-lanes ?k ?l)))

(speed-up-faster-than (?x)
 :start                    (slower-than ?x)
 :objective                (faster-than ?x)
 :requires                 ( )
 :components               ((accelerate)))

(change-lanes (?l ?k)
 :start                    (lane self ?l)
 :objective                (lane self ?k)
 :requires                 (left-of ?k ?l)
 :components               ((shift-left)))

(overtake (?x)
 :start                    (behind ?x)(different-lane ?x)
 :objective                (ahead ?x)
 :requires                 (different-lane ?x)(faster-than ?x)
 :components               ((shift-left)))
```

```
(pass (?x)
 :start                    (behind ?x)(same-lane ?x)
 :objective                (ahead ?x)(same-lane ?x)
 :requires                 (lane ?x ?l)
 :components               ((speed-up-faster-than ?x)
                            (change-lanes ?l ?k)
                            (overtake ?x)
                            (change-lanes ?k ?l)))

(speed-up-faster-than (?x)
 :start                    (slower-than ?x)
 :objective                (faster-than ?x)
 :requires                 ( )
 :components               ((accelerate)))

(change-lanes (?l ?k)
 :start                    (lane self ?l)
 :objective                (lane self ?k)
 :requires                 (left-of ?k ?l)
 :components               ((shift-left)))

(overtake (?x)
 :start                    (behind ?x)(different-lane ?x)
 :objective                (ahead ?x)
 :requires                 (different-lane ?x)(faster-than ?x)
 :components               ((shift-left)))
```

Revising Expected Reward Functions

ICARUS uses a hierarchical variant of Q learning to revise estimated reward functions based on internally computed rewards:



Update $Q(S) = \theta \bullet \phi$ with $R(t)$, $Q(s')$

** This method learns 100 times faster than nonhierarchical ones.*

Intellectual Precursors

Our work on ICARUS has been influenced by many previous efforts:

- earlier research on integrated cognitive architectures
  - especially influenced by ACT, Soar, and Prodigy
- earlier work on architectures for reactive control
  - especially universal plans and teleoreactive programs
- research on learning value functions from delayed reward
  - especially hierarchical approaches to Q learning
- decision theory and decision analysis
- previous versions of ICARUS (going back to 1988).

However, ICARUS combines and extends ideas from its various predecessors in novel ways.

Directions for Future Research

Future work on ICARUS should introduce additional methods for:

- forward chaining and mental simulation of skills;
- allocation of scarce resources and selective attention;
- probabilistic encoding and matching of Boolean concepts;
- flexible recognition of skills executed by other agents;
- caching of repairs to extend the skill hierarchy;
- revision of internal reward functions for concepts; and
- extension of short-term memory to store episodic traces.

Taken together, these features should make ICARUS a more general and powerful architecture for constructing intelligent agents.

Concluding Remarks

ICARUS is a novel integrated architecture for intelligent agents that:

- includes separate memories for concepts and skills;
- organizes concepts and skills in a hierarchical manner;
- associates affective values with all cognitive structures;
- calculates these affective values internally;
- combines reactive execution with cognitive repair; and
- uses expected values to nominate tasks and abandon them.

This constellation of concerns distinguishes ICARUS from other research on integrated architectures.