Available online at www.sciencedirect.com
ScienceDirect
Cognitive Systems Research xxx (2011) xxx–xxx
Cognitive Systems RESEARCH
www.elsevier.com/locate/cogsys

Reactive goal management in a cognitive architecture

Action editor: Joachim Funke

Dongkyu Choi *

Department of Aeronautics & Astronautics, Stanford University, Stanford, CA 94305, USA

Received 26 April 2010; accepted 16 September 2010

Abstract

Goals play an important role in human cognition. Different aspects of human mind influence the generation of goals they pursue, and the goals guide their behaviors. In psychology, researchers made significant efforts to study goals and their origin, and cognitive architectures include various facilities to handle goals of artificial agents. One such architecture, ICARUS, supports goal-driven behaviors while maintaining reactivity, and the top-level goals play the role of guiding ICARUS agents' behaviors. However, the architecture covers neither the origin of its top-level goals nor the management of them, and this imposes various restrictions on ICARUS, like the limited autonomy. In this paper, we extend the architecture to provide the capability to nominate top-level goals using the notion of long-term, general goals, and manage the nominated goals by prioritizing them. For prioritization of goals, we introduce a novel capability to match concepts in a continuous manner. We show some illustrative examples in an urban driving domain, and discuss related and future work in this direction before we conclude.

Keywords: Goal nomination; Goal prioritization; Reactive goal management; Cognitive architecture

1. Introduction and motivation

Goals are important in human life. People have ideas on what they want to do or what they should do, and these give rise to goals. The environment influences this process heavily, by affecting both the selection and the prioritization of goals. Selected and prioritized goals then guide people's behaviors by restricting the space of possible actions to take.

Traditionally, this fundamental role of goals caught a lot of attention among psychologists, and we can find numerous accounts in the literature (Simon, 1967; Sloman, 1987; Gray & Braver, 2002). As frameworks for computational models of cognition, most cognitive architectures (Newell, 1990), too, provide facilities for goals. At the very least, these architectures allow the specification of goals or subgoals that guide the artificial agent's behaviors in either explicit or implicit fashion. One such architecture, ICARUS (Langley & Choi, 2006), operates in a goal-oriented fashion, in which it uses multiple, reactive top-level goals.

But some architectures provide more than others, including nomination and retraction of goals. For instance, CLARION (Sun, 2007) has drive and goal mechanisms that correspond to a psychological account of goal nomination. Soar (Laird, Rosenbloom, & Newell, 1986) can nominate its top-level operators as action-like goals or intentions. The original ICARUS architecture can select an operator for execution at random as part of its exploration strategy, but it lacks any mechanism to add, delete, or reorder its top-level goals, limiting its capabilities significantly. In this paper, we present an extension to the architecture with a new goal management mechanism that works reactively to the environment. We extended the explicit architectural distinction that exists between long-term knowledge and short-term structures to cover goals as well, by introducing the notion of long-term, general descriptions of goals. The

* Present address: Department of Psychology, University of Illinois at Chicago, 1007 W. Harrison St. (M/C 285), Chicago, IL 60607, USA.
E-mail address: dongkyuc@uic.edu.

1389-0417/$ - see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.cogsys.2010.09.002

system instantiates these long-term goals based on the current situation of the world to get short-term, specific goals to guide its own behavior. The extended ICARUS has the ability to nominate, retract, and prioritize its top-level goals, allowing more suitable behaviors in reaction to its surroundings.

In the subsequent sections, we briefly review the ICARUS architecture and introduce an urban driving domain that we use as a testbed. Then we explain the extension for reactive goal management in detail and provide some illustrative examples in the urban driving domain. We conclude after discussions on related and future work.

2. Review of the ICARUS architecture

ICARUS (Langley & Choi, 2006) shares its basic features with other cognitive architectures like Soar (Laird et al., 1986) and ACT-R (Anderson, 1993). It makes commitments to its representation of knowledge, memory structures, and mechanisms for inference, execution, and learning. The system provides a computational framework for intelligent agents, which stays constant across different domains. In this section, we review the basic capabilities of the architecture before we continue our discussion on the extensions for reactive goal management. We start with ICARUS's representation of knowledge and memories that support this, and then cover the architecture's inference and execution processes.

2.1. Representation and memories

The architecture distinguishes conceptual and procedural knowledge. ICARUS's *concepts* describe various aspects of the environment, whereas its *skills* define procedures that are known to achieve corresponding concepts when executed to completion. ICARUS also distinguishes long-term knowledge and short-term structures. Long-term knowledge includes general descriptions of the environment and procedures. Short-term structures are instantiations derived from ICARUS's long-term knowledge, which are relevant to the current situation.

The distinctions along these two directions result in four main memories in ICARUS. Its long-term conceptual memory stores general definitions of concepts that involve variables for object symbols and their attributes to describe various situations. A long-term skill memory houses variablized skills that define general procedures to achieve certain concepts, namely, the *goals*. There also are short-term memories that correspond to these long-term memories. A short-term conceptual memory stores instantiated concepts, which the system believes to be true in the current situation. A short-term skill memory holds instantiated skills, along with their corresponding goals. For this reason, we often call the short-term memories as the *belief memory* and the *goal memory*, respectively.

Table 1 shows the syntax for concepts in ICARUS. Each concept includes a head, and a body that takes one of the four forms shown. The first two concepts are *primitive*, and they include only perceptual matching conditions that ground on object information from the environment. On the other hand, the other two concepts are *non-primitive*, since they refer to other concepts in the `:relations` field. This hierarchical organization of concepts allows multiple levels of abstraction, and facilitates the description of complex situations in the world. Meanwhile, Table 2 provides the syntax for skills in ICARUS. Each skill has its goal as the head and includes a body in either of the forms shown. In a similar fashion to its conceptual counterparts, there are primitive skills and non-primitive ones. The first skill shown is primitive, and it consists of perceptual matching conditions, preconditions, and direct references to immediate actions in the world. The second skill, however, is non-primitive and it provides a way to decompose the task into subgoals instead of referencing directly to actions. In the next section, we cover ICARUS's processes that work over these knowledge structures.

Table 1
Syntax for ICARUS's concepts.

(⟨*head*⟩	
`:percepts`	(⟨*perceptual matching conditions*⟩))
(⟨*head*⟩	
`:percepts`	(⟨*perceptual matching conditions*⟩)
`:tests`	(⟨*tests against variables*⟩))
(⟨*head*⟩	
`:percepts`	(⟨*perceptual matching conditions*⟩)
`:relations`	(⟨*references to other concepts*⟩))
(⟨*head*⟩	
`:percepts`	(⟨*perceptual matching conditions*⟩)
`:tests`	(⟨*tests against variables*⟩)
`:relations`	(⟨*references to other concepts*⟩))

2.2. Inference and execution

The ICARUS architecture operates in distinct cycles. On each cycle, the system invokes a series of processes including the inference of current beliefs and the execution of skill paths relevant to the situation (see Fig. 1). At the beginning of each cycle, ICARUS receives the sensory input from the environment in its perceptual buffer. Based on this information, the system infers its beliefs, namely, all the concept instances that are true in the current state. It starts with the lowest-level structures (i.e., primitive concepts), and moves up the hierarchy to non-primitive concepts. ICARUS goes through this process every cycle, and therefore, any naive

Table 2
Syntax for ICARUS's skills.

(⟨*head*⟩	
`:percepts`	(⟨*perceptual matching conditions*⟩)
`:start`	(⟨*preconditions*⟩)
`:actions`	(⟨*direct actions*⟩))
(⟨*head*⟩	
`:percepts`	(⟨*perceptual matching conditions*⟩)
`:start`	(⟨*preconditions*⟩)
`:subgoals`	(⟨*a subgoal decomposition*⟩))

Please cite this article in press as: Choi, D. Reactive goal management in a cognitive architecture. *Cognitive Systems Research* (2011), doi:10.1016/j.cogsys.2010.09.002

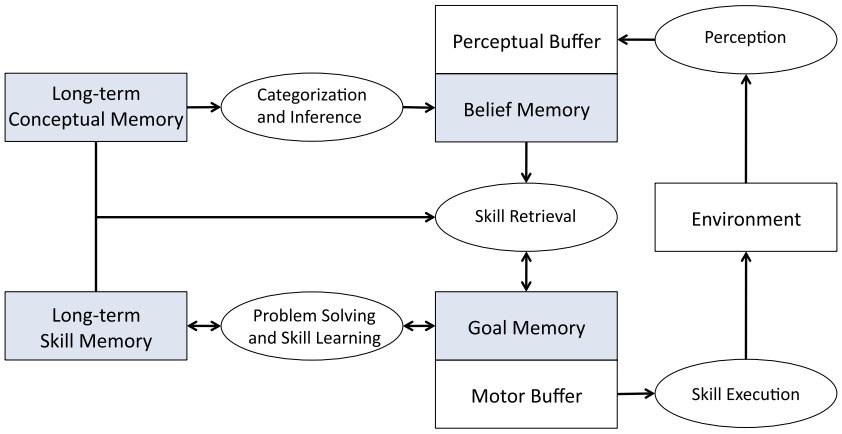

Fig. 1. ICARUS's memories and the processes that work over them. The shaded rectangles represent memories, while the oval shapes stand for the processes that use them as inputs and produce outputs. Note that the figure also shows ICARUS's problem solving and skill learning process that we do not use in this paper.

approach to the inference process is susceptible to the combinatorial effect the system encounters in domains with many objects. There have been several efforts to alleviate this problem including a prioritized inference mechanism by Asgharbeygi, Nejati, Langley, and Arai (2005).

When the system finishes inferring all its beliefs, it attempts to execute skills based on them. ICARUS retrieves skills that are relevant to its top-level goals, and finds one or more executable paths through its skill hierarchy.[1] A skill path is executable when all the skill instances on the path are executable, from top to bottom. Although a path can include a single primitive skill that achieves an ICARUS agent's top-level goal, most of the time a skill path starts with a non-primitive skill for a top-level goal and continues down several levels until it reaches a primitive skill at the bottom. The primitive skill provides actions the system should take in the environment. The architecture deposits these actions in its motor buffer and executes them to make changes in its surroundings. In turn, the perceptual input on the next cycle changes, and the system repeats all of the above processes based on the new sensory data.

But when the system encounters an impasse, in which it cannot find any executable skills for its goals, ICARUS invokes its means–ends problem solver. The architecture chains off of its goals until it finds a subgoal for which it has an applicable skill. The system also learns from problem solving traces it generates in this manner. Although a goal management mechanism will influence the problem solving and learning, the details of these mechanisms are not directly relevant here. In the following section, we introduce an urban driving domain that we use as a testbed throughout this paper before we continue our discussion on ICARUS in the context of the new extensions.

[1] By default, ICARUS stops searching through its skill hierarchy upon finding the first skill path that is executable. However, there are cases in which the system should find some or all such paths, and the architecture supports the retrieval of multiple skill paths and the coordinated execution of them. For more details, see Choi (2010).

3. Urban driving domain

Cognitive architectures are integrated models of general intelligence, and they often aim for understanding human-level intelligent behaviors. As such, they become a large collection of machinery that processes a variety of knowledge and other information. There are components that cover high-level aspects of intelligence, while others deal with lower-level cognition and control. Therefore, evaluating such architectures naturally requires some test domains that can afford various challenges at different levels of cognition. Domains like the Blocks World, Tower of Hanoi, or FreeCell Solitaire provide challenges suitable to evaluate symbolic operations, but they usually omit the aspects of physical control and assume perfect and instantaneous manipulation of objects. On the other hand, typical control domains like inverted pendulum, cruise control, or robot arms pose continuous control problems in a physical environment, but they lack any high-level aspects like the reason why such control is necessary. Therefore, we want a complex physical domain that supports both of these aspects, providing high-level challenges due to its complexity and creating opportunities for continuous, physical control.

Driving a car is one such task. It includes classic vehicle control problems like accelerating and decelerating comfortably, moving straight, and turning smoothly through the simultaneous control of multiple mechanisms in a vehicle. But it also involves the perception and recognition of both static and dynamic objects, and high-level cognition like decision making in reaction to the surroundings, navigation to a target location, and so on. Furthermore, compared to other settings like on highways, driving in an urban setting can be even more complicated, due to the various objects to interact with, including pedestrians, other cars, traffic signals and signs, and buildings with addresses. For this reason, we developed a simulation environment of urban driving using a three-dimensional game engine, in which we test ICARUS. Some basic features of this

Please cite this article in press as: Choi, D. Reactive goal management in a cognitive architecture. *Cognitive Systems Research* (2011), doi:10.1016/j.cogsys.2010.09.002

simulation are from an older development of the domain in two dimensions (Choi, Kaufman, Langley, Nejati, & Shapiro, 2004), but it includes a vast amount of improvements and additional capabilities. We cover this simulation environment in detail and describe basic behavior of ICARUS in this domain below.

3.1. Simulated environment

We developed our simulation of urban driving from ground up using a commercial game engine, TORQUE (http://www.garagegames.com/). Fig. 2 shows a screenshot of the simulated environment. In this world, we have a city downtown that consists of square blocks populated with numerous buildings and streets that run between these blocks. The streets in this city include various objects like street segments, intersections, lane lines, crosswalks, and sidewalks. Streets are either horizontal or vertical as in a planned city, and there are no deadends. We can have as many lanes as we wish on a particular street. At each intersection, we can choose to have traffic signals, stop signs, or none of these. We can place dynamic objects like vehicles and pedestrians on the streets. Each block has several buildings in them, and the buildings have unique addresses on each street.

Cars in this city are governed by realistic vehicle dynamics, which model them with a certain mass, center of mass, aerodynamic drag, engine torque, brake torque, tire friction, life time, and so on. They have three control variables: the percentage values of the gas and brake pedals, and the angle of the steering wheel. The simulation directly controls most of these cars to drive around the city. They accelerate up to a predefined speed, change lanes to left or right before making turns in the respective directions, and swerve around slower car in front of them. They randomly make turns at intersections, and stops to avoid collisions. However, depending on the probability of illegal moves defined in the simulation, they do violate these rules and get involved in accidents with each other or with other objects. When they are stuck somewhere in the city due to such accidents, the simulation spawns them afresh at a random location after a while.

Another type of dynamic objects, pedestrians, are also controlled by the simulation. They start off on sidewalks, move along the sidewalks, cross streets at designated crosswalks or near intersections. They do not react to cars around them, making them at the mercy of drivers. Just as drone cars, pedestrians, too, can violate rules, depending on the probability of jaywalk defined in the simulation. If this probability is set to a positive value, pedestrians randomly cross streets with the probability, increasing the chance for accidents. When accidents happen and they collide with cars, pedestrians die at the spot, but they get spawned at a random location after a while just as drone cars do.

What makes this simulated environment interesting to us is the fact that there are various interactions between the objects there. An agent driving a car in this city should watch for other cars, careless pedestrians, traffic signals, and stop signs. Sometimes the agent should also look at the addresses of buildings on the street. It needs to consider multiple of these factors at the same time while driving around the city. On top of that, the agent can have higher-level motivations or goals for driving. For example,

Fig. 2. A screenshot from the urban driving domain.

Please cite this article in press as: Choi, D. Reactive goal management in a cognitive architecture. *Cognitive Systems Research* (2011), doi:10.1016/j.cogsys.2010.09.002

it might want to drive as fast as it can to let out some stress. It might want to deliver packages to some addresses. Or, it might need to take a patient in a critical condition to a hospital. In this way, we can create numerous scenarios in the domain with relative ease, since driving is a very familiar task to us. At the same time, however, it still poses challenging problems, as it does in our everyday life.

3.2. Experimental support

Like other domains we use with ICARUS, there are two types of communication that should happen in the urban driving domain. One of these transfers perceptual information from the domain to the architecture, while the other passes control actions from ICARUS to the environment. Unlike some domains, especially the ones driven from existing games, the driving simulation is built from scratch with this in mind, and we have a complete control over what the agent can perceive and what it can do in this domain, facilitating development and evaluation of ICARUS agents.

When connected to this simulation, an ICARUS agent drives a particular car in the environment. The car has the same dynamic properties as all other cars in the city. From inside this car, the agent perceives various objects in an agent-centered polar coordinate system. Details of object attributes that ICARUS perceives vary between different object types. Table 3 shows what the ICARUS agent perceives of various objects in the simulation. For example, when the agent sees itself, it senses its name, speed, heading, steering wheel angle, amount of throttle that is open, maximum throttle, amount of brake applied, the street segment it is on, and the number of hits with pedestrians. On the other hand, when it perceives a lane line, the agent gets information on the name of the line, color, distance and angle from itself to the line, and the name of the segment the line belongs to.

In addition, the simulation offers some actions ICARUS can perform in the world. Each of these actions adjusts a set of control variables available in the simulation in a certain way. Table 4 shows the actions provided in the domain. The first three of them are crucial for ICARUS to drive, and they control the gas pedal, the brake pedal, and the steering wheel, respectively. The actions take a single argument each, the percentage of pedal application for the first two and the desired steering wheel angle for the last. The agent also has two additional actions provided for convenience, to coast the car (i.e., neither gas pedal nor brake pedal is applied) and to straighten the steering wheel, respectively. With these five actions, ICARUS has a complete control of its car.

The effects of these actions are simulated in a realistic way. The application of the gas and brake pedals and the rotation of the steering wheel are durative, and it may take several cycles to achieve the desired application or angle. For instance, if the current angle of the steering wheel is 0° and the desired angle is 10°, the agent can reach the desired angle in one cycle. But if the current angle of the steering wheel is −90° and the desired angle is +90°, then it might take more than one cycle to reach the desired angle depending on the maximum rate of change defined as a parameter in the simulation. In addition, the pedals are spring-loaded, and they return to zero after the application at some percentage. Therefore, the agent should continuously execute relevant pedal actions if it desires to maintain the amount of the pedal application constant.

Meanwhile, the domain features sophisticated parameter control over the simulation for evaluation purposes. Using a user-friendly interface, experimenters can specify the number of horizontal and vertical streets, the length of each block, the number of drones and pedestrians in the simulation, the probability of illegal moves for drone cars, the probability of jaywalks for pedestrians, the speed of simulation, ICARUS's visibility distance, number of lanes on each street, and the type of traffic signals at intersections. They can also choose whether they want predefined placement of cars and pedestrians from a file, or they want the simulation to place them at random. This feature is useful when we want to arrange a particular situation from the start, so that we can evaluate a certain behavior of ICARUS.

Table 3
Various attributes of objects ICARUS perceives in the urban driving domain.

objects	percepts
itself	name, speed, heading, steering wheel angle, throttle, maximum throttle, brake, current street segment, number of pedestrian hits
drone cars	name, distance, angle, heading, speed, value
pedestrians	name, distance, angle, heading, speed, alive
street segments	name, street name, distance, angle
intersections	name, street name, cross street name, distance
lane lines	name, color, distance, angle, segment name
buildings	name, address, street name, distance to the closest corner, angle to the closest corner, distance to the second closest corner, angle to the second closest corner

Table 4
Five actions ICARUS can perform in the urban driving domain.

actions	control variables		
	gas pedal	brake pedal	steering wheel
(*gas percent)	percent	0	N/A
(*brake percent)	0	percent	N/A
(*steer angle)	N/A	N/A	angle
(*cruise)	0	0	N/A
(*straighten)	N/A	N/A	0

Please cite this article in press as: Choi, D. Reactive goal management in a cognitive architecture. *Cognitive Systems Research* (2011), doi:10.1016/j.cogsys.2010.09.002

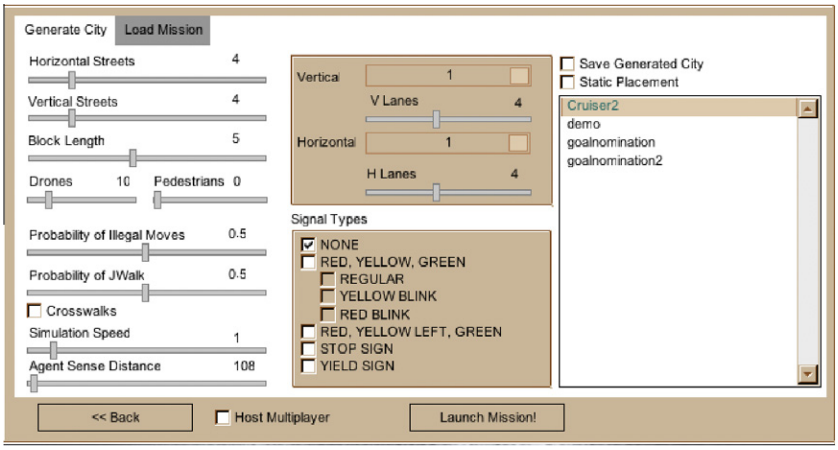

Fig. 3. User interface for controlling city environment in the driving domain.

Fig. 3 shows the user interface through which we can control these parameters in the city.

3.3. Basic driving behavior in ICARUS

To determine whether ICARUS supports interesting high-level behavior in this domain, we first needed to create a program that exhibits basic driving capabilities. These include vehicle acceleration and deceleration, cruising at constant speed, aligning to lane lines, lane changes, and turns. To this end, we wrote an ICARUS program with 70 concepts and 32 skills, some of which are shown in Table 5. Among them, 18 concepts and 22 skills are primitive. The concept hierarchy is five levels deep and the skill hierarchy is three levels deep, the latter with some recursive structures. With this program, ICARUS can maintain its cruising or turning speed, stay on the right side of the street, keep aligned and centered in a lane, change lanes to the left or right, make a right turn, and stop its vehicle.

The quality of driving behavior varies slightly on machines with different speeds, but, on a typical dual-core computer, this minimal agent manages to execute the basic

Table 5
Some sample concepts and skills for ICARUS's basic driving behavior in the urban driving domain.

```
((yellow-line ?line)
  :percepts                              ((lane-line ?line color YELLOW)))
((at-turning-speed ?self)
  :percepts                              ((self ?self speed ?speed))
  :tests                                 ((>= ?speed 15)
                                         (<= ?speed 20)))
((ready-for-right-turn ?self)
  :relations                             ((in-rightmost-lane ?self ?l1 ?l2)
                                         (at-turning-speed ?self)))
((in-intersection-for-rt ?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)))
((on-street ?self ?tg)
  :percepts                              ((self ?self)
                                         (street ?st)
                                         (street ?tg)
                                         (intersection ?int))
  :start                                 ((intersection-ahead ?self ?int ?tg))
  :subgoals                              ((ready-for-right-turn ?self)
                                         (in-intersection-for-rt ?self ?int ?st ?tg)
                                         (on-street ?self ?tg)))
((ready-for-right-turn ?self)
  :percepts                              ((self ?self))
  :subgoals                              ((in-rightmost-lane ?self ?l1 ?l2)
                                         (at-turning-speed ?self)))
```

maneuvers competently. However, the agent has limited flexibility to adapt to changing situations. It simply considers specific goals that it is initially programmed with, and does not and cannot change its strategy during runtime. As we discuss in the next section, the limitation observed in the basic agent's behavior suggests the need for substantial extensions to ICARUS that overcome this problem.

4. Reactive goal management

As seen earlier, ICARUS has a goal memory that stores information on its top-level goals and subgoals along with their corresponding skill instances. Most contents of the memory are very specific and short-lived, and they change as the agent moves along its path toward achieving its goals. But the top-level goals themselves did not change so far. It was as if an oracle gave the agent a set of goals it should always pursue, which does not change over time.

This, however, is not very reasonable. When people are pursuing some goals, more urgent matters come up sometimes and they should deal with them first. Or, they get distracted by something that happens in the environment. In the extended architecture, therefore, the top-level goals for agents change dynamically, rather than staying constant throughout the course of execution. The system has a new goal nomination process that generates top-level goals for an agent on each cycle. The nominated goals are based on the generalized descriptions of goals in a *long-term goal memory*. In this new memory, we can program both general and domain-specific rules for the nomination (and implicitly, the retraction) of goals.

Once some goals are nominated for the cycle, ICARUS sorts them according to their relative priorities, calculated from the default priority value associated with the corresponding long-term goals. Since the default priority values are constants, however, the architecture requires a dynamic measure to modulate these based on the situation. Otherwise, the ordering of goals will stay fixed. We use the relevance of each goal as the measure, computed through continuous matching of the relevance conditions associated with the goal. In this section, we first introduce a new representation for goals and then describe the processes for nomination, retraction, and prioritization of goals. We also present the notion of continuous matching of concepts.

4.1. Representation

Probably, the best way to introduce the new representation of goals is by examples. Table 6 shows some sample goals stored in the long-term goal memory. Each element takes the form of <*condition*, *goal*> pair that specifies the generalized goal and the conditions under which it is relevant. These pairs resemble Ohlsson and Rees' (1991) constraints, but here they applies only in the context of goals. The relevance conditions stored in `:nominate` fields are concepts that the system can match against its beliefs, and the goal concepts that appear as heads of the elements

Table 6
Some sample ⟨*relevance conditions*, *generalized goal*⟩ pairs stored in ICARUS's long-term goal memory.

```
((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)
```

use some common variables from these relevance conditions. The relationship between the long-term goal memory and the existing (short-term) goal memory is similar to those between long-term concept and skill memories and their respective short-term counterparts. This is a feature that has some architectural significance, which shows the unified nature of ICARUS.

The elements of ICARUS's long-term goal memory also have priority values associated with them, which represent the relative importance of the goals compared to others in the memory. Users provide a default prioritization measure by defining the fixed priority values. This corresponds to the general idea people seem to have on what is more important and what is less so. For instance, most people would agree that saving one's life has priority over saving his or her possessions. Many people will also save a child before saving an adult if caught in an accident. There are many examples like these, and we believe that the default priorities among ICARUS's generalized goals represent this behavior. Below, we continue our discussion on the processes that use the new representation.

4.2. Nomination and retraction processes

When the ICARUS architecture finds a match for any relevance condition stored in its long-term goal memory, it instantiates the corresponding goal accordingly. The system then stores the instantiated goal in its short-term goal memory. When this nomination process is complete, the system has a series of top-level goals, which guides the behavior during the cycle as in the original architecture. The nomination process starts after the architecture infers its beliefs based on the perceptual information from the environment. The system then goes through each <relevance conditions, generalized goal> pair stored in the long-term goal memory, and makes attempts to match the relevance conditions against the current beliefs. Whenever this attempt is successful, ICARUS instantiates the corresponding goal with the variable bindings it has found from the match. This also means that the retraction of goals happens without any additional mechanisms. If a currently nominated goal loses its match for relevance in the subsequent cycles, the system no longer nominates the goal, in effect, retracting it from the short-term goal memory. During this retraction, however, ICARUS retains

Please cite this article in press as: Choi, D. Reactive goal management in a cognitive architecture. *Cognitive Systems Research* (2011), doi:10.1016/j.cogsys.2010.09.002

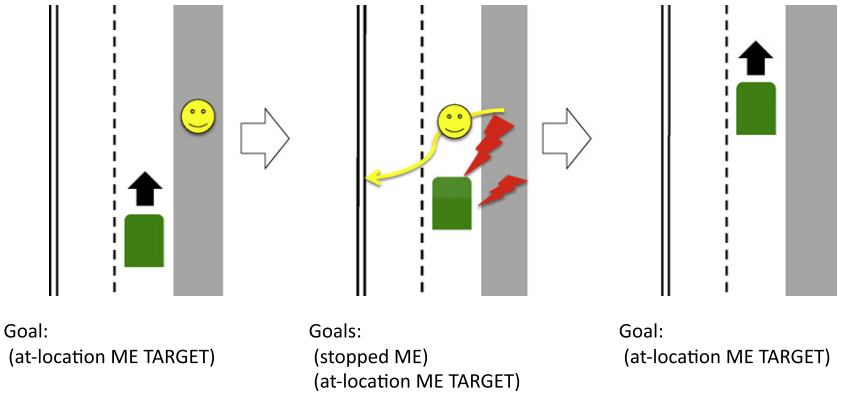

Fig. 4. An example of goal nomination process in an urban driving domain.

some information on the previous nomination, and uses it at a later time if the same goal instance is nominated again.

For example, Fig. 4 shows a simple situation that involves the nomination and retraction of a goal. Initially, there is nothing ahead of the agent's car (shown as a green box) moving upwards in the figure. Therefore, it has only a single goal to get to its target location. Then a pedestrian, *ped1* (shown as a yellow smily face), suddenly starts to jaywalk the street in front of the agent's car and it causes a concept instance, *(pedestrian-ahead me ped1)* to match in the state. In response, the system generates the corresponding goal, *stopped*, and now it has two goals as shown in the second column. When the pedestrian moves away, the relevance condition disappears and the goal is retracted. The agent now has a single goal again, as shown in the last column.

4.3. Prioritization process

In the above, we found that it suffices to have the existing boolean descriptions of the world if we are only concerned with the nomination and retraction of goals. This is due to the all-or-none characteristic of the nomination process, which decides whether or not the system should have certain goals at any given time but does not change the ordering of goals from what is initially given to the system. However, we see the necessity for goal prioritization on top of the simpler goal nomination and retraction processes. For instance, in the urban driving domain, a firetruck driver will observe the traffic rules most of the time as regular drivers would, but when there is an emergency, he might drive on the left side of the road to avoid traffic and get to the destination quickly. Situations like this require a change of the priorities among goals, and for this we need a dynamic measure that will affect the priorities.

Since we already have constant priority values, perhaps we need some measure that changes in reaction to the environment, with which we can modulate the priority values and allow reactive prioritization. We focus on the fact that the severity of conditions affects the reactions in human behavior (e.g., whether or not a firetruck driver ignores traffic rules depends on the seriousness of the emergency situation), and we propose modulating the constant priority values based on the degree with which the relevance conditions match in the current state. We can compute the *degree of match* for concept instances as scalar numbers between zero, which means that the instance does not hold in the state, and one, which means that the instance is true in the state.

Using the degree of match for the relevance conditions, we modulate the corresponding goal's priority value simply by multiplying them together. Now the priority value for a certain situation would be between zero and the default priority value of the goal. If the relevance condition is highly true, then the corresponding goal instance gets almost all its default priority value. If the condition is less true, its goal gets a lower priority value than the default one. This modulation allows changes in the order of the nominated goals based on the situation around an ICARUS agent. Since the continuous match of concepts plays such an important role in this process, we will describe the mechanism in a separate section below.

4.4. Continuous concept matching

Agents that operate in some environment react to what is happening around them, and ICARUS computes a set of beliefs about its environment for this purpose. As seen earlier in this paper, the architecture infers its beliefs using its concepts stored in long-term memory and the information received from the environment. In the original ICARUS, the inference process involves matching variables against objects in the world and their attribute values. There are also certain conditions imposed on these variables. Inference leads to a Boolean value that indicates whether or not a concept instance is true. This *symbolic* pattern matching has served the architecture well in a variety of domains, but we need a more continuous approach to concept inference for reactive goal prioritization.

We can find a source of continuity in concepts that incorporate numeric tests in their definitions. By applying a monotonic curve at the boundaries of these tests, we

Please cite this article in press as: Choi, D. Reactive goal management in a cognitive architecture. *Cognitive Systems Research* (2011), doi:10.1016/j.cogsys.2010.09.002

can get some continuous measures of how close we are to satisfying the tests. We will refer to the variables in these tests as *pivots* to emphasize that they serve as references for the computation of continuous degree of match for concepts. This approach is very similar to Black's (1937) notion of *vagueness* and Zadeh's (1965) *fuzzy sets*, in which a function defines degree of membership as a number between zero and one. It is important to distinguish the degree of match from the probability of match, which would describe the possibility of a concept being true.

As an example, let us use the case of a right turn in the urban driving domain. Depending on the angle of the turn, there will be an ideal range of the speed and the steering wheel angle of the car. For a typical 90 degree turn, we have found that a speed between 15 and 20 (measured in the simulation's internal unit for speed) and a steering angle of 10 degrees produce reasonable behavior. As shown in Table 7, we define two concepts, *at-turning-speed* and *at-steering-angle-for-right-turn*, that describe the situation that the agent is ready for a right turn in terms of speed and steering wheel angle, respectively. The first two concepts are for the original architecture, while the rest use the extended representation for the continuous matching.

Assuming that we have an agent named 'me,' the traditional inference process would output (*at-turning-speed me*) when the agent's speed is between 15 and 20, but nowhere else. For the second concept, it would say (*at-steering-angle-for-right-turn me*) is true only when the steering wheel

Table 7
ICARUS concepts for right turns in the urban driving domain.

```
((at-turning-speed ?self)
  :percepts                                ((self ?self speed ?speed))
  :tests                                   ((>= ?speed 15)
                                           (<= ?speed 20)))
((at-steering-angle-for-right-turn ?self)
  :percepts                                ((self ?self steering ?angle))
  :tests                                   ((=?angle 10)))

((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))
```

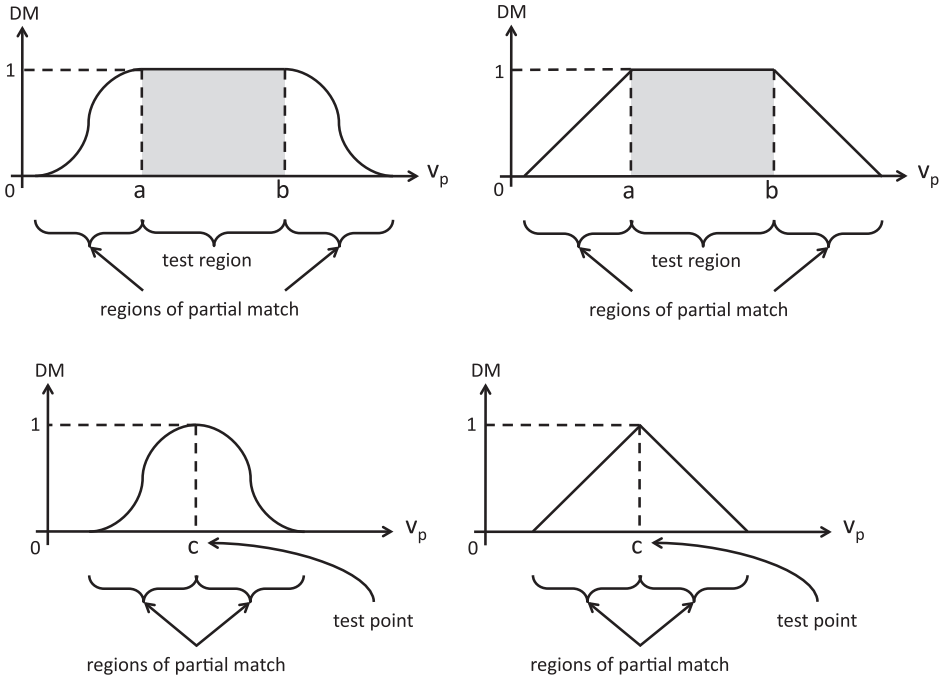

Fig. 5. Some monotonic functions applied to numeric tests against pivot variable v_p.

Please cite this article in press as: Choi, D. Reactive goal management in a cognitive architecture. *Cognitive Systems Research* (2011), doi:10.1016/j.cogsys.2010.09.002

angle is 10 degrees. When the speed and the angle fall out of the regions, even by a miniscule amount, the system does not infer any of these instances, and ICARUS cannot tell how bad the situation is for a right turn. However, if it uses the continuous matching capability with a new field, `:pivot`, for each of these concepts, the degree of match will tell the system how close the current situation is to satisfying these conditions.

When inferring instances for a primitive concept, the extended ICARUS still performs the pattern matching against percepts in the same way as before, but it applies a monotonic curve around the threshold values for the pivot variable in the numeric tests. For example, in Table 7, the third concept has a pivot variable, `?speed`, and ICARUS constructs a curve like those in the first row in Fig. 5. For the last concept, the numeric test involves an equality of a pivot variable, `?angle`, to a single number, 10, and the system superimposes two monotonic curves around the number. This results in a bell shape like the ones in the second row in the figure. Although the piecewise linear curves in the right column are very simplistic, they serve our purpose of modeling a decreasing degree of match away from the test regions. Ultimately, each test in primitive concepts should have different curves based on their sensitivity to changes in their pivot variables, but, for the purpose of this work, we assume they are fixed across all the tests.

Now that we have a way to extract continuous values from numeric tests, we can propagate them through ICARUS's concept hierarchy. Some primitive concepts have more than one numeric test, so it is possible for a concept to have two or more pivot variables. This leads to the need for combining multiple degrees of match along these variables. Furthermore, higher-level concepts consist of other concepts, which may include multiple primitive concepts with their own degrees of match, so we need a mechanism to combine two or more degrees of match. For this, we treat each degree of match as an axis in a multi-dimensional space and compute the combined degree of match as the vector sum specified by individual degrees of match on different Cartesian axes. After a normalization, the continuous degree of match propagates upward through the concept hierarchy, enabling higher-level concepts to serve as relevance conditions for goal prioritization.

5. Illustrative examples

With the extensions described, we believe ICARUS provides a psychologically plausible account of goal management. Testing this claim, however, does not lend itself to standard techniques, because cognitive capabilities like goal management occur at a very high level. We want to show performance improvements achieved in the extended system, but doing so using quantitative measures would be very difficult. Instead, we can demonstrate the qualitative behavior of the extended system and show that it improves functionality while aligning more closely with our intuitions about human cognition. We might also compare the extended system's behavior to that of other architectures, but it is difficult to give credit to or blame a particular mechanism in these architectures based on quantitative results. In response, Cassimatis, Bello, and Langley (2008) have suggested that models of higher-order cognition should be evaluated on three dimensions: their ability compared to humans; the breadth of situations they cover; and the parsimony of their mechanisms. We believe the extended system improves significantly over the original architecture on all three accounts, but, in this paper, we focus on issues of ability and parsimony.

In this section, we examine the behavior of both the original and the extended systems in two scenarios. This comparison shows the advantages of having the goal management capability in terms of programmability and human-like behavior. Often the original system that lacks goal management cannot demonstrate the desired behavior at all, while the extended system can do so easily.

5.1. Scenario 1: cruiser

Imagine someone in a sports car cruising down the street. The driver notices a car slowing down, and he swerves around it by changing lanes. Later, some pedestrians suddenly run into the street. In response, he stops to avoid hitting them and continue onward down the road. Situations of this sort should be familiar to any urban driver.

In the previous version of ICARUS, we would produce such responses by giving the agent two goals, *(stopped-and-all-clear me)* and *(cruising-in-lane me ?line1 ?line2)*, in this order. The resulting system would give higher priority to the first goal, so it would focus attention on maintaining a safe distance from pedestrians before worrying about maintaining its cruising speed. However, this approach has a number of drawbacks. Not only the system has the first goal regardless of its relevance, but the goal also does not mention any specific pedestrian, so the system must pick a pedestrian dynamically within the skills for this goal. As a result, the agent can deal with only one pedestrian at a time. We might program things so that the closest pedestrian receives attention, but then the agent would have no way to consider any other pedestrians.

In contrast, the goal nomination capability of the extended architecture lets us program three long-term goals, such as (*stopped-and-clear me ?ped*) with the nomination condition (*pedestrian-ahead me ?ped*), (*clear me ?car*) with the nomination condition (*vehicle-ahead me ?car*), and (*cruising-in-lane me ?line1 ?line2*) with a null nomination condition. Table 8 shows ICARUS concepts and skills for the extended system that take this approach. One advantage of this strategy is that the agent will have only the relevant set of goals at any given moment. More important, the ICARUS agent can consider each goal instance separately. For example, if three pedestrians are jaywalking in front of the agent's car, then three instances of the generalized goal (*stopped-and-clear me ?ped*) are deposited into

Please cite this article in press as: Choi, D. Reactive goal management in a cognitive architecture. *Cognitive Systems Research* (2011), doi:10.1016/j.cogsys.2010.09.002

Table 8
ICARUS's concepts and skills for the Cruiser scenario using the extended architecture.

```
((stopped-and-clear ?self ?obj)
  :percepts                                   ((self ?self))
  :relations                                  ((stopped ?self)
                                              (clear ?self ?obj)))

((clear ?self ?obj)
  :percepts                                   ((self ?self)
                                              (pedestrian ?obj))
  :relations                                  ((not (pedestrian-ahead ?self ?obj))))
((clear ?self ?obj)
  :percepts                                   ((self ?self)
                                              (car ?obj))
  :relations ((not (vehicle-ahead ?self ?obj))))

((stopped-and-clear ?self ?obj)
  :percepts                                   ((self ?self))
  :actions                                    ((*brake 1000)))
((clear ?self ?obj)
  :percepts                                   ((self ?self))
  :start                                      ((in-leftmost-lane ?self ?linel ?line2))
  :subgoals                                   ((in-rightmost-lane ?self ?line3 ?line4)))
((clear ?self ?obj)
  :percepts                                   ((self ?self))
  :start                                      ((in-rightmost-lane ?self ?linel ?line2))
  :subgoals                                   ((in-leftmost-lane ?self ?line3 ?line4)))
```

short-term goal memory, so the system can consider all of them in the order of their priorities. This lets it take an action for the highest priority goal and continue to the subsequent ones if resources are available. Moreover, the system does not require a complicated goal concept. Instead of using multiple disjunctive definitions of a goal concept to cover complex cases, individual goals for different pedestrians are instantiated from a single generalized goal description, and deposited into the system's current goal memory.

Let us analyze a typical run with the extended system. The agent starts in the leftmost lane of a street segment. There are several other cars in that stretch of the street and the first one, *c6120*, is far ahead of the agent in the same lane. For the first 10 cycles, the agent has a single goal, (*cruising-in-lane me ?line1 ?line2*), that is nominated and retained from the beginning. On cycle 11, as the ICARUS agent gets closer to the car, *c6120*, it detects that the car is blocking its way when (*vehicle-ahead me c6120*) becomes true. In response, the system nominates (*clear me c6120*) as a goal. There are no pedestrian in sight, so there are no other goals. On the next cycle, ICARUS retrieves a skill for the first goal with the same name, *clear*, which leads to the action (**steer 35*). While the agent is changing its lane to the right, it notices on cycle 13 that its speed is below the predefined cruising speed and the second goal, *cruising-in-lane*, is unsatisfied. This leads the agent to execute (**gas 20*) concurrently with (**steer 35*) to adjust its speed. The system continues steering to the right while it performs the speed adjustments as needed until cycle 21, when it notices that its car is in the target lane and starts aligning itself. By this time, the agent has successfully avoided the blocking vehicle and the concept instance (*vehicle-ahead me c6120*) is no longer true. As a result, the goal (*clear me c6120*) that was triggered previously by this concept instance disappears.

5.2. Scenario 2: ambulance

To make the task more complicated, let us think about driving an emergency vehicle, say an ambulance. When we walk down a street, we sometimes notice that an ambulance is driving normally, waiting for pedestrians to pass, observing the speed limit, and stopping for red lights. Yet other times we see an ambulance speeding by almost as though driven by a reckless driver, blinking its lights and blaring its siren. We may guess that they are responding to problems of different severities that affect the behavior of the drivers.

Modeling this difference in the original ICARUS is very difficult, requiring the programmer to write concepts and skills for all possible cases. Even then, the system would require so many pattern matchings that it would run very

Table 9
ICARUS concepts and skills for the Ambulance scenario using the extended architecture.

```
((emergency ?self)
  :percepts                   ((self ?self status ?status level
                              ?level))
  :tests                      ((equal ?status 'emergency)
                              (= ?level 10))
  :pivot                      (?value))
((not-emergency ?self)
  :percepts                   ((self ?self))
  :relations                  ((not (emergency ?self))))
```

slowly. However, the extended system supports this behavior easily by using generalized goals and their continuous conditions for goal nomination. Table 9 shows two new concepts that we added for this scenario to support continuous matching.

To let the agent reach the hospital with the proper urgency, we encode a goal, (*okay-to-go me ?signal*) with priority 2, to have nomination conditions (*signal-ahead me ?signal*) and (*not-emergency me*). This goal forces the agent to observe traffic signals when there is no emergency. But when in emergency, the degree of match for the concept, (*emergency me*), starts to increase from zero. This, in turn, causes the degree of match for the negated concept, (*non-emergency me*), to decrease from one. When this happens, the relevance of the goal drops, eventually making the agent focus first on the other goal, (*cruising-in-lane me ?line1 ?line2*) with priority 1, which moves the ambulance to the hospital.

Now let us consider how the system behaves during a typical run. As before, the agent starts out by accelerating itself to reach its cruising speed. On cycle 7, it finds a car blocking its path and starts steering to the right to avoid the car. With occasional accelerations to maintain speed, it continues steering to the right. On cycle 13, the agent notices that it is in the target lane and starts to cruise. It soon finds another car that it avoids in a similar manner, this time on the left, and finishes the maneuver by cycle 21. The agent then arrives at an intersection. Because the traffic signal is red, it brakes to stop. During the wait, the emergency level changes to 8, which, in turn, changes the degree of match for the concept instance *(emergency me)* to 0.8. The negation of this instance, *(not-emergency me)*, therefore matches with degree 0.2. This is a nomination condition for one of the current goals, *(okay-to-go me c27224)*. Hence the system modulates the priority value of the goal to be $2 \times 0.2 = 0.4$. This causes the goal to be less important than the default, *(cruising-in-lane me ?line1 ?line2)*, which has priority 1. Therefore, the system stops observing traffic signals and continues driving even through red lights. Later on cycle 95, when it reaches the next intersection, the emergency level drops back to 3 and the modulated priority value for *(not-emergency me)* becomes 0.7. This again places the goal to obey traffic signals before the default goal of driving ahead, and the system starts observing signals again.

6. Related work

Our work has been heavily influenced by related research in psychology, where one can find considerable work on motivation and goal selection. Many psychologists have recognized that the internal and external states of an agent influence motivation (Miller, Galanter, & Pribram, 1960; Norman & Shallice, 1986; Bargh, 1990; Moskowitz & Gesundheit, 2009). For example, Norman and Shallice's Norman and Shallice (1986) detailed model for control of behavior includes the environmental stimuli, motivational factors, and an attentional system to govern the activation of goals and the selection of action schemas. A *supervisory attentional system* activates goals under influence from motivational factors and the environmental stimuli control behavior. In this model, as Bargh (1990) summarizes, there is no explicit connection between environmental stimuli and the activation of goals. In contrast, ICARUS has an explicit link between the environment (represented as an agent's internal beliefs) and the selection of goals. Nomination rules stored in long-term goal memory controls this process by matching the relevance conditions associated with goals against current beliefs.

Simon (1967) proposed goal-terminating and interruption mechanisms that enable an essentially serial information processor to deal with unpredictable situations in real time. His termination mechanism stops further actions when a goal is achieved, which ICARUS incorporates as one of its basic features. Simon's interruption mechanism assumes a motivational system similar in spirit to the one we have developed. He argues that an urgent need should interrupt ongoing actions and force an agent to focus its attention to the immediate problem. The extended ICARUS can achieve this functionality in either of two ways. First, if the interruption is commonly observed, a developer can program long-term goals with the interrupting situation as a negated nomination condition. Alternatively, one can have a separate goal to deal with such a situation and let the goal prioritization mechanism change the goal orders to focus on the urgent problem first. Of course, ICARUS's concurrent execution capability still lets it pursue other goals subject to coordination constraints.

More recently, Sloman (1987, 2002) proposed 'motivators' to resolve conflicts among goals. In contrast to ICARUS, which uses degree of relevance to prioritize goals, he proposed three different measures – insistence, urgency, and intensity – that should affect how a system manages goals. ICARUS's notion of relevance does not map directly to these measures. Rather, the architecture computes the degree of conceptual match and the concepts can encode any of them. ICARUS provides a developer the ability to use any concepts for the relevance conditions on goals, and the developer must decide what type of measure to use.

Blech and Funke (2010) extended their perspectives to different types of goal relations and studied the effect of independent, compatible, and conflicting goals to problem solving performance. The authors found that goal conflicts induce decreased motivation and increased stress, and that these factors can affect behavior and performance during problem solving. In ICARUS, all top-level goals are essentially in competition with one another. Priorities (default or modulated) determine the order of attention to these goals, and later goals are inaccessible to the agent until either earlier goals are satisfied or the system has remaining resource after executions for them. It is possible to implement a similar mechanism that simulates increased stress in ICARUS by reducing the number of resources available when the number of conflicting goals increase.

Please cite this article in press as: Choi, D. Reactive goal management in a cognitive architecture. *Cognitive Systems Research* (2011), doi:10.1016/j.cogsys.2010.09.002

Gray and Braver (2002) approached this topic in the broader context of emotion, which, they claimed, can prioritize conflicting alternatives and trade-offs. They further argued the need for integration of emotion and cognitive control, assuming this aids adaptation to the environment. They also outlined a set of emotion-related processing stages that relate a situation to behavior through approach and withdrawal responses that lead in turn to corresponding goals. We should note that researchers in this field sometimes use the term *emotion* very loosely. However, regardless of the details, there is general agreement that a motivational system generates goals for agents and that the environment influences this process. Our work on ICARUS follows this trend, but it adapts the explicit notion of long-term goals with associated relevance conditions.

There also are some cognitive architectures that address motivations and goals. One such system is CLARION (Sun, 2007), which incorporates implicitly represented drives and explicitly specified goals. Two subsystems in the architecture interact to nominate goals. A *motivational subsystem* maintains an implicit, value-based network that relates the state of the world and the strength of an agent's drives. A *meta-cognitive subsystem* uses a multiple vote approach to determine the current goal. Each internal drive proposes multiple goals in the order of their assigned numeric preference. The subsystem chooses the goal that receives most votes across all the drives and passes it to the execution module. In contrast, ICARUS does not divide this process into two submodules. The architecture maintains long-term goals as direct links from beliefs to goals. Unlike CLARION, which has a single goal that is active at a given moment, it also supports the nomination of multiple top-level goals.

Another architecture that incorporates an explicit goal nomination mechanism is Broersen, Dastani, Hulstijn, and van der Torre (2002)'s BOID. This is based on BDP logic (Thomason, 2000), which explains goals as a result of interactions between beliefs and desires, but the architecture also includes obligations and intentions. An agent in this framework computes beliefs from observations of the world, while desires and obligations that are consistent with these beliefs trigger goals. The system treats previously generated goals as intentions that it uses to generate successive goals. Interactions among these four *conditional mental attitudes* may produce more than one candidate goal set, and the architecture supports different 'agent types' that resolve conflicts in different ways. For example, a realistic agent tends to choose a goal derived from beliefs over one from obligations, while a dogmatic agent does the opposite. Although ICARUS handles beliefs and intentions in a similar fashion, it includes neither obligations nor desires. However, we can program many of these aspects using ICARUS's goal management mechanism, since the architecture does not have any restrictions on the relevance conditions for its long-term goals. As long as we can represent obligatory or desire-driven rules as conditionalized long-term goals, ICARUS can achieve similar effects.

Some other architectures provide limited capabilities for goal management, with Gordon and Logan's GRUE (2005) being a good example. This closely resembles ICARUS in that it extends the teleoreactive framework (Nilsson, 1994) in a number of directions. The authors include a goal arbitration mechanism that uses resources, but GRUE cannot nominate or retract goals, which ICARUS achieves using its goal management mechanism. In contrast, Soar (Laird et al., 1986) has the ability to nominate its top-level operators as its action-like goals or intentions. But the architecture proposes a single intention at a time, removing both the need for prioritization and the potential advantages of interactions among goals.

This topic of goal management has also been discussed in the context of planning under nondeterministic conditions. A good example is Molineaux et al.'s ARTUE system (2010), which integrates a hierarchical task network planner (Nau et al., 2003) with four additional components: discrepency detection between expected and observed states; explanation of the unexpected events; goal generation; and goal prioritization. Upon detection of symbolic or numeric discrepencies, the system attempts to find hidden factors that influence the state and explain the detected discrepencies by abduction. Then ARTUE generates goals using background knowledge in the form of *principles*, which consist of a set of participants: a condition, a fixed intensity level, and a goal form. This representation is similar to ICARUS's long-term goals, which include relevance conditions, a default priority value, and a generalized goal. During the goal management process, the ARTUE system chooses a single goal with the highest intensity, while ICARUS modulates the default priority values of goals with the degree of relevance and reorders goals using the resulting values.

The field of robotics also includes related research. For example, Hanheide et al. (2010) propose a goal generation and management framework for the PECAS robot architecture (Hawes et al., 2009). Following Beaudoin and Sloman (1993), this framework encodes an agent's drives as goal generators, which react to the internal and external states. An *attention filter* blocks some of the generated goals based on their importance and urgency, thus protecting the higher-level management process. This latter process either activates or suspends the filtered goals. The authors does not describe details of these processes, but they appear to use heuristics like information gain to determine the importance of candidate goals.

Motivation and self-control are also topics of interest in multi-agent systems literature. For instance, Luck and d'Inverno (1998) investigate the motivated generation of goals as a prerequisite for transfer or adoption of goals by multiple entities. The authors extend their previous work on agent autonomy by adding mechanisms for generation and adoption of goals. The system adopts a four-tiered hierarchical view of entities, objects, agents, and autonomous agents.

From a representational standpoint, the long-term goals in ICARUS resemble the constraints that Ohlsson and Rees

Please cite this article in press as: Choi, D. Reactive goal management in a cognitive architecture. *Cognitive Systems Research* (2011), doi:10.1016/j.cogsys.2010.09.002

(1991) use in their HS system. This encodes constraints as pairs of relevance and satisfaction conditions, which it uses to detect violated states and to revise rules to prevent further failures. Their notion of constraints is more general than our notion of conditionalized goals, in that they use constraints to evaluate or judge the state of the environment, which may include not only goals but also other internal and external aspects of an agent.

Although the extended ICARUS architecture does not yet offer a complete framework for general intelligence, the work reported in this chapter is an important step toward an architectural account of goal management. In addition to providing mechanisms for goal nomination and prioritization, it provides a unified approach to representing and interpreting goal knowledge in relation to other types of cognitive structures. Furthermore, it serves as a general mechanism that should let us investigate the effects of emotion, desires, and other factors that influence goals. As seen in the demonstrations above, the extended architecture can model the delicate interactions among instantiated goals that are needed for fine-grained judgements. ICARUS also has potential to model individual differences that occur in such prioritizations. In summary, our main contribution lies in proposing a unified framework that incorporates capabilities for goal management into a general cognitive architecture. This provides a solid foundation for future research, which we discuss in the next section.

7. Future work

Despite the novel contributions of this work, ICARUS is still not a complete framework for modeling human cognition. There is ample room for further improvements in goal management. In this section, we describe these open problems and propose solutions to them.

7.1. Concept-dependent continuous matching

An important aspect of the current system is the continuous matching of concepts. ICARUS applies monotonic curves at the boundary of numeric tests within concepts to compute the degree of match. In this paper, we used the same curve across all concepts. But we can imagine situations where a single, system-wide curve is inappropriate and where different concepts require distinct slopes at the edges of their numerical regions.

For example, we might have two concepts, *aligned-with-lane* and *heading-north*, in the urban driving domain. Since the accuracy we need to check whether we are driving straight in a lane is probably higher than that needed when measuring the heading, we want to match the former more strictly than the latter, with less tolerance permitted. In such cases, we should apply two different curves to guide the continuous matching of concepts, which would impose two distinct levels of tolerance.

Supporting the use of multiple curves does not require major changes to the architecture, but we need more than a selection of curves. For this feature to be useful, ICARUS should update the shape of its curves based on experience. However, the learning process must be closely related to the outcome of a goal achievement. This requirement makes the extension more complicated than simple value learning.

To address this problem, ICARUS will need to adjust the shape of the curves based on the results of execution – a process that is influenced by several factors. We can incorporate ideas from work on fuzzy logic controllers for a solution. For example, Berenji (1992) proposed a learning mechanism that updates monotonic membership functions of fuzzy sets using a method similar to temporal difference learning. ICARUS's monotonic functions for partial matching maps directly onto these membership functions for fuzzy sets, so this approach to learning should be applicable.

7.2. Uncertainty of beliefs

We also recognize the need for further research on uncertainty in ICARUS. Here we plan to focus on the issue of inaccurate percepts caused by noisy sensor outputs. The continuous matching capability provides one response to this problem, in that the mechanism lets ICARUS adapt to sensor errors by inferring beliefs outside the boundaries of concepts' numeric tests. For example, consider a concept that checks whether the distance to a vehicle is less than five. If ICARUS perceives a vehicle at distance six when, in fact, the vehicle is at five, continuous matching would still return a high degree of match for the concept instance, despite the fact that the numeric test is not satisfied. This feature lets the system tolerate inaccurate sensor values, much as fuzzy logic controllers.

However, continuous matching cannot handle missing percepts due to sensor limitations, such as limited visual fields and occlusion. One response to such issues would involve building a probabilistic version of the ICARUS inference process that lets the system recover from missing sensor outputs. There is a long history of work that addresses this type of challenge, with Markov logic networks (Richardson & Domingos, 2006) being a recent, popular example. These networks combine first-order logic and probabilistic models into a single representation. Each clause in the knowledge base constrains the truth values of the variables and has an associated weight that reflects the strength of the constraint. Given a set of observations, the system carries out a hill-climbing search through a space of possible worlds, returning the one it considers most probable. This world will contain beliefs that are consistent with the observations, given the available rules, including ones that cannot be derived deductively.

ICARUS's inference mechanism is quite modular, and replacing it with a Markov logic interpreter would be a straightforward task. Once integrated, it would address the problems of missing percepts. The increased stability of perception and inference would let ICARUS agents behave

Please cite this article in press as: Choi, D. Reactive goal management in a cognitive architecture. *Cognitive Systems Research* (2011), doi:10.1016/j.cogsys.2010.09.002

more reliably and realistically. For example, consider an agent driving its car down the street. It might see only the front half of a car coming around a corner. Although this partial sensing does not give the agent a complete percept of the car, a Markov logic interpreter would suggest the car is very probable, and the agent can make proper decisions in response.

7.3. Learning goal priority values

The final target for further research concerns the current mechanism for goal management. ICARUS assumes constants for the default priority values of long-term goals. Although we argued that these constants represent most people's notion of ethics, this does not mean that they cannot change. It seems plausible that values should drift slowly over time, as a person might change his ethical position over the years. We believe that experience can motivate this change, especially the experience of failures.

One approach to updating default priorities would introduce utility values associated with concepts and use them as default priorities for goals. Previous work by Asgharbeygi, Stracuzzi, and Langley (2006) has explored learning such utility values within a variant of ICARUS in the context of multi-player games with delayed rewards. These domains yield only a small number of conceptual predicates that have innate non-zero values. The system uses a version of temporal difference learning over its relational concept structures to propagate reward and update expected value functions for other predicates that may lead to concepts with innate values. Such value functions represent the predicates' importance in the domain, letting them serve as reasonable default values for goal prioritization.

However, as we discussed in the previous section, utility to the agent might not be the only factor for deciding goal priorities. Sloman (1987) proposed insistence and urgency as factors that influence goal priorities, and we can incorporate this idea to ICARUS. For this, it will be beneficial to have a separate priority learning mechanism. One approach would use a simple reward mechanism in which repeated failure to achieve a goal decreases the insistence value associated with that goal, and in which failure caused by delayed execution increases the urgency value associated with it.

8. Conclusions

In this paper, we introduced extensions to ICARUS that support goal management. In addition to its architectural significance, the approach has close connections to previous work in psychology. The extended framework nominates goals based on the current belief state, and it reacts to changes to pursue only relevant goals. We also described the new capability to prioritize nominated goals. Through a mechanism that computes continuous degrees of match for nomination conditions, the system modulates the default priority values assigned to goals and uses the resulting values to reprioritize them. The extended framework can handle dynamic tasks like comparing multiple instances of the same goal or choosing the most important among a set of goals. Using two examples from the urban driving domain, we demonstrated the ability to manage goals that the original architecture lacks.

We argue that the contribution of these extensions goes beyond the relative advantages we have shown. They also provide a unified approach to representing and interpreting goal knowledge within the context of ICARUS's existing long-term and short-term cognitive structures. Our comparisons to related work also suggest that these extensions hold promise for modeling emotion, desires, obligations, and other factors that affect the behavior of intelligent agents.

Acknowledgments

The research presented in this paper was performed while supported in part by Grant HR0011-04-1-0008 from DARPA IPTO and Project No. 2E20870 from Korea Institute of Science and Technology. No endorsement should be inferred. The author would also like to thank Pat Langley and Stellan Ohlsson. Discussions with them contributed to many ideas presented here.

References

Anderson, J. R. (1993). *Rules of the mind*. Hillsdale (NJ): Lawrence 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.

Asgharbeygi, N., Stracuzzi, D., & Langley, P. (2006). Relational temporal difference learning. In *Proceedings of the twenty-third international conference on machine learning* (pp. 49–56). Pittsburgh, PA.

Bargh, J. A. (1990). Auto-motives: Preconscious determinants of thought and behavior. In E. T. Higgins & R. M. Sorrentino (Eds.). *Handbook of motivation and cognition: Foundations of social behavior* (Vol. 2, pp. 93–130). New York (NY): Guilford Press.

Beaudoin, L. P., & Sloman, A. (1993). A study of motive processing and attention. In A. Sloman, D. Hogg, G. Humphreys, A. Ramsay, & D. Partridge (Eds.), *Prospects for artificial intelligence: Proceedings of AISB-93* (pp. 229–238). Amsterdam: IOS Press.

Berenji, H. R. (1992). A reinforcement learning-based architecture for fuzzy logic control. *International Journal of Approximate Reasoning, 6*, 267–292.

Black, M. (1937). Vagueness: an exercise in logical analysis. *Philosophy of Science, 4*, 427–455.

Blech, C., & Funke, J. (2010). You cannot have your cake and eat it, too: How induced goal conflicts affect complex problem solving. *Open Psychology Journal, 3*, 42–53.

Broersen, J., Dastani, M., Hulstijn, J., & van der Torre, L. (2002). Goal generation in the BOID architecture. *Cognitive Science Quarterly, 2*, 428–447.

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). Coordinated execution and goal management in a reactive cognitive architecture. Ph.D. Thesis. Stanford University.

Please cite this article in press as: Choi, D. Reactive goal management in a cognitive architecture. *Cognitive Systems Research* (2011), doi:10.1016/j.cogsys.2010.09.002

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.

Gordon, E., & Logan, B. (2005). Managing goals and resources in dynamic environments. In D. N. Davis (Ed.), *Visions of mind: Architectures for cognition and affect* (pp. 225–253). Idea Group.

Gray, J. R., & Braver, T. S. (2002). Integration of emotion and cognitive control: A neurocomputational hypothesis of dynamic goal regulation. In M. Oaksford & S. C. Moore (Eds.), *Emotional cognition: From brain to behaviour* (pp. 289–316). Philadelphia (PA): John Benjamins Publishing Company, chapter 12.

Hanheide, M., Hawes, N., Wyatt, J., Göbelbecker, M., Brenner, M., Sjöö, K., et al. (2010). A framework for goal generation and management. In *Proceedings of the AAAI – 2010 workshop on goal-directed autonomy*. Atlanta (GA): AAAI Press.

Hawes, N., Zender, H., Sjöö, K., Brenner, M., Kruijff, G.-J.M., & Jensfelt, P. (2009). Planning and acting with an integrated sense of space. In *Proceedings of the first international workshop on hybrid control of autonomous systems* (pp. 25–32).

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., & Choi, D. (2006). A unified cognitive architecture for physical agents. In *Proceedings of the twenty-first national conference on artificial intelligence*. Boston: AAAI Press.

Luck, M., & d'Inverno, M. (1998). Motivated behaviour for goal adoption. In C. Zhang & D. Lukose (Eds.), *Multi-agent systems: theories, languages and applications – proceedings of the fourth australian workshop on D istributed artificial intelligence LNAI 1544* (pp. 58–73). Springer-Verlag.

Miller, G. A., Galanter, E., & Pribram, K. H. (1960). *Plans and the structure of behavior*. New York: Holt, Rinehart & Winston.

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*. Atlanta, GA: AAAI Press.

Moskowitz, G. B., & Gesundheit, Y. (2009). Goal priming. In G. B. Moskowitz & H. Grant (Eds.), *The psychology of goals* (pp. 203–233). New York (NY): Guilford 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.

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

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

Norman, D. A., & Shallice, T. (1986). Attention to action: Willed and automatic control of behavior. In R. J. Davidson, G. E. Schwartz, & D. Shapiro (Eds.), *Consciousness and self-regulation: Advances in research and theory* (pp. 1–18). New York: Plenum Press.

Ohlsson, S., & Rees, E. (1991). Adaptive search through constraint violations. *Journal of Experimental & Theoretical Artificial Intelligence, 3*, 33–42.

Richardson, M., & Domingos, P. (2006). Markov logic networks. *Machine Learning, 62*, 107–136.

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.

Sloman, A. (2002). How many separately evolved emotional beasties live within us? In R. Trappl, P. Petta, & S. Payr (Eds.), *Emotions in humans and artifacts* (pp. 35–114). MIT Press.

Sun, R. (2007). The motivational and metacognitive control in CLARION. In W. Gray (Ed.), *Modeling integrated cognitive systems* (pp. 63–75). New York, NY: Oxford University Press.

Thomason, R. H. (2000). Desires and defaults: A framework for planning with inferred goals. In *Proceedings of the seventh international conference on the principles of knowledge representation and reasoning* (pp. 702–713). San Mateo (CA): Morgan Kaufman.

Zadeh, L. A. (1965). Fuzzy sets. *Information and Control, 8*, 338–353.

Please cite this article in press as: Choi, D. Reactive goal management in a cognitive architecture. *Cognitive Systems Research* (2011), doi:10.1016/j.cogsys.2010.09.002