Nomination and Prioritization of Goals in a Cognitive Architecture

Dongkyu Choi
Department of Aeronautics and Astronautics
Stanford University, Stanford, CA 94305
dongkyuc@stanford.edu

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 behavior. 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 an important role by guiding the behavior of ICARUS agents. However, the architecture does not cover the origin of its goals or the management of them, and this imposes restrictions like limited autonomy in ICARUS. In this paper, we extend the architecture to provide the capability to manage top-level goals using the notion of long-term, general goals. We show some illustrative examples in an urban driving domain, and discuss related and future work in this direction.

Introduction and Motivation

Goals play an important role in human cognition. People have ideas on what they want to do or what they should do, and these give rise to many different goals. Such goals, in turn, guide people's behavior by restricting the space of possible actions to take. Traditionally, psychologists put significant efforts on the study of this process (Simon, 1967; Sloman, 1987; Gray & Braver, 2002 to name a few). As computational frameworks for models of cognition, most cognitive architectures (Newell, 1990), too, have some supports for goals. At the very least, these architectures allow the specification of goals or subgoals that guide the artificial agent's behavior. But some architectures provide more, including nomination and prioritization of goals. For instance, CLARION (Sun, 2007) has drive and goal mechanisms that correspond to psychological accounts of goal nomination. In Soar (Laird et al., 1986), the top-level operators can act as reactive goals and there are rules that govern their nomination as goals.

Another cognitive architecture, ICARUS (Langley & Choi, 2006), operates in a goal-directed fashion, and uses multiple top-level goals. However, the architecture lacks any mechanism to add, delete, or reorder such goals, limiting its capabilities significantly. In this paper, we present the ICARUS architecture with a new goal management mechanism that is reactive to the environment. We extended the existing architectural distinction between long-term knowledge and short-term structures to goals by introducing general goal descriptions associated with their own relevance conditions. The system instantiates these goals with respect to the current situation of the world and nominates them as its own top-level goals to guide its behavior. The extended architecture also has a new ability to prioritize its nominated top-level goals by modulating their priority values with continuous degrees of match for the relevance conditions.

In the subsequent sections, we briefly review the ICARUS architecture and explain the extension for nomination and prioritization of goals in detail. Then we provide some illustrative examples in an urban driving domain. After that, we conclude after a discussion on related and future work.

Review of the ICARUS Architecture

ICARUS 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 nomination and prioritization of goals. We start with ICARUS' representation of knowledge and memories that support this, and then cover the architecture's inference and execution processes. Throughout this section, we show examples from an urban driving domain, which we also use for demonstration purposes in a later section.

Representation and Memories

The ICARUS architecture distinguishes conceptual and procedural knowledge. Its *concepts* describe various aspects of the environment, whereas its *skills* define procedures that are known to achieve corresponding concepts when executed until completion. ICARUS also distinguishes long-term knowledge and short-term structures. Long-term knowledge includes general descriptions of the environment and procedures. The architecture instantiates them and gets short-term structures 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 use variablized objects and their attributes to describe situations. A long-term skill memory houses variablized skills that define general procedures to achieve certain concepts, namely their goals. When the system instantiates these general concepts and skills, it deposits them in the corresponding short-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 some sample concepts in an urban driving domain. The first two concepts are *primitive*, and they include only perceptual matching conditions that ground on object information from the environment in the `:percepts` and `:tests` fields. On the other hand, the last concept is *non-primitive*, since it refers 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 some examples of skills in this domain. In a similar fashion to their conceptual counterparts, there are primitive and non-primitive skills. The first skill shown is primitive, and it consists of perceptual matching conditions, preconditions, and a direct reference to an immediate action in the world. The other two skills, however, are non-primitive, and they provide subgoal decompositions instead of references to actions. In the next section, we cover ICARUS' processes that work over these knowledge structures stored in its memories.

Table 1: Some sample ICARUS concepts for 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)))
```

Table 2: Some sample ICARUS skills for the urban driving domain.

```
((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)))
```

Inference and Execution

The ICARUS architecture operates in distinct cycles. On each cycle, the system invokes a series of processes including the inference of the current belief state and the execution of skill paths relevant to the situation. ICARUS receives sensory data from the environment at the beginning of each cycle. Based on the perceptual information, the system infers its belief state, namely, all the concept instances that are true in the current state. It starts with primitive concepts at the lowest level and moves up the hierarchy to non-primitive ones. ICARUS performs this process on every cycle, and therefore, any naive approach to the belief inference is susceptible to the combinatorial effect found in domains with many objects. In response, there have been several efforts to alleviate this problem including Asgharbeygi et al. (2005).

When the system finishes inferring its belief state, it attempts to execute its skills accordingly. ICARUS retrieves skills that are relevant to its top-level goals, and finds one or more executable paths through the hierarchy that start from these skills. A skill path is executable when all the skill instances on it are executable, from top to bottom. Although a path might include a single primitive skill that achieves an ICARUS agent's top-level goal, a skill path usually 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 includes some actions the system needs to perform in the environment. The ICARUS architecture takes these actions and applies them to make changes in its surroundings. Then the system repeats the processes based on the updated sensory data. In the following section, we continue our discussion on the architecture in the context of the new extension.

Reactive Goal Management

As seen in the previous section, the ICARUS architecture 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 in this memory. It was as if a godly entity 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 of their own, they do get distracted from the environment, and sometimes more urgent matters come up and they should deal with them first. To support this kind of behavior, the top-level goals change dynamically in the extended architecture, rather than staying constant throughout the course of execution. The system has a new goal nomination process that generates top-level goals for its agent on each cycle. The nominated goals from this process are based on the generalized descriptions stored in a new long-term goal memory. In this new memory, we can program both general and domain-specific rules for the nomination of goals. These rules collectively represent a basic form of motivational structure in ICARUS.

Once the architecture finishes nominating goals that are relevant to the current situation, it prioritizes them before start executing for the goals. The programmer assigns a default priority value to each general goal, and ICARUS modulates this value based on a continuous measure for relevancy of the

goal. The architecture computes the degree of match for the relevance conditions of a goal whenever possible, and uses this continuous matching value during the goal prioritization process. This continuous degree of match represents the degree of relevance for the goal in the current situation, and anything less than the complete relevance will reduce the priority value accordingly. In the subsequent sections, we explain the new representation and processes in detail.

Representation

Perhaps the best way to describe the new representation is through examples. Table 3 shows some sample goals stored in the long-term goal memory. Each element takes the form of a *<conditions, goal>* pair that specifies the generalized goal and the conditions under which it is relevant. The relevance conditions stored in `:nominate` fields are templates for concepts that the system can match against its beliefs, and the goals are concepts that use some common variables that appear in the relevance conditions. The relation between this 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 counterparts. This is a feature that has an architectural significance, which shows the unified nature of ICARUS.

Table 3: Some sample *<relevance conditions, generalized goal>* pairs stored in ICARUS' 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)
```

The elements of ICARUS' 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 predefine the goals and their associated priority values, providing a default prioritization measure. This corresponds to the general idea people seem to have on what is more important and what is less so. For instance, most people 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 consider the default priorities assigned to generalized goals in ICARUS' long-term goal memory as representing this behavior. Next, we continue our discussion on the new processes that use this memory.

Nomination Process

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 guide the behavior during the particular cycle.

The nomination process starts after the architecture infers its belief state based on the perceptual information from the environment. The system goes through each *<relevance condition, generalized goal>* pair stored in the long-term goal memory, and makes attempts to match the relevance conditions against the current state. Whenever its 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 relevance in the subsequent cycles, the system no longer nominates the goal, effectively retracting it from the short-term goal memory. During this retraction, however, ICARUS stores some information on the previous nomination, and uses it at a later time if the same goal instance is nominated again.

Figure 1 shows a simple situation that involves the nomination and retraction of a goal. Initially, there is nothing in front of the agent's car (shown as a green box) moving upwards in the figure. Therefore, it has 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 this causes a concept instance, *(pedestrian-ahead me ped1)*, to match in the state. In response, the system generates the corresponding goal, *(stopped me)*, 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 has a single goal again, as shown in the last column.

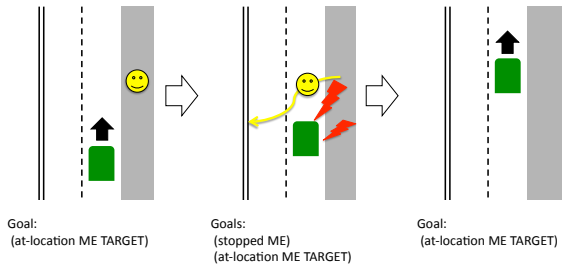

Figure 1: An example of goal nomination process in an urban driving domain.

Prioritization Process

Once ICARUS completes the nomination process, it attempts to reorder the currently nominated goals to prioritize them under the given circumstances. Since all the top-level goals have default priority values associated with them and ICARUS orders the goals according to these values, we need a mechanism to modulate these fixed values based on the current situation of the world. This modulation will then give goals with lower default priorities a chance to overtake higher-priority

ones. Our approach uses the continuous matching of concepts, more specifically, the relevance conditions associated with each goal.

As shown in the previous section, ICARUS' concepts include perceptual matching conditions. Especially, some primitive concepts have numeric tests in their bodies that often involve continuous variables. We take such variables as the source of continuous matching. For example, consider a concept that includes a numeric test on a variable, *?var*, as in $0 <?var < 10$. ICARUS normally checks if the value of the variable is within the specified range, and returns true (1) if it is larger than 0 and smaller than 10, but returns false (0) otherwise. But if we make the boundaries of the tests smoother as shown in Figure 2, we can get some partial matches between 0 and 1 when the variable falls right outside of the specified region.

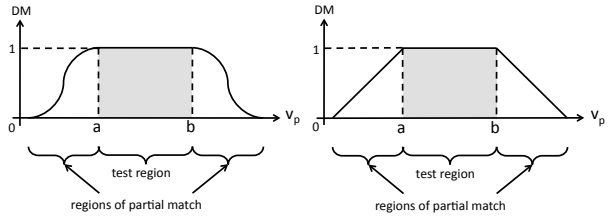

Figure 2: Curves applied to the boundaries of numeric tests for continuous matching.

When the relevance conditions associated with ICARUS' goals include a primitive concept, we can get the degree of match between zero and one using this mechanism. This value will then represent how relevant the associated goal is, and we can use it to modulate the default priority value of the goal. In this manner, a very relevant goal with a low default priority can overtake a barely relevant goal with a high default priority. We believe this explains people's behavior in extreme conditions like when people are extremely hungry or thirsty. In such cases, people will probably drink fluids with a bad smell that they would normally reject.

Illustrative Examples

With the extensions described so far, we believe the ICARUS architecture provides a reasonable account of goal management. Testing this hypothesis, however, is not of the standard affair. As is often the case in the evaluation of cognitive architectures, capabilities like the goal management are innately at a very high-level. We want to show performance improvements we can get from the extended system over the previous one, but doing so using several quantitative measures is not immediately possible in this case, and those results will not be quite representative either. Instead, we can demonstrate the qualitative behavior of the extended system and confirm that it is far more aligned with our intuition about human cognition than the previous system. Cassimatis et al. (2008) suggested that models of higher-order cognition should be evaluated in three aspects: their ability compared to humans, the breadth of situations they cover, and the parsimony of their mechanisms.

In this section, we challenge the original and the extended systems with two scenarios. By comparing the two systems, we show the advantages of the goal management in various aspects like programmability and human-like behavior. Often the original system is not capable of demonstrating the desired behavior at all, while the extended system can easily simulate it.

Scenario 1: Cruiser

Imagine that you are driving a sports car cruising down the street. You notice a car slowing down and stopping in front of you, and you swerve around the car by changing your lane. After a while, a group of careless pedestrians jump out to the road all of a sudden and jaywalk the street. Startled, but decisively you make a move to avoid hitting the pedestrians and continue your cruise down the road. Unless you are driving exclusively on freeways, this kind of situation should sound very familiar.

In the previous version of the ICARUS architecture, we would program this behavior by giving the system two goals, *(stopped-and-all-clear me)* and *(cruising-in-lane me ?line1 ?line2)* in this order. The system gives higher priority to the first goal than the second one, so it correctly focuses its attention to maintaining a safe distance from pedestrians before worrying about cruising on the street. However, we find several issues with this program. In addition to the fact that the system will have the first goal regardless of whether it is relevant or not, a more notable problem is that the first goal does not mention any specific pedestrian, and that the system will need to pick a pedestrian dynamically within the skills for this goal. This means that the system can cover for only one pedestrian at a time. We will probably program it so that the closest pedestrian from the ICARUS agent's position gets the attention, but no matter what we do, the system has no way to consider any other pedestrians.

On the other hand, using the extended system with the goal nomination capability, we would program three long-term goals like, *(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 4 shows ICARUS concepts and skills for the extended system that we wrote this way. The first advantage of this system over the previous one is that the agent has only the relevant set of goals at any given moment, much like people would. But what is more important in this particular case is that, the ICARUS agent can consider each instance of the goals separately. For instance, if there are multiple pedestrians jaywalking the street in front of the agent's car, multiple instances of the generalized goal, *(stopped-and-clear me ?ped)* will be deposited into the system's short-term goal memory, and the system will be able to consider all of them in the order of their corresponding priorities. By doing so, the system can take an action for the highest priority goal

and continue to the subsequent ones if resources are available. It is also notable that the system no longer requires a complicated goal concept. Instead, all the individual cases of different pedestrians are instantiated from a generalized goal description, and deposited into the system's short-term goal memory.

Table 4: ICARUS 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 ?line1 ?line2))
 :subgoals ((in-rightmost-lane ?self ?line3 ?line4)))

((clear ?self ?obj)
 :percepts ((self ?self))
 :start     ((in-rightmost-lane ?self ?line1 ?line2))
 :subgoals ((in-leftmost-lane ?self ?line3 ?line4)))
```

Let us analyze a typical run with this 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 always nominated. On cycle 11, as the ICARUS agent gets closer to the car, *c6120*, it detects that the car is blocking its way and the predicate, *(vehicle-ahead me c6120)*, becomes true in the state. So, the system nominates *(clear me c6120)* as its goal. On the next cycle, ICARUS retrieves a skill for the first goal with the same name, *clear*, and the skill leads to an 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. The agent now executes *(*gas 20)* concurrently with *(*steer 35)* to adjust its speed. It continues steering to the right while it performs the speed adjustments as needed until cycle 20, but then it notices that it is in the target lane, and starts aligning itself in that lane. By this time, the agent successfully avoided the blocking vehicle, and the concept instance, *(vehicle-ahead me c6120)*, is no longer true. So the goal, *(clear me c6120)*, that was triggered by this concept instance disappears.

Scenario 2: Ambulance

Now, to make the task more complicated, let us think about driving an emergency vehicle, say, an ambulance. We sometimes see that an ambulance is moving quite normally, waiting for pedestrians to pass, observing the speed limit, and even stopping for red lights, although it has its lights and siren on. Yet some other times we see an ambulance speeding by almost like one driven by a reckless driver, blinking every single light it has equipped on and making a very loud sound. We can guess that the difference is on the severity of the problem at their destinations, or onboard, and this factor affects the behavior of the drivers.

Modeling this behavior in the previous version of ICARUS is close to impossible, unless the programmer is patient enough to write concepts and skills for all possible cases there are. Even then, the space of search will be so large that the performance will be below what is required during the execution. However, the extended system supports this behavior easily, with some generalized goals encoded in its long-term memory, coupled with their corresponding triggers. Table 5 shows the new concepts that we added for this scenario.

Table 5: 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))))
```

To handle the task to get to the hospital with the proper urgency based on the current situation, we encode the goal, *(okay-to-go ME ?signal)* with priority 2, to have nomination conditions, *(signal-ahead me ?signal)* and *(not-emergency me)*. This goal is what forces the agent to observe traffic signals when there is no emergency. But when the emergency strikes and the degree of match for the concept *(emergency me)* starts to increase from zero, that for the concept *(non-emergency me)* starts to decrease from one accordingly. When this happens, the relevance of the above goal drops with them, eventually making the architecture focus on the other goal of getting to the hospital first.

Now we will show how the system behaves during a typical run. In a similar fashion 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 clear the car. With occasional accelerations to maintain its speed, it continues steering to the right. On cycle 13, it notices that it is in the target lane, and starts to cruise there. But it soon finds another car, and clear it in a similar manner, but this time to the left lane, and finishes the move by cycle 21. The agent then sees a traffic signal that is red, and brakes to stop. During the wait, its 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, gets its degree of match at 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 0.4 ($= 2 \times 0.2$). This causes the goal to be less important than the default goal, *(cruising-in-lane me ?line1 ?line2)* that has the priority of 1.

Therefore, the system now stops observing traffic signals, and starts cruising even with the red traffic light. Later on cycle 95 when it reaches the next intersection, however, the emergency level is back to 3, and the modulated priority value for *(not-emergency me)* becomes 0.7. This once again puts the goal to observe traffic signals before the default goal of cruising, and the system starts observing signals again.

The two programs shown above, one for the original architecture and the other for the extended architecture, both result in equivalent behaviors at the high level. However, the two systems still have differences at lower level for basic driving maneuvers, and the extended system shows much smoother driving behavior. What is important to note in this scenario is that the goal nomination capability leads to a much simpler program that is more intuitive and reasonable to us.

Related and Future Work

Our work has been heavily influenced by related work in the psychology literature. One can find a fair amount of research related to motivation and goal selection there. Typically, these also cover the topic of emotion. Simon (1967) recognized that the central nervous system, despite being a serial information processor, serves multiple needs in an organism surrounded by unpredictable situations. He suggested that two mechanisms, a goal-terminating mechanism and an interruption mechanism, would satisfy this requirements. Simon further described the relationship among interruption, motivation, and emotion, and outlined an information-processing system that covers these as wells as learning in relation to them. More recently, Sloman (1987; 2002) suggested that any system with priority in beliefs and actions naturally have emotions. He argued that goals often conflict with each other, and systems must have a mechanism to resolve such conflicts. The author proposed that motivators can serve this purpose.

As mentioned earlier in this paper, there are also some related work in the architectural perspective. CLARION (Sun, 2007) and Soar (Laird et al., 1986) architectures possess their own accounts of goal management. The former is more psychologically positioned, providing interactions between drives and goals. The latter has a rule-based mechanism to nominate its top-level operators as its goals, which resembles the conditionalized goals ICARUS has. Unlike ICARUS, however, the Soar architecture proposes a single goal at a time, removing the need for prioritization or the advantage of interactions among multiple goals.

Although the current work is an important first step toward a cognitive architecture with the full capability for goal management, it still ignores a vast amount of psychological accounts on human motivation and goal handling. First of all, people can change priorities among different goal in a flexible manner, depending on the current situation. We have a way to model this behavior, and hope to report in this direction in a near future. More broadly, we should explain where the long-term knowledge about goals comes from. It is very likely that we will deal with even higher-level cognitions like motivations, emotion, and obligations. We expect the the evidences in the social psychology literature will help us in the modeling process.

Conclusions

In this paper, we introduced an extension to the ICARUS architecture for reactive goal management. We first conceived the idea in the architectural perspective, but the extension makes close connections to previous work in psychology and other related fields. The extended framework supports the nomination, retraction, and prioritization of goals based on the current belief state. We have demonstrated in an urban driving domain that the extension leads to simpler programs while supporting new behaviors that connects to the context better than the original architecture.

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.

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

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

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.

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

Simon, H. A. (1967). Motivational and emotional controls of cognition. *Psychological Review*, *74*(1), 29–39.

Sloman, A. (1987). Motives, mechanisms, and emotions. *Cognition & Emotion*, *1*(3), 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.* New York, NY: Oxford University Press.