Adding Goal Priorities to Teleoreactive Logic Programs

Nan Li[1] (nan.li.3@asu.edu)
Dongkyu Choi[2] (dongkyuc@stanford.edu)
Pat Langley[1,2] (langley@csli.stanford.edu)
[1] School of Computing and Informatics, Arizona State University, Tempe, AZ, USA
[2] Computational Learning Laboratory, CSLI, Stanford University, Stanford, CA, USA

Abstract

When people have more than a single goal, they usually start with the most important one and continue on to the next. Artificial agents can use the same scheme, letting them work on multiple goals in the order of relative importance. Previously, we introduced the notion of a *teleoreactive logic program* that provides intelligent agents with reactive but goal-directed behavior. However, this formalism does not support multiple goals effectively, especially when they have different priorities. In this paper, we report a new framework, *prioritized teleoreactive logic programs*, that extends the previous formalism and that handles multiple goals in a modular fashion. We illustrate this approach in the domain of urban driving and we report experimental evidence of its advantages therein. We also consider the implications of prioritized goals for skill learning, along with related work on multi-goal processing.

Keywords: teleoreactive control, multiple goals, goal priorities

Introduction

Previous work in cognitive science suggests that people carry out complex activities in a goal-directed manner by using skills that are organized hierarchically. To model this ability, we introduced a representational formalism, *teleoreactive logic programs*,[1] that encodes complex skills in a hierarchical manner (Langley & Choi, 2006). These programs index skills by the goals they achieve, and they let agents produce goal-directed behavior while remaining reactive to their current surroundings.

When people have multiple goals, they typically set priorities and try to achieve them one by one. Similarly, artificial agents can prioritize their goals and work on each of them in sequence. However, since goals can interact with each other, some achieved objectives may become unsatisfied at a later time. For this reason, an agent must have some way to maintain the satisfied goals, either by avoiding actions that will undo them or by returning to more important goals whenever they become untrue. However, the actions that violate the satisfied goals may be carried out not only by the agent itself, but also by other agents in the world, which makes it difficult to take the first approach. Therefore, we have focused on methods for switching the agent's attention back to higher priority goals when they no longer hold. To this end, we have developed an extended formalism – *prioritized teleoreactive logic programs* – that supports processing of multiple top-level goals.

In the following sections, we review the basic assumptions and features of teleoreactive logic programs, along with a physical domain that we use to illustrate and evaluate the framework. In addition, we describe a mechanism that lets agents handle multiple goals with different priorities, and we report experimental evidence for its advantages over the original formalism. We also discuss some implications for learning in the extended framework. We conclude with a short review of related research on agent architectures and some suggestions for future work.

A Review of Teleoreactive Logic Programs

Before presenting our extension to teleoreactive logic programs, we should first review our previous work on this topic. We begin by describing the basic formalism, after which we consider the inference and execution mechanisms that operate over structures encoded in this framework. We have embedded these representational and processing assumptions in ICARUS, a cognitive architecture for physical agents that supports goal-driven but reactive behavior. We will occasionally use ICARUS and teleoreactive logic programs as synonyms, although other interpreters for this formalism are possible.

Representation of Knowledge

A teleoreactive logic program assumes both long-term and short-term memories. Long-term structures reside in a skill memory, which contains methods for achieving goals, and a conceptual memory, which specifies abstractions of situations the agent may encounter. Short-term stores include a belief memory, which contains instantiated concepts that match against the current situation, and a goal memory, which includes objectives the agent wants to achieve.

The skill memory is organized hierarchically. Each skill clause provides a method the agent can execute in the world to achieve a desired situation. Such a clause consists of a head, which is a defined concept that specifies the desired situation, a set of start conditions that must hold to initiate the skill, requirements that must hold during its execution, and

[1] We have borrowed the term *teleoreactive* from Nilsson (1994), who coined it to indicate systems that are goal directed but that also respond to their environment.

Table 1: Examples of skills for the urban driving domain.

```
((at-desired-speed ?self)
 :percepts ((self ?self))
 :start    ((stopped ?self))
 :actions  ((*gas 6.25)))

((at-address ?self ?ad ?st)
 :percepts ((self ?self) (street ?st))
 :start    ((address-behind ?self ?ad ?st))
 :subgoals ((address-in-front ?self ?ad ?st)
            (at-address ?self ?ad ?st)))
```

Table 2: Examples of concepts for the urban driving domain.

```
((at-desired-speed ?self)
 :percepts  ((self ?self speed ?speed
                       limit ?limit))
 :tests     ((<= ?speed (- ?limit 5))
             (> ?speed (- ?limit 15))))

((at-street ?self ?street)
 :percepts  ((self ?self) (segment ?sg))
 :relations ((in-segment ?self ?sg)
             (segment-at-street ?sg ?street)))
```

steps that should make the head true. Table 1 presents some sample skills, which we can divide into two groups.

Primitive skills have an action field that refers to actions the agent can execute directly in the environment. For example, the skill *(at-desired-speed ?self)* in Table 1 is a primitive skill which states that, when the agent's vehicle is stopped, the agent should invoke the action *(*gas 6.25)* to ensure the vehicle will eventually achieve the desired speed.

Nonprimitive skills have similar structure, but specify subgoals the agent should achieve instead of actions it should execute. For example, *(at-address ?self ?ad ?st)* in Table 1 is a nonprimitive skill that takes the agent to a specified address by decomposing the task into two subgoals, *(address-in-front ?self ?ad ?st)* and *(at-address ?self ?ad ?st)*. This means that, if the target address is behind the car, it should first change the heading to face the address and then recursively achieve the original goal of reaching the address.

The conceptual memory characterizes world situations at different levels of abstraction, as shown in Table 2. Each conceptual clause has a head, which consists of a predicate with arguments, and a body, which defines the class of situations that match the concept. Every predicate that occurs in skill memory has a corresponding predicate in conceptual memory. There are two forms of concepts, just as with skills.

Primitive concepts have a test field in their bodies that refers only to percepts that agent can observe directly in the environment, while nonprimitive concepts have a relation field that refers to other concepts. Table 2 shows examples of concepts from the driving domain. The primitive concept *(at-desired-speed ?self)* specifies that, if the vehicle's speed is neither too slow nor too fast, *(at-desired-speed ?self)* is true. In contrast, the nonprimitive concept *(at-street ?self ?street)* refers to two lower-level concepts, *(in-segment ?self ?segment)* and *(segment-at-street ?sg ?street)*. This clause indicates that, if the vehicle is in a certain segment and the segment is part of a given street, then the vehicle is on that street.

Belief memory is a short-term store that contains instances of concepts that the current situation satisfies. Each belief includes a predicate defined in conceptual memory and zero or more arguments that refer to objects in the environment. For instance, if the speed of vehicle *veh* is under the speed limit, thus satisfying the concept *(at-desired-speed ?self)*, then belief memory will include the structure *(at-desired-speed veh)*. However, if the vehicle exceeds the speed limit later, *(at-desired-speed veh)* will no longer be present, even though the concept *at-desired-speed* remains in long-term memory.

Goals reside in another short-term memory. A goal is some concept that the agent wants to satisfy. Like beliefs, each goal consists of a defined predicate and its arguments, but in this case some arguments may be pattern-match variables rather than constants. For instance, the goal *(at-desired-speed veh)* means the agent wants to drive its vehicle *veh* below the speed limit, whereas the goal *(at-desired-speed ?any)* means it wants every vehicle to obey the speed limit.[2]

Taken together, the contents of these memories provide an agent with the information needed to produce goal-directed yet reactive behavior. We assume that skills and concepts change only gradually over time, as the result of learning, whereas beliefs change more rapidly, as the environment changes. We typically include skills and concepts when we refer to a teleoreactive logic program, but not beliefs. Goals are somewhat more ambiguous, but we will return to this issue later in the paper.

Inference and Execution Mechanisms

As noted earlier, the ICARUS architecture (Langley, 2006) provides an interpreter for teleoreactive logic programs. This operates in discrete cognitive cycles, as Figure 1 depicts. On each cycle, the system first carries out an inference process that infers beliefs from low-level percepts in the environment and its conceptual knowledge. The inferred concept instances describe the agent's belief state, which it builds on in later decision-making processes.

The inference module starts by extracting information about objects the agent perceives in its immediate surroundings, as shown in Table 3. After this, it initiates a bottom-up, data-driven process that matches the resulting percepts against concept definitions stored in long-term memory. The system instantiates the satisfied concepts based on the current environmental state by replacing variables with constants and stores them in belief memory. ICARUS begins by matching percepts against primitive concepts and adds instances of satisfied concepts to memory. These trigger matching of higher-level concepts, which produces additional beliefs. The entire process halts when the system can draw no more inferences.

For example, given the percepts in Table 3, the system infers a variety of beliefs from primitive concepts, such as that the vehicle is in segment S2724 and segment S2724 is on

[2] Basic teleoreactive logic programs assume the agent has a single top-level goal, but one can augment them to include a goal stack with subgoals that supports means-ends problem solving, which we discuss later.

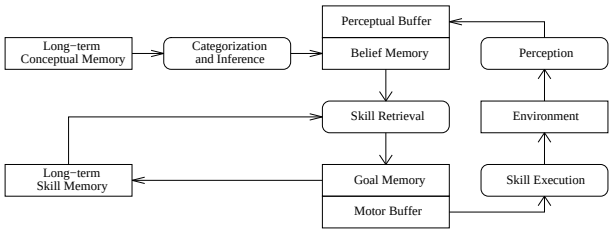

Figure 1: Execution of teleoreactive logic programs.

street B. After this, the interpreter further infers that the vehicle is on street B, which depends on a higher-level concept. The result is a rich description of the environment at different levels of abstraction, which the system can use to drive its execution of skills. The inference process repeats on every cycle, thus ensuring that the agent's beliefs reflect the current state of its surroundings.

Given an inferred set of beliefs, ICARUS' execution module carries out a goal-directed evaluation of the clauses in skill memory. On the first cycle, if the top-level goal is unsatisfied, the system attempts to find an executable path through the skill hierarchy. A skill path starts with a skill that achieves the current goal. If this skill is a primitive one, it will be the only skill on the path. Otherwise, the path will have a skill that achieves the first unsatisfied subgoal as its next element. The path continues downward through the hierarchy, terminating with a primitive skill that describes what actions to perform in the world. A skill path is *executable* when the start conditions of every skill on the path are satisfied. Once ICARUS has selected such a path, it applies the actions listed in the primitive skill in the environment.

On subsequent cycles, the interpreter prefers the path that it executed on the previous cycle. Instead of finding a new path from scratch, it takes the previous path and starts evaluating from the highest-level skill. If the skill is still executable in the updated state, then it follows the path to the next skill. When it encounters a skill instance on the path that is no longer executable, the system finds another executable skill that achieves the subgoal, thus diverging from the previous path. This control strategy lets the system remain reactive to its surroundings while generating persistent and goal-directed behavior.

An Illustrative Domain

Our examples of teleoreactive logic programs have drawn on an illustrative domain, a simulated urban driving environment (Choi et al., 2007) that we have used in our prior work on this topic. This has proven to be a rich and challenging testbed that involves complex goal-directed activities. The environment is inherently dynamic and involves many objects to which the agent must react based on limited perceptions. The agent must also follow certain rules, such as driving on the right side of the road, staying under the speed limit, and not hitting pedestrians. It must also decide how to respond when these constraints conflict with each other, such as when collision with another vehicle interferes with the lower-priority goal of delivering a package.

Figure 2: A screen shot of the urban driving domain.

Figure 2 presents a screenshot of the simulated driving environment. This includes both static objects, such as road segments, intersections, lane lines, and buildings, and dynamic objects, such as pedestrians and vehicles. Every road is divided into multiple segments, each of which includes five lane lines, a yellow one in the middle, two white ones on both sides, and two sidewalks at the street boundary. Buildings sit on both sides of the street and have address numbers. A typical package-delivery task involves taking a package to a specified street address. Pedestrians walk across the road at unpredictable intervals, and other simulator-controlled vehicles drive through the city, obeying the traffic laws. The agent also controls a vehicle, which it must use to achieve its goals.

The driving agent can perceive objects in the environment, but only ones in its immediate surroundings. Table 3 shows some examples of perceived objects, with distances and angles in agent-centered polar coordinates. These include two streets named SECOND and B, along with their intersection, S2708, which is 88.7815 meters from the car. Segment S2724 in street B is 88.986 meters from the agent's car, while a lane line, sidewalk1, is 30.503 meters away. The number 5.27616 indicates the angle between the car's heading and the lane line's direction. A building, B2677, has address 2 on street B, and the two of its corners nearest to the car have distances 42.4071 and 54.092 meters, respectively. The agent also perceives the angles and distances of pedestrians around it.

The environment also provides the agent with actions it can invoke to control the vehicle, as shown in Table 4. In particular, the agent can push the gas pedal to speed up and step on the brake to slow down. The agent can also control the speed of this acceleration and deceleration, and it can control the vehicle's direction by turning the steering wheel to different angles. When combined with the environment's complex surroundings, these functions provide enough control to support a rich set of driving activities.

Table 3: Sample object information that the agent perceives in the driving environment.

```
((street SECOND) (street B)
 (segment S2724 street B dist 88.986)
 (building B2677 address 2 street B
   c1angle -49.5323 c1dist 42.4071
   c2angle -34.5953 c2dist 54.092)
 (intersection S2708 street SECOND
   cross B dist 88.7815)
 (lane-line SIDEWALK1 color SIDEWALK
   dist -30.503 angle 5.27616
   segment S2724)
 (man M31 angle 16.5915 dist 87.035)
 (self ME speed 20.0 heading -5.60023
   wheel-angle 0 throttle 0 limit 25
   breakthrottle 100 segment S2724)
 (package PACKET1 address 5 street A
  delivered NIL))
```

Adding Goal Priorities

In previous work (Langley, 2006), we have reported a teleoreactive logic program that controls an urban driving agent. The system drives around the city and performs various tasks like package delivery. However, our experience with this environment also revealed a key limitation of teleoreactive logic programs. Because the domain includes a number of distinct constraints and driving rules, we were forced to reproduce these constraints repeatedly throughout the skill hierarchy to ensure the agent always obeyed them in its activities.

For example, during the process of delivering the packages, the driving agent must operate within constraints such as staying under the speed limit and staying within the lanes. In a basic teleoreactive logic program, this means that we must include each constraint as a requirement of skills that control high-level activities. For example, to make sure that the agent always drives under the speed limit, every skill must handle both the situations in which *(at-desired-speed ME)* holds and those in which it does not hold, as shown in Table 5. Moreover, the complexity of the program will continue to grow as we add new constraints on the agent's behavior.

Furthermore, the basic formalism cannot deal easily with the fact that some constraints, such as not hitting pedestrians, are more important than others, such as driving at the desired speed. When the vehicle is heading toward a pedestrian, the appropriate response is to avoid running him down by slowing down the car quickly, even if this means failing to satisfy *(at-desired-speed ME)*. In such cases, the skills in Table 5 are insufficient, since they maintain the same priority for the two driving laws. The skill hierarchy needs further modification to make sure the agent maintains its speed only when it is not about to hit a pedestrian. This makes the encoding even more complicated, as Table 6 indicates.

This is a general phenomenon we observed in programming agents, which motivated us to extend the original framework to address its limitations. In response, we developed a new formalism, *prioritized teleoreactive logic programs*, that differs only in that it maintains a list of top-level goals,

Table 4: Actions available in the driving environment.

`(*cruise) :`	`NO OP`
`(*gas pedal) :`	`push the gas pedal at the given amount`
`(*brake pedal) :`	`push the brake pedal at the given amount`
`(*straighten) :`	`straighten the steering wheel`
`(*steer angle) :`	`turn the steering wheel to the given angle`

such as *(not-hitting-pedestrians ME)*, *(at-desired-speed ME)*, and *(package-delivered Package1)*. On every cycle, the interpreter selects the first unsatisfied goal from this list and works toward this objective until satisfying it. Note that this mechanism does not forget about a goal once it has been achieved, but rechecks its status at each time step. During this process, if some higher-priority goal becomes unsatisfied, the agent suspends work on its current goal and switch to the high-priority one. After it has reachieved this goal, the agent resume the work on the suspended objective.

This mechanism ensures that the agent considers a goal only when all more important goals are satisfied, so that they always hold during execution of low-priority tasks. For instance, suppose that the current goal is *(package-delivered Package1)* and all higher-priority goals, including *(not-hitting-pedestrians ME)* and *(at-desired-speed ME)*, are satisfied. In this situation, the agent will attempt to deliver Package1 to its destination. However, suppose that, after several cycles, the agent detects that it is no longer under the speed limit. Since *(at-desired-speed ME)* has a higher priority than *(package-delivered Package1)*, the agent will stop delivering the package and will adjust its speed until it slows down to the speed limit. Afterward, the agent will return to its delivery task.

Some readers may question why we view the prioritized goal list as part of the teleoreactive logic program, arguing that, since goals are ephemeral, we should treat them as separate. However, certain goals encode constraints that the agent must continuously obey, making them a form of long-term knowledge. By incorporating these constraints into the prioritized goal list, we effectively encode them as knowledge about the domain. From this point of view, the new formalism supports a more efficient way to state such knowledge than the original scheme, making it reasonable to view the goal list as part of the program.

With this extension, the introduction of new driving laws requires only a small number of additional skills to produce the desired behavior, provided that the goal list encodes their priorities. This involves only incremental modification, with no changes to the skills in the original knowledge base. Therefore, if we change the constraints later, we need only modify the newly added skills and the goal list. In summary, the program's modularity makes it easy to maintain over time, as requirements develop.

Table 5: Skills for the driving domain with one law in original formalism.

```
((at-address ?self ?ad ?st)
 :percepts ((self ?self) (street ?st))
 :start    ((address-behind ?self ?ad ?st))
 :requires ((at-desired-speed ?self))
 :subgoals ((address-in-front ?self ?ad ?st)
            (at-address ?self ?ad ?st)))

((at-address ?self ?ad ?st)
 :percepts ((self ?self) (street ?st))
 :start    ((not (at-desired-speed ?self))
            (address-behind ?self ?ad ?st))
 :subgoals ((at-desired-speed ?self)
            (at-address ?self ?ad ?st)))
```

Table 6: Skills for the driving domain with two ordered laws in the original formalism.

```
((at-address ?self?ad ?st)
 :percepts ((self ?self) (street ?st))
 :start    ((address-behind ?self ?ad ?st))
 :requires ((not-hitting-pedestrians ?self)
            (at-desired-speed ?self))
 :subgoals ((address-in-front ?self ?ad ?st)
            (at-address ?self ?ad ?st)))

((at-address ?self ?ad ?st)
 :percepts ((self ?self) (street ?st))
 :start    ((not (not-hitting-pedestrians ?self))
            (address-behind ?self ?ad ?st))
 :subgoals ((not-hitting-pedestrians ?self)
            (at-address ?self ?ad ?st)))

((at-address ?self ?ad ?st)
 :percepts ((self ?self) (street ?st))
 :start    ((not (at-desired-speed ?self))
            (address-behind ?self ?ad ?st))
 :require  ((not-hitting-pedestrians ?self))
 :subgoals ((at-desired-speed ?self)
            (at-address ?self ?ad ?st)))
```

Empirical Benefits

As just discussed, because our original formalism was inefficient at handling certain constraints, such as driving laws, we have proposed an extended framework, prioritized teleoreactive logic programs, that should support greater modularity. This approach embeds information about priority in a list of top-level goals and extends the execution mechanism to support their pursuit. This should produce a simplified skill hierarchy that is easier for the developer to modify when domain constraints change.

However, these are empirical claims that we must support with evidence. To evaluate the benefits of the extended formalism, we designed experiments that compared the two frameworks' abilities to encode agent behavior. First, to test the hypothesis that the new formalism maintains a more compact and modular knowledge base than the original one, we varied the number of driving laws that the agent must obey and measured the number of skills required to show equivalent behaviors in the two notations.

Table 7 shows the results from this study. With no driving laws, the agent needed 54 skills in both formalisms. These skills encode basic driving behavior, such as accelerating, decelerating, and steering to align the vehicle with lane lines. However, when we add a constraint related to *at-desired-speed*, the number of skills doubled in the original formalism, while we needed a mere three additional skills in the extended one. Adding a second law related to *not-hitting* pedestrians required another 55 skills in the old framework but only one new skill in the expanded one.

Second, to test the hypothesis that the extended formalism is easier to modify when domain constraints are changed, we encoded three driving laws in both the original formalism and the extended one, and then changed some of them. For example, suppose that we were no longer interested in ensuring that the agent always drives below the speed limit, and instead wanted to model an aggressive driver that always exceeds it. In this case, we might need to delete some skills and add others. Thus, we varied the number of laws changed in the domain and recorded the number of skills that we needed to revise in both frameworks.

Table 8 presents the results from this experiment, which reveal that there was no need to modify the skills in the extended formalism. To produce an agent that obeys the new laws, we only needed to change the top-level goals. However, the original formalism required that we delete 54 skills and add the same number of skills when we replaced one driving constraint from *at-desired-speed* to *driving-as-fast-as-possible*. Moreover, we had to revise twice this number of skills when we changed a second constraint on the agent's driving behavior. The number of altered skills appears to grow proportionally with the number of changed driving laws in the original framework, but does not grow in the extended one.

These results support our claims about the benefits of prioritized goal lists. Basic teleoreactive logic programs require around 50 new skills to ensure compliance with a new constraint, but the extended formalism allows equivalent behavior with only a handful of additional skills. The old framework needs over 50 skill revisions to satisfy each modified constraint, whereas the new one requires only minor changes to the skills and goal list.

Implications for Learning

As described elsewhere (Langley & Choi, 2006), we have also demonstrated methods for learning teleoreactive logic programs from experience. In this section, we review this earlier work and then discuss ways to extend the approach to support prioritized goal lists.

Problem Solving and Skill Learning

Our previous work has assumed that skill learning is a side effect of problem solving, which occurs whenever the system hits an impasse during execution, in that it cannot find any applicable path through the skill hierarchy that addresses the

Table 7: Number of skills required to encode different numbers of driving laws in the original and extended formalisms.

Driving Laws	Teleoreactive Logic Programs	Prioritized Teleoreactive Logic Programs
0	54	54
1	111	57
2	166	58

Table 8: Number of skills modified for different numbers of driving laws in the original and extended formalism.

Changed Laws	Teleoreactive Logic Programs	Prioritized Teleoreactive Logic Programs
0	0	0
1	54	0
2	108	0

current goal. In such cases, a means-ends problem solver decomposes the current goal by chaining backwards off either conceptual clauses or primitive skills, with a priority on the latter alternative.

In skill chaining, the system retrieves a skill clause that contains the goal in its effect, whereas concept chaining uses the concept definition to decompose the goal into multiple subgoals based on subconcepts. The system pushes the selected subgoal onto a goal stack and records details about its decisions for future use. This process continues recursively until it finds a subgoal for which there is an executable method in the current situation or until it must backtrack.

Once the system finds an applicable skill path that will achieve the current subgoal, it enacts the associated actions in the environment, thus interleaving problem solving with execution. Although this may put the agent in a situation where it cannot backtrack in the world, it allows incremental learning of new skills upon achieving each subgoal. We can explain this process with an example from the Blocks World, which we use here for its clarity; however, we have obtained similar results in the urban driving domain.

Figure 3 shows a trace of problem solving in the Blocks World. The agent is given the goal of clearing block A, but it cannot find any applicable skill path in the current state. In response to this impasse, it invokes problem solving, which chains off the goal using the instantiated skill *(unstack B A)*. This has the start condition *(unstackable B A)*, which the system pushes onto the goal stack. On the next cycle, the agent uses concept chaining to decompose this subgoal into *(clear B)* and *(hand-empty)*, and the system chooses the currently unsatisfied *(clear B)* as its next subgoal.

Since there exists an executable skill, *(unstack C B)*, that would achieve this subgoal, the system executes it. This produces a state in which *(clear B)* is true, but in which *(hand-empty)* has become untrue. Therefore, the agent pushes this subgoal onto its stack and retrieves *(putdown C T)*, which it executes in turn. This makes *(unstackable B A)* true, which means the agent can finally execute *(unstack B A)* to achieve the top-level goal *(clear A)*. Although this example is simple enough to let the system find the solution without much effort, the problem solver must often carry out search before succeeding on a task.

Throughout this process, whenever the system achieves a goal or subgoal, it creates a new skill based the information stored in the goal stack. Thus, learning is fully integrated with both problem solving and execution, producing the incremental addition of skills as the agent gains experience in the domain. The head of the new skill is a generalized version of the goal in which constants are replaced by variables, whereas the body depends on whether the system produced the subgoal through skill chaining or concept chaining.

If the agent achieved the goal through skill chaining, the two subgoals of the new skill are the heads of the skill used to achieve the start condition of the chained skill and the head of the chained skill, in the same order as their execution. The start condition is simply the start condition of the first subgoal. Thus, if the agent first applied skill S_1 to satisfy the start condition of skill S_2 and then applied S_2 to achieve the goal, the subgoals of the new skill are S_1 and S_2, in this order, and the start condition of the new skill is the start condition of S_1. The effect of the new skill is the same as the effect of the chained skill, in this case S_2.

On the other hand, if the agent achieved the goal through concept chaining, the subgoals of the new skill are subconcepts that were initially unsatisfied, in the order the system achieved them. For example, suppose that the subconcepts of the goal are S_1, S_2, and S_3, that S_1 held in the initial situation, and that the agent first achieved S_2 and then S_3. In this case, the subgoals of the new skill clause would be S_2 and S_3. The start condition consists of those subconcepts that held initially, in this case S_1.

On subsequent problems, the system will take advantage of these learned skills whenever they apply to the current situation. These let the agent achieve goals that require many steps purely through skill execution, rather than by resorting to deliberative problem solving. Experiments in the Blocks World, urban driving, and other domains indicate that such learning improves the agent's performance substantially.

Learning with Multiple Goals

The learning mechanism described above focuses on a single top-level goal, but clearly the introduction of a prioritized goal list offers opportunities for other forms of skill acquisition. Here we discuss a some proposals along these lines.

One idea involves the gradual propagation of recurring constraints into the bodies of skills. Such compilation of explicit goals into implicit procedures happens regularly in humans. For instance, when a person first learns to drive, he must keep the driving rules in memory and constantly focus on following them. As the driver gains experience, obeying these constraints becomes automated and unconscious, thus freeing attention for other activities. However, note that such compilation also has negative impacts, in that it makes the agent less adaptive; a skilled driver has difficulty when placed in a setting that involves different driving laws. Nevertheless, modeling this compilation would produce more human-like

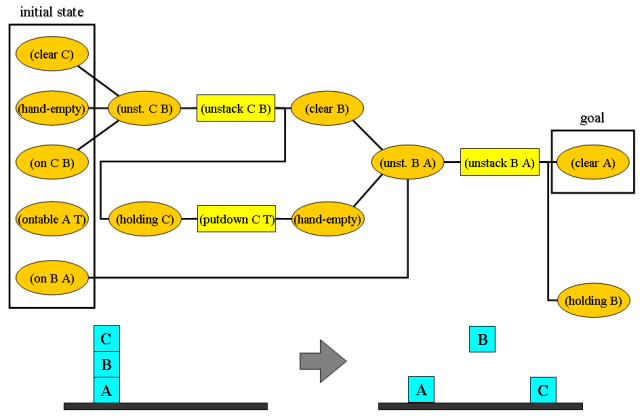

Figure 3: A problem-solving example in the Blocks World. Conceptual clauses are shown in circles, while skill clauses are in rectangles.

behavior at the expense of flexibility and adaptivity.

A mechanism that may lead to such compilations concerns learning from failure. Consider a situation in which the agent is pursuing a low-priority goal, but in the process takes an action that violates a higher-priority one. Upon analyzing the cause of this unexpected event and identifying alternative actions that would have avoided it, the system would incorporate these into new, more specific skills that have precedence over the original ones. These would effectively incorporate the high-priority goal in procedural form, letting the agent drop it as an explicit objective once it has acquired enough such specialized skills.

Another idea is that, when an agent pursues multiple goals, it should also learn skills that take into account interactions among them. This should lead to more efficient execution, in that it can avoid redundant efforts and violations of previously achieved goals. The system would store the acquired skills under a new goal that is the conjunction of more basic constraints; these procedures should take relative priorities into account in an attempt to keep the different constraints satisfied as often as possible. We hypothesize that analytic learning from failure can also contribute to the construction of such integrated skills.

In summary, the introduction of a prioritized goal list suggests a number of extensions to learning teleoreactive logic programs. Together, these should let the framework more closely reflect the nature of human skill acquisition and support more automatized behavior that plays an important role in complex domains like urban driving.

Related Research

The formalism we have reported in this paper shares some basic principles with previous work. The idea that skills are organized hierarchically goes back to the earliest days of artificial intelligence and cognitive science (Miller et al., 1960). Recently, research on hierarchical task networks (Nau et al., 1999) has built upon this tradition. Our formalism encodes skills in a similar format, but it indexes skills by the goals they achieve. Both frameworks can represent more complex goal-directed activities than purely reactive approaches like those used in behavioral cloning (Sammut, 1996).

There has been some other work on handling multiple goals in agent architectures, including multi-tasking mechanisms that deal with prioritized goals by assigning resources properly. For example, Freed's (1998) APEX manages multiple tasks in complex, dynamic environments under time pressure. His architecture calculates its own task priorities based on resource conflicts, and thus takes a more sophisticated approach than the one we have reported.

More recently, Salvucci and Taatgen (in press) have attempted to unify various multi-tasking models in ACT-R. Their approach, which they call *threaded cognition*, posits that goals correspond to threads of processing across a set of resources, and they report an algorithm that acquires and releases these resources. Our approach differs by focusing on priorities among constraints that are unrelated to resources, such as giving some driving laws precedence over others.

Haigh and Veloso's (1996) Rogue also generates and executes plans that address multiple goals. Their system uses goal-selection rules to prioritize goals that may arrive asynchronous and to pick one goal from among those pending ones. This framework treats all objectives as achievement goals and does not check their status after achieving them. In contrast, our approach handles both short-term goals like package delivery and long-term goals like obeying traffic laws.

Directions for Future Work

Despite the promise of prioritized teleoreactive logic programs, our research on this topic remains in its early stages. We have already discussed the need to integrate the new approach with learning, but there remain a number of other open issues that we should address in our future work.

First, we plan to demonstrate the benefits of the extended formalism in domains that involve even more complex skill hierarchies. As before, we will build agent programs in both frameworks, introduce additional constraints to the task, and then measure the sizes of the skill hierarchies. We expect these studies to replicate our initial results, but further evidence will strengthen our claims about the advantages of prioritized goal lists for modularity.

Second, we will explore mechanisms that adjust goal priorities in response to learning. Elsewhere (Asgharbeygi et al., 2006), we have reported a method for assigning expected values to concepts in response to delayed rewards. If we interpret these estimated values as priorities, then we can use them to order the agent's top-level goals and revise that order in the light of its experience.

Finally, we plan to augment the extended system with resource management capabilities that involve the parallel execution of skills. When the agent notes that multiple top-level goals are unsatisfied at the same time, it would attempt to retrieve skill paths that draw upon different resources, so that it can execute them in parallel. For example, in attempting to avoid a collision, the agent should be able to turn the steering wheel and step on the brake simultaneously. This should improve the system's ability to achieve its goals in complex and dynamic environments.

Concluding Remarks

In summary, our previous work on teleoreactive logic programs has shown that it offers a promising framework for building intelligent agents, but we also found that it does not scale well to tasks that involve many constraints. The domain of urban driving, which requires the agent to obey a number of laws, illustrates that such settings can lead to quite complex programs for controlling agent behavior.

In response, we have developed an extended formalism, *prioritized teleoreactive logic programs*, that incorporates an ordered list of top-level goals. We demonstrated the benefits of this approach in an urban driving domain, showing experimentally that the complexity of its programs increases far more slowly with the number of driving laws than in the original formalism. Nevertheless, we must still extend the framework along a number of dimensions, including its integration with learning, to achieve its full potential.

Acknowledgements

This paper reports research sponsored by DARPA under agreement FA8750-05-2-0283. The U. S. Government may reproduce and distribute reprints for Governmental purposes notwithstanding any copyrights. The authors' views and conclusions should not be interpreted as representing official policies or endorsements, expressed or implied, of DARPA or the Government.

We would like to thank Michael Morgan for his development of the urban driving environment and GarageGames (http://www.garagegames.com/) for allowing free use of its products in that effort, including TorqueGame Engine 1.4.2 for Linux, BTCP Car Pack, and Urban Pack 1.0.0.

References

Asgharbeygi, N., Langley, P., & Stracuzzi, D. (2006). Relational temporal difference learning. *Proceedings of the Twenty-Third International Conference on Machine Learning* (pp. 49–56). Pittsburgh, PA.

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

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

Freed, M. (1998). Managing multiple tasks in complex, dynamic environments. *Proceedings of the National Conference on Artificial Intelligence* (pp. 921–927).

Haigh, K., & Veloso, M. (1996). Interleaving planning and robot execution for asynchronous user requests. *Proceedings of the International Conference on Intelligent Robots and Systems* (pp. 148–155). Osaka, Japan: IEEE Press.

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

Langley, P., & Choi, D. (2006). Learning recursive control programs from problem solving. *Journal of Machine Learning Research*, *7*, 493–518.

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

Nau, D., Cao, Y., Lotem, A., & Munoz-Avila, H. (1999). SHOP: A simple hierarchical ordered planner. *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence* (pp. 968–973).

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

Salvucci, D. D., & Taatgen, N. A. (in press). Threaded cognition: An integrated theory of concurrent multitasking. *Psychological Review*.

Sammut, C. (1996). Automatic construction of reactive control systems using symbolic machine learning. *Knowledge Engineering Review, 11*, 27–42.