Implementing a Rule-based System to Represent Decision Criteria for On-Road Autonomous Navigation

N. Zimmerman, C. Schlenoff and S. Balakirsky
Intelligent Systems Division

National Institute of Standards and Technology (NIST)
Gaithersburg, MD 20899-8230.
Tel: (301) 975-8554 Fax: (301) 990-9688
Email: {noah.zimmerman, craig.schlenoff, stephen.balakirsky}@nist.gov

Abstract

The purpose of this paper is to explore an implementation of a rule-based system for generating costs for an on-road autonomous vehicle using the C Language Production System (CLIPS). In this context, costs are numeric values that represent a penalty the vehicle incurs by taking a certain action or occupying a state.

Introduction

For the purpose of this paper, we define an autonomous vehicle as an embodied intelligent vehicular system that can operate for extended periods of time without human supervision. As part of an effort to teach the vehicle to behave in an intelligent manner, the Intelligent Systems Division, a part of the National Institute of Standards and Technology (NIST), is applying the 4D/RCS (Albus & et.al. 2002) architecture to serve as the underlying reference model architecture to control the autonomous vehicle. In (Albus & et.al. 2002) (p.2), 4D/RCS is described as follows:

> *"The 4D/RCS architecture provides a reference model for military unmanned vehicles on how their software components should be identified and organized. It defines ways of interacting to ensure that missions, especially those involving unknown or hostile environments, can be analyzed, decomposed, distributed, planned, and executed intelligently, effectively, efficiently and in coordination. To achieve this, the 4D/RCS reference model provides well defined and highly coordinated sensory processing, world modeling, knowledge management, cost/benefit analysis, behavior generation, and messaging functions, as well as the associated interfaces."*

The 4D/RCS architecture is hierarchical in nature, and is composed of a common node structure at each level. A typical node is shown in Figure 1 (Albus & et.al. 2002) (p.28). The functional elements within an RCS node are behavior generation, sensory processing, world modeling, and value judgment. These are supported by a knowledge database, and a communication system that serves as a bridge between the functional processes and the knowledge layer. These functional elements, along with their interconnections, provide the infrastructure needed to allow a system to act autonomously.

Within the autonomous vehicle, actions are proposed by a planner (a part of the behavior generation component) which evaluates several different plans concurrently and determines the optimal plan to satisfy the higher level goals of the vehicle. One common approach to planning is based on a cost model (often referred to as cost-based planning) (Balakirsky 2003). In this approach, costs are assigned to actions that a vehicle performs and states that a vehicle occupies. By summing all of the costs that a vehicle incurs in each of the plans, a metric is created that can be used to compare the proposed plans against one another. Costs are usually assigned *a priori* to actions and states that are appropriate to the context in which the vehicle is operating. In the case of on-road driving, costs may be associated with:

- Running a stop sign
- Being too close to another vehicle
- Exceeding the speed limit by a certain threshold
- Changing lanes
- etc;

These costs are ubiquitous, and we intend for them to be used by many systems within the autonomous vehicle. For example, in addition to planning our own vehicle's path, we also expect these costs to be used by the vehicle's subsystem that anticipates the actions of other moving objects in the environment. Since we anticipate that these costs will be generally useful throughout the architecture, capturing them independently of any individual system makes them more generally accessible throughout the architecture.

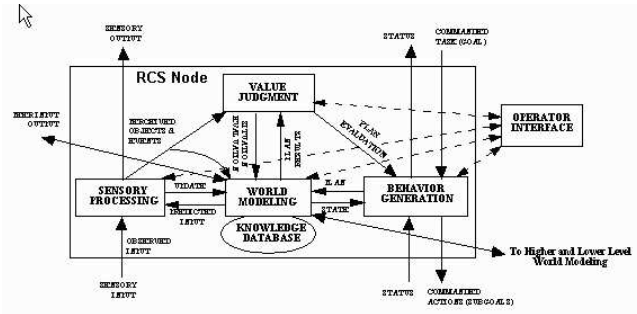

Figure 1: Typical RCS node structure

Clients within the system can then access the cost generator using a well-defined interface, thereby reducing dependencies among individual systems. This paper describes on-going research in exploring the use of the C Language Integrated Production System (CLIPS) to capture the cost models in an implementation-independent format within the 4D/RCS framework.

For the purpose of this paper, we will limit our examples and scope to cost models pertaining to on-road driving. However the results of this research are not on-road driving specific, and should be able to lend themselves to any domain in which real-time planning and control is applied.

We begin the paper by describing on-going work at NIST in cost-based planning and show how the planning system is expected to utilize and interface with the cost model. In subsequent sections we describe how the driving rule base has been expanded and refined since (Zimmerman, Schlenoff, & Balakirsky 2003). Finally, we discuss the ongoing evaluation of the suitability of CLIPS for this application and future work to be done on the project.

Planning With Costs

As previously mentioned, the cost generator will be utilized by many subsystems within the architecture. However for the purposes of this evaluation a single subsystem - the planner component of behavior generation - was chosen to model the interfaces between RCS subsystems and the cost generator.

The planning system used by the autonomous vehicle is an implementation of the incrementally created graph planning approach described in (Balakirsky 2003). This approach incorporates a graph search algorithm that determines the lowest cost path through a graph that is composed of nodes (representing system states) connected by edges (representing system actions). The cost of a path through the graph is defined as the sum of the action costs (the edges) plus the costs of having occupied the traversed states (the nodes).

One such graph search algorithm is Dijkstra's shortest path algorithm (Dijkstra 1959). An example of this algorithm is shown in Figure 2 and may be summarized as follows:

1. Initialize the search. This includes setting the initial cost of all nodes (in the figure nodes are shown as circles and node costs are the bold numbers next to them) to infinity, and creating a set of *open* nodes that only contains the goal node (n_g) at a cost of zero. An *open* node is a node that the search has reached but not evaluated. Nodes that have been fully evaluated are shown as bold circles in the figure.
2. Find the least expensive member of the *open* set (denote this node by n_{cheap}) and remove it from the *open* set.
3. Compare n_{cheap} to the start node (n_s). This search proceeds from the goal to the start, so if n_{cheap} is equal to the start node the search is finished. It can be noted that this search may also proceed from start to goal without loss of generality.

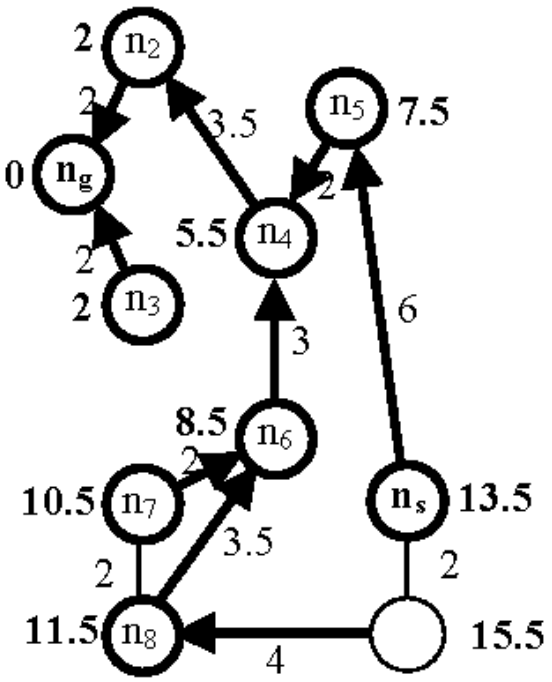

Figure 2: Graph search from node n_g to n_s

4. Expand n_{cheap}. During this step, the cost of reaching each of n_{cheap}'s predecessors (nodes connected by lines in the figure) must be determined. The following steps occur for each predecessor:
   (a) Determine the cost of the edge that connects n_{cheap} to the predecessor and the cost of occupying the predecessor.
   (b) If the sum of these two costs plus the cost of n_{cheap} is less then the current cost of the predecessor, the edge is maintained as a forward pointing edge (set to bold in the figure), any previous forward pointing edge is removed, and the predecessor is added to the *open* set.
5. Go to step 2.

An example of this algorithm's application is shown in Figure 2. The optimal path from any expanded node to n_g lies along the decreasing cost path of bold edges (follow the arrows). For this example, the search proceeds from the node labeled n_g to the node labeled n_s. The search terminates at the optimal answer when the node n_s is examined for expansion. The optimal path found may be seen to be $n_s - n_5 - n_4 - n_2 - n_g$.

As seen from the above algorithm description, each loop of the algorithm must make multiple calls to a cost generating function (step 4a). A single plan may entail several hundred or even thousands of algorithm loops, and the cost generator is at the heart of the loop, making its performance critical.

The actual interface to the planning system is quite simple. As the planner expands nodes, it passes the current node (and its associated state information) and the predecessor node (along with its associated state information) to

the cost generator. The cost generator then returns the incremental cost of the transition plus the cost of occupying the predecessor node.

On-Road Driving Rule Base

In previous work (Zimmerman, Schlenoff, & Balakirsky 2003), we examined rule-based and functional tools to determine which are best suited for cost generation in an autonomous vehicular system. This paper represents the second phase of this research, which expands, refines and continues to analyze the effectiveness of the rule base using the tool selected in (Zimmerman, Schlenoff, & Balakirsky 2003).

Expanding the Rule Base

In the previous version a small subset of the rules governing on-road driving were selected to implement in three different tools in order to gage their suitability for cost-generation in an autonomous vehicular system. These rules included such trivial situations as speed-limit violations (over,under), lane change violations, and stop sign violations. It also included slightly more complex rules that dictate the legal lane traversals at the intersection of two, two-lane roads in the presence of moving objects.

With the exception of the lane change constraints mentioned above, these rules dealt entirely with situational awareness and not with the preparation and execution of an activity. In the extended version of the Driving Rule Base (DRB), we have implemented additional functionality to allow the vehicle to assess the safety of a standard passing maneuver. This situation presents a variety of interesting intricacies which force the DRB to reason about information gained from the world model, as well as meta-reasoning using information about itself.

The constraints for a passing maneuver are grouped under an umbrella rule, *conditions-good-to-pass*, which can only be activated after several subordinate rules have been activated as shown in figure 3. Each of the specific conditions required to perform a passing maneuver inherit from an abstract parent class. The abstract *condition* class contains fields which identify the name of the condition, the certainty with which the condition is met (0 if it is not met, 1 if it is certain), and a weighted risk associated with violating the condition. When the rules are activated, a message-handler in CLIPS which acts as a constructor, initializes a third field, risk-index, which is a function of the weighted risk, the certainty, and the current context, and is used to compute the overall cost of the maneuver. For instance, say the vehicle is approaching an intersection and attempting to change lanes. The intersection may carry a weighted risk of 30 units with a certainty of .8, yielding a risk-index of 24, which can than be more finely tuned depending on the context (a change-lanes operation in this example, as opposed to, say, a toll-booth operation). Using this approach, we are able to adjust the aggressivity of the vehicle by altering its risk-index threshold to make it more or less sensitive to the risk involved in a maneuver. The expected input for this formula is information from the sensory processing system

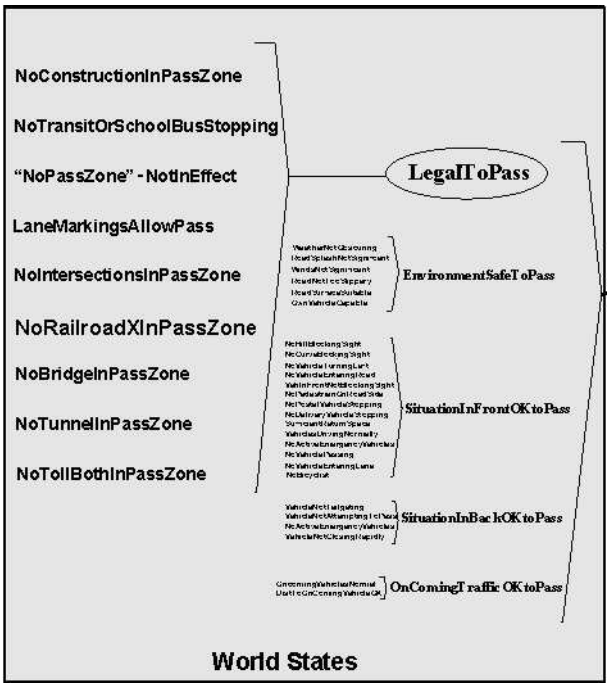

Figure 3: A tree-based representation of the conditions-good-to-pass rule. The sub-rule legal-to-pass is expanded for clarity.

which includes its prediction of what it is sensing and an associated certainty with which this observation was made.

The first sub-rule under *conditions-good-to-pass* checks whether it is legal to execute a passing maneuver given the current state of the world. As defined by Barbera (Barbera *et al.* 2003) the legal to pass condition is based on a series of sub-conditions (which all inherit from the parent class *condition*); lane markings, tunnels, intersections, no passing signage, school buses, construction zones, railroads, bridges, and tollbooths. A rule to determine legal lane markings to pass was already described in (Zimmerman, Schlenoff, & Balakirsky 2003) and is utilized by this rule. If these conditions are satisfied with a risk-index below the threshold for the vehicle another condition, legal-to-pass, is asserted to the knowledge base. This contains a new risk-index calculated as a function of the individual risk-indices of the prerequisite conditions.

The second sub-rule determines if the environment is safe to pass in. It checks if there are weather or other road conditions that would make it unsafe to pass. It also makes sure that the vehicle is capable of passing. Here we see an example of the meta-reasoning described earlier - it involves vehicle introspection in addition to situational awareness and allows the vehicle to make judgments about itself. It can take into account "self-knowledge", such as a malfunction or current speed, and use it to determine if it is capable of carrying out the pass. These rules are logically separate from the DRB as they do not deal specifically with the rules of the road. In future work these will be abstracted away from the DRB in order to preserve this distinction between rules of the road and meta-reasoning. If the environ-

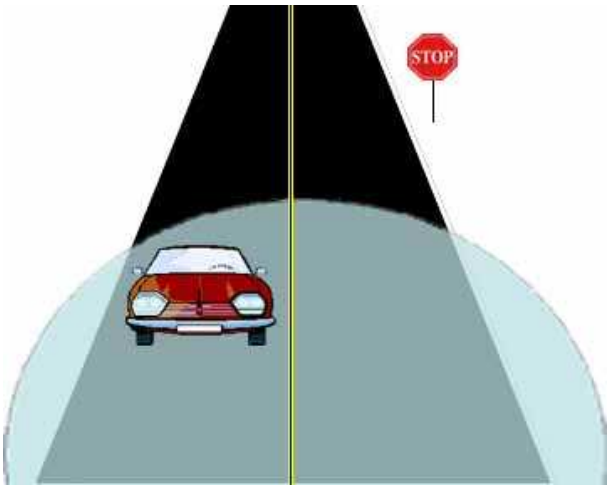

Figure 4: The shaded region indicates the sphere of influence of the vehicle. In this case the oncoming vehicle falls within the sphere of influence while the stop sign does not.

ment is determined to be safe to pass, another condition, environment-safe-to-pass is created which is similar to the legal-to-pass condition mentioned earlier.

The third and fourth sub-rules check the conditions in front and behind the vehicle. These conditions include such observations as pedestrians, obstructed shoulder, a vehicle in the rear attempting a pass, or a curve blocking visibility. The final sub-rule determines if the oncoming traffic is conducive to a pass by checking that the vehicle can safely avoid oncoming vehicles. Each sub-rule creates a new condition with the combined risk index of its respective conditions. Once these have been initialized for each of the sub-rules, the system activates the umbrella rule, *conditions-good-to-pass*. This calculates the overall risk of the maneuver based on the individual risk indices calculated by the sub-rules and returns a single cost to the planner.

Refining the Rule Base

In (Zimmerman, Schlenoff, & Balakirsky 2003) the knowledge base was composed of flat facts which relied on their order to determine a relationship between the individual fields and their values. This had the disadvantage of requiring the maintenance of a unique set of rules that applied to different types of ordered facts. In this extended version, the knowledge base has been refined to utilize the polymorphic subtyping capabilities available in CLIPS. By creating a hierarchical structure of objects in the world model, we can create rules which apply to entire classes of objects instead of individual types of facts. For instance, a parent class called object-of-interest is loosely defined as any object in the world model which may affect the behavior generation and planning of the vehicle. This is a broad and heterogeneous group which includes things like signs, intersections and bridges, to name a few. Using this technique, we can leverage CLIPS' ability to perform dynamic dispatch (a run-time determination of actual class type) to decide what actions should be taken depending on the specific type of the object-of-interest. This translates into a smaller and more robust rule base where more generic rules can be applied to a variety of different objects.

Objects-of-interest that arrive from the world model are filtered based on whether or not they exists within some predetermined sphere of influence of the vehicle. In this domain, objects of interest are loosely defined as objects in the world model which may affect the behavior generation and planning of the vehicle. The area of influence around the vehicle is variable based on the velocity and type of the vehicle 4. As the velocity of the vehicle increases, the sphere expands to encompass a larger area surrounding the vehicle. For example, the vehicle at rest may have a radius of influence of 10 m, while the same vehicle traveling at 35 km/h would have a sphere of influence of 623 m. The formula 1 (where K is a vehicle dependent constant) for determining the size of the sphere is arbitrary at this stage and was designed merely to exhibit how the velocity of the vehicle affects the relative size of the planning horizon. This allows objects to freely enter and exit the vehicle's sphere of influence as necessary.

$$\mathrm{radius-of-influence} \quad = \quad K[(V^2 + 20)/2] \quad (1)$$

By isolating the objects that have an immediate influence on the vehicle, we can significantly decrease the load on the inference engine as it processes the rules. Modules are then created that deal specifically with the higher-fidelity objects within the vehicle's sphere of influence and ignore object's that are not of immediate interest. This allows us to create context specific rules and facts and mask them from irrelevant contexts. While significant timing trials have not yet been completed, we anticipate that partitioning the information from the world model in this manner should significantly increase the overall performance of the system by substantially reducing the working knowledge base relevant to the DRB.

Analysis and Conclusions

The cost-generating sub-system within the autonomous vehicle framework will play an essential role in a number of other systems in the vehicle, including path planning and moving object prediction. An adequate means for representing the constraints that generate these costs, as well as managing and relaying these costs to other systems is necessary. As research becomes more comprehensive, CLIPS has continued to exhibit its proclivity for this type of application. In addition to the initial conditions satisfied in (Zimmerman, Schlenoff, & Balakirsky 2003), the object oriented capabilities allow us to use inheritance, polymorphism, and dynamic-dispatch to simplify an increasingly complex rule-base.

Future work will focus on integrating the existing cost-generator with the world-model to obtain real-time data about our surroundings. This will then be interfaced with the planner, and more specifically the node-generator, to assist in the planning algorithm described earlier.

Acknowledgments

The authors would like to thank Tony Barbera for the use of his task decomposition of the passing maneuver described in the section *Expanding the Rule Base*

References

Albus, J., and et.al. 2002. 4D/RCS Version 2.0: A Reference Model Architecture for Unmanned Vehicle Systems". Technical Report NISTIR 6910, National Institute of Standards and Technology, Gaitherburg, MD 20899, U.S.A.

Balakirsky, S. B. 2003. *A Framework for Planning with Incrementally Created Graphs in Attributed Problem Spaces*. Akademische Verlagsgesellschaft Aka GmbH.

Barbera, T.; Horst, J.; Schlenoff, C.; Wallace, E.; and Aha, D. 2003. Developing World Model Data Specifications as Metrics for Sensory Processing for On-Road Driving Tasks. Technical Report NIST Special Publication 990, National Institute of Standards and Technology, Gaitherburg, MD 20899, U.S.A.

Dijkstra, E. 1959. A Note on Two Problems in Connexion with Graphs. *Merische Mathematik* 269–271.

Zimmerman, N.; Schlenoff, C.; and Balakirsky, B. 2003. Performance Evaluation Of Tools and Techniques for Representing Cost-Based Decision Criteria for On-Road Autonomous Navigation. Technical report, National Institute of Standards and Technology, Gaitherburg, MD 20899, U.S.A.