Logic-based Subsumption Architecture

Eyal Amir and Pedrito Maynard-Reid II
Computer Science Department
Stanford University
Stanford, CA 94305
{eyala,pedmayn}@cs.stanford.edu

Abstract

In this paper we describe a logic-based AI architecture based on Brooks' Subsumption Architecture. We axiomatize each of the layers of control in his system separately and use independent theorem provers to derive each layer's outputs given its inputs. We implement the subsumption of lower layers by higher layers using circumscription to make assumptions in lower levels and nonmonotonically retract them when higher levels come up with some new conclusions. We give formal semantics to our approach. Finally, we describe an empirical experiment showing the feasibility of robot control using our architecture.

1 Introduction

In (Brooks 1986), Rodney Brooks provided a decomposition of the problem of robot control into layers corresponding to levels of behavior, rather than according to a sequential, functional form. Within this setting, he introduced the idea of subsumption, that is, that more complex layers could not only depend on lower, more reactive layers, but could also influence their behavior. The resulting architecture was one that could service simultaneously multiple, potentially conflicting goals in a reactive fashion, giving precedence to high priority goals.

Because of its realization in hardware, the architecture lacks declarativeness, making it difficult to implement higher-level reasoning and making its semantics unclear. Furthermore, the increasing hardware complexity with new layers introduces scaling problems. Finally, relying on hardware specifications, the architecture is specifically oriented towards robot control and is not applicable to software agents or other software-based intelligent agents. The problem of extending similar architectures to more complex tasks and goals and to agents that are not necessarily physical, has already been raised and discussed in general lines by (Minsky 1985) and (Stein 1997), but, to our knowledge, no practical AI architecture has been developed along these lines.

In this paper we describe an architecture modeled in the spirit of Brooks' Subsumption Architecture but which relies on a logical framework and which has wider applicability and extendability in the manner described above. Our Logic-Based Subsumption Architecture (LSA) includes a set of logical theories, each corresponding to a layer in the sense of Brooks' architecture. Each layer is supplied with a separate theorem prover, allowing the system of layers to operate concurrently. We use an approximation of nonmonotonic reasoning to model the connections between the theories. By allowing the layers to make nonmonotonic assumptions, each layer's performance is independent of the performance of other layers, thus supporting reactivity. We demonstrate our architecture modeling Brooks' first two layers, showing empirically that the layer in greatest demand of reactivity is sufficiently fast (0.2–0.3 seconds per control-loop cycle). This empirical result shows that general-purpose theorem provers can be used in intelligent agents without sacrificing reactivity.

The remainder of the paper is organized as follows: After giving a brief introduction to Brooks' system and logical AI, we describe a general architecture that embodies a collection of theories and uses their decoupling and interactions to exhibit complex behaviors. Then, we describe our representation for the first two layers of Brooks' system and describe how subsumption and the general operation of the system can be implemented using theorem proving and negation-as-failure techniques. We give formal semantics to our approach using circumscription, discuss implementation issues and conclude with comparisons to related work and a sketch of future directions.

This work is a first step towards creating a general logic-based AI architecture that is efficient, scalable and supports reactivity, an architecture that is our long-term goal.

2 Background

2.1 Brooks' Subsumption Architecture

Brooks showed that decomposing a system into parallel tasks or behaviors of increasing levels of competence, as opposed to the standard functional decomposition, can be advantageous. Whereas a typical functional decomposition might resemble the sequence:

sensors → perception → modeling → planning →

task recognition → motor control,

Brooks would decompose the same domain as follows:

avoid objects < wander < explore < build maps < monitor changes < identify objects < plan actions < reason about object behavior

where < denotes increasing levels of competence. Potential benefits from this approach include increased robustness, concurrency support, incremental construction, and ease of testing.

An underlying assumption is that complex behavior is a product of a simple mechanism interacting with a complex environment. This focus on simplicity led to a design where each individual layer is composed of simple state machine modules operating asynchronously without any central control.

In general, the different layers are not completely independent. For example, in the decomposition above, wandering and exploring depend on the robot's ability to avoid objects. Often, the system may be able to service these multiple goals in parallel, despite the dependence. However, occasionally, the goals of one layer will conflict with those of another layer. In such instances we would expect higher priority goals to override lower priority ones. To address this issue, the Subsumption Architecture provides mechanisms by which higher layers may interfere with the operation of lower layers. First, it is possible for higher layers to observe the state of lower layers. Second, it is possible for higher layers to inhibit outputs and/or suppress (that is, override) inputs to modules in a lower layer. Consequently, more competent layers can adjust the behavior of more reactive layers. At the same time, it is possible to have high priority tasks in a lower layer (such as halting when an object is dead-ahead) continue to have high precedence by simply not allowing any higher layers to tamper with those particular tasks.

In (Brooks 1986), Brooks describes in detail the first three layers of one particular robot control system he implemented using the Subsumption Architecture concept. The first two are shown in Figure 1. We briefly describe the three layers here:

Avoid The most basic layer in the system endows the robot with obstacle avoidance capabilities. When an obstacle appears directly ahead in the robot's path, it halts before colliding. In general, whenever there are obstacles in its vicinity, it uses their direction and distance with respect to the robot to compute a new heading which moves it away from the obstacles as much as possible.

In more detail, the layer accepts sonar readings of the robot's surroundings into its `sonar` module which outputs a map of the vicinity based on these readings. The `collide` module checks if there is an obstacle directly ahead and, if there is, forces the robot to stop regardless of what other modules are doing. The `feelforce` module uses the map to calculate a combined repulsive "force" that the surrounding objects exert on the robot. The `runaway` module checks if this force is significant enough to pay attention to and, in the case that it is, determines the new heading and speed for the robot to move away from the force. The `turn` module commands the robot to make the required turn, then passes the speed on to the forward module which, if not in a halt state, commands the robot to move forward with the specified speed. The further away the robot gets, the smaller the speed computed by the `runaway` module.

Wander The `wander` layer consists of two modules which, together with the `avoid` layer, cause the robot to move around aimlessly when it is not otherwise occupied. Every so often, the `wander` module chooses a new random direction for the robot to move in. The `avoid` module combines it with the output of the `avoid` layer's feelforce module, computing an overall heading that suppresses the input to the `avoid` layer's `turn` module. Hence, when wander mode is active, it overrides the default heading computed by the `avoid` layer.

Explore This layer begins to add some primitive goal-directed behavior to the robot's repertoire. The robot periodically checks to see if it is idle and, if so, chooses some location in the distance to head towards and explore. While in exploration mode, it inhibits the `wander` layer so that it remains more or less on track towards its destination. However, like the `wander` layer, it takes advantage of the `avoid` layer's capabilities to prevent collisions. As we don't model this layer in this paper (the first two layers are sufficient to demonstrate our proposal), we refer the reader to (Brooks 1986) for details.

2.2 Logic & Circumscription

Since the early days of AI, logic held a promise to serve as a main knowledge representation language in the future intelligent machine (McCarthy 1958). In the last decade, theorem provers became wide spread as formal verification tools and lately a few robot systems wielding logic have emerged (e.g., (Shanahan 1996),(Giacomo, Lesperance, & Levesque 1997)).

In the logical paradigm, McCarthy's Circumscription (McCarthy 1980) is one of the first major nonmonotonic reasoning tools. Since its debut, the nonmonotonic reasoning line of work has expanded and several textbooks now exist that give a fair view of nonmonotonic reasoning and its uses (e.g., (Brewka 1991), (Antoniou 1997), (Brewka, Dix, & Konolige 1997), (D.M. Gabbay 1994), (Sandewall 1994), (Shanahan 1997)). The motivations for nonmonotonic reasoning vary from formalizing Common Sense reasoning through Elaboration Tolerance and representing uncertainty to Belief Revision. We do not expand on these motivations here; the reader may look at (Shanahan 1997),(McCarthy 1998),(Pearl 1990) and (Antoniou 1997) for further details in these directions.

McCarthy's Circumscription formula (McCarthy

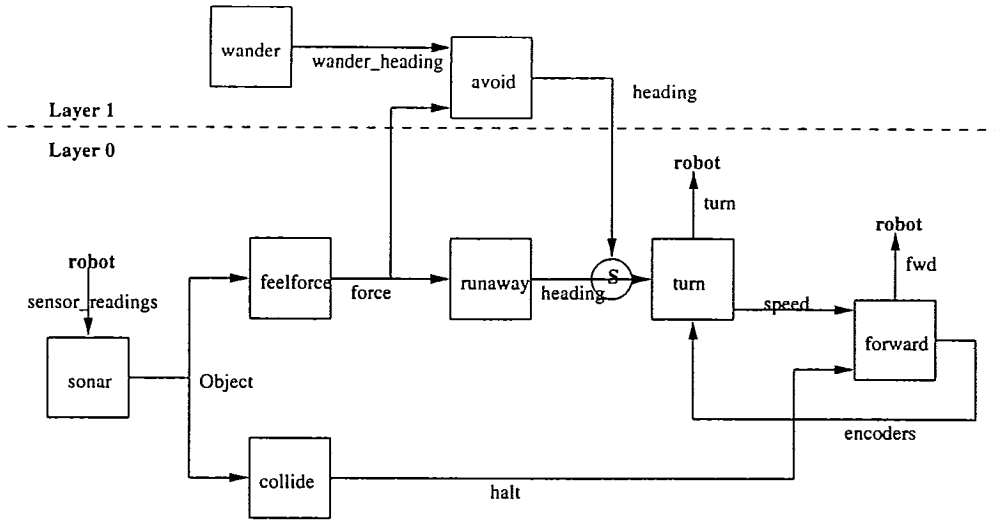

Figure 1: Layers 0 and 1 of Brooks' Subsumption Architecture robot control system.

1980)

$$Circ[A(P,Z);P;Z] = \\ A(P,Z) \wedge \forall p,z\ (A(p,z) \Longrightarrow \neg(p < P))$$

says that in the theory A, with parameter relations and function sequences P, Z, P is a minimal element such that $A(P,Z)$ is still consistent, when we are allowed to vary Z in order to allow P to become smaller.

Take for example the following simple theory:

$$T \equiv block(B_1) \wedge block(B_2)$$

Then, the circumscription of *block* in T, varying nothing, is $Circ[T;block;] = T \wedge \forall p\ [T_{[block/p]} \Longrightarrow \neg(p < block)]$ which is equivalent to

$$Circ[T;block;] \equiv \forall x\ (block(x) \Leftrightarrow (x = B_1 \vee x = B_2))$$

By minimizing *block* we concluded that there are no other blocks in the world other than those mentioned in the original theory T.

3 Logic-Based Subsumption Architecture

This section is dedicated to describing the proposed architecture. We first give an intuitive account and an approximation of the semantics of the system. Then we describe the architecture in more detail and give the ideas and goals embodied in it. Finally we describe how to build a system that approximates that of Brooks' in the proposed way.

3.1 Intuition

The *Logic-Based Subsumption Architecture* (LSA) is composed of a sequence of logical theories, each supplied with its own theorem prover. Each of the theories communicates with some of those "underneath" it in the sequence (those that are "subsumed"), modifying and controlling the behavior of these "lower-level" theories. Aside from this collection of theories there are sensors that affect the theories and manipulators that are affected by the theories (more precisely, by the results found by the theorem provers). In the description that follows we assume that the architecture is used to control a cylindrical robot that has sonar sensors on its perimeter and wheels that control its motion. No other sensors or manipulators are assumed (at this stage).

We want a system based on these premises to work in a loop as follows: First, the physical sonars collect their data and assert it in the form of logical axioms, such as $sonar\_reading(sonar\_number) = dist$. These axioms are added[1] to the appropriate theory in the sequence of theories comprising the system. At the same time, we assert any input coming to each layer from a higher-level theory (both kinds of *input* are replaced each cycle with the new inputs). Then, we ask the theorem prover of each layer to find the required outputs for that layer, having some of the outputs specify actions for the manipulators. After reaching the conclusions, we transmit the relevant ones (those outputs that specify actions for the manipulators) to the robot manipulators and, while the robot executes the requested actions, the loop starts again. Figure 2 illustrates this process.

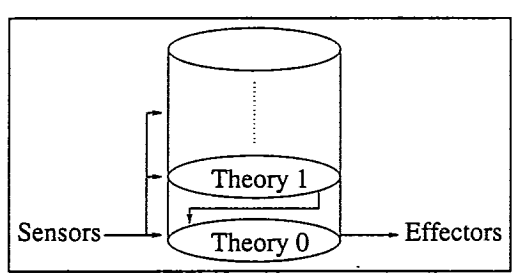

Figure 2: An abstract diagram of the LSA.

[1] At each *iteration* of sonar readings, we replace the previous inputs with the new ones.

The Logic-Based Subsumption Architecture should follow this general pattern, making sure that the loop is fast enough so that at the time we get to our conclusions the world is not too different from the world we based these conclusions on (e.g., making sure that the robot does not fall off the cliff while planning a way to avoid the cliff edge). Notice that, in fact there are several loops with different times (for the different layers and theorem provers) and the reactive loop is the one that counts for the speed of the overall loop. We describe how this behavior is achieved in the next section.

3.2 The architecture

Behavioral decomposition The first important idea we borrow from Brooks' architecture is that of decomposing the domain along behavioral lines rather than along the standard sequential functional lines. This change in paradigm makes it possible to view the robot control problem as a set of loosely coupled processes and, hence, parallelizable. We claim that we get similar performance benefits when applying this paradigm to a logical version of the Subsumption Architecture.

To build the LSA, we represent each layer with an axiomatization of the layer's behavior, that is, the layer's input, output and state, including any dependencies between these that support the task of this layer (in our architecture, these layer-inputs and layer-outputs are predicates/functions that are intended to go either from/to sensors/actuators or from/to lower layers).

Ignoring inter-layer interactions for a moment, the output of each layer is determined by running a separate theorem prover for that layer only. These treatment and representation buy us a few benefits. First, because the axiomatization of a layer is generally much smaller than that of the whole system, each cycle will be less computationally expensive than running one theorem prover over the whole compound axiomatization. Second, by decoupling the different layers of behavior in this way, it becomes possible to achieve more reactive behavior. As in Brooks' system, lower layers controlling basic behaviors do not need to wait on higher layers to have completed their computations before they can respond to situations. Rather, since lower layers are trusted to be autonomous (if the higher layer is not active, the lower layer will still behave validly) and those layers will have simpler axiomatizations in general, the cycle time to compute their outputs can be shorter than that of higher, more complex layers, leading to an overall high performance. This is important if, for example, we want the robot to continue avoiding obstacles as it tries to plan its next line of action.

Note that, although the module is an important construct in Brooks' architecture, in our representation modules serve mostly as "syntactic sugar." They provide conceptual clarity with regard to the operation of a layer a given theory denotes. We hasten to point out, however, that the relative independence between module axiomatizations could also be exploited, e.g., to have a separate theorem proving session to determine the intermediate module outputs, making it possible to pipeline the operation of a layer.

Subsumption Principles Of course, the layers are *not* fully independent. A fundamental feature of Brooks' Subsumption Architecture is the ability of higher layers to observe and interfere with the operation of the lower layers. In particular, the suppression and inhibition capabilities provide a means by which the otherwise independent layers may interact, allowing high-level goals to override/adjust default low-level reactive behavior. We adopt the view that, together with the task-based decomposition idea, this coupling approach represents an important and natural paradigm for an intelligent agent in general and robot control in particular (see (Stein 1997)).

However, implementing this idea in a logical setting raises the following issue: In general, when one layer overrides another, the two disagree on what some particular module input should be. Therefore, the two corresponding theories will be inconsistent. We need to formalize the higher-layer theory's precedence over the lower-layer's in such a way that (a) if there is no conflict, all the facts in either theory hold in the overall state of the system, (b) in the event of a conflict, the overall state sides with the higher layer, and (c) independent facts (e.g., the inputs to either layer) remain unchanged.

A number of techniques developed in the logic community, such as nonmonotonic techniques and belief revision, are applicable. We have chosen to use circumscription, although we agree that other approaches may be equally interesting and appropriate.

Circumscription-based Subsumption As described earlier, each layer is a logical theory. We distinguish three parts of the logical theory: (1) the *Body* of the level, (2) the *Sensory Latch* and the *Input Latch* and (3) the *Output* (see figure 3). The *Body* of the layer is the constant theory for that layer. The *Latches* are used to accept the input and replace it every cycle (rather than accumulate it). The *Output* is simply the output sentences proved by our layer (including the latches).

In the following, assume an arbitrary level i and that the theory in that layer is in a language $\mathcal{L}$. We distinguish the "input language" $\mathcal{L}_I \subset \mathcal{L}$, which constitutes the language that is allowed in the *Latches* of that layer. This is the language by which the theory is influenced and it serves for the assertions coming from the sensors and higher-level layers. The "output language" $\mathcal{L}_O \subset \mathcal{L}$ is used to limit the queries that the theory can be asked. This language includes the outputs to be asserted in lower-level layers and used for actuator controls.

To implement the idea of subsumption, we let each layer have "assumptions" about the inputs that may later be adjusted by other (higher-level) layers. These assumptions can take the form of an "abnormality"

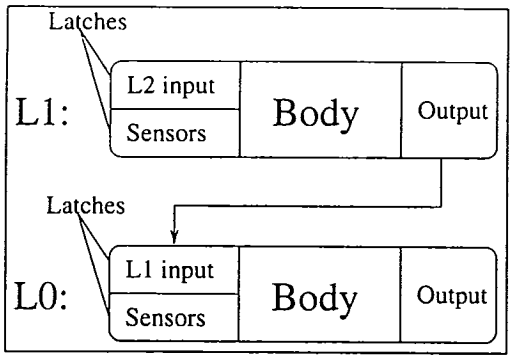

Figure 3: A detailed look at two layers.

predicate ab_i whose negation is a precondition for some sentences in the language $\mathcal{L}_I$ in the *Body* of that layer. The assumptions can also take the form of the Closed-World-Assumption (CWA), by minimizing a predicate in $\mathcal{L}_I$. In all these minimizations we vary all of $\mathcal{L}$ to make sure that our assumptions propagate. For example, a higher-level layer can assert (in a lower-level layer) the existence of an object that was previously excluded (using our CWA).

During each cycle of any particular layer, we first assert in that layer's Latches any sentences that higher layers may have inferred (in the respective output language for those higher layers). We then apply our "assumptions" by circumscribing the *ab* predicates or input predicates for which we enforce the CWA in the theory while varying all other predicates and functions in $\mathcal{L}$. A theorem prover can then obtain the appropriate outputs (for that layer), taking into account (asserted) interference from higher layers. More formally, let $Layer_i$ be the theory of layer i, ab_i the additional "abnormality" constant symbol and C_i a set of predicates in $\mathcal{L}_I$, for which we wish to assert CWA. Then, subsumption is achieved by using the parallel circumscription policy

$$Circ[Layer_i; ab_i, C_i; \mathcal{L}]$$

From an implementation point of view, many times this formula can be substituted with a simple (external to the logic) mechanical interference determining the value of the minimized predicates; we discuss this issue in section 4.

Semantics If we ignore the mechanism that runs behind the scenes for a moment (e.g., ignore the time difference between the theorem provers in different layers) and consider the entire system of layers as one logical theory, we can formalize the logical theory as follows. Let $Layer_i$, ab_i, C_i be as mentioned above. Then, the combined system described above is equivalent to

$$\begin{array}{l} Circ[Layer_0; ab_0, C_0; P_0] \wedge \\ Circ[Layer_1; ab_1, C_1; P_1, P_0, ab_0, C_0] \wedge \\ \vdots \\ Circ[Layer_N; ab_N, C_N; P_0, ..., P_N, ab_0, ..., ab_{N-1}, \\ \qquad C_0, ..., C_{N-1}] \end{array}$$

3.3 A Model of Brooks' System

In this part we describe the logical theory corresponding roughly to layers 0 and 1 in Brooks' Subsumption Architecture. We divide our theory to conceptually correspond to the layers and the modules mentioned in figure 1. For simplicity, we omit some parts of the description, and refer the reader to appendix A.

Our layer 1 differs slightly from Brooks'. Instead of implementing random wandering, this layer supports simple movements towards a goal location. This goal location is specified by layer 2 which, we can imagine, first constructs a plan of exploration then, at each step of this plan, asserts in the theory of layer 1 (via a subsumption latch) where the next goal location is. Layer 1 makes a simple calculation to determine in which of the eight quadrants surrounding it the goal position is located (see Figure 4). Layer 1 then asserts in the theory of layer 0 (by way of another subsumption latch) the existence of a "virtual pushing object" in the opposing quadrant. The avoidance capabilities of layer 0 effectively push the robot away from the object. The robot heads in the direction of the goal although it may deviate from a direct path, depending on the physical objects in its vicinity.

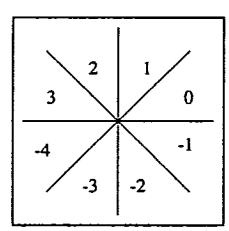

Figure 4: Quadrants for the pushing object.

During each cycle of layer 0, the theorem prover of layer 0 is asked to find the required actions for the modules *Turn* and *Forward* described below. It attempts to prove *fwd*(*heading_speed*) and *turn*(*heading_angle*), where *heading_speed* and *heading_angle* are instantiated by the proof. The results are translated into the appropriate robot commands.

LAYER 0

The inputs for this layer are the sonar data and the output from Layer 1. The input language includes the symbols *sonar_reading*, *sonar_direction*, *Object*, *Direction*, *Distance*, and *halt_robot*. The output includes *fwd* and *turn*.

Sonar The Sonar module takes the input from the physical sonars, asserted in the form of the axiom schema $sonar\_reading(sonar\_number) = dist$, and translates it to a map of objects (the type of each of the symbols is defined in the appendix).[2]

$$\begin{aligned}
&\forall dist, dir. \\
&\quad (\exists sonar_number. \\
&\qquad sonar_reading(sonar_number) = dist \wedge \\
&\qquad sonar_direction(sonar_number) = dir \wedge \\
&\qquad dist \geq 0 \wedge dir > -\pi \wedge dir \leq \pi) \Longrightarrow \\
&\quad (\exists obj.\ Object(obj) \wedge Distance(obj) = dist \wedge \\
&\qquad Direction(obj) = dir) \\
&\forall sonar_number.\ sonar_direction(sonar_number) = \\
&\quad \frac{2\pi}{NSONARS} * sonar_number
\end{aligned} \tag{1}$$

The reason we have only an implication from sonars to objects is that we minimize *Object* in our circumscription below. for the same reason, we don't include axioms stating that there is at most one object at any point.

Collide We take the predicate Object and check to see if it has detected objects lying directly in front of us.

$$Object\_Ahead \Longrightarrow halt\_robot$$

$$\begin{aligned}
&Object_Ahead \Longleftrightarrow \\
&\quad (\exists obj.\ (Object(obj) \wedge Distance(obj) < MIN_DIST \wedge \\
&\qquad Direction(obj) = dir \wedge dir > 2\pi - \tfrac{\pi}{4} \vee dir < \tfrac{\pi}{4})
\end{aligned} \tag{2}$$

Feelforce Feelforce does the dirty work of computing the combined repulsive force from the different detected objects[3].

$$\begin{aligned}
&force_direction = \tan^{-1}(\tfrac{force_y}{force_x}) \\
&force_strength = \sqrt{{force_y}^2 + {force_x}^2} \\
&force_x = sum_x(0, 2\pi) \\
&force_y = sum_y(0, 2\pi)
\end{aligned} \tag{3}$$

Runaway

$$\begin{aligned}
&\neg ab_{avoid} \Longrightarrow \\
&\quad heading_angle = \\
&\qquad ((2\pi + force_direction) \bmod 2\pi) - \pi \\
&\neg ab_{avoid} \Longrightarrow \\
&\quad heading_speed = force_strength
\end{aligned} \tag{4}$$

[2]We assume that the robot's 0 radians reference point is straight ahead, the front sonar is numbered 0, and the sonars are numbered consecutively counter-clockwise from 0 to $NSONARS - 1$.

[3]Feelforce may be implemented as a library function rather than as a logical theory, as it does not gain much by the logical representation, it can be implemented more efficiently as a procedure and the logical representation uses some library functions anyway.

Turn

$$need\_turn(heading\_angle) \Longrightarrow turn(heading\_angle) \tag{5}$$

Forward

$$\begin{aligned}
&\neg halt_robot \wedge \neg need_turn(heading_angle) \wedge \\
&\quad need_fwd(heading_speed) \Longrightarrow \\
&\qquad fwd(heading_speed) \\
&halt_robot \vee need_turn(heading_angle) \Longrightarrow fwd(0)
\end{aligned} \tag{6}$$

Circumscribing the Theory

Finally, we add the parallel circumscription formula

$$Circ[Layer_0; ab_{avoid}, Object, halt\_robot; \mathcal{L}(Layer_0)]$$

LAYER 1

The inputs for this layer are the current location data from the robot and the output from Layer 2. The input language includes the symbols *got_move_cmd* and *curr_location*. The output includes *Object*, *Direction*, *Distance*.

Note that, unlike in layer 0, all the coordinates in this layer are in terms of some fixed coordinate system independent of the robot's location[4].

Simple Move

$$\begin{aligned}
&\forall x_0, y_0, x, y.\ curr_loc(x_0, y_0) \wedge got_move_cmd(x, y) \Longleftrightarrow \\
&\quad pushing_object(quadrant(x_0 - x, y_0 - y))
\end{aligned} \tag{7}$$

Push

$$\begin{aligned}
&\forall quad.\ pushing_object(quad) \Longleftrightarrow \\
&\quad Object(PUSH_OBJECT) \wedge \\
&\quad Direction(PUSH_OBJECT, quad * \tfrac{2\pi}{NQUADS}) \wedge \\
&\quad Distance(PUSH_OBJECT, PUSH_OBJ_DIST)
\end{aligned} \tag{8}$$

Circumscribing the Theory

Again, we add the parallel circumscription formula

$$Circ[Layer_1; got\_move\_cmd; \mathcal{L}(Layer_1)]$$

4 Implementation issues

We have implemented the above theory using the PTTP theorem prover ((Stickel 1988b), (Stickel 1988a), (Stickel 1992)) on a Sun Ultra60 Creator3D with 640MB RAM running Solaris 2.6 with Quintus Prolog as the underlying interpreter for PTTP. The theory is not yet implemented on a physical robot, yet the simulations done on the above-described machine helped us identify some points of difficulty in using a theorem prover for the task of controlling an intelligent agent.

[4]The robot is able to maintain its position in a cartesian space with origin at the position where it was last "zeroed" (e.g., where it was powered on).

4.1 Choice of a theorem prover

The first difficulty we encountered was in fact finding a suitable theorem prover. Our theory includes several mathematical computations (such as several trigonometric functions (see the appendix A)) that are much better suited for a systematic algorithm than a theorem prover. Since we also wanted to have some algebraic sophistication in our theory, we needed semantic attachments. We examined many theorem provers and none of them seemed to support semantic attachments easily, nor did we have any convenient control over the theorem proving process (via strategies or otherwise).

Some of the provers that we examined more closely are Otter[5] (a resolution theorem prover), ACL2[6] (an industrial-strength version of the Boyer Moore theorem prover) and ATP[7] (a model elimination theorem prover). The major difficulties we encountered with them (not all difficulties were encountered with all) were the inability to append semantic attachments easily, complexity of making the theorem prover run on a given platform, the inability to control the inference process easily, and the lack of documentation.

In addition we also examined a few proof checkers such as PVS[8] (a proof checker), HOL[9] and GETFOL[10], all of which were found unsuitable due to their need for at least some human intervention.

PTTP (Prolog Technology Theorem Prover) is a model-elimination theorem prover. Given a theory made of clauses (not necessarily disjunctive) without quantifiers, the PTTP produces a set of PROLOG-like horn clauses, it makes sure only sound unification is produced and avoids the negation-as-failure proofs that are produced by the PROLOG inference algorithm. It also makes sure the inference algorithm is complete by using ID (Iterative Deepening) in the proof space. Together these ensure the PTTP is a sound and complete theorem prover.

One of the features that we liked the most about the PTTP was that, despite the lack of suitable documentation (although there are a fair number of examples), the theorem prover is very easy to customize and its reliance on the underlying language (it was implemented in both PROLOG and LISP) allows the easy use of semantic attachments. The collection of examples together with the PTTP software is very illustrative and, despite some difficulties incurred by the use of semantic attachments and built-in predicates (such as the algebraic $<$ relation), it gave us relatively few troubles in either installation or use.

[5] http://www-unix.mcs.anl.gov/AR/otter/

[6] http://www.cs.utexas.edu/users/moore/acl2/acl2-doc.html

[7] http://www.ksl.stanford.edu/people/axf/reference-manual.html

[8] http://www.csl.sri.com/pvs.html

[9] http://www.comlab.ox.ac.uk/archive/formal-methods/hol.html

[10] http://www-formal.stanford.edu/clt/ARS/Entries/getfol

4.2 Running PTTP with our theory

We embodied the *Feelforce* module in a C function *get_force* that needs no input and returns the force vector $[Strength, Direction]$. It does its work by calling the prolog bagof operator to collect all the objects for which existence proofs can be found and then simply computes the sum of the forces subjected by each of the objects. The CWA (Closed World Assumption) is achieved here by limiting the sizes of proofs to be no longer than a specified constant (we experimented a little and got the constant to be 20. At about 16 all the objects had a proof. At around 36 all the objects got a second proof and the same happened at about 48. We did not experiment further).

Finally, $get\_force(L), <, =<, +, *,$ abs and others are treated as "built-in" functions/predicates and are given to the underlying prolog for evaluation (which in turn may call the C function *get_force* for the prolog predicate). This way one manages to get the semantic attachments without the need to be able to prove many properties of algebraic formulas.

The rest of the theory stays the same, with the provision that constants such as *NSONARS* and *MIN_DIST* are incorporated with the aid of predicates such as $nsonars(X)$ and $min\_dist(Y)$.

We ran our theory with different simulated sensory inputs (results from a sample run are shown in figures 5, 7, and 6 in the appendix) and the results were achieved in 0.2 to 0.3 seconds, depending on whether a turn was needed (leaning towards 0.2 seconds) or a forward move was needed (leaning mostly towards 0.3 seconds). It is worth mentioning that in the computation of *get_force* we applied caching (of the computed force), and since PTTP does not apply caching of results, there was a major improvement in efficiency (from several seconds for a proof to 0.2–0.3) using this simple scheme. This is due to the fact that every proof "re-proved" *get_force* many times.

4.3 Nonmonotonicity considerations

As mentioned above, the way we treated nonmonotonicity in this experiment was using NAF (Negation-as-Failure). If a default was not proved to be wrong after a certain amount of time, then we treated it as false. In particular, there were three points in which nonmonotonicity was required:

1. CWA for the objects in our world.
2. *halt_robot* was assumed to be false unless otherwise proved.
3. ab_{avoid}, which allows complete overriding of the force constraints input of layer 0 by layer 1, is assumed to be false, since we did not use that facility (overriding) in our implementation of level 1.

For the CWA for the objects, we looked for proofs no longer than 20 and we did the same for the *halt_robot* proposition.

5 Related Work

In our work we showed that theorem provers can be used to implement robot control. We also showed that an extended version of Brooks' Subsumption Architecture can be implemented using theorem provers and logical theories. This extended version is proposed as a general architecture for building intelligent agents. Notice that we did not include a specific action theory above L0, but only showed how can such a theory influences L0.

We compare our work to those of Shanahan ((Shanahan 1996), (Shanahan 1998)), Baral and Tran ((Baral & Tran 1998)) and Reiter et al ((Reiter 1998), (Lesprance *et al.* 1996), (Reiter 1996)): (Shanahan 1998) describes a map-building process using abduction. His abduction is specialized for the specific scenario of spatial occupancy and noise that one may wish to include. He then implements the theory in an algorithm that abides by the logical theory, and to his account of abduction.

In contrast, our work is not implemented in an algorithm but rather using a general-purpose theorem prover. We showed that we can use theorem provers for control as long as the theory remains small and relatively simple. We described a way of joining such theorem provers and theories in a structure that allows for larger theories to interact and for more complicated behaviors to be established.

Although our control theory is much simpler than that of (Shanahan 1998), we can in fact include a version of the original theory presented by Shanahan as an upper layer. Also, since our robot is a more sophisticated mobile robot, any inclusion of such a theory will have to take into account the different sensors (sonars instead of switches).

The work of Baral and Tran ((Baral & Tran 1998)) focuses on the relationship between the members of the family of action languages $\mathcal{A}$ ((Gelfond & Lifschitz 1993), (Giunchiglia, Kartha, & Lifschitz 1997), (Kartha & Lifschitz 1994)) and reactive control modules. They define control modules to be of a form of Stimulus-Response (S-R) agents (see (Nilsson 1998)) where a state is defined by a set of fluent values (either sensory or memory) and a sequence of rules defines the action that the agent should take, given conditions on the state. They provide a way to check that an S-R module is correct with respect to an action theory in $\mathcal{A}$ or $\mathcal{AR}$. Finally, they provide an algorithm to create an S-R agent from an action theory.

Their work, although dealing with reactivity, does not seem to be able to deal with the world-complexity of our model. Our sensors have too many possible input values to be accounted for by several input fluents. If one decides to use the algorithm described by Baral and Tran to produce a simple S-R module, the complexity of the algorithm (which is worst-case exponential in the number of fluents) will not allow it to end in our life time. Also, they lack the hierarchy-of-theories model that we use in our work.

Finally, the work of Reiter, Levesque and their colleagues ((Levesque *et al.* 1997), (Giacomo, Lesperance, & Levesque 1997), (Reiter 1998), (Lesprance *et al.* 1996), (Reiter 1996), (Giacomo, Reiter, & Soutchanski 1998)) focuses on the language GOLOG (and its variants) for the specification of high-level robot actions. In their paradigm, there is a planner that computes/plans the GOLOG program off-line, then lets the robot execute the GOLOG program on-line. Their language includes (among others) testing for a truth condition, performing a primitive action, performing a sequence of actions, a nondeterministic choice of two actions and a nondeterministic iteration.

While we use a hierarchical model for reasoning, merging both planning and execution, their work splits planning and execution, having the planning done off-line. Also, their use of logic is only for the semantics of their GOLOG programs, which is given using Situation Calculus ((McCarthy & Hayes 1969)).

6 Discussion and Future Work

In the last 5 years, the logical approach to AI got reinvigorated with positive results on different frontiers, from planning (e.g., the work of Bibel and of Selman and Kautz ((Kautz, McAllester, & Selman 1996))) to sensing and AI architectures (e.g., (Shanahan 1996), (Lesperance *et al.* 1994), (Giacomo, Lesperance, & Levesque 1997)).

In this paper, we have presented a logic-based architecture that formalizes Brooks' Subsumption Architecture, using circumscription to implement subsumption. In so doing, we have combined the reactivity advantages of Brooks' architecture with the declarative advantages of logic to produce a first cut at an architecture that can perform sensing, planning and acting concurrently.

At the moment, the system is only partially implemented (level 0 only) on a simulating computer. Besides implementing the system on a mobile robot, our future work plan includes expanding the layers described in this paper to contain planning layers, map-creating layers (e.g., creating a map of the world (possibly following the work of Shanahan)), layers that contain beliefs about the world (e.g., we may want to doubt our conclusion that we are in a certain location if we believe that a moment ago we were in a distant location and no reasonable change was done to the world to put us in that new location), etc. This project also serves as an experiment in the Elaboration Tolerance of the layering approach.

7 Acknowledgments

We wish to thank Mark Stickel for allowing us to use his PTTP sources (both for PROLOG and LISP) and providing helpful answers to our inquiries regarding its use.

This research was supported by an ARPA (ONR) grant N00014-94-1-0775 and by a National Physical Science Consortium (NPSC) fellowship.

References

Antoniou, G. 1997. *Nonmonotonic Reasoning.* Cambridge, Massachusetts: MIT Press.

Baral, C., and Tran, S. C. 1998. Relating theories of actions and reactive control. *Electronic Transactions on Artificial Intelligence.* Under Review.

Brewka, G.; Dix, J.; and Konolige, K. 1997. *Nonmonotonic Reasoning: An Overview*, volume 73 of *CSLI Lecture Notes.* CSLI Publications.

Brewka, G. 1991. *Nonmonotonic Reasoning: Logical Foundations of Common Sense.* Cambridge University Press.

Brooks, R. A. 1986. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation* RA-2(1):14–23.

D.M. Gabbay, C.J.Hogger, J., ed. 1994. *Handbook of Logic in Artificial Intelligence and Logic Programming, Volume 3: Nonmonotonic Reasoning and Uncertain Reasoning.* Great Britain: Oxford University Press.

Gelfond, M., and Lifschitz, V. 1993. Representing Actions and Change by Logic Programs. *Journal of Logic Programming* 17:301–322.

Giacomo, G. D.; Lesperance, Y.; and Levesque, H. J. 1997. Reasoning about concurrent execution, prioritized interrupts, and exogenous actions in the situation calculus. In Pollack, M., ed., *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97)*, 1221–1226. Morgan Kaufmann.

Giacomo, G. D.; Reiter, R.; and Soutchanski, M. 1998. Execution monitoring of high-level robot programs. In Cohn, A.; Schubert, L.; and Shapiro, S. C., eds., *Proceedings of the 6th International Conference on Knowledge Representation and Reasoning (KR-98)*, 453–464. Morgan Kaufmann.

Giunchiglia, E.; Kartha, G. N.; and Lifschitz, V. 1997. Representing Action: Indeterminacy and Ramifications. *Artificial Intelligence.* to appear.

Kartha, G. N., and Lifschitz, V. 1994. Actions with indirect effects: Preliminary report. In *Proceedings of KR-94*, 341–350.

Kautz, H.; McAllester, D.; and Selman, B. 1996. Encoding plans in propositional logic. In Doyle, J., ed., *Proceedings of KR'96*, 374–384. Cambridge, Massachusetts: KR.

Lesperance, Y.; Levesque, H. J.; Lin, F.; Marcu, D.; Reiter, R.; and Scherl, R. B. 1994. A logical approach to high-level robot programming: A progress report. In Kuipers, B., ed., *Control of the Physical World by Intelligent Systems: Papers from the 1994 AAAI Fall Symposium*, 79–85.

Lesprance, Y.; Levesque, H.; Lin, F.; Marcu, D.; Reiter, R.; and Scherl, R. 1996. Foundations of a logical approach to agent programming. In *Intelligent Agents II. Agent Theories, Architectures, and Languages. IJCAI-95 Workshop (ATAL) Proceedings*, 331–346 of xviii+437. Springer-Verlag.

Levesque, H.; Reiter, R.; Lesprance, Y.; Lin, F.; and Scherl, R. 1997. Golog: A logic programming language for dynamic domains. *Journal of Logic Programming* 31:59–84.

McCarthy, J., and Hayes, P. J. 1969. Some Philosophical Problems from the Standpoint of Artificial Intelligence. In Meltzer, B., and Michie, D., eds., *Machine Intelligence 4.* Edinburgh University Press. 463–502.

McCarthy, J. 1958. Programs with Common Sense. In *Mechanisation of Thought Processes, Proceedings of the Symposium of the National Physics Laboratory*, 77–84. London, U.K.: Her Majesty's Stationery Office. Reprinted in McC90.

McCarthy, J. 1980. Circumscription—A Form of Non-Monotonic Reasoning. *Artificial Intelligence* 13:27–39. Reprinted in (McCarthy 1990).

McCarthy, J. 1990. *Formalization of common sense, papers by John McCarthy edited by V. Lifschitz.* Ablex.

McCarthy, J. 1998. Elaboration Tolerance. In CommonSense '98 and in McCarthy's web page http://www-formal.stanford.edu/jmc/elaboration.html.

Minsky, M. 1985. *The Society of Mind.* New York: Simon and Schuster.

Nilsson, N. J. 1998. *Artificial Intelligence, A new Synthesis.* Morgan-Kaufmann.

Pearl, J. 1990. Reasoning under uncertainty. *Annual Review of Computer Science* 4:37–72.

Reiter, R. 1996. Natural actions, concurrency and continuous time in the situation calculus. In *Principles of Knowledge Representation and Reasoning, Proceedings of the fifth International Conference (KR '96).* Morgan Kaumann.

Reiter, R. 1998. *Knowledge in Action: Logical Foundations for Describing and Implementing Dynamical Systems.* National Academy Press.

Sandewall, E. 1994. *Features and Fluents.* Oxford University Press.

Shanahan, M. P. 1996. Robotics and the common sense informatic situation. In *Proceedings ECAI 96*, 684–688.

Shanahan, M. 1997. *Solving the Frame Problem, a mathematical investigation of the common sense law of inertia.* Cambridge, MA: MIT press.

Shanahan, M. 1998. A logical account of the common sense informatic situation for a mobile robot. *Electronic Transactions on Artificial Intelligence.* Under Review.

Stein, L. A. 1997. Postmodular systems: Architectural principles for cognitive robotics. *Cybernetics and Systems* 28(6):471–487. Available from http://www.ai.mit.edu/people/las/cv.html.

Stickel, M. E. 1988a. A Prolog technology theorem prover. In Lusk, E., and Overbeek, R., eds., *Proc. 9th International Conference on Automated Deduction*, 752–753. Springer LNCS, New York.

Stickel, M. E. 1988b. A Prolog Technology Theorem Prover: implementation by an extended Prolog compiler. *Journal of Automated Reasoning* 4:353–380. Stickel, M. E.

Stickel, M. E. 1992. A Prolog Technology Theorem Prover: a new exposition and implementation in Prolog. *Theoretical Computer Science* 104:109–128. Stickel, M. E.

A The Logical Theory

In this appendix we draw the entire logical theory for our version of Brooks' Subsumption Architecture example from (Brooks 1986).

A.1 Language

We will write our theories using the following language $\mathcal{L}$. $\mathcal{L}$ is a double-sorted language with the three sorts Real (real numbers), Int (integer numbers) and objects.

object constants *NSONARS*, π, *numbers*, *MIN_DIST*, *MIN_SPEED*, *MIN_ANGLE*, *force_direction*, *force_strength*, $force_x$, $force_y$, ab_{avoid}, *heading_angle*, *heading_speed*, *NQUADS*, *PUSH_OBJECT*, *PUSH_OBJ_DIST*.

variables Real: *dist*, *dir*, dir_1, dir_2, x_0, y_0, x, y.
Objects: *obj*, *obj1*, *obj2*, *obj'*.
Integer: *sonar_number*, *quad*.

functions *sonar_reading(int)*, *Direction(object)*, *sonar_direction(int)*, *Distance(object)*, $sum_x(int, int)$, $sum_y(int, int)$, *quadrant*.

predicates *Object(object)*, *Object_Ahead(object)*, *halt_robot*, *Exactly_One_Object(real, real, object)*, *need_turn(real)*, *turn(real)*, *need_fwd(real)*, *fwd(real)*. $curr\_loc(x_0, y_0)$, $got\_move\_cmd(x, y)$, *pushing_object(quad)*.

library functions tan (tangent), sin (sine), cos (cosine), square root, $*$, $+$, $-$, $/$, abs

A.2 LAYER 0

Sonar

$$\begin{aligned}
&\forall sonar_number, dist.\\
&\quad sonar_reading(sonar_number) = dist \Longrightarrow\\
&\qquad sonar_number \geq 0 \wedge\\
&\qquad sonar_number < NSONARS\\
&\forall dist, dir.\\
&\quad (\exists sonar_number.\\
&\qquad sonar_reading(sonar_number) = dist \wedge\\
&\qquad sonar_direction(sonar_number) = dir \wedge\\
&\qquad dist \geq 0 \wedge dir > -\pi \wedge dir \leq \pi) \Longrightarrow\\
&\quad (\exists obj.\ Object(obj) \wedge Distance(obj) = dist \wedge\\
&\qquad Direction(obj) = dir)\\
&\forall sonar_number.\ sonar_direction(sonar_number) =\\
&\quad \frac{2\pi}{NSONARS} * sonar_number
\end{aligned} \tag{9}$$

Collide

$$Object\_Ahead \Longrightarrow halt\_robot$$

$$\begin{aligned}
&Object_Ahead \Longleftrightarrow\\
&\quad (\exists obj.\ (Object(obj) \wedge Distance(obj) < MIN_DIST \wedge\\
&\qquad Direction(obj) = dir \wedge dir > 2\pi - \tfrac{\pi}{4} \vee dir < \tfrac{\pi}{4})
\end{aligned} \tag{10}$$

Feelforce

$$\begin{aligned}
&force_direction = \tan^{-1}(\tfrac{force_y}{force_x})\\
&force_strength = \sqrt{{force_y}^2 + {force_x}^2}\\
&force_x = sum_x(0, 2\pi)\\
&force_y = sum_y(0, 2\pi)
\end{aligned} \tag{11}$$

The following is a somewhat inefficient way of saying "sum the forces".

$$\begin{aligned}
&\forall dir_1, dir_2, obj.\ Exactly_One_Object(dir_1, dir_2, obj) \Longrightarrow\\
&\quad sum_x(dir_1, dir_2) = \cos(Direction(obj)) * \tfrac{1}{Distance(obj)^2}\\
&\forall dir_1, dir_2.\ sum_x(dir_1, dir_2) =\\
&\quad sum_x(dir_1, dir_2 - \tfrac{dir_2 - dir_1}{2}) +\\
&\quad sum_x(dir_1 + \tfrac{dir_2 - dir_1}{2}, dir_2)\\
&\forall dir_1, dir_2, obj.\ Exactly_One_Object(dir_1, dir_2, obj) \Longrightarrow\\
&\quad sum_y(dir_1, dir_2) = \sin(dir) * \tfrac{1}{dist^2}\\
&\forall dir_1, dir_2.\ sum_y(dir_1, dir_2) =\\
&\quad sum_y(dir_1, dir_2 - \tfrac{dir_2 - dir_1}{2}) +\\
&\quad sum_y(dir_1 + \tfrac{dir_2 - dir_1}{2}, dir_2)
\end{aligned}$$

$$\begin{aligned}
&\forall dir_1, dir_2.\ Exactly_One_Object(dir_1, dir_2, obj) \Longleftrightarrow\\
&\quad (Object(obj) \wedge dir_1 \leq Direction(obj) \wedge\\
&\qquad Direction(obj) < dir_2 \wedge\\
&\qquad (\forall obj'.\ Object(obj') \wedge dir_1 \leq Direction(obj') \wedge\\
&\qquad\quad Direction(obj') < dir_2 \Longrightarrow obj' = obj))
\end{aligned} \tag{12}$$

Runaway

$$\neg ab_{avoid} \Longrightarrow \\ heading\_angle = \\ ((2\pi + force\_direction) \bmod 2\pi) - \pi \tag{13}$$

$$\neg ab_{avoid} \Longrightarrow \\ heading\_speed = force\_strength$$

Turn

$$need\_turn(heading\_angle) \Longrightarrow turn(heading\_angle)$$

$$\forall angle.\ (need\_turn(angle) \iff \\ heading\_angle = angle \wedge angle > MIN\_ANGLE) \tag{14}$$

Forward

$$\neg halt\_robot \wedge \neg need\_turn(heading\_angle) \wedge \\ need\_fwd(heading\_speed) \Longrightarrow \\ fwd(heading\_speed)$$

$$\forall speed.\ (need\_fwd(speed) \iff \\ heading\_speed = speed \wedge speed > MIN\_SPEED)$$

$$halt\_robot \vee need\_turn(heading\_angle) \Longrightarrow fwd(0) \tag{15}$$

LAYER 1

Simple Move

$$\forall x_0, y_0, x, y.\ curr\_loc(x_0, y_0) \wedge got\_move\_cmd(x, y) \iff \\ pushing\_object(quadrant(x_0 - x, y_0 - y))$$

$$\forall x, y.\ x > y \geq 0 \Longrightarrow quadrant(x, y) = 0$$

$$\forall x, y.\ y \geq x \geq 0 \Longrightarrow quadrant(x, y) = 1$$

$$\forall x, y.\ y > 0 > x \wedge y < abs(x) \geq \frac{x}{y} > -1 \Longrightarrow \\ quadrant(x, y) = 2$$

$$\forall x, y.\ y > 0 > x \wedge y \geq abs(x) \Longrightarrow quadrant(x, y) = 3$$

$$\forall x, y.\ x > 0 > y \wedge x > abs(y) \Longrightarrow quadrant(x, y) = -1$$

$$\forall x, y.\ x > 0 > y \wedge x \leq abs(y) \Longrightarrow quadrant(x, y) = -2$$

$$\forall x, y.\ 0 \geq x > y \Longrightarrow quadrant(x, y) = -3$$

$$\forall x, y.\ 0 \geq y > x \Longrightarrow quadrant(x, y) = -4 \tag{16}$$

Push

$$\forall quad.\ pushing\_object(quad) \Longrightarrow \\ Object(PUSH\_OBJECT) \wedge \\ Direction(PUSH\_OBJECT, quad * \frac{2\pi}{NQUADS}) \wedge \\ Distance(PUSH\_OBJECT, PUSH\_OBJ\_DIST) \tag{17}$$

B Proofs Done by the Theorem Prover

The figures below show a sample proof session by the PTTP theorem prover. Figures 5 shows the axioms corresponding to the sonar input, 7 gives an example of a proof by the system, and 6 shows the proof for the existence of a set of objects.

```
sonar_reading(0, 0.5),
sonar_reading(1, 2),
sonar_reading(2, 2),
sonar_reading(3, 2),
sonar_reading(4, 2),
sonar_reading(5, 2),
sonar_reading(6, 2),
sonar_reading(7, 2),
```

Figure 5: An example of simulated sonar input asserted in the theory.

```
[PROOF of prove((object(Obj), direction(Obj,Dir), distance(Obj,Dist)),
            20,0,1,[],_) (one of the proofs)]

search for cost 0 proof...  8 inferences so far.
search for cost 1 proof...  16 inferences so far.
search for cost 2 proof...  27 inferences so far.
search for cost 3 proof...  50 inferences so far.
⋮
search for cost 15 proof...
Proof:
Goal#  Wff#  Wff Instance
-----  ----  ------------
  [0]    0   query :- [1] , [6] , [11].
  [1]   27      object(obj_sk1(2,3.14159/2)) :- [2] , [3] , [4].
  [2]    1         pi(3.14159).
  [3]   19         sonar_reading(2,2).
  [4]    8         sonar_direction(2,3.14159/2) :- [5].
  [5]    1            pi(3.14159).
  [6]   27      direction(obj_sk1(2,3.14159/2),3.14159/2) :- [7] , [8] , [9].
  [7]    1         pi(3.14159).
  [8]   19         sonar_reading(2,2).
  [9]    8         sonar_direction(2,3.14159/2) :- [10].
 [10]    1            pi(3.14159).
 [11]   27      distance(obj_sk1(2,3.14159/2),2) :- [12] , [13] , [14].
 [12]    1         pi(3.14159).
 [13]   19         sonar_reading(2,2).
 [14]    8         sonar_direction(2,3.14159/2) :- [15].
 [15]    1            pi(3.14159).
Proof time: 1222 inferences in 0.02 seconds, including printing
```

Figure 6: One of the sub-proofs (of existing objects) by the theory.

```
| ?- prove((turn(A);fwd(S))),print('turn angle = '),
        print(A),nl, print('fwd speed = '),print(S)
search for cost 0 proof...  3 inferences so far.
search for cost 1 proof...
⋮
search for cost 9 proof...
[MAIN PROOF]
Proof:
Goal#  Wff#  Wff Instance
-----  ----  ------------
  [0]    0   query :- [1].
  [1]   34      turn(-314) :- [2] , [5].
  [2]   32         heading_angle(-314) :- [3] , [4].
  [3]   41            not_ab_avoid.
  [4]   33            get_move_dir(0,-314).
  [5]   36         need_turn(-314) :- [6] , [7].
  [6]    5            min_angle(5).
  [7]   32            heading_angle(-314) :- [8] , [9].
  [8]   41               not_ab_avoid.
  [9]   33               get_move_dir(0,-314).
Proof time: 12664 inferences in 0.24 seconds,
including printing
turn angle = -314
fwd speed = _6872
```

Figure 7: An example proof by the theory.