LiSA: A Robot Driven by Logical Subsumption

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

Abstract

This paper describes an implemented robot-control system that is based on Brooks-style subsumption [3] of logical theories. It implements Brooks-style subsumption between layers using nonmonotonic reasoning. We describe the control and reasoning algorithms and some of the experiments that we did with the system, running on a Nomad200 robot and a set of computers. Our experimental study shows that commonsense theories and general-purpose first-order logic theorem provers can be used to control real-time agents and robots in particular. Our system improves over traditional subsumption systems in several ways. It allows the user to send new axioms to each of the layers as the robot is running, allowing the user to give advice to the robot and to correct behaviors in runtime. Our system has no voting scheme for deciding on the behavior that should be followed. Instead, the layers work in synergy to provide the compound behavior. Our system improves over other robot-control systems that are based on logic in that it allows full first-order expressivity and that it is fully declarative.

1 Introduction

Logic is promising for AI because it has the ability to represent virtually every domain and problem that humans and other intelligent beings may be interested in. An approach advocated by AI researchers that study logical approaches (e.g., [16]), is to represent the world and the computer's view of it in logical theories and then let the computer use these logical theories to reason and act in the world. This approach has recently been reinvigorated with the successes of several implementations (e.g., [14; 21]). These use logical theories for semantics of robot-control systems and for specifying high-level control. Recently, [1] suggested that the subsumption architecture of [3] may be used to revisit this point of view. Each layer in a subsumption architecture can represent the behavior of the layer with a logical theory. The real-time use of theorem provers or other more specialized reasoners with these theories is then used to control a mobile robot. In this paper we report on an empirical validation of this suggestion.

This paper shows that real-time commonsense control of an AI agent can be achieved with an architecture based on logical theories and general-purpose theorem provers with the use of subsumption. We report on a system that follows these ideas, and a set of experiments executed with this architecture on a mobile robot. Our architecture uses layers of logical theories and theorem provers put in a Brooks-style subsumption system. Each layer performs autonomous theorem proving of a pre-specified goal, sending the result to lower layers. We use nonmonotonic reasoning in each layer to implement subsumption by assuming defaults that may be subsumed when information arrives from higher layers or from the agent's sensors.

The architecture has been implemented and tested on a mobile robot. It exhibits real-time performance and performs navigation and control tasks. The layers can receive and incorporate new axioms from the user at run-time, allowing the user to give advice to the robot and to correct behaviors that are erroneous. The architecture also allows incorporating layers that perform diagnosis and layers that remember experiences for other layers.

Previous work on using subsumption with logical theories and theorem provers [1] has shown that the time taken by theorem provers to prove the needed assertions is low enough to promise real-time control of AI agents, but has presented little empirical evidence of such control on a running system. Our work also improves over results presented using GOLOG (e.g. [14]) and the work of [21] in that our system is fully declarative and has the full expressiveness of FOL. We provide a more detailed comparison to these and other related work at the end of the paper.

2 Logic-Based Subsumption

This section describes how we implement the principles discussed above. The first important idea that this architecture borrows from Brooks' architecture is that of decomposing the domain along behavioral lines into simple layers. Unlike systems that followed Brooks' work, it allows the layers to work in synergy to produce the compound behavior.

2.1 Basic Machinery

A *Logic-Based Subsumption Architecture* (LSA) is built of layers corresponding to behaviors (see Figure 1). The layers work concurrently and asynchronously to each other.

We distinguish four parts of a logical layer: (1) the *body* of the layer, (2) the *sensory and input Latches*, (3) the *output*, and (4) the *default assumptions*. The *body* of the layer is a fixed axiomatization describing the behavior of that layer. The *latches* are used to accept input axioms from the sensors and from higher layers and replace them at the beginning of every cycle (rather than accumulate this input). The *output* is a fixed set of *goal* sentences (possibly with some free variables) whose proof and instantiation determine the behavior sanctioned by the layer's theory (including the latches axioms). The *default assumptions* are used to implement the idea of subsumption between layers. These assumptions are implemented using nonmonotonic reasoning methods, which we describe in more detail in Section 2.2.

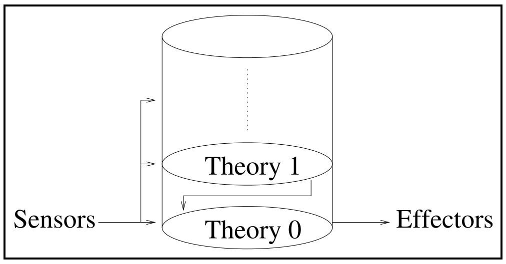

Figure 1: An abstract diagram of the LSA.

A logic-based subsumption system is comprised of a set of layers, each equipped with a theorem prover and concurrently running its own processing loop. The processing loop of each layer proceeds as follows: First, collect any pertinent sensor data and assert it in the form of logical axioms. Simultaneously, assert any inputs from higher-level theories. The theorem prover of that layer then attempts to prove the layer's goal, from the theory including the default assumptions. Upon proving its goal, the layer transmits the goal instantiation to the layer below or (in the case of the lowest layer) to the robot manipulators.

Because the axiomatization of a layer is usually much smaller than that of the whole system, each cycle is less computationally expensive than running one theorem prover over the whole compound axiomatization. As in Brooks' system, lower layers controlling basic behaviors are trusted to be autonomous and do not need to wait on results from higher layers (they assume some of them by default) before being able to respond to situations.

We will typically include a form of nonmonotonicity that is not computationally expensive or that is a fast approximation for a more computationally heavy form of nonmonotonicity. Using a fast form of nonmonotonicity for implementing default assumptions and having lower layers typically having simpler axiomatizations, the cycle time to compute these layers' outputs can be significantly shorter than that of more complex layers.

2.2 Circumscription-Based Subsumption

We use nonmonotonic reasoning to introduce defaults for each layer. Without nonmonotonicity in each layer, goals that were proved once without input from higher layers cannot be rejected upon the introduction of new axioms arriving from higher layers.

An example of a suitable nonmonotonic-reasoning system, is McCarthy's circumscription [17] formula:

$$\begin{aligned}&Circ[A(P,Z);P;Z] = \\ &\quad A(P,Z) \wedge \forall p,z\ (A(p,z) \Rightarrow \neg(p < P))\end{aligned}$$

It says that in the theory A, with parameter relations and function sequences P, Z, P is a minimal element such that $A(P,Z)$ still holds while Z is allowed to vary in order to allow P to become smaller. Roughly speaking, adding this formula allows us to say that the predicate P is true for only those elements for which it must be true. In other words, P is false by default. To state more complicated defaults one can add axioms and predicates. For example, if we want to say that P *is true by default*, then we can add a new predicate symbol, P', and the axiom $\forall x\ P(x) \iff \neg P'(x)$, and minimize P' in the circumscription formula.

Take, for example, the theory

$$A \;\equiv\; block(B_1) \wedge block(B_2)$$

The circumscription of $block$ in A, varying nothing, is $Circ[A;block;] = A \wedge \forall p\ [A_{[block/p]} \Rightarrow \neg(p < block)]$ and is equivalent to $\forall x\ (block(x) \Leftrightarrow (x = B_1 \vee x = B_2))$. By minimizing $block$, we have concluded that there are no other blocks in the world besides those mentioned in the original theory A.

In the LSA, we use circumscription for two distinct tasks: assuming defaults in the layers and giving semantics to the system of layers as one big logical system. The first is the one used to implement subsumption in the actual system, influencing both the semantics and the implementation. The second is needed for giving semantics to the directional nature of the complete system (i.e., that messages between layers go only in one direction).

To implement the idea of subsumption, we let each layer make default "assumptions" about the inputs that later may be adjusted by other (higher-level) layers. These assumptions typically take the form of the Closed-World Assumption (CWA) by minimizing a predicate in the layer's input language (Extended CWA, a generalization of CWA, was shown to be equivalent to circumscription [8]).

More formally, for a set of axioms, A, let $L(A)$ be the set of nonlogical symbols (predicates, functions, and constants) that appear in A. Also, let $\mathcal{L}(A)$ be the FOL language built using the symbols in $L(A)$ (a language here is the set of all FOL sentences that can be built from those symbols). Let $Layer_i$ be the combined theory of layer i, i.e., the combination of the body axioms, $Base_i$, the sensory-latch axioms, $Sensors_i$, and the input-latch axioms, $Input_i$. Let $\vec{C}_i$ be a set of predicates in $\mathcal{L}(Layer_i)$ for which we wish to assert CWA. Then, subsumption is achieved for layer i by using the parallel circumscription policy

$$Circ[Layer_i; \vec{C}_i; L(Layer_i)] \tag{1}$$

When implemented, this formula often 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. Other systems for nonmonotonic reasoning can also be used instead of circumscription, depending on the intended behavior and the designer's choice of tradeoffs (e.g., time versus expressivity).

2.3 Putting It All Together

In the case of using Circumscription for nonmonotonic assertions, each layer tries to prove

$$Circ[Layer_i; \vec{C}_i; \vec{Z}_i] \models \exists \vec{x} Goal_i(\vec{x})$$

Here, $\vec{C}_i, \vec{Z}_i$ are specified as part of the defaults for layer i, $Layer_i$ is the set of axioms including the body and the latches and $Goal_i(\vec{x})$ is a goal formula specified for layer i ($\vec{x}$ is a vector of variables open in $Goal_i(\vec{x})$). Upon proving $Goal_i(\vec{a})$, the layer transmits $Goal_i(\vec{a})$ either to the layer below or (in the case of the lowest layer) to the robot manipulators. Figure 2 summarizes this algorithm, while Figure 3 illustrates the process.

PROCEDURE LSA($\{Layer_i\}_{i\leq n}$, $\{Goal_i\}_{i\leq n}$)
$\{Layer_i\}_{i\leq n}$ a layered theory T, $Goal_i$ a fixed goal in $\mathcal{L}(Layer_i)$ ($i \leq n$).

Concurrently, for each layer, i:

1. Request sensory data from the robot and assert it into the Sensory Latch, $Sensors_i$.
2. Combine axioms in the Body theory with those in the sensory and input latch: $Layer_i \leftarrow Base_i \cup Sensors_i \cup Input_i$.
3. Let the theorem prover for this layer attempt to prove $Circ[Layer_i; \vec{C}_i; \vec{Z}_i] \models \exists \vec{x}\ Goal_i(\vec{x})$ (i.e., from $Layer_i$ given the default assumptions).
4. If $Goal_i(\vec{a})$ was proved for assignment $\vec{a}$, assert $Goal_i(\vec{a})$ in the Input Latches of layer $i-1$, $Input_{i-1}$.

Figure 2: The LSA algorithm.

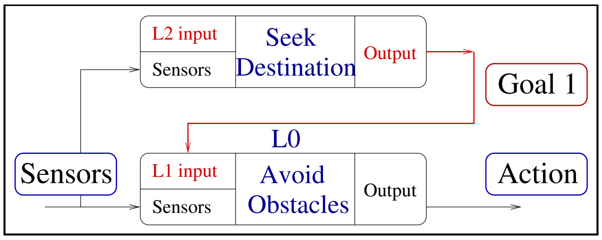

Figure 3: A detailed look at two layers.

This description of LSA hides two issues: First, what happens when a layer cannot prove something? Second, what happens to the input latch of a receiving layer after some time has passed? For the first question, in general we assume that the theorem prover for each layer works without interruptions until it finds a proof. If the theorem prover did not find a proof after some timeout, we restart the prover (possibly on a different sub-space of the search space) with the new latch information. Alternatively, one can assume that the sensory latch and the input latch are refreshed asynchronously, and the prover immediately takes any new information into account, discarding any old information from that latch (and any of the consequences it may have made on the basis of the old latches). For the second question, we assume in this paper that latch information disappears after some time. Thus, if layer 1 did not prove its goes for the last few seconds, then layer 0 will no longer consider an axiom sent previously by layer 1 as valid.

3 Logical Layers for a Mobile Robot

In this section we describe the logical theories used in a control system we have implemented for a Nomad200 mobile robot operating in a multi-story office building. All theories but layer 1 are significant extensions and modifications of those presented in [1]. The Nomad200 is a cylindrical robot with sonar sensors on its perimeter, wheels that control its motion, and encoders that compute an estimate of the robot's position and angular heading (see Figure 4).

Figure 4: Nomad 200

The system includes five logical layers. Each layer's body theory can be seen as having two main parts that we categorize roughly into sensory-focused and goal-focused. Most theories also include other axioms that describe domain-dependent relationships in the world. This distinction into parts is not reflected in the system or in its behavior at the moment. The following description uses upper-case letters to denote variables and lower-case letters for constants. We describe the theories but not the goals or defaults taken. Those are similar to those specified in [1].

3.1 LAYER 3: Wide-Range Motion Planning

The top layer, *layer 3*, is responsible for high-level robot motion planning. The theory can be seen as comprising of three main parts: sensory-focused, goal-focused and spatial relationships in the world. The goal-focused part represents the effects of robot motions in situation calculus [19]. There is only one fluent, the robot's location, and only one action schema, $moveto(L)$, where L is a location variable. For this simple situation calculus theory it is convenient to consider the actions as having duration and the situations as histories of actions from the initial situation, $S0$. This theory has a single effect axiom:

$$\forall L0, L, S\ at(r, L0, S) \wedge visuallyConnected(L0, L) \Rightarrow at(r, L, result(moveto(L), S))$$

where $visuallyConnected(L0, L)$ means that there is a line of sight between L and $L0$. No frame axioms or explanation closure axioms are needed, as this effect axiom specifies the value of the only fluent in our theory.

$S0$ is considered to be the situation the robot is in when the layer receives the sensory and other input axioms. The sensory-focused part of the theory includes a representation of the relationships between landmarks in the world and the cartesian coordinates supplied by the robot's odometry sensors. Among other things, the robot knows when it is between landmarks using the axiom

$$\begin{aligned}&visuallyConnected(P1, P2) \land curr\_loc(X, Y) \land \\ &\quad cartesian(P1, C1) \land cartesian(P2, C2) \land C1 \neq C2 \land \\ &\quad pos\_between(C1, [X, Y], C2) \Rightarrow \\ &\qquad current\_landmark(between(Pos1, Pos2))\end{aligned}$$

where $[X, Y]$ is considered between $C1, C2$ if it is close enough to the straight line passing through them.

Axioms for spatial relationships describe the relationships between rooms, room entrances, corridors, floors and elevators. For example, rooms are visually linked to their entrances, and locations that are in the same corridor are visually connected as well. Other axioms describe invariants of the domain, such as the commutativity of $visuallyConnected$ and the fact that a position between two visually connected positions is visually connected to both positions.

3.2 LAYER 2: Local Action Planning

Layer 2 is responsible for translating target landmarks into cartesian coordinates for the robot, and for planning and low-level interaction and control with the elevators. The theory can be seen as comprising of four main parts: sensory-focused, motion-focused, elevator-focused and spatial relationships in the world.

The motion-focused subtheory uses a map and a simple axiom to translate landmarks to cartesian locations. The lower layer, layer 1, does not know about the existence of floors, and the move command that is to be executed by lower layers does not include a floor parameter.

$$\begin{aligned}&target\_landmark(L) \land \neg elevator\_related(L) \\ &\qquad \land cartesian(L, [X, Y]) \Rightarrow move\_cmd(X, Y)\end{aligned}$$

The elevator-focused subtheory is a situation calculus theory with four main fluents: the location of the robot, the locations of the two elevators and whether two locations are visually connected. There are three action schemas: $moveto(L)$, which moves the robot to location L; $callElev$, which calls the elevator; and $orderElev(floor(F))$, which commands the elevator to go to floor F. There are three effect axioms for the three schemas. For example,

$$\begin{aligned}&\forall F, S\ (at(r, front(elev(floor(F))), S) \Rightarrow \\ &\quad \exists S2, E\ temporally\_close(result(callElev, S), S2) \land \\ &\qquad elevator(E) \land at(E, floor(F), S2))\end{aligned}$$

says that at some point after calling the elevator one of the elevators (there are two) would come. For this situation calculus theory we need some frame or explanation-closure assumptions. Since the number of effect axioms and fluents is small, there is no harm in specifying frame axioms, such as

$$\begin{aligned}&\forall F, S\ (at(r, front(elev(floor(F))), S) \Rightarrow \\ &\quad at(r, front(elev(floor(F))), result(callElev, S)))\end{aligned}$$

The sensory-focused and spatially-focused theories are similar to the ones used in *layer 3*. The main difference from *layer 3* is that the property of two locations being visually connected is now dependent on the situation (the elevators may be connected to their entrances or not).

3.3 LAYER 1: Destination-seeking

Layer 1 supports simple movements towards a goal location. Given the orientation of the robot and the distance and direction to the goal location, it concludes the existence of a *pushing object* in a particular location close to the robot (later, layer 0 will use this extra object to guide the robot in a particular direction).

The theory can be seen to have two main parts: sensory-focused and goal-focused. The sensory-focused part translates the subjective odometry and direction of the robot to a global view of the robot in the world. The goal-focused part uses the goal location, the distance to the goal and a set of quadrants to decide where to place a pushing object (if at all). For example,

$$\begin{aligned}&\forall X, Y, X2, Y2, Qd\ \ curr\_loc(X, Y) \land dest(X2, Y2) \land \\ &\quad quadrant(X{-}X2, Y{-}Y2, Qd) \Rightarrow push\_object(Qd)\end{aligned}$$

3.4 LAYER 0: Obstacle-avoidance

Layer 0 is responsible for deciding what low-level action the robot should perform. The theory has two main parts: sensory-focused and control-focused. The sensory-focused part considers sensory input only from the sonars. It takes its input, asserted in the form of the axiom schema $SonarReading(sonar\_number) = dist$, from the physical sonars and translates it into a map of objects, recording their distance and direction (relative to the robot).

The control-focused part decides which of the actions to perform, summing up the forces that the different objects around the robot exert on it (forces that are correlated to the distances of the objects to the robot). It uses the resulting force to determine whether the robot should turn or move forward and with what velocities so as to maximally avoid the objects.

$$\begin{aligned}&\forall A, S, Asp\ heading\_angle(A) \land need\_turn(A) \land \\ &\qquad heading\_speed(S) \land need\_fwd(S) \land \\ &\qquad\quad angle\_speed(Asp, A) \Rightarrow turn(Asp)\end{aligned}$$

This axiom uses $heading\_speed(S)$ and $need\_fwd(S)$ to prevent the robot from turning in one direction and then a different direction due to sensory noise (only if we need to go to a particular place will we turn to get there).

3.5 LAYER -1: Halt or Go

The lowest layer, *layer -1*, is responsible for sending actions to the robot or halting the robot. It has a theory similar to *layer 0* but with additional axioms that check if there are objects that the robot is about to collide with.

$$\begin{aligned}&object(Obj) \land distance(Obj, Dist) \land Dist < min\_dist \land \\ &\quad direction(Obj, Dir) \land (Dir < \tfrac{1}{3}\pi \lor Dir > 1\tfrac{2}{3}\pi) \Rightarrow \\ &\qquad object\_ahead\end{aligned}$$

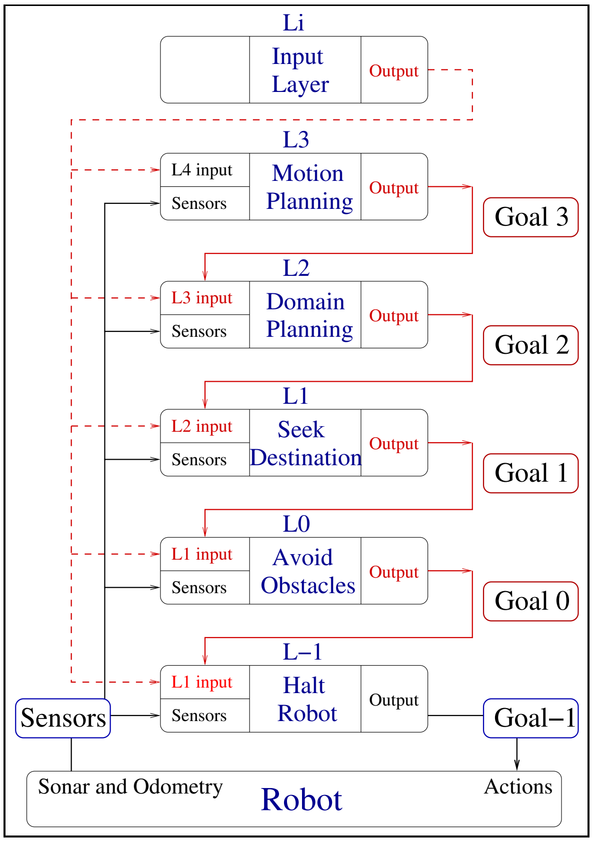

Figure 5: Diagrammatic view of an LSA system controlling a robot.

4 Implementation on a Mobile Robot

4.1 The Software and Dynamics of the System

We have implemented the above architecture using the PTTP theorem prover ([22]) on a cluster of Sun SuperSparc 1 stations running SWI Prolog or Quintus Prolog[1] as the underlying interpreter for PTTP. The system runs on a Nomad 200 robot.

PTTP (Prolog Technology Theorem Prover) is a model-elimination theorem prover using iterative deepening in the proof space. Given a theory made of clauses (not necessarily disjunctive) without quantifiers, PTTP produces a set of Prolog-like Horn clauses, ensures that only sound unification is produced, and avoids the negation-as-failure proofs that are produced by the Prolog inference algorithm. It is sound and complete for refutation in FOL (general first-order sentences are translated into clausal form in the usual way, using Skolemization).

Our implementation[2] is written in C++ with classes allowing prover-specific implementation: *Layer* is the superclass of *Layer_qp* (Quintus prolog with PTTP), *Layer_swi* (SWI prolog with PTTP) and *Layer_input* (a layer used in an executable that allows the user to send new axioms to the other running layers). An executable consists of a *Layer* object (the central piece of the executable), objects for communication (TCP/IP), and an embedded theorem prover (in the case of PTTP, this means an object file consisting of a prolog implementation and a compiled PTTP). There is a separate layer for the communication with the robot that translates *layer -1*'s proven goals to robot motion commands, and sends sensory information to the layers on request.

Each time a layer is run (with whichever theorem prover implementation) a configuration file specifies the theory it should initially load and the communication pattern of the layer. The communications part specifies the layers from which it should accept axioms, the host/port of that layer, and the mode of communication, i.e., whether it is synchronous (request data explicitly) or asynchronous (use the data in the latch).

After initializing the communication, it initializes the theorem prover and loads the body theory into it. Then it runs the following infinite loop: First, the layer reads the messages that are on the ports and asserts the latest ones from each port into the theorem prover; then, the layer attempts to prove the goal; finally, upon successful conclusion, it sends the result of the proof to listening layers below it.

The information that the layer reads from the ports overrides previous latch data. However, if no information has arrived on a specific port, it reuses the information that arrived previously. This allows the layers to work at different frequencies without confusing delay in communication or computation for a directive that overrides the latest information from that layer. To avoid ambiguity, most layers prove *layer_i_failed* if they failed to prove the goal (depending on the defaults asserted for each layer). In that case, this proved assertion is sent to the listening layers, which use this message to override previously received assertions.

4.2 LiSA's behavior

We ran several experiments on LiSA with goals of traveling to different rooms in the building. Figure 6 presents average total time measurements for each layer during these experiments (the final paper adds other statistics and a breakdown of timing according to tasks/scenarios). This figure shows that the current implementation is somewhat slow for a truly reactive behavior. Currently we are working on speeding up several components of the system. We plan to report on the improved behavior in the final version of this paper.

Each experiment with LiSA starts with running the logical layers, the nomad layer and the input layer. LiSA has a given map that is used by layers 2 and 3. We reset the robot in a position and heading that matches this map. Using the input layer, we tell the robot that its goal is to navigate to one of the rooms or across the lab. It takes from a few seconds to a minute before layer 3 finds a plan from its current location, and sends a goal landmark to layer 2. Layer 2 instantaneously translates the landmark into a goal location and sends it to layer 1. Layer 1 then provides a pushing object to layer 0. Layer 0 sends a motion command to layer -1, and layer -1 executes it, if there are no direct obstacles in front of it. The

[1] Quintus Prolog 3.4 has a bug that prevents it from running with some of the layers without crashing. For some other layers it is faster than SWI, though.

[2] The implementation of this system can be found at http://www-formal.stanford.edu/eyal/lsa/.

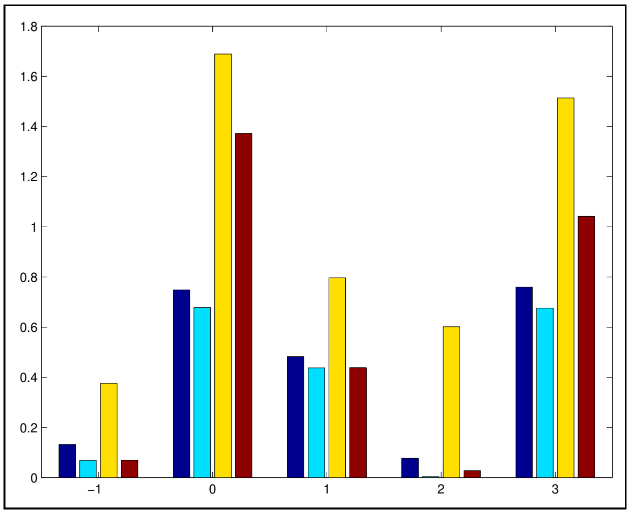

Figure 6: Average time (in seconds) per cycle for each of the layers[4]. The bars for each layer correspond to total cpu-time per cycle, cpu-time for proofs in a cycle, total clock time per cycle and clock time for proofs in a cycle, in this order from left to right.

robot typically starts turning until it faces in the direction that it intends to go and then proceeds forward towards that target. The transition between turning and moving forward is smooth and without delays.

In our experiments, LiSA took from 30 seconds to two minutes to move from one landmark to the next, with landmarks that are approximately 4-10 meters apart and with obstacles that are close to its path. When we put obstacles such as chairs and humans in front of the robot, it managed to go around them without colliding or hesitating.

In our earlier experiments, before we improved the theory of layer 3, LiSA sometimes got *lost*. If an obstacle was put in LiSA's path and she had to go around it, the next time layer 3 tried to plan a path to the goal it sometimes did not know where LiSA was, as she was not close to any landmark (and may have moved significantly away from the path to the next landmark). In that case, we were able to use the input layer and send an axiom into LiSA's layer 3 telling it that LiSA was between the two landmarks. Currently, LiSA's layer 3 re-uses the last proven goal landmark as the default that is sent to layer 2 if layer 3 fails. This sidesteps the problem, but is not consistent with our semantics, as our theoretical layers have no *memory* or *state*. We plan to extend the system and the semantics using models of belief update to allow memory and belief change in a consistent manner.

[4]There is some discrepancy between these measurements and the real-time behavior of the system. These measurements are given for the system running with all logs registering the advance of proofs and other messages that are required to collect statistics for these runs. These mechanisms typically slow the system down by a factor of about 4.

4.3 Tuning the System

Theorem provers are notoriously slow, which is one of the main reasons they usually are not used for time-sensitive applications. However, in this implementation we were able to sidestep this difficulty by using only small and simple theories and using a fast approximation of nonmonotonic reasoning to conclude defaults. We attribute the speed achieved by our system to several optimizations that we mention below.

During our experiments it became clear to us that much of the bottleneck in some layers was in concluding that there is no proof. Without this conclusion, the layer cannot terminate the cycle and start a new cycle (with new sensory information and new latch information). For this reason, each layer has an associated limit on the depth of the search allowed for a proof of the goal. If no proof is found up to this depth, we conclude that for any run-time purpose there is no proof. This depth limit is determined experimentally and is specified together with the goal formula.

Depth limit is also used to implements a rough approximation for defaults. For example, layer 0 implements the CWA for objects by aggregating all the objects that it can find up to a specified proof depth. Under the same assumption we consider the depth limit on the proof of the goal as a default that says that the goal is false by default.

The same mechanism allows us to provide *islands* in the search space of the prover. For example, in layer 3 we first try to prove that the robot is currently at a particular location. If we succeed, we add the assertion $at(r, CurrLandmark, s0)$ as a temporary new axiom to the prover, where $CurrLandmark$ is the landmark that we found the robot to be at. This sometimes cuts the depth of the proof search space by 10, which has a significant influence on the proof time (cuts the proof time in those cases by a factor of approximately 10).

Applying caching to the proof of *get_force* has a similar effect. Since every proof in layer 0 re-proves *get_force* many times, caching improved the performance of Layer 0 significantly (from approximately 10 seconds to 0.1 seconds per proof on a Sun UltraSparc 60).

Finally, we use a semantic attachment in Layer 0 for the predicate get_force. It is embodied in a C function that returns the force vector $[Strength, Direction]$. It calls Prolog's setof operator to collect all the objects for which existence proofs can be found, then computes the sum of the forces contributed by each object. This CWA is achieved by limiting proofs to be no longer than a specified constant, as described above.

5 Related Work

Compared to other approaches to robot-control that use logic, LiSA is the only one using full FOL theorem provers for reasoning. This is the first presentation of a robot-control architecture that is built on theorem provers and is suitable for realizing complex tasks in real time.

Shanahan [21] describes a map-building process using abduction, but then implements his theory in an algorithm that is proved to have an abductive semantics. Baral and Tran [2] define control modules to be of a form of Stimulus-Response

(S-R) agents, relating them to the family of action languages $\mathcal{A}$ (e.g., [7]). They provide a way to check that an S-R module is correct with respect to an action theory in $\mathcal{A}$ or $\mathcal{AR}$ and provide an algorithm to create an S-R agent from an action theory. Systems based on the GOLOG project (e.g., [14; 9]) have a planner that computes/plans the GOLOG program off-line, only later letting the robot execute the nondeterministic GOLOG program on-line. Logic and situation calculus ([19]) are used to give semantics for GOLOG programs. Another, somewhat earlier line of work, concentrated on compilation of logical description into control languages (e.g., [11; 12; 13]).

Compared to this work, our system is the first to allow declarative representation and reasoning in real time, enabling the full power of first-order logic. Also, being able to send logical-formulae advice to the robot at run-time is a property that has not seen (to our knowledge) since Shakey the robot [20].

Compared to subsumption systems for robot control (e.g., [4; 5; 15; 10; 6]) our system allows the user to send new axioms to each of the layers as the robot is running. This allows the user to give advice to the robot and to correct behaviors in runtime. In addition, our system has no voting scheme for deciding on the behavior that should be followed. Instead, the layers work in synergy, sending messages to each other, together providing the compound behavior.

Clearly, we have not solved the age-old problems with using theorem provers, and there are limitations to our approach. However, with proper tuning and given recent advances in automated reasoning, this kind of system seems to support high-level reasoning that is still reactive, offering a major advantage to robotic systems and systems that wish to perform commonsense reasoning online.

6 Conclusion

We have shown that theorem provers can be used for robot control by employing them in a layered architecture. We have demonstrated that the architecture and the versatility of theorem provers allow us to realize complex tasks, while keeping individual theories simple enough for efficient theorem proving.

There are many important avenues for extending this system. *Memory* and *state* can be added to the system easily, but they do not fit the current semantics. We plan to use belief update semantics to extend this framework and allow such modifications as defaults that change according to the beliefs of the robot and diagnosis of the robot behavior and whereabouts for determining its location. Also, we wish to create such reactive systems automatically from first-order theories that describe the intended behavior. We plan to pursue these goals in the near future.

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. We also thank Nils Nilsson and Jean-Claude Latombe for allowing us to use their Nomad 200 robots, and to Héctor González-Baños for a lot of help and advice with using (and occasionally fixing) the robots. This research was supported by an AFOSR grant AF F49620-97-1-0207, and by a National Physical Science Consortium (NPSC) fellowship.

References

[1] E. Amir and P. Maynard-Reid. Logic-based subsumption architecture. In *Proc. Sixteenth International Joint Conference on Artificial Intelligence (IJCAI '99)*, pages 147–152, 1999.

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

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

[4] R. A. Brooks. Elephants don't play chess. *Journal of robotics and autonomous systems(1-2)*, 6:3–15, June 1990.

[5] R. A. Brooks and A. M. Flynn. Robot beings. In *Proceedings of the IEEE/RSJ Int'l Conference on Intelligent Robotics and Systems (IROS-89)*, pages 2–10, 1989.

[6] R. A. Brooks and L. A. Stein. Building brains for bodies. *Autonomous Robots*, 1(1):7–25, 1994.

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

[8] M. Gelfond, H. Przymusinska, and T. C. Przymusinski. On the relationship between circumscription and negation as failure. *Artificial Intelligence*, 38(1):75–94, Feb. 1989.

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

[10] I. Horswill. Polly: A vision-based artificial agent. In *Proceedings of the 11th National Conference on Artificial Intelligence*, pages 824–829, Menlo Park, CA, USA, July 1993. AAAI Press.

[11] L. P. Kaelbling. REX: A symbolic language for the design and parallel implementation of embedded systems. In *Proceedings of the AIAA conference on computers in aerospace*, pages 255–260, 1987.

[12] L. P. Kaelbling. An architecture for intelligent reactive systems. In J. Allen, J. Hendler, and A. Tate, editors, *Readings in Planning*, pages 713–728. Morgan Kaufmann publishers Inc.: San Mateo, CA, USA, 1990.

[13] L. P. Kaelbling and S. J. Rosenschein. Action and planning in embedded agents. *Robotics and Autonomous Systems(1–2), June 1990*, 6:35–48, 1990.

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

[15] M. J. Matarić. Integration of representation into goal-driven behavior-based robots. *IEEE Transactions on Robotics and Automation*, 8(3):304–312, June 1992.

[16] J. McCarthy. Programs with common sense. In *Mechanisation of Thought Processes, Proceedings of the Symposium of the National Physics Laboratory*, pages 77–84, London, U.K., 1958. Her Majesty's Stationery Office. Reprinted in McC90.

[17] J. McCarthy. Applications of Circumscription to Formalizing Common Sense Knowledge. *Artificial Intelligence*, 28:89–116, 1986. Reprinted in [18].

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

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

[20] N. J. Nilsson. Shakey the robot. Technical Report 323, SRI International, Menlo Park, California, 1984.

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

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