Logic-Based Subsumption Architecture: Empirical Evaluation (Status Report)

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

Abstract

We describe an implementation of the Logic-based Subsumption Architecture (LSA) [Amir and Maynard-Reid, 1999], an AI architecture inspired by Brooks' subsumption architecture but based on logic. In this architecture, layers of control are axiomatized in First-Order Logic (FOL) and independent theorem provers derive each layer's goals given its inputs. Default assumptions in the lower layers implement subsumption by higher layers. These assumptions are nonmonotonically retracted when higher layers draw new conclusions. We describe four layers designed for the task of robot control and experiments that empirically show the feasibility of using fully expressive FOL theorem provers for robot control with 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 a sequential, functional form. Within this setting, he introduced the idea of subsumption, that is, that more complex layers not only depended on lower, more reactive layers, but could also influence their behavior. The resulting architecture was one that could simultaneously service multiple, potentially conflicting goals in a reactive fashion, giving precedence to high-priority goals. The architecture was further developed into the *behavior language* [Brooks, 1990].

The subsumption architecture is not easily used for developing intelligent agents, despite its attractive properties. It is procedural, making it difficult to incorporate representations of knowledge and implementations of higher-level reasoning. It also makes the semantics of resulting systems unclear which introduces scaling problems. Furthermore, implementations of subsumption are restricted to prioritization of the actions derived by each layer [Arkin, 1998], as opposed to more generally allowing layers to *influence* other layers' behaviors. Finally, the architecture is specifically oriented towards robot control and is not easily applicable to 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 terms by [Minsky, 1985] and [Stein, 1997].

In this paper we describe an implementation of a *Logic-Based Subsumption Architecture* (LSA) that was introduced in [Amir and Maynard-Reid, 1999]. The LSA is modeled in the spirit of Brooks' subsumption architecture but relies on a logical framework and has wider applicability and extendibility. It includes a set of First-Order Logic (FOL) 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 allow each layer to make nonmonotonic assumptions, which can be later retracted when higher layers introduce new axioms/messages into this layer. This makes each layer's performance essentially independent of the performance of other layers. Together with Brooks' idea of having each layer focus on a separate, simple goal, this enables the overall system to be reactive.

We demonstrate these claims using four layers modeled for the task of a mobile robot control. We describe the architecture, the software built and the theorem prover used. We also discuss what demands our architecture raises for theorem provers. We report running times measured for the different layers in our architecture and the behavior of the overall system. We observed that the layer in greatest need of reactivity is sufficiently fast (0.1–0.2 seconds per control-loop cycle). This is particularly surprising given the common belief that general-purpose FOL theorem provers are not suitable for implementing intelligent agents. This system also allows the incorporation of decomposed theories in a subsumption architecture in the sense of [Amir and McIlraith, 2000; Amir, 2000].

The remainder of the paper is organized as follows: After giving a brief introduction to Brooks' system and behavioral decomposition, we describe the LSA. We then describe the robot control system we have implemented using the architecture. We conclude with a discussion of implementation issues, comparisons to related work and a description of future directions.

2 Subsumption and Decomposition

2.1 Brooks' Subsumption Architecture

Brooks showed that it is often advantageous to decompose a system into parallel tasks or behaviors of increasing levels of competence rather than the standard functional decomposition. Whereas a typical functional decomposition might resemble the sequence

> sensors $\rightarrow$ perception $\rightarrow$ modeling $\rightarrow$ planning $\rightarrow$ task recognition $\rightarrow$ motor control,

Brooks would decompose the same domain as

> 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 can be the product of many simple mechanisms interacting with each other and 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.

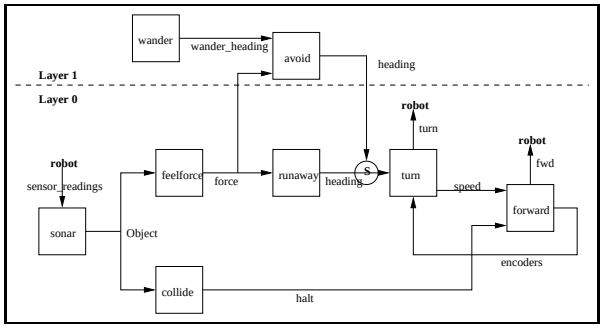

Figure 1: Layers 0 and 1 of Brooks' subsumption architecture robot control system.

In general, the different layers are not completely independent. In the decomposition above, wandering and exploring depend on the robot's ability to avoid objects. But the system may be able to service multiple goals in parallel, despite the dependence. The goals of one layer will occasionally conflict with those of another layer, in which case higher-priority goals should override lower-priority ones. Consequently, the subsumption architecture provides mechanisms by which higher, more competent layers may observe the state of lower layers, inhibit their outputs and override their inputs, thus adjusting their behavior. High-priority tasks in lower layers (such as reflexively halting when an object is dead ahead) will still have a default precedence if the designer disallows any tampering with these particular tasks.

Brooks implemented a control system of layers corresponding to the first three levels of competence described above (avoidance, wandering and exploration). The first two layers are shown in Figure 1. Briefly, the `avoid` layer endows the robot with obstacle avoidance capabilities by moving it in directions that avoid obstacles as much as possible and forcing it to stop if a head-on collision is imminent. The `wander` layer causes the robot to move around aimlessly when it is not otherwise occupied. The `explore` layer gives the robot some primitive goal-directed behavior by periodically choosing a location in the distance and heading the robot towards it if idle. While in explore mode, this layer inhibits the `wander` layer so that the robot remains on track towards its destination. When either the `wander` or the `explore` layer is active, it overrides the default heading computed by the `avoid` layer, but the `avoid` layer still ensures that the robot does not have a collision. We refer the reader to [Brooks, 1986] for further details.

2.2 Behavioral Decomposition in the LSA

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. A *Logic-Based Subsumption Architecture* (LSA) is composed of a sequence of FOL theories. Each represents a layer with an axiomatization of the layer's behavior, that is, the layer's inputs, outputs (goal), state and any dependencies between them. The inputs are axioms coming from either the sensors or higher layers. The outputs are proved theorems determined by running a separate theorem prover for that layer only. These outputs may be sent to lower layers or to the robot effectors.

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, leading to an overall higher performance. Another advantage of the layer-decoupling is the possibility of achieving more reactive behavior. 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. Because these layers typically have simpler axiomatizations, and given the default assumptions, the cycle time to compute their outputs can be shorter than that of the more complex layers.

2.3 Subsumption Principles

Of course, the layers are *not* fully independent. We adopt the view that, together with the task-based decomposition idea, the coupling approach represented by subsumption in the subsumption architecture is an important and natural paradigm for intelligent agents in general, and robot control in particular (see [Stein, 1997]). We want each layer in an LSA to be able to communicate with those underneath it in the hierarchy.

In general, however, when one layer overrides another, the two disagree on what some particular input should be. In a classical logic setting, 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, both layers keep their facts and the higher

layer asserts its relevant conclusions in the lower layer, and (b) if there is conflict, the lower layer tries to give up some assumptions to accommodate the higher layer's conclusions. A number of techniques developed in the logic community are applicable, e.g., nonmonotonic techniques and belief revision. We have chosen to use the nonmonotonic system of circumscription, although other approaches may be equally interesting and appropriate.

3 Logical Subsumption

This section describes in detail how we implement the principles discussed above.

3.1 Basic Machinery

We distinguish three parts of the logical theory associated with each layer: (1) the *Body* of the layer, (2) the *Sensory and Input Latches*, and (3) the *Output*. The *Body* of the layer is the invariant theory for that layer. The *Latches* are used to accept the input and replace it at the beginning of every cycle (rather than accumulate it). The *Output* is the set of *goal* sentences proved from the layer's theory (including the latches).

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. Upon reaching the conclusions, transmit the relevant ones either to the layer below or (in the case of layer 0) to the robot manipulators. Figure 2 illustrates this process.

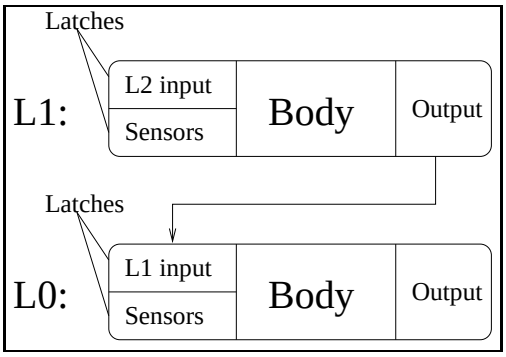

Figure 2: A detailed look at two layers.

3.2 Circumscription-Based Subsumption

In the logical paradigm, McCarthy's circumscription [McCarthy, 1986] is one of the first major nonmonotonic reasoning tools. McCarthy's circumscription formula

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

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

Take, for example, the theory $T \equiv block(B_1) \wedge block(B_2)$. The circumscription of $block$ in T, varying nothing, is $Circ[T;block;] = T \wedge \forall p\,[T_{[block/p]} \Rightarrow \neg(p < block)]$ 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 T.

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. We do not elaborate on the semantics here. The reader is referred to [Amir and Maynard-Reid, 1999] where the LSA is given semantics using prioritized circumscription [Lifschitz, 1989].

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 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 [Gelfond *et al.*, 1989]).

During each cycle of any particular layer, before applying any default assumptions, we include those sentences that were asserted in the layer's latches. Then, the layer's theorem prover can try to prove the appropriate goal for that layer. This grants higher layers the right to override those default assumptions.

More formally, let $Layer_i$ be the theory of layer i, and C_i, 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

$$Circ[Layer_i; C_i; \mathcal{L}(Layer_i)]$$

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 5.

4 Logical Theories for the Layers

We briefly describe the logical theories for a control system we have implemented for a robot operating in a multi-story office building. The first two layers correspond roughly to the first three layers in Brooks' system. The layers are diagrammatically described in Figure 3.

For simplicity, we list only selected axioms from the theories. The system is used to control 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.

4.1 LAYER 0: Obstacle-avoidance

Layer 0 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)[1]. It may also dis-

[1]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 $NSONARS - 1$.

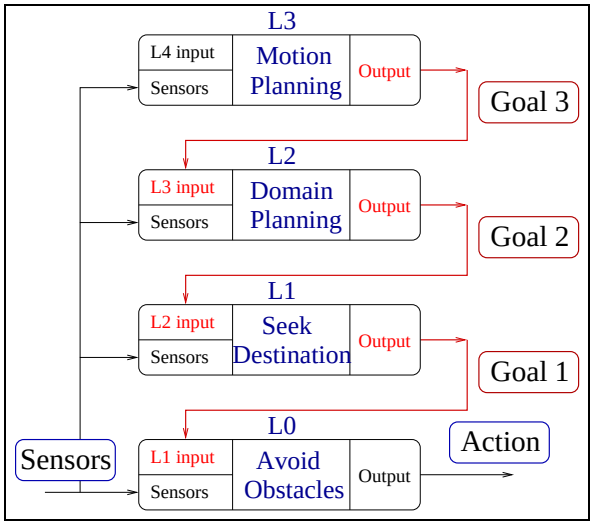

Figure 3: Diagrammatic view of an LSA system controlling a robot.

cover "virtual" objects by way of layer 1's subsumption latch.

$$\forall dist, dir.\ (\exists sonar_number.\ SonarReading(sonar_number) = dist \wedge SonarDirection(sonar_number) = dir \wedge dist \geq 0 \wedge dir > -\pi \wedge dir \leq \pi) \Rightarrow (\exists obj.\ Object(obj) \wedge Distance(obj) = dist \wedge Direction(obj) = dir).$$

Layer 0 checks to see if it has detected objects lying directly in front of it, and halts the robot if it has.

$$ObjectAhead \iff (\exists obj.\ (Object(obj) \wedge Distance(obj) < MIN_DIST \wedge Direction(obj) = dir \wedge dir > -\tfrac{\pi}{4} \wedge dir < \tfrac{\pi}{4}).$$

$$ObjectAhead \Rightarrow HaltRobot.$$

Using the map, layer 0 executes the function *GetForce*, computing the combined "repulsive force" exerted on the robot by the detected objects as *Force_direction* and *Force_strength*. It uses the former to specify a heading angle for the robot away from this force. Once headed in the right direction, the robot is commanded to move away at a speed proportional to the strength of the force, slowing down as it moves farther away from the objects.

$$Heading_angle = (Force_direction \bmod 2\pi) - \pi.$$
$$Heading_speed = Force_strength.$$

$$NeedTurn(Heading_angle) \Rightarrow Turn(Heading_angle).$$

$$\neg HaltRobot \wedge \neg NeedTurn(Heading_angle) \wedge NeedFwd(Heading_speed) \Rightarrow Fwd(Heading_speed).$$
$$HaltRobot \vee NeedTurn(Heading_angle) \Rightarrow Fwd(0).$$

During each cycle of layer 0, it applies the CWA to the symbols *HaltRobot*, *Object*, *Distance*, *Direction* in the input language. It then uses its theorem prover to try to prove both $Fwd(speed)$ and $Turn(angle)$, where *speed* and *angle* are instantiated by the proof. The results are translated into the appropriate actuator commands.

4.2 LAYER 1: Destination-seeking

Layer 1 supports simple movements towards a goal location, more closely resembling the exploration layer of Brooks' system than the wandering layer. The sensory inputs that it considers are the current location and orientation data from the robot's encoders (or any other source, such as encoder data corrected using a gyroscope or compass [Borenstein and Feng, 1996a; 1996b]).

Given a particular pair of coordinates specified by the input $Destination(X, Y)$ from layer 2 (or an outside source) and given the robot's current location and orientation,[2] it determines in which of the eight quadrants surrounding the robot this goal position is, and asserts the existence of a *virtual pushing object* in the opposing quadrant. If $Destination(X, Y)$ cannot be concluded or the robot is close enough to that target position, it concludes that there is no need for a *pushing objects*.

$$Object(PUSH_OBJECT).$$

$$\forall x_0, y_0, x, y.\ CurrLoc(x_0, y_0) \wedge Destination(x, y) \Rightarrow HasPushObject(Quadrant(x_0 - x, y_0 - y)).$$

$$\forall quad.\ HasPushObject(quad) \Rightarrow Direction(PUSH_OBJECT) = quad * \tfrac{2\pi}{NQUADS} \wedge Distance(PUSH_OBJECT) = PUSH_OBJ_DIST.$$

During each cycle, layer 1's theorem prover attempts to prove $Object(obj)$, $Direction(obj) = dir$, and $Distance(obj) = dist$, and introduces them into layer 0's input latch if successful. The avoidance capabilities of layer 0 effectively push the robot away from the object in the direction of the goal, although it may deviate from a direct path if there are physical objects in the vicinity.

4.3 LAYER 2: Local Motion/Action Planning

Layer 2 performs two tasks: (1) translate logical locations into Cartesian coordinates and (2) reason in the situation calculus [McCarthy and Hayes, 1969] about small-range movements and using the elevators.

The inputs for this layer are the current location data from the robot ($CurrLoc$) and the output from layer 3 ($TargetLandmark$). The situation calculus theory includes three fluents: the two elevators' locations and the robot's location (we explicitly state frame axioms). Since the domain and depth are small, planning here is simple.

$$Cartesian(RM218, [0, 0]).$$

$$\forall x, y, lm.\ \neg NeedElevPlan \wedge TargetLandmark(lm) \wedge Cartesian(lm, [x, y]) \Rightarrow Destination(x, y).$$

$$\forall e, f, s. \neg Elevator(e) \vee \neg At(e, Floor(f), s) \vee VLinked(e, ElevEntr(e, Floor(f)), s)$$

$$\forall f, s. At(Robot, Front(Elev(Floor(f))), s) \Rightarrow At(Robot, Front(Elev(Floor(f))), Result(CallElev, s)) \wedge \exists s', e. (\neg TempClose(Result(CallElev, s), s') \vee Elevator(e) \wedge At(e, floor(f), s'))$$

[2] These coordinates are with respect to the fixed coordinate system of the domain.

During each cycle, layer 2 tries to plan the motion to the next landmark and to prove $Destination(next_landmark_x, next_landmark_y)$. It does so by finding a plan that brings the robot to $TargetLandmark$. If the action concluded by the robot is either $CallElev$ (call the elevator) or $OrderElev(loc)$, then this action is sent to another layer that controls the hand motions of the robot (at the moment this layer does not exist, as our robot does not have an arm). If the next action to be performed is a motion, it sends an axiom asserting the next destination to layer 1's latches.

4.4 LAYER 3: Wide-Range Motion Planning

Layer 3 performs high-level robot motion planning using situation calculus. There is only one fluent, the robot's location. Deeper reasoning can be performed and longer plans can be formed in a reasonable time, due to this restriction (e.g., no frame axioms are needed because there is only one fluent). The input for this layer is the current location of the robot (same as for layer 2). The goal is $TargetLandmark(landmark)$, which is asserted from the outside (or from a higher layer).

$$Room(RM218) \wedge Room(ELEV1) \ldots$$
$$Corridor(C2A1) \wedge Corridor(C2AELEV) \ldots$$
$$InCorridor(Front(RM218), C2A1) \ldots$$

$$\forall l, l', s. At(Robot, l, s) \wedge Vlinked(l, l') \Rightarrow At(Robot, l', Result(MoveTo(l'), s)).$$

$$\forall l. Room(l) \Rightarrow VLinked(l, Front(l)).$$

During each cycle, layer 3 tries to plan the motion from its current location to the target landmark. It has a map that abstracts cartesian coordinates to landmarks, similar to layer 2. It tries to create a plan that will lead the robot to this landmark and prove $NextDestination(next_landmark_x, next_landmark_y)$. If a plan is found, the next action to be performed is sent as an axiom asserting the next destination to layer 2's latches.

5 Implementation Details

5.1 Implementation setup

We have implemented the above theory using the PTTP theorem prover ([Stickel, 1988; 1992]) on a cluster of Sun Sparc stations running Quintus Prolog as the underlying interpreter for PTTP. The system runs on a Nomad 200 robot (see Figure 4) in the Computer Science Robotics Lab at Stanford University.

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, ensure 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 first implementation[3] was written as a sequential procedure. It was written in Quintus Prolog, and performed the proofs for each of the layers one after the other. The main purpose of this original implementation was to show that the proof time for the lower layers was small enough to allow real-time control (see Figure 5). However, it was not possible to control a robot with this implementation: an interface for the robot had to be written, and the software had to become *hands-free* and robust enough to sustain the robot for hours of operation.



Figure 4: Nomad 200

The current implementation 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 sending 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 (translating layer 0's proven goals to robot motion commands, and sending 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 (is it synchronous (request data explicitly) or asynchronous (if there is data in the latch, use it)).

Every theory file includes a theory and a goal formula to be proved by the layer. When each layer is run, we put a limit on the depth of the search allowed for a proof (this depth is determined experimentally). If no proof has been found up to this depth, we conclude that for any run-time purpose there is no proof. This limit is specified together with the goal formula.

5.2 Choosing a Theorem Prover

Choosing a theorem prover is not easy. A theorem prover that is embedded in an autonomous agent needs to be a fully *automated reasoner* (no human intervention can be used here), needs to allow easy analysis of its proof progression (the system designer uses this information to change the theory or to detect problems in it), needs to allow for proof strategies (to be specified either by a human or by another layer), needs to allow for easy embedding of semantic attachments (and possibly theory resolution [Stickel, 1985]), and needs to allow for nonmonotonic reasoning (or at least some approximation of it).

[3]The first implementation of the LSA can be found at http://www-formal.stanford.edu/eyal/lsa/lsaV1.0.tar.gz.

	Layer 0				Layer 1				Layer 2		Layer 3	
	Time		Infer.		Time		Infer.		Time	Infer.	Time	Infer.
	Mean	SD	Mean	SD	Mean	SD	Mean	SD				
Scen. 1	0.09	0.02	3598	629	0.02	0.01	394	2	0.01	4	0.00	20
Scen. 2	0.10	0.01	3703	613	0.02	0.01	384	4	0.52	27184	0.47	34056
Scen. 3	0.09	0.02	3575	640	0.02	0.01	389	1	0.00	4	11.24	694966

Figure 5: Proof time and inference steps measurements for the LSA *ver1.0* during experiments in three different scenarios: (1) single-floor planning, (2) lower-level elevator planning, and (3) multi-floor planning. (*SD* is *standard deviation*.)

The need for semantic attachments can be seen by observing layer 0. Layer 0's theory includes several mathematical computations (such as several trigonometric functions) that are much better suited for a systematic algorithm than a theorem prover. Since we wanted to have some algebraic sophistication in our theory, we needed semantic attachments. The aggregation in layer 0 of all the objects who's existence we can prove illustrates the need for nonmonotonic reasoning.

We examined several theorem provers. In the first group were *proof checkers*, such as PVS[4] (natural deduction), HOL[5] and GETFOL[6]. These theorem provers are highly expressive, but they are not suitable for embedding in autonomous agents, as they need at least some human intervention.

Compared to other theorem provers we examined (such as Otter[7] (a resolution theorem prover), ACL2[8] (an industrial-strength version of the Boyer Moore theorem prover), and ATP[9] (a model elimination theorem prover)), we found PTTP to be simple and easily customizable. Its close relationship with the underlying programming language (it has been implemented in both Prolog and LISP) allows for easy embedding of semantic attachments. Also, its output at run-time together with its iterative-deepening procedure allow for proof-progression analysis and control. It also allows for an approximate form of nonmonotonic reasoning by limiting the lengths of proofs.

5.3 Experiments with LSA *v1.0*

As mentioned earlier, LSA *ver1.0* served as a testbed for verifying the theories we developed for each layer. We subjected it to several experiments in a simulated office building environment (no active simulator was involved, and the sensory information (and map) were entered by hand). Figure 5 summarizes the results for three scenarios of varying difficulty: (1) planning a path towards a location on the same floor as the robot, (2) creating a plan that requires a low-level plan for using the elevator, and (3) planning a path towards a location on a different floor. In each scenario, we experimented with various robot orientations and obstacle positions in the robot's vicinity. For each layer, we measured the number of inference steps and time taken to prove its goal.[10] Results from a sample run are shown in Figure 6.

Layer 0, the critical layer, achieved its results in an average of 0.1 seconds when a turn action was required, and 0.3 seconds when a forward action was required. Layers 1, 2, and 3 worked fairly fast, although the long planning involved in Scenario 1 took more than 10 seconds (for a depth of 30 in the proof space). However, because we rely on the speed of only layer 0, safety is not compromised; the avoidance capabilities ensure that the robot does not fall off a cliff while planning a way to avoid the cliff edge.

We attribute the speed achieved to three optimizations. First, we used a few semantic attachments in Layer 0. In particular, the predicate $GetForce$ was embodied in a C function that returns the force vector $[Strength, Direction]$. It calls Prolog's bagof 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 (after some experimentation, we settled on a constant of 20.)

Second, we applied caching to the proof of $GetForce$. Since every proof "re-proved" $GetForce$ many times, this improved the performance of Layer 0 significantly (from approximately 10 seconds to 0.1 seconds per proof).

Third, we divided the planning so that layer 2 executes "local planning" for the elevator domain. This allowed layer 3 to avoid an explosion of the proof space, which otherwise would have occurred since there are four principal actions as well as a number of frame axioms associated with the robot and the elevator. The separation also helped prevent complex unifications.

5.4 Experiments with LSA *ver2.0*

Unfortunately, at the moment, the system is not yet debugged enough to be extensively tested. The layers are working together, and the system is stable for a few minutes, but a layer (executable) will crash, typically after one to five minutes of work. However, we observed that the combined system running on the Nomad 200 performed its task of planning and moving between different landmarks.

At the time of this writing we have decided to switch our development effort to the use of SWI prolog instead of Quintus Prolog. We discovered after much debugging of the system that Quintus Prolog (version 3.4) has a bug that causes our layers to crash after about 1000 iterations through the control loop. In fact, we managed to construct a fairly small example that makes Quintus Prolog crash in about 5 seconds.

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

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

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

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

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

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

[10] We do not list averages or standard deviations for layers 2 and 3 because their performances are independent of both the robot's orientation and sonar readings.

```
| ?- prove((turn(A); fwd(S))),
       print('turn angle = '), print(A), nl,
       print('fwd speed = '), print(S)
search for cost 0 proof...  3 infer-
ences 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 sec-
onds,
including printing
turn angle = -314
fwd speed = _6872
```

Figure 6: An example proof by the theory.

That said, Quintus Prolog is still the fastest Prolog implementation known to us for our problem. It is five times faster than SWI Prolog and ten times faster than SICStus Prolog on example runs of PTTP. We are currently communicating with both Rational soft. (developers of *Purify*) and SICS (developing and supporting Quintus Prolog), trying to solve the Quintus Prolog problem.

6 Related Work

Compared to other approaches to agent architecture and robot control using logic, LSA is the only one using full FOL theorem provers for the low-level control loop and the first one to propose an architecture built on theorem provers that is suitable for realizing complex tasks.

[Shanahan, 1996] describes a map-building process using abduction, but then implements his theory in an algorithm that is proved to have his abductive semantics. [Baral and Tran, 1998] define control modules to be of a form of Stimulus-Response (S-R) agents (see [Nilsson, 1998]), relating them to the family of action languages $\mathcal{A}$ (e.g., [Gelfond and Lifschitz, 1993], [Giunchiglia *et al.*, 1997]). Baral and Tran 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. [Levesque *et al.*, 1997], [Giacomo *et al.*, 1998], and other work in the GOLOG project have a planner that computes/plans the GOLOG program off-line, only later letting the robot execute the GOLOG program on-line. Here again, logic is used only to give semantics for GOLOG programs by way of situation calculus ([McCarthy and Hayes, 1969]).

None of this work uses FOL theorem provers for controlling robots at run-time. To our knowledge, there has been no such system since Shakey [Nilsson, 1984].

7 Conclusion

We have shown that theorem provers can be used for robot control by employing them in a layered architecture. We 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. Furthermore, we have grounded our proposal by giving it formal semantics based on circumscription.

At this time, the system is implemented in four layers on a simulating computer. Besides installing the system on a mobile robot, our future work plan includes adding layers that create maps and layers that reason about and update explicit beliefs about the world. We are currently working on incorporating vision sensory capabilities and implementing concurrency.

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

8 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 ARPA (ONR) grant N00014-94-1-0775, AFOSR grant AF F49620-97-1-0207, and by a National Physical Science Consortium (NPSC) fellowship.

References

[Amir and Maynard-Reid, 1999] Eyal Amir and Pedrito Maynard-Reid. Logic-based subsumption architecture. In *Proc. Sixteenth International Joint Conference on Artificial Intelligence (IJCAI '99)*, 1999.

[Amir and McIlraith, 2000] Eyal Amir and Sheila McIlraith. Paritition-based logical reasoning. In *Principles of Knowledge Representation and Reasoning: Proc. Seventh Int'l Conference (KR '2000)*, pages 389–400, 2000.

[Amir, 2000] Eyal Amir. (De)composition of situation calculus theories. In *Proc. National Conference on Artificial Intelligence (AAAI '00)*, pages 456–463. AAAI Press/MIT Press, 2000.

[Arkin, 1998] Ronald C. Arkin. *Behavior-based robotics*. MIT Press, 1998.

[Baral and Tran, 1998] Chitta Baral and Son Cao Tran. Relating theories of actions and reactive control. *Electronic Transactions on Artificial Intelligence*, 1998. Under Review.

[Borenstein and Feng, 1996a] Johann Borenstein and Liqiang Feng. Gyrometry: A new method for combining data from gyros and odometry in mobile robots. In *IEEE Intl' conf. on robotics and automation*, pages 423–428. IEEE, 1996.

[Borenstein and Feng, 1996b] Johann Borenstein and Liqiang Feng. Measurement and correction of systematic odometry erros in mobile robots. *IEEE Trans. on Robotics and Automation*, 12(5):869–880, 1996.

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

[Brooks, 1990] Rodney A. Brooks. The behavior language; user's guide. A.I. Memo 1227, Massachusetts Institute of Technology, April 1990.

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

[Gelfond *et al.*, 1989] Michael Gelfond, Halina Przymusinska, and Teodor C. Przymusinski. On the relationship between circumscription and negation as failure. *Artificial Intelligence*, 38(1):75–94, February 1989.

[Giacomo *et al.*, 1998] Giuseppe De Giacomo, Ray Reiter, and Mikhail 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.

[Giunchiglia *et al.*, 1997] Enrico Giunchiglia, G. Neelakantan Kartha, and Vladimir Lifschitz. Representing Action: Indeterminacy and Ramifications. *Artificial Intelligence*, 1997. to appear.

[Levesque *et al.*, 1997] H.J. 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.

[Lifschitz, 1989] Vladimir Lifschitz. Circumscriptive theories: A logic-based framework for knowledge representation. In R. H. Thomason, editor, *Philosophical Logic and Artificial Intelligence*, pages 109–159. Kluwer Academic Publishers, 1989.

[McCarthy and Hayes, 1969] John McCarthy and Patrick 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.

[McCarthy, 1986] John McCarthy. Applications of Circumscription to Formalizing Common Sense Knowledge. *Artificial Intelligence*, 28:89–116, 1986. Reprinted in [McCarthy, 1990].

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

[Minsky, 1985] Marvin Minsky. *The Society of Mind*. Simon and Schuster, 1985.

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

[Nilsson, 1998] Nils J. Nilsson. *Artificial Intelligence, A new Synthesis*. Morgan-Kaufmann, 1998.

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

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

[Stickel, 1985] Mark E. Stickel. Automated deduction by theory resolution. *Journal of Automated Reasoning*, 1:333–355, 1985.

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

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