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

We describe a logic-based AI architecture based on Brooks' subsumption architecture. In this architecture, we axiomatize different layers of control in First-Order Logic (FOL) 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 layers, and nonmonotonically retract them when higher layers come up with new conclusions. We also give formal semantics to our approach. Finally, we describe four layers designed for the task of robot control, and an experiment, empirically showing 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 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 simultaneously service 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. Fhe increasing hardware complexity with new layers introduces scaling problems. And, relying on hardware specifications, the architecture is specifically oriented towards robot control and is not 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], 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 extendibility in the manner described above. Our *Logic-Based Subsumption Architecture* (LSA) 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 use nonmonotonic reasoning to model the connections between the theories. In addition, 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 by modeling four layers for the task of robot control, the bottom two of which are Brooks' first two layers. We show empirically that the layer in greatest need of reactivity is sufficiently fast (0.1–0.3 seconds per control-loop cycle). This 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 behavioral decomposition, we describe the LSA and give formal semantics to the approach using circumscription. We then describe a 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 → perception → modeling → planning → task recognition → 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.

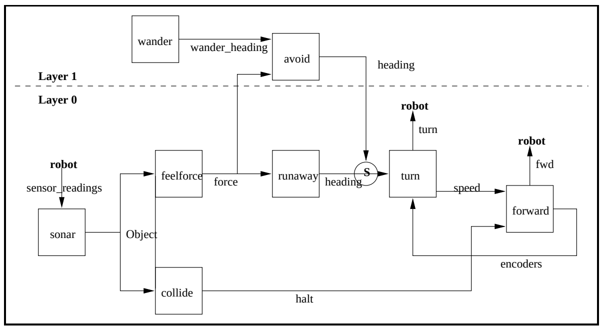

Figure 1: Layers 0 and 1 of Brooks' subsumption architecture robot control system.

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. However, occasionally the goals of one layer will 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 simply 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 which 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

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 sensor or from 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 accomodate 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 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.

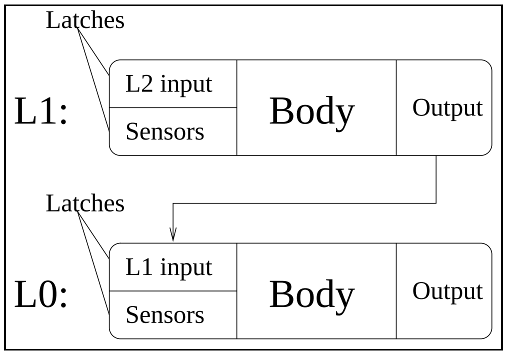

Figure 2: A detailed look at two layers.

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

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

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.

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 equivalent to circumscription [Gelfond *et al.*, 1989]).

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.

3.3 Semantics for LSA

If we ignore the time differences between the theorem provers in different layers and consider the entire system of layers as one logical theory, we can give the system a simple semantics. Let $Layer_i$ be the theory of layer i (including any CWA as a FOL schemata), $G_i(x)$ be the goal formula of layer i (i.e., the formula that we try to prove in that layer) and let $G'_i(f_i(x))$ be its translation to $Layer_{i-1}$'s input. We call such a system of layers T a *layered theory.* For $\varphi \in \mathcal{L}(Layer_i)$ we write $T \vdash \varphi$, if the mechanical entailment we described above derives φ.

Let $\varphi_i^{ab} = \forall x.\neg ab_i(x) \Rightarrow (G_i(x) \Rightarrow G'_i(f_i(x)))$, where every ab_i is a relation symbol that does not show in T.

Definition 3.1 (Semantics for Layered Theories) *$\mathcal{M}$ is a model of the layered theory T (written $\mathcal{M} \models_c T$) iff it is a first-order model of the circumscriptions*

$$\bigwedge_{i \leq n} Circ[Layer_i \cup Layer_{i-1} \cup \{\varphi_i\}; ab_i; \mathcal{L}(Layer_{i-1})]$$

This semantics assumes (1) we are interested in the set of results $\varphi \in \mathcal{L}(Layer_i)$ for some i (as opposed to $\varphi \in \mathcal{L}(Layer_i \cup Layer_j)$, for example) and (2) all the symbols in the various theories are different (e.g., the symbol 1 actually has the name $L_0.1$ in layer 0 and $L_2.1$ in layer 2).

Let $\mathcal{L}_i^I$ be the FOL language including only G_i, $=$, and the constant and function symbols of $\mathcal{L}(Layer_i)$. For a sentence $\phi \in \mathcal{L}_i^I$, let ϕ' be the translation of ϕ by replacing G with G' and every term t by $t' = f_i(t)$. The following theorem validates the semantics for our proof-theoretic system of transferring goals from one layer to another.

Theorem 3.2 (Completeness) *Assume that $T = \{Layer_i\}_{i \leq m}$ is a layered theory and φ is a formula in the language $\mathcal{L}(Layer_0)$. If $T \vdash \varphi$, then there is $k \geq 0$ and a sequence of sentences $\varphi_k, \ldots, \varphi_0$ s.t. $\varphi = \varphi_0$, $Layer_k \models \varphi_k$, and $1 \leq \forall i \leq k$ $\varphi_i \in \mathcal{L}_i^I$ and $Layer_{i-1} \models \varphi'_i \Rightarrow \varphi_{i-1}$.*

Our LSA obeys this semantics, assuming that transferring a single instantiation of the goal between any pair of layers is sufficient. In case one layer proves only a disjunction of goal instantiations, we need to refine our LSA to support such a transfer, but this refinement can be done for any size of disjunctions. Additionally, there is no need to consider quantification in our LSA, since we assume skolemization of the clauses (see Section 5).

We omit the proof for lack of space, but mention that it relies on Craig's Interpolation Theorem for FOL, and on the following lemma.

Lemma 3.3 *If $Circ[Layer_1 \cup Layer_0 \cup \{\varphi_1^{ab}\}; ab_1; \mathcal{L}(Layer_0)] \not\models \forall x \neg ab_1(x)$, then there is a formula $\phi \in \mathcal{L}_1^I$, negative in G_1, s.t. $Layer_1 \not\models \phi$ and $Layer_0 \models \phi'$.*

Furthermore, if $Layer_1'$ is the result of adding all such ϕ's to $Layer_1$, then $Circ[Layer_1' \cup Layer_0 \cup \{\varphi_1^{ab}\}; ab_1; \mathcal{L}(Layer_0)] \models \forall x \neg ab(x)$.

For soundness, we need to assume that the set of layers of T is consistent with $\{\varphi_i, \forall x \neg ab_i(x)\}_{i \le n}$ and that all the circumscriptions in $\models_c$ for T have *smooth* preference relations (i.e., every model is either minimal or has a minimal model that is preferred to it).

Theorem 3.4 (Soundness) *Assume that there are $\varphi_k, ..., \varphi_l$ s.t. $k \ge l$ and $Layer_k \models \varphi_k$ and $k \ge \forall i > l$, $\varphi_i \in L_i^I$, positive in G_i, and $Layer_i \models \varphi_{i+1}' \Rightarrow \varphi_i$, where φ_i' is the translation of φ_i from layer i to layer $i-1$. Then, $T \models_c \varphi_l$.*

4 A Model of Brooks' System

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. For simplicity, we only list selected axioms from the theories and refer the reader to the full version of the paper for the complete system.

We assume the architecture is used to control a cylindrical robot with sonar sensors on its perimeter and wheels that control its motion. We also assume that it is able to determine its current location and orientation.

4.1 LAYER 0: Obstacle-avoidance

Layer 0 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 discover "virtual" objects by way of layer 1's subsumption latch.

$$\begin{aligned}&\forall dist, dir.\ (\exists sonar\_number.\\ &\quad SonarReading(sonar\_number) = dist \wedge\\ &\quad SonarDirection(sonar\_number) = dir \wedge\\ &\quad dist \ge 0 \wedge dir > -\pi \wedge dir \le \pi) \Rightarrow\\ &\qquad (\exists obj.\ Object(obj) \wedge Distance(obj) = dist \wedge\\ &\qquad\quad Direction(obj) = dir).\end{aligned}$$

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

$$\begin{aligned}&ObjectAhead \iff\\ &\quad (\exists obj.\ (Object(obj) \wedge Distance(obj) < MIN\_DIST \wedge\\ &\qquad Direction(obj) = dir \wedge dir > -\tfrac{\pi}{4} \wedge dir < \tfrac{\pi}{4}).\end{aligned}$$

$$ObjectAhead \Rightarrow HaltRobot.$$

[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 $NSONARS - 1$.

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 further away from the objects.

$$\begin{aligned}&Heading\_angle = (Force\_direction \bmod 2\pi) - \pi.\\ &Heading\_speed = Force\_strength.\end{aligned}$$

$$NeedTurn(Heading\_angle) \Rightarrow Turn(Heading\_angle).$$

$$\begin{aligned}&\neg HaltRobot \wedge \neg NeedTurn(Heading\_angle) \wedge\\ &\quad NeedFwd(Heading\_speed) \Rightarrow Fwd(Heading\_speed).\\ &HaltRobot \vee NeedTurn(Heading\_angle) \Rightarrow Fwd(0).\end{aligned}$$

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 and prove $Fwd(speed)$ 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. Given a particular pair of coordinates specified by the input *MoveCmd* from layer 2 and given the robot's current location,[2] it makes a simple calculation to find 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.

$$Object(PUSH\_OBJECT).$$

$$\begin{aligned}&\forall x_0, y_0, x, y.\ CurrLoc = \{x_0, y_0\} \wedge MoveCmd(x, y) \Rightarrow\\ &\quad HasPushObject(Quadrant(x_0 - x, y_0 - y)).\end{aligned}$$

$$\begin{aligned}&\forall quad.\ HasPushObject(quad) \Rightarrow\\ &\quad Direction(PUSH\_OBJECT) = quad * \tfrac{2\pi}{NQUADS} \wedge\\ &\quad Distance(PUSH\_OBJECT) = PUSH\_OBJ\_DIST.\end{aligned}$$

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: Mid-level 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 using the elevators.

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

The inputs for this layer are the current location data from the robot ($CurrLandmark$) and the output from layer 3 ($TargetLandmark$). During each cycle, it tries to plan for the next landmark and prove $MoveCmd(next\_landmark)$.

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

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

$$\forall x, y, lm, s_{l1}, inter, s_1.\ NeedElevPlan \wedge TargetLandmark(lm) \wedge At(ROBOT, lm, s_{l1}) \wedge FirstSit(s_{l1}, s_1) \wedge At(ROBOT, inter, s_1) \wedge Cartesian(inter, [x, y]) \Rightarrow MoveCmd(x, y).$$

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.

4.4 LAYER 3: High-level planning

Layer 3 performs high-level robot motion planning using situation calculus. Here there is only one fluent (the robot's location); thus, deeper reasoning can be performed in a reasonable time.

The input for this layer is the current location of the robot. The goal is $TargetLandmark(landmark)$.

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

$$\forall l, l', s. At(ROBOT, l, s) \wedge VConnected(l, l') \Rightarrow At(ROBOT, l', Result(MoveTo(l'), s)).$$

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

5 Implementation Issues

We have implemented the above theory using the PTTP theorem prover ([Stickel, 1988], [Stickel, 1992]) on a Sun Sparc station, running Quintus Prolog as the underlying interpreter for PTTP. PTTP (Prolog Technology Theorem Prover) is a model-elimination theorem prover. Given a theory made of clauses (not necessarily disjunctive) without quantifiers, PTTP produces a set of Prolog-like Horn clauses, makes sure 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.

We subjected our system to a battery of experiments in a simulated office building environment. Figure 3 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.[3]

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. (Due to space concerns, we have included in figure 3 only the data for cases of the former kind.) 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 only rely on the speed of layer 0, safety is not compromised; the avoidance capabilities ensure that the robot does not fall off the 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 occured since there are four principle actions as well as a number of frame axioms associated with the robot and the elevator. The separation also helped prevent complex unifications.

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]). 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. [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

[3]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.

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

program on-line. Here again, logic is only used 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, scalable, and 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. This research was supported by an ARPA (ONR) grant N00014-94-1-0775 and by a National Physical Science Consortium (NPSC) fellowship.

References

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

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

[Gelfond and Lifschitz, 1993] M. Gelfond and V. Lifschitz. Representing actions and change by logic programs. *J. Logic Programming*, 17:301–322, 1993.

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

[Giacomo *et al.*, 1998] G. De Giacomo, R. Reiter, and M. Soutchanski. Execution monitoring of high-level robot programs. In *Proc. KR-98*, pages 453–464, 1998.

[Giunchiglia *et al.*, 1997] E. Giunchiglia, G.N. Kartha, and V. Lifschitz. Representing action: Indeterminacy and ramifications. *Artificial Intelligence*, 95(2):409–438, 1997.

[Levesque *et al.*, 1997] H.J. Levesque, R. Reiter, Y. Lesperance, F. Lin, and R. Scherl. Golog: A logic programming language for dynamic domains. *J. Logic Programming*, 31:59–84, 1997.

[McCarthy and Hayes, 1969] J. McCarthy and P.J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In *Machine Intelligence*, volume 4, pages 463–502. 1969.

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

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

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

[Nilsson, 1998] N.J. Nilsson. *Artificial Intelligence: A New Synthesis*. Morgan-Kaufmann, 1998.

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

[Stein, 1997] L.A. Stein. Postmodular systems: Architectural principles for cognitive robotics. *Cybernetics and Systems*, 28(6):471–487, September 1997.

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

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