Logic-Based Subsumption Architecture

Eyal Amir [a,1] Pedrito Maynard-Zhang [b,2]

[a] *Stanford University, Computer Science Department, Stanford, CA 94305-9020, USA*

[b] *Miami University, Computer Science & Systems Analysis Department, Oxford, OH 45056, USA*

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 nonmonotonic reasoning principles. In particular, we use circumscription to make default assumptions in lower layers, and nonmonotonically retract those assumptions when higher layers draw new conclusions. We also give formal semantics to our approach. Finally, we describe layers designed for the task of robot control and a system that we have implemented that uses this architecture for the control of a Nomad 200 mobile robot.

Our system combines the virtues of using the represent-and-reason paradigm and the behavioral-decomposition paradigm. It allows multiple goals to be serviced simultaneously and reactively. It also allows high-level tasks and is tolerant to different changes and elaborations of its knowledge in runtime. Finally, it allows us to give more commonsense knowledge to robots. We report on several experiments that empirically show the feasibility of using fully expressive FOL theorem provers for robot control with our architecture and the benefits claimed above.

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 depend on lower, more reactive layers, but could also influence their behavior. The resulting architecture was one that

[1] E-mail: eyal.amir@cs.stanford.edu

[2] E-mail: maynarp@muohio.edu

could simultaneously service multiple, potentially conflicting goals in a reactive fashion, giving precedence to high-priority goals.

Because of its realization in hardware, the subsumption architecture lacks declarativeness, making it difficult to implement higher-level reasoning and making its semantics unclear. The 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 that is modeled in the spirit of Brooks' subsumption architecture but relies on a logical framework and has wider applicability and extensibility 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. After proving a goal, each layer sends the result to lower layers. We use nonmonotonic reasoning to allow layers to make default assumptions. These assumptions may be subsumed when information arrives from higher layers or from the agent's sensors. This allows each layer's performance to be independent of the performance of other layers, supporting reactivity. We also use nonmonotonic reasoning to provide semantics for the combined (static) system.

The benefits of our system are the results of combining the direct use of logical theories with a procedural-behavioral overall structure. In particular, the benefits of using logical theories directly are those that are found when comparing the declarative approach with the procedural approach to building intelligent systems (e.g., (Levesque and Brachman, 1985)). A declarative system can receive advice at runtime, is easily extensible and reusable, and is more understandable to the outside observer. There is no need to aim for a specific application when designing the theories involved in the system. Our choice of FOL as the basic representation language allows the system to use varying representations of knowledge that are taken from different streams in AI, including some that require elaborate logical languages and axioms, such as probability theory, frame systems, set theory, and utility theory. With little additional axioms, we can include information that is both quantitative and qualitative. Thus, our system uses a single representational and reasoning mechanism to capture all parts of the complex system while keeping the overall behavior fast. Finally, our system allows adding layers that reason explicitly about recommended inference, relying on theories such as those in (Russell and Wefald, 1989).

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 (much in the spirit of the Advice Taker of (McCarthy, 1958)). The architecture also allows incorporating layers that perform diagnosis and can be extended to layers that remember experiences for other layers. Our experiments show that real-time commonsense control of AI agents can be achieved with an architecture based on logical theories and general-purpose theorem provers with the use of subsumption. These theorem provers can be replaced with special-purpose reasoners, for improved efficiency, but we show that in simple cases this is not needed with current theorem-proving technology. We report on the set of experiments that we executed with this system on a Nomad200 mobile robot.

This work improves over results presented using GOLOG (e.g. (Levesque et al., 1997)) and the work of (Shanahan, 1996) 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.

Some of the results in this paper appeared previously in (Amir and Maynard-Reid II, 1998; Amir and Maynard-Reid II, 1999; Amir and Maynard-Reid II, 2001). (Note, that the second author's surname in the corresponding proceedings was Maynard-Reid II.)

2 Principles of Subsumption Architectures

2.1 Brooks' Subsumption Architecture

Brooks (Brooks, 1986) 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 behaviors interacting with each other and a complex environment. This focus on simplicity leads to a design where each individual layer is composed of simple state machine modules operating asynchronously without any central control.

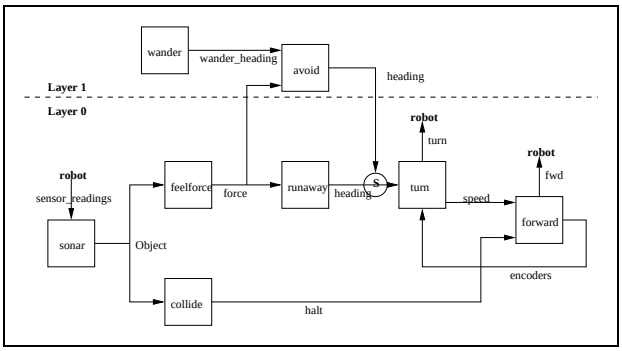

Fig. 1. Layers 0 and 1 of Brooks' subsumption architecture robot control system.

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.

Following Brooks' work and others (e.g., see (Arkin, 1998)) there has been much work on implementing behavior-based robots when the different behaviors are simply switched by an automaton or are stacked in a subsumption architecture. In the latter, layers typically avoid intervening with the internals of layers below, opting to inhibit the output of lower layers, replacing it with the higher layer's output.

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. 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 `avoid` layer accepts sonar readings of the robot's surroundings into its `sonar` module which outputs a map of the vicinity based on these readings. The `collide` module checks if there is an obstacle directly ahead and, if there is, forces the robot

to stop regardless of what other modules are doing. The `feelforce` module uses the map to calculate a combined repulsive "force" that the surrounding objects exert on the robot. The `runaway` module checks if this force is significant enough to pay attention to and, in the case that it is, determines the new heading and speed for the robot to move away from the force. The `turn` module commands the robot to make the required turn, then passes the speed on to the forward module which, if not in a halt state, commands the robot to move forward with the specified speed. The further away the robot gets, the smaller the speed computed by the `runaway` module.

Every so often, the `wander` module chooses a new random direction for the robot to move in. The `avoid` module combines it with the output of the `avoid` layer's feelforce module, computing an overall heading that suppresses the input to the `avoid` layer's `turn` module.

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 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 (we verified this claim experimentally with the PTTP theorem prover). 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 circumscription, although other approaches may be equally interesting and appropriate.

3 Logic-Based Subsumption

This section describes how we implement the principles discussed above. Following Brooks, the architecture decomposes a domain along behavioral lines into simple layers. But, unlike systems that followed Brooks' work, it allows the layers to work in synergy to produce the compound behavior.

3.1 Basic Machinery

A *Logic-Based Subsumption Architecture* (LSA) is built of layers corresponding to behaviors (see Figure 2). The layers work concurrently and asynchronously with respect 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 3.2.

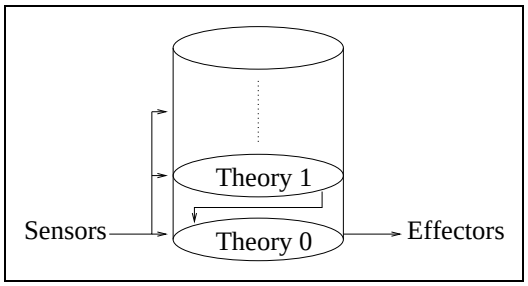

Fig. 2. An abstract diagram of the LSA.

Each layer is 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.

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

3.2 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 (McCarthy, 1986) formula:

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

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.

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 (Gelfond et al., 1989)).

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 $\overrightarrow{C}_i$ 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

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

When implemented, this formula often can be replaced with a simple (external to the logic) mechanical interference determining the value of the minimized predicates; we discuss this issue in section 6. 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).

3.3 *Putting It All Together*

Each layer tries to prove

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

Here, $\overrightarrow{C}_i$, $\overrightarrow{Z}_i$ are specified as part of the defaults for layer i, $Layer_i$ is the set of axioms including the body and the latches and $Goal_i(\overrightarrow{x})$ is a goal formula specified for layer i ($\overrightarrow{x}$ is a vector of variables open in $Goal_i(\overrightarrow{x})$). Upon proving $Goal_i(\overrightarrow{a})$, the layer transmits $Goal_i(\overrightarrow{a})$ either to the layer below or (in the case of the lowest layer) to the robot manipulators. Figure 3 summarizes this algorithm, while Figure 4 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) Try to prove $Circ[Layer_i; \overrightarrow{C}_i; \overrightarrow{Z}_i] \models \exists \overrightarrow{x}\ Goal_i(\overrightarrow{x})$ (try to prove the goal from $Layer_i$ given the default assumptions).
(4) If $Goal_i(\overrightarrow{a})$ was proved for assignment $\overrightarrow{a}$, assert $Goal_i(\overrightarrow{a})$ in the input latches of layer $i-1$, $Input_{i-1}$.

Fig. 3. The LSA algorithm.

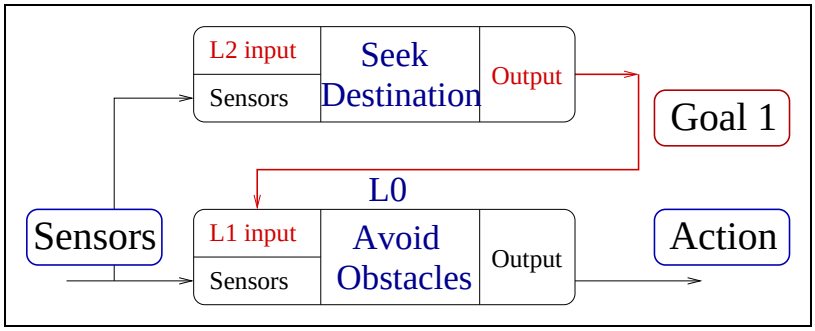

Fig. 4. A detailed look at two layers.

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 has not found a proof after some pre-specified time period, 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 has not proven its goal in the last few seconds, then layer 0 will no longer consider an axiom sent previously by layer 1 as valid.

4 Static Semantics for LSA

In the previous section we showed how we use circumscription to implement subsumption. We also use circumscription to give semantics to the system of layers as one big logical system. Specifically, it is needed to give semantics to the directional nature of the complete system (i.e., that messages between layers go only in one direction).

An LSA system has layers that are asynchronous and acting in a changing world. There is also a discrepancy between sensing and acting, as the action is never executed in exactly the same world in which the sensing was done. Also, information collected by the sensors is always imprecise. We do not try to model these here. In this section we assume that sensing and acting is done in a stationary world (i.e., that the robot is not allowed to act before the formula in Definition 4.1 is computed).

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. In what follows we are interested in the output of the system to the actuators of the robot.

Let $Layer_i$ be the theory of layer i, and assume that we use first-order circumscription to assume defaults for layer i. We include any default as a FOL schemata. Let $Goal_i(\overrightarrow{x})$ be the goal formula of layer i (i.e., the formula that we try to prove in that layer). We call such a system of layers, T, a *layered theory*. Let T have $n+1$ layers (i.e., layers are numbered $0, ..., n$). For $i \leq n$, $\varphi \in \mathcal{L}(Layer_i)$, we write $T \vdash \varphi$ if the mechanical entailment we described above derives φ in step 3 (for any layer $j \leq n$).

Definition 4.1 (Output Semantics for Layered Theories) $\mathcal{M}$ is a model of the layered theory T *(written $\mathcal{M} \models T$) iff it is a first-order model of*

$$Circ[Layer_0 \cup Circ[Layer_1 \cup ...; \overrightarrow{C}_1; \overrightarrow{Z}_1]; \overrightarrow{C}_0; \overrightarrow{Z}_0]$$

where $\overrightarrow{Z}_i = L(\bigcup_{j=i}^{n} Layer_j)$, and $\overrightarrow{C}_i$ is taken as in formula (1).

We assume that $L(Layer_i) \cap L(Layer_j)$ includes no predicate symbols if $i \neq j$ and $i \neq j+1$, and that the predicate symbols that appear in $L(Layer_i) \cap L(Layer_{i+1})$

appear also in $Goal_{i+1}(\overrightarrow{x})$.

Let T be a layered theory with $n+1$ layers, $0, ..., n$, and assume that for every $i \leq n$, $Goal_i(\overrightarrow{x})$ is a single literal. The LSA obeys this semantics, assuming that transferring a single instantiation of the goal between any pair of layers is sufficient (i.e., that the only prime implicate of $Layer_{i+1}$ in $L(Goal_{i+1})$ is a single literal). In case one layer proves only a disjunction of goal instantiations, we need to refine LSA to support such a transfer, but this refinement can be done for any fixed size of disjunctions.

We borrow a technical result due to (Amir, 2002) which presents an interpolation theorem for circumscription.

Theorem 4.2 *Let T_1, T_2 be two theories, P, Q vectors of symbols in $L(T_1) \cup L(T_2)$ such that $P \subseteq L(T_1)$ and $P \cup Q \supset L(T_2)$. Let γ be the set (possibly infinite) of prime implicates of T_2 in $\mathcal{L}(L(T_1) \cap L(T_2))$ (or a logically equivalent set of sentences). Then, for every $\varphi \in \mathcal{L}(T_1)$,*

$$Circ[T_1 \cup \gamma; P; Q] \models \varphi \iff Circ[T_1 \cup T_2; P; Q] \models \varphi$$

Corollary 4.3 (Interpolation Theorem for Circumscription) *Let T be a theory, P, Q vectors of symbols in $L(T)$ such that $(P \cup Q) \supseteq L(T)$, $P \subseteq L(\varphi)$. Then, if $Circ[T; P; Q] \models \varphi$, then there is a set of sentences $\gamma \subset \mathcal{L}(T) \cap \mathcal{L}(\varphi)$ such that*

$$T \models \gamma \quad \textit{and} \quad Circ[\gamma; P; Q] \models \varphi.$$

Furthermore, this holds if γ is chosen to be the set of prime implicates of T in $L(T) \cap L(\varphi)$.

We now show that the LSA obeys the proposed semantics.

Theorem 4.4 (Output Completeness) *Assume that $T = \{Layer_i\}_{i \leq n}$ is a layered theory and $\varphi \in \mathcal{L}(Layer_0)$. If $T \models \varphi$, then there is $k \geq 0$ and a sequence of sentences $\varphi_k, \ldots, \varphi_0$ such that $\varphi = \varphi_0$, $Circ[Layer_k; \overrightarrow{C}_k; \overrightarrow{Z}_k] \models \varphi_k$, and for all i such that $0 \leq i < k$, φ_i includes predicate symbols only from $L(Goal_i)$ and*

$$Circ[Layer_i \cup \{\varphi_{i+1}\}; \overrightarrow{C}_i; \overrightarrow{Z}_i] \models \varphi_i.$$

PROOF See Appendix A.1.

Theorem 4.5 (Output Soundness) *Assume that there are sets of formulae $\varphi_n, ..., \varphi_0$ such that $Circ[Layer_n; \overrightarrow{C}_n; \overrightarrow{Z}_n] \models \varphi_n$ and for all i such that $0 \leq i < n$, φ_i is the set of prime implicates of $Circ[Layer_i \cup \varphi_{i+1}; \overrightarrow{C}_i; \overrightarrow{Z}_i]$ in $L(Goal_i) \cup L_f$ (L_f is the set of function and constant symbols in T). Then, $T \models \varphi_0$.*

PROOF See Appendix A.2.

5 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. 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 5).

Fig. 5. Nomad 200

The system includes five logical layers. A schematic view of the combined system is given in Figure 6. Each layer's body theory contains three main types of axioms: sensory-focused, goal-focused, and domain-dependent relationships in the world. We describe these theories, the default assumptions made by each associated theorem prover, and the goal each prover attempts to prove.

We follow the convention that constant, function, and predicate names use all lower-case letters, whereas variable names have at least the first letter capitalized. For the sake of clarity, we describe our axioms using a standard first-order logic notation. The translation into the notation of our PTTP/Prolog theorem prover is straightforward, with a couple of notable exceptions:

- We must skolemize existentially quantified variables to satisfy PTTP's requirement that all sentences be represented as clauses.
- For most functions and non-numeric constants, we use predicates. This is to accommodate PTTP's limited ability to handle equality. PTTP handles equality ("=") by a unification test. Thus, $=$ means *unifiable*, and $=\backslash=$ means *not unifiable*. Thus, we restrict the use of equality to cases where the unification test is a correct mechanism for testing equality or when equality is tested between arithmetic terms that can be evaluated at the time of the equality test. This gives rise to the modeling choices of replacing functions and constants with predicates. For example, we model the number pi (π) using the PTTP sentence

$$pi(3.14159), not\_pi(C0) :- CO =\backslash= 3.14159$$

  instead of $pi = 3.14159$.

We include the complete theories in the actual PTTP (clausal-like) notation in Appendix B for the interested reader. (The beginning of the appendix also contains an

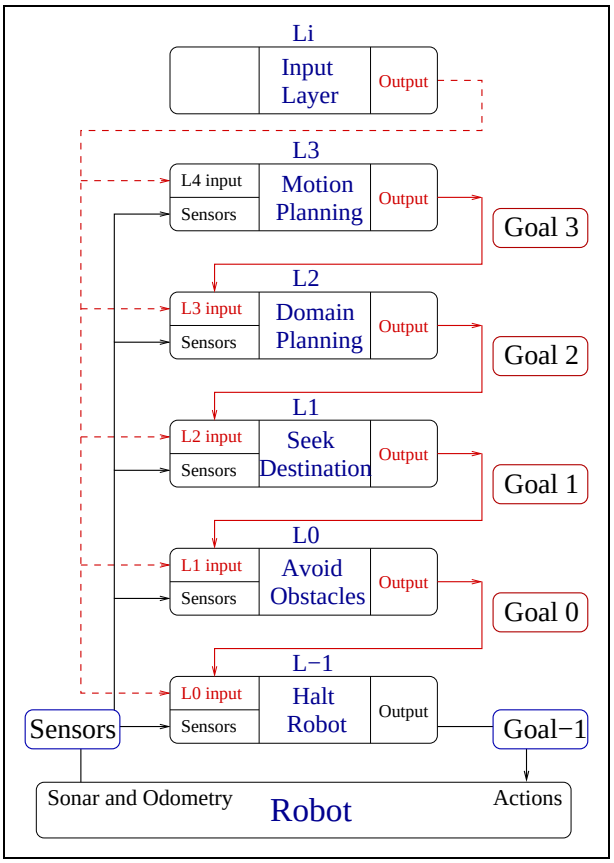

Fig. 6. Diagrammatic view of an LSA system controlling a robot.

introduction to PTTP's syntax.)

5.1 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 three main parts: goal-focused, sensory-focused and spatial relationships in the world. The following description uses predicates, functions and constant symbols that are identical to those used in the implemented theory. The complete theory and an index of its symbols appear in Appendices B.1 and B.2.

The goal-focused part represents the effects of robot motions in situation calculus. There is only one fluent, the robot's landmark, and only one action schema, $moveto(L)$, where L is a landmark 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:

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

(cf. Appendix B.2, formula 77 in the sample proof) where $vConnected(L0, L)$ means that there is a line of sight between $L, L0$. 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. For example, the robot knows when it is between Cartesian positions of landmarks using the axiom

$$vConnected(L1, L2) \wedge curr\_loc(X, Y) \wedge cartesian(L1, C1) \wedge$$
$$cartesian(L2, C2) \wedge C1 \neq C2 \wedge pos\_between(C1, [X, Y], C2) \Rightarrow$$
$$current\_landmark(between(L1, L2))$$

(cf. Appendix B.2, formula 45 in the sample proof) where $[X, Y]$ is considered between $C1, C2$ if it is close enough to the line crossing them (there is another axiom that describes this), and $curr\_loc(X, Y)$ is the Cartesian input from the robot odometry.

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 landmarks that are in the same corridor are visually connected as well. Other axioms describe invariants of the domain, such as the commutativity of $vConnected$, and the fact that a position between two visually connected positions is visually connected to both positions.

It is important to notice that the landmarks in our domain designate regions in space rather than specific cartesian positions in the world. For example, $current\_landmark(zero\_pt)$ holds if the robot is in the circle defined by the center of $zero\_pt$ and a radius defined by the predicate $short\_distance(L1, L2)$. In our current implementation the regions are not assumed to be exhaustive or disjoint. Thus, the robot may be in more than one landmark or none at all. The first choice does not interfere with our system, and the second did not raise problems in our experiments.

The goal for this layer is proving $target\_landmark(L)$ (with L a variable that gets instantiated in the proof) from this theory, the message $goal\_location(corridor2\_cross)$ (received from Input Layer) and the sensory information ($curr\_loc(0, 0)$ and others). For efficiency, we find the proof in four stages. First, we find a landmark at which the robot is (proving $at(r, L, s0)$, with L a free variable that gets instantiated in the proof) [3] .

[3] The "Wff#" in the layer-proofs in this section refer to the index of the axiom used to derive the consequence. Every such index is given with respect to the theory loaded into

```
Goal#  Wff#  Wff Instance
-----  ----  ------------
  [0]    0   query :- [1].
  [1]   74      at(r, zero_pt, s0) :- [2].
  [2]  138         current_landmark(zero_pt) :- [3],[4],[5].
  [3]  159            curr_loc(0, 0).
  [4]  102            cartesian(zero_pt, [0, 0]).
  [5]  139            short_distance([0, 0], [0, 0]) :- [6].
  [6]  140               distance_threshold(100).
'proved the robot is in 'zero_pt'
```

Then, we find a plan that achieves the robot's goal (proving $atgoal(r, S)$, with S a free variable that gets instantiated in the proof).

```
Goal#  Wff#  Wff Instance
-----  ----  ------------
  [0]    0   query :- [1].
  [1]   82      atgoal(r, result(moveto(corridor2_cross), result(moveto(mid_lab),
                      result(moveto(corridor_cross), s0)))) :- [2] , [3].
  [2]  156         goal_location(corridor2_cross).
  [3]   77         at(r, corridor2_cross, result(moveto(corridor2_cross),
                         result(moveto(mid_lab), result(moveto(corridor_cross),
                         s0)))) :- [4] , [16].
  [4]   77            at(r, mid_lab, result(moveto(mid_lab),
                            result(moveto(corridor_cross), s0))) :- [5] , [14].
  [5]   77               at(r, corridor_cross, result(moveto(corridor_cross),
                               s0)) :- [6] , [12].
  [6]   74                  at(r, zero_pt, s0) :- [7].
  [7]  132                     current_landmark(zero_pt):- [8],[9],[10].
  [8]  153                        curr_loc(0, 0).
  [9]   96                        cartesian(zero_pt, [0,0]).
 [10]  133                        short_distance([0,0], [0,0]):- [11].
 [11]  134                           distance_threshold(100).
 [12]   71                  vConnected(zero_pt, corridor_cross) :- [13].
 [13]  113                     vConnected(corridor_cross, zero_pt).
 [14]   71               vConnected(corridor_cross, mid_lab)  :- [15].
 [15]  114                  vConnected(mid_lab, corridor_cross).
 [16]  118            vConnected(mid_lab, corridor2_cross).
'proved the plan is 'result(moveto(corridor2_cross), result(moveto(mid_lab),
    result(moveto(corridor_cross), s0)))'
```

Then, we find the first situation in the plan, proving $firstSit(S, S1)$ with S instantiated (to $result(moveto(corridor2\_cross), result(moveto(mid\_lab), result(moveto(corridor\_cross))))$ in this case) and $S1$ a free variable that gets instantiated in the proof.

```
Goal#  Wff#  Wff Instance
-----  ----  ------------
  [0]    0   query :- [1].
  [1]   83      firstSit(result(moveto(corridor2_cross), result(moveto(mid_lab),
                      result(moveto(corridor_cross),s0))),
                      result(moveto(corridor_cross),s0)) :- [2].
  [2]   83         firstSit(result(moveto(mid_lab),
                         result(moveto(corridor_cross),s0)),
                         result(moveto(corridor_cross),s0)) :- [3].
  [3]   84            firstSit(result(moveto(corridor_cross),s0),
                            result(moveto(corridor_cross),s0)).
```

Finally, we find the first landmark associated with the first situation. We do so by proving $at(r, L1, S)$, with S instantiated (to $result(moveto(corridor\_cross))$ in this case) and $L1$ a free variable that gets instantiated in the proof.

PTTP for that layer (Section B.2).

```
Goal#  Wff#  Wff Instance
-----  ----  ------------
  [0]    0   query :- [1].
  [1]   77      at(r, corridor_cross, result(moveto(corridor_cross), s0)) :-
                      [2], [8].
  [2]   74         at(r, zero_pt, s0) :- [3].
  [3]  132            current_landmark(zero_pt):-[4],[5],[6].
  [4]  153               curr_loc(0, 0).
  [5]   96               cartesian(zero_pt, [0, 0]).
  [6]  133               short_distance([0,0], [0,0]) :- [7].
  [7]  134                  distance_threshold(100).
  [8]   71         vConnected(zero_pt, corridor_cross):- [9].
  [9]  113            vConnected(corridor_cross, zero_pt).
'found the landmark 'corridor_cross
```

5.2 Layer 2: Local Action-Planning

Layer 2 translates target landmarks (given to it from Layer 3) into Cartesian coordinates for the robot (sent to Layer 1), and for planning, low-level interaction, and control of the elevators. (Currently, there is no low-level implementation of the elevator control instructed by this layer.) The theory can be seen as comprising four main parts: sensory-focused, motion-focused, elevator-focused and spatial relationships in the world. The following description uses predicates, functions and constant symbols that are identical to those used in the implemented theory. The complete theory and an index of its symbols appear in Appendices B.1 and B.3.

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 landmarks being visually connected is now dependent on the situation (the elevators may be connected to their entrances or not).

The inputs for this layer are the current location data from the robot ($curr\_loc$) and the output from Layer 3 ($target\_landmark$). During each cycle, it tries to plan for the next landmark in Cartesian coordinates and prove $move\_cmd([X, Y])$. If successful, it inputs $destination(X, Y)$ into Layer 1. The motion-focused sub-theory uses a map and a simple axiom to translate landmarks to Cartesian locations.

$$target\_landmark(L) \wedge cartesian(L, [X, Y]) \wedge \neg elevator\_related(L) \Rightarrow move\_cmd(X, Y)$$

(cf. Appendix B.3, formula 1 in the sample proof.)

The theory also has additional axioms describing the different logical positions

involved, and when the elevator is relevant. For example,

$$elevator(L) \iff (L = elev1 \vee L = elev2)$$

$$(L \neq front(elev(floor(F))) \vee \neg current\_landmark(elev(floor(F))))\wedge$$
$$L \neq elev(floor(F)) \wedge \neg elevator(L) \Rightarrow \neg elevator\_related(L)$$

$$cartesian(corridor\_cross, [805, -300])$$

(cf. Appendix B.3, formulae 11-13,5,31 in the sample proof.)

An example proof of $move\_cmd(X, Y)$ (with X, Y variables that get instantiated in the proof) from this theory and the message $target\_landmark(corridor\_cross)$ (received from Layer 3) is

```
Goal#  Wff#  Wff Instance
-----  ----  ------------
  [0]    0   query :- [1].
  [1]    1      move_cmd(805, -300) :- [2] , [3] , [5].
  [2]   89         target_landmark(corridor_cross).
  [3]    5         not_elevator_related(corridor_cross):-[4].
  [4]   13            not_elevator(corridor_cross).
  [5]   31         cartesian(corridor_cross, [805, -300]).
'proof succeeded. move to coordinates ('805,-300')'
```

Finally, the elevator-focused part of this layer is a situation calculus theory (McCarthy and Hayes, 1969) with four main fluents: the landmark of the robot, the landmarks of the two elevators and whether two landmarks are visually connected. There are four action schemas: $moveto(L)$, which moves the robot to landmark L; $callElev$, which calls the elevator; $orderElev(floor(F))$, which commands the elevator to go to floor F, and $wait$, which results in waiting an undetermined amount of time. The effect axioms for these actions are the following:

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

$$at(r, front(elev(floor(F))), S) \Rightarrow$$
$$(at(r, front(elev(floor(F))), result(callElev, S))\wedge$$
$$(at(elev1, floor(F), result(wait, result(callElev, S)))\vee$$
$$at(elev2, floor(F), result(wait, result(callElev, S)))))$$

$$at(r, X, S) \Rightarrow at(r, X, result(wait, S))$$

$$at(r, E, S) \wedge elev(E) \Rightarrow$$
$$(at(r, E, result(orderElev(floor(F)), S))\wedge$$
$$at(E, floor(F), result(wait, result(orderElev(floor(F)), S))))$$

(cf. Appendix B.3, formulae 18-21 in the sample proof.) The first says that if two positions are visually linked in a situation then moving from one to the other results

in the robot being in the other position. The second axiom says that after calling the elevator and waiting, one of the elevators (there are two) will come. For this situation calculus theory we need some frame axioms, which we add by simply specifying them by hand (the number of effect axioms and fluents is small, so there is no harm in specifying them explicitly), interleaved with the effect axioms above.

5.3 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 $destination$ from Layer 2 and given the location and orientation of the robot,[4] it concludes the existence of a "virtual pushing object" in a particular location close to the robot and opposite the destination. When input into Layer 0's theory, Layer 0's obstacle avoidance behavior effectively moves the robot towards the destination.

We use the following symbols in our description of Layer 1:

- Predicate symbols that should ideally be constants: $push\_object(z)$ (where z is a special constant that denotes a pushing object), $nquads(8)$, $push\_obj\_dist(20)$, $curr\_loc(\langle X0\rangle, \langle Y0\rangle)$, $destination(\langle Xd\rangle, \langle Yd\rangle)$.
- Predicate symbols that should ideally be functions: $quadrant(\langle X\rangle, \langle Y\rangle, \langle Qd\rangle)$, $direction(\langle Obj\rangle, \langle Dir\rangle)$, $distance(\langle Obj\rangle, \langle Dist\rangle)$.
- Remaining predicates symbols: $marginal\_distance(\langle X0\rangle, \langle Y0\rangle, \langle Xd\rangle, \langle Yd\rangle)$, $object(\langle Obj\rangle)$, $has\_push\_object(\langle Qd\rangle)$.

The meaning of these symbols will become clearer in the theory description below. Furthermore, a complete list of the symbols used in the layer and their intended meanings are described in detail in Appendices B.4 and B.5.

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 divides the world into a set of quadrants and uses the goal location to decide where to place a pushing object (if at all). First, this object, represented by the predicate $push\_object$, is only legitimized as a bona fide object if the destination isn't already near enough:

$$\forall X0, Y0, Xd, Yd, PO.\ push\_object(PO) \wedge curr\_loc(X0, Y0) \wedge$$
$$destination(Xd, Yd) \wedge \neg marginal\_distance(X0, Y0, Xd, Yd) \Rightarrow$$
$$object(PO).$$

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

(cf. Appendix B.5, formula 6 in the sample proof.) The pushing object is placed in the middle of the quadrant opposite the destination quadrant:

$$\forall X0, Y0, Xd, Yd, Qd.\ curr\_loc(X0, Y0) \wedge destination(Xd, Yd) \wedge$$
$$quadrant(X0{-}Xd, Y0{-}Yd, Qd) \Rightarrow$$
$$has\_push\_object(Qd).$$
$$\forall Qd, NQ, PO, Dist\_po.\ nquads(NQ) \wedge has\_push\_object(Qd) \wedge$$
$$push\_object(PO) \wedge push\_obj\_dist(Dist\_po) \Rightarrow$$
$$direction(PO, (Qd + 0.5) * \tfrac{2\pi}{NQ}) \wedge distance(PO, Dist\_po).$$

(cf. Appendix B.5, formulae 21–23 in the sample proof.) $nquads$ is the number of quadrants and $push\_obj\_dist$ is the distance from the robot at which to place the pushing object.

The complete theory appears in Appendices B.4 and B.5.

The theorem prover attempts to find an object PO such that it can prove $object(PO)$ and $push\_object(PO)$, and attempts to find values $Dist\_po$ and Dir_po that make $distance(PO, Dist\_po)$ and $direction(PO, Dir\_po)$ true. If successful, it inserts these sentences into Layer 0. The following is a sample of a successful proof of the need for a pushing object.

```
Goal#  Wff#  Wff Instance
-----  ----  ------------
  [0]    0   query :- [1] , [10].
  [1]    6      object(z) :- [2] , [3] , [7] , [8].
  [2]    5         push_object(z).
  [3]   31         curr_loc_internal(38, -103) :- [4] , [5].
  [4]   58            curr_loc(38, -103).
  [5]   35            offset_internal(0, 0, 0) :- [6].
  [6]   60               offset(0, 0, 0).
  [7]   61         destination(805, -300).
  [8]    8         not_marginal_distance(38, -103, 805, -300) :- [9].
  [9]    3            margin(50).
 [10]    5      push_object(z).

Goal#  Wff#  Wff Instance
-----  ----  ------------
  [0]    0   query :- [1] , [23].
  [1]   22      distance(z, 20) :- [2] , [3] , [4] , [11] , [14].
  [2]    5         push_object(z).
  [3]    9         push_obj_dist(20).
  [4]   21         has_push_object(3) :- [5] , [9] , [10].
  [5]   31            curr_loc_internal(38, -103) :- [6] , [7].
  [6]   58               curr_loc(38, -103).
  [7]   35               offset_internal(0, 0, 0) :- [8].
  [8]   60                  offset(0, 0, 0).
  [9]   61            destination(805, -300).
 [10]   16            quadrant(38-805, -103- -300, 3).
 [11]   23         quad_angle(3, 2.74889) :- [12] , [13].
 [12]    1            pi(3.14159).
 [13]   11            nquads(8).
 [14]   24         angle_world_vs_robot(2.74889, 2.74889- -1.11352-2*3.14159) :-
                         [15] , [21] , [22].
```

```
[15]   32            curr_dir_internal(-1.11352) :- [16] , [17] , [19].
[16]   59               curr_dir(2962).
[17]   35               offset_internal(0, 0, 0) :- [18].
[18]   60                  offset(0, 0, 0).
[19]   33               angle_deg_rad(2962-0, -1.11352) :- [20].
[20]    1                  pi(3.14159).
[21]    1            pi(3.14159).
[22]   27            between_minus_and_plus(3.14159, 2.74889- -1.11352,
                        2.74889- -1.11352-2*3.14159).
[23]   22      direction(z, 2.74889- -1.11352-2*3.14159) :-
                     [24] , [25] , [26] , [33] , [36].
[24]    5         push_object(z).
[25]    9         push_obj_dist(20).
[26]   21         has_push_object(3) :- [27] , [31] , [32].
[27]   31            curr_loc_internal(38, -103) :- [28] , [29].
[28]   58               curr_loc(38, -103).
[29]   35               offset_internal(0, 0, 0) :- [30].
[30]   60                  offset(0, 0, 0).
[31]   61            destination(805, -300).
[32]   16            quadrant(38-805, -103- -300, 3).
[33]   23         quad_angle(3, 2.74889) :- [34] , [35].
[34]    1            pi(3.14159).
[35]   11            nquads(8).
[36]   24         angle_world_vs_robot(2.74889, 2.74889- -1.11352-2*3.14159) :-
                        [37] , [43] , [44].
[37]   32            curr_dir_internal(-1.11352) :- [38] , [39] , [41].
[38]   59               curr_dir(2962).
[39]   35               offset_internal(0, 0, 0) :- [40].
[40]   60                  offset(0, 0, 0).
[41]   33               angle_deg_rad(2962-0, -1.11352) :- [42].
[42]    1                  pi(3.14159).
[43]    1            pi(3.14159).
[44]   27            between_minus_and_plus(3.14159, 2.74889- -1.11352,
                        2.74889- -1.11352-2*3.14159).
push object distance = 20
push object direction = -2.42077
```

5.4 Layer 0: Obstacle-Avoidance

Layer 0 is responsible for deciding what low-level action the robot should perform. Our description of this theory uses the following symbols:

- Predicate symbols that should ideally be constants: $nsonars(16)$, $min\_speed(10)$, $min\_angle(0.3)$, $fwd(\langle Speed \rangle)$, $heading\_speed(\langle Speed \rangle)$, $turn(\langle Angle \rangle)$, $heading\_angle(\langle Angle \rangle)$, $get\_force([\langle ForceMag \rangle, \langle ForceDir \rangle])$.
- Predicate symbols that should ideally be functions: $adjacent\_right\_sonar(\langle SonarL \rangle, \langle SonarR \rangle)$, $sonar\_direction(\langle Sonar \rangle, \langle Dir \rangle)$, $sonar\_reading(\langle Sonar \rangle, \langle Dist \rangle)$, $correct\_distance(\langle Sonar \rangle, \langle Dist \rangle)$, $distance(\langle Obj \rangle, \langle Dist \rangle)$, $direction(\langle Obj \rangle, \langle Dir \rangle)$.
- Remaining predicate symbols: $object(\langle Obj \rangle)$, $sonar(\langle Sonar \rangle)$, $need\_turn$, $need\_fwd$.

A complete list of symbols and their intended meanings are described in detail in Appendices B.4 and B.6. (Note that in our implementation $correct\_distance$, $need\_turn$, and $need\_fwd$ each take an additional argument which we ignore here

– without loss of generality – for the sake of clarity.)

The theory has two main parts: sensory-focused and control-focused. We divide it to conceptually correspond to the modules shown in Figure 1. During each cycle of layer 0, the theorem prover of layer 0 attempts to prove $fwd(Speed)$ and $turn(Angle)$, where $Speed$ and $Angle$ are instantiated by the proof. If successful, the results are passed to Layer -1.

The inputs for this layer are the sonar data and the output from Layer 1. The input language includes the symbols $sonar\_reading$, $object$, $distance$ and $direction$. The output includes fwd and $turn$.

The sensory-focused part considers sensory input only from the sonars. It takes its input, asserted in the form of the axiom schema

$$sonar\_reading(Sonar\_number, Dist)$$

from the physical sonars and translates it into a map of objects, one per sensor, recording their distance and direction (relative to the robot) [5]. The direction of each object is the corresponding sonar's direction. This interpretation of the data relies, of course, on the assumptions that each sonar observes a different object and objects are points. These assumptions are not significant for the purpose of obstacle-avoidance.

$direction$ is *effectively* made a function by restricting angles to be in the range $[0, 2\pi]$. To account for noise, the distance of each object is computed by applying a coarse filter to the sonar reading by taking a weighted average of the readings of the sonar and its two neighboring sonars. This is based on the assumption that objects observed by neighboring sensors tend to be close. Obviously, there are special scenarios where this assumption breaks down, but we found it to be effective in practice.

$$\begin{aligned}
&\forall Sonar, LeftS, RightS, Dist, DistL, DistR.\\
&\quad adjacent_right_sonar(Sonar, RightS) \wedge\\
&\quad adjacent_right_sonar(LeftS, Sonar) \wedge\\
&\quad sonar_reading(LeftS, DistL) \wedge sonar_reading(Sonar, Dist) \wedge\\
&\quad sonar_reading(RightS, DistR) \wedge Dist \leq DistL + DistR \Rightarrow\\
&\qquad correct_distance(Sonar, (DistL + 4 * Dist + DistR)/6).
\end{aligned}$$

[5] 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 $nsonars - 1$.

$$\forall Sonar, LeftS, RightS, Dist, DistL, DistR.$$
$$adjacent\_right\_sonar(Sonar, RightS) \wedge$$
$$adjacent\_right\_sonar(LeftS, Sonar) \wedge$$
$$sonar\_reading(LeftS, DistL) \wedge sonar\_reading(Sonar, Dist) \wedge$$
$$sonar\_reading(RightS, DistR) \wedge DistL + DistR < Dist \Rightarrow$$
$$correct\_distance(Sonar, (DistL + 2 * Dist + DistR)/4).$$

$$\forall Sonar, Dir, Dist.\ sonar(Sonar) \wedge$$
$$sonar\_direction(Sonar, Dir) \wedge correct\_distance(Sonar, Dist) \Rightarrow$$
$$(\exists Obj.\ object(Obj) \wedge direction(Obj, Dir) \wedge distance(Obj, Dir)).$$

(cf. Appendix B.6, formulae 14–16 in the sample proof.) In the implementation, we replace Obj in the last axiom with a Skolem function obj_sk1. We only have an implication from sonars to objects because we minimize $object$ in our circumscription (see Section 5.6). For the same reason, we do not include axioms stating that there is at most one object at any location. Also, like $direction$, $sonar\_direction$ is restricted to the range $[0, 2\pi]$ to effectively make it a function. The sensory-focused part may also discover "virtual" objects by way of layer 1's subsumption latch.

The control-focused part decides which of the actions to perform, summing up the "repulsive forces" that the different objects around the robot exert on it; these forces are correlated to the distances of the objects from the robot. It uses the resulting force to determine whether the robot should turn (and how much) or move forward (and how fast).

get_force does the dirty work of computing the combined repulsive force from the different detected objects. It is equal to a pair $[ForceMag, ForceDir]$ where

$$ForceMag = \sqrt{Force_y{}^2 + Force_x{}^2}.$$
$$ForceDir = \tan^{-1}(\tfrac{Force_y}{Force_x}).$$
$$Force_x = \sum_{object(Obj), distance(Obj,Dist), direction(Obj,Dir)} Dist \cdot \cos(Dir).$$
$$Force_y = \sum_{object(Obj), distance(Obj,Dist), direction(Obj,Dir)} Dist \cdot \sin(Dir).$$

We implement get_force as a library function rather than as a logical theory: it does not gain much by the logical representation, it can be implemented more efficiently as a procedure, and the logical representation would use some library functions anyway. In the future, if we want to enjoy the benefits of the declarative approach (discussed in Section 1) for this part as well, then our implementation will have to use of a more advanced semantic-attachments or theorem proving technology.

The layer uses the $ForceDir$ 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.

$$\forall ForceMag, ForceDir.\ get\_force([ForceMag, ForceDir]) \Rightarrow$$
$$heading\_angle((ForceDir \bmod 2\pi) - \pi) \wedge heading\_speed(ForceMag).$$

(cf. Appendix B.6, formula 41 in the sample proof.) Note that, like the direction of each object, $ForceDir$ and $heading\_angle$ are computed relative to the robot's current orientation.

$heading\_angle$ and $heading\_speed$ are only taken seriously if they are larger than constant thresholds min_angle and min_speed, respectively.

$$\forall Angle, MIN\_ANGLE.\ min\_angle(MIN\_ANGLE) \wedge heading\_angle(Angle) \Rightarrow$$
$$(Angle > MIN\_ANGLE \iff need\_turn).$$
$$\forall Speed, MIN\_SPEED.\ min\_speed(MIN\_SPEED) \wedge heading\_speed(Speed) \Rightarrow$$
$$(Speed > MIN\_SPEED \iff need\_fwd).$$

(cf. Appendix B.6, formulae 45, 46, 48, and 49 in the sample proof.) If $heading\_angle$ is significant, the output $turn$ is set to it. If $heading\_speed$ is significant and no turn is needed, the output fwd is set to it (we disallow simultaneous forward and rotational motion).

$$\forall Angle.\ need\_turn \wedge need\_fwd \wedge heading\_angle(Angle) \Rightarrow turn(Angle).$$
$$\forall Speed.\ \neg need\_turn \wedge need\_fwd \wedge heading\_speed(Speed) \Rightarrow fwd(Speed).$$

(cf. Appendix B.6, formulae 44 and 47 in the sample proof.)

The complete theory appears in Appendices B.4 and B.6.

The Layer 0 theorem prover attempts to prove $turn(A)$ and $fwd(S)$. If either proof is unsuccessful, it sets the corresponding constant to the default 0. It then inserts the sentences into Layer -1's subsumption latch.

We now consider a sample proof. First, we collect the set of objects for get_force to use:

```
Goal#  Wff#  Wff Instance
-----  ----  ------------
  [0]    0   query :- [1] , [3] , [5].
  [1]   59      object(z) :- [2].
  [2]   83         external_object(z).
  [3]   60      distance(z, 20) :- [4].
  [4]   84         external_distance(z, 20).
```

```
  [5]   61      direction(z, -3.08225) :- [6].
  [6]   85         external_direction(z, -3.08225).

Goal#  Wff#  Wff Instance
-----  ----  ------------
  [0]    0   query :- [1].
  [1]   14      object(obj_sk1((91+65+4*140)//6, 1* (2*3.14159/16))) :-
                     [2] , [3] , [5] , [8].
  [2]    2         pi(3.14159).
  [3]   52         sonar_reading_internal(1, 140) :- [4].
  [4]   65            sonar_reading(1, 140).
  [5]   13         sonar_direction(1, 1* (2*3.14159/16)) :- [6] , [7].
  [6]    2            pi(3.14159).
  [7]    4            nsonars(16).
  [8]   15         correct_dist(1, 140,  (91+65+4*140)//6) :-
                        [9] , [10] , [11] , [13].
  [9]   17            adjacent_right_sonar(1, 0).
 [10]   18            adjacent_right_sonar(2, 1).
 [11]   52            sonar_reading_internal(2, 91) :- [12].
 [12]   66               sonar_reading(2, 91).
 [13]   52            sonar_reading_internal(0, 65) :- [14].
 [14]   64               sonar_reading(0, 65).

 ... [proofs for sonars 0, 2-15]
```

A subsequent attempt to prove $turn(A)$ fails, so we attempt to prove $fwd(S)$, which succeeds.

```
Goal#  Wff#  Wff Instance
-----  ----  ------------
  [0]    0   query :- [1].
  [1]   47      fwd(26) :- [2] , [6] , [10] , [12].
  [2]   41         heading_speed(26) :- [3] , [4] , [5].
  [3]    1            not_ab_avoid.
  [4]   42            get_move_speed(26, 26).
  [5]   43            get_move_dir(2.90998, -0.23).
  [6]   41         heading_angle(-0.23) :- [7] , [8] , [9].
  [7]    1            not_ab_avoid.
  [8]   42            get_move_speed(26, 26).
  [9]   43            get_move_dir(2.90998, -0.23).
 [10]   45         not_need_turn(-0.23) :- [11].
 [11]    8            min_angle(0.3).
 [12]   49         need_fwd(26) :- [13].
 [13]   10            min_speed(10).
'turn angle = '0
'fwd speed = '26
```

5.5 Layer -1: Halt or Go

The lowest layer, *Layer -1*, is responsible for sending actions to the robot or halting the robot if there are objects with which the robot is about to collide. Our description of this theory uses the following symbols:

- Predicate symbols that should ideally be constants: $min\_dist(30)$, $go\_fwd(Speed)$, $go\_turn(\langle Angle \rangle)$.
- Predicate symbols that should ideally be functions: $sonar\_reading(\langle Sonar \rangle, \langle Dist \rangle)$, $sonar\_direction(\langle Sonar \rangle, \langle Dir \rangle)$,

$distance(\langle Obj \rangle, \langle Dist \rangle)$, $direction(\langle Obj \rangle, \langle Dir \rangle)$.

- Remaining predicates: $object(\langle Obj \rangle)$, $sonar(\langle Sonar \rangle)$, $object\_ahead$, $fast\_halt\_robot$, $halt\_robot$.

A complete list of symbols and their intended meanings are described in detail in Appendices B.4 and B.6.

If Layer 0 proves $fwd(S)$ and/or $turn(A)$, these are inserted into Layer -1 as $external\_fwd(S)$ and $external\_turn(A)$ and translated by the sensor module into $go\_fwd(S)$ and $go\_turn(A)$. These commands get passed to the robot if Layer -1 finds no reason to halt.

Layer -1 axiomatizes two halt predicates: $fast\_halt\_robot$ and $halt\_robot$. $fast\_halt\_robot$ is an easy-to-compute approximation of whether the robot should halt or not. It is made true if the front sonar's raw data indicates that an object may be too close (i.e., less than the constant min_dist away) in front of the robot (any direction between $-\pi/3$ and $\pi/3$ radians relative to the robot).

$$\begin{aligned}
&\forall Sonar, Dir, Dist, MIN_DIST.\ sonar(Sonar) \wedge \\
&\quad sonar_direction(Sonar, Dir) \wedge Dir \leq \pi/3 \wedge Dir \geq 2\pi - \pi/3 \wedge \\
&\quad sonar_reading(Sonar, Dist) \wedge min_dist(MIN_DIST) \wedge Dist \leq MIN_DIST \\
&\qquad \Rightarrow fast_halt_robot.
\end{aligned}$$

(cf. Appendix B.6, formula 36 in the sample proof.) If $fast\_halt\_robot$ is proven to be false and Layer 0 has sent a forward command, the LSA concludes it is reasonably safe to pass the command on to the robot. Otherwise, a more rigorous check is required: $halt\_robot$.

$halt\_robot$ is true iff an object is detected directly ahead.

$$object\_ahead \iff halt\_robot$$

(cf. Appendix B.6, formulae 37 and 38 in the sample proof.) An object is considered directly in front of the robot iff there is an object determined to be too close in front of the robot.

$$\begin{aligned}
&object_ahead \iff \\
&\quad (\exists Obj, Dist, Dir.\ object(Obj) \wedge \\
&\quad distance(Obj, Dist) \wedge Dist < min_dist \wedge \\
&\quad direction(Obj, Dir) \wedge Dir < \pi/3 \wedge Dir > 2\pi - \pi/3)\,.
\end{aligned}$$

(cf. Appendix B.6, formulae 39 and 40 in the sample proof.) In the implementation, we replace Obj, $Dist$, and Dir with Skolem functions obj_sk2, $dist\_sk1$, and

dir_sk1, respectively, for the $\Rightarrow$ direction.

Objects, their distances, and their directions are computed as in Layer 0. The complete theory appears in Appendices B.4 and B.6.

The Layer -1 theorem prover first attempts to prove $go\_turn(A)$. If it succeeds, Layer -1 directs the robot to turn A degrees. The theorem prover then attempts to prove $fast\_halt\_robot$. If it succeeds, it attempts to prove $halt\_robot$. If both proofs succeed, the layer sends a halt command to the robot. Otherwise, the theorem prover attempts to prove $go\_fwd(S)$. If it succeeds, then the layer sends the robot a command to move forward at a speed of S. If it doesn't, it commands the robot to halt.

Consider the following example proofs. In the first example, Layer 0 has not computed any commands yet and no inputs have been received from the sonars:

```
'start prove go_turn(A)'
'failed proof. '
'start prove fast_halt_robot'
'proof failed'
'start prove go_fwd(S)'
'failed proof. '
'turn angle = '0
'fwd speed = '0
```

None of the proofs succeed, so the default behavior is to halt. In the second example, the sonars have reported values and Layer 0 has sent a command to go forward at a speed of 24:

```
'start prove go_turn(A)'
Proof:
Goal#  Wff#  Wff Instance
-----  ----  ------------
  [0]    0   query :- [1].
  [1]   63      go_turn(0) :- [2].
  [2]   84         external_turn(0).
'start prove fast_halt_robot'
'proof failed'
'start prove go_fwd(S)'
Proof:
Goal#  Wff#  Wff Instance
-----  ----  ------------
  [0]    0   query :- [1].
  [1]   62      go_fwd(24) :- [2].
  [2]   83         external_fwd(24).
'turn angle = '0
'fwd speed = '24
```

None of the halt proofs succeed, so Layer -1 passes the forward command on to the robot.

5.6 The Default Assumptions of Robots' Layers

Each one of the layers has some default assumptions that it makes, as described in Section 3. In this section we list those nonmonotonic assumptions that are made by the layers of the system described in Figure 6. The current implementation of these assumptions on the robot are not always logically equivalent to this circumscription, as we use the negation-as-failure mechanism of Prolog together with a hard bound on the resources used in proof finding. Thus, no actual circumscription computation is involved in the implementation.

5.6.1 Layer 3's Assumptions

We add the circumscription formula

$$Circ[Layer_3; \Phi_3; L(Layer_3)]$$

where $\Phi_3(L) = \exists S, Sg.(at(r, L, S) \wedge firstSit(S, Sg) \wedge atgoal(r, Sg))$. This layer tries to prove that the next action to be performed by the robot is a movement to a particular location. The minimization of Φ_3 implies that it is assumed that there is no target landmark unless we prove so.

5.6.2 Layer 2's Assumptions

We add the circumscription formula

$$Circ[Layer_2; move\_cmd; \overrightarrow{Z}_2].$$

This formula says that we minimize the movements of the robot: if we cannot prove that there is a target landmark, we should stay put. Thus, we wish to minimize the movement of the robot, varying everything else that we know. If we prove that there is a movement command, then the robot must move. Otherwise, it can stay put. Notice that the layers below may have to take this decision into account and revise it if there are other factors that they need to consider (e.g., objects about to collide with the robot).

5.6.3 Layer 1's Assumptions

We add the circumscription formula

$$Circ[Layer_1; \Phi_1 < \Phi_2; \overrightarrow{Z}_1].$$

where $\Phi_1 = object(Z) \wedge push\_object(Z)$ and $\Phi_2 = \exists D, A.(distance(Z, D) \wedge direction(Z, A))$. This formula says roughly that unless we can prove that there

is another object (z) that is a pushing object and with some distance and direction, then we assume that there is no such object. In fact, Layer 1's theorem prover attempts to prove $object(z)$, $direction(z, A)$, and $distance(z, D)$ during each cycle. Upon successfully proving these, it introduces them into Layer 0's input latch, which then uses the new object to modify its behavior (avoid this object as well).

5.6.4 Layer 0's Assumptions

We add the circumscription formula

$$Circ[Layer_0; object < distance < direction < turn < fwd; \overrightarrow{Z}_0].$$

During each cycle of Layer 0, it applies the CWA to the symbols $object$, $distance$, and $direction$ in the input language. It then uses its theorem prover to try to prove $fwd(Speed)$ and $turn(Angle)$, where $Speed$ and $Angle$ are instantiated by the proof. The results are sent as new axioms into Layer -1's latch. This is modeled by minimizing all of the predicates $object$, $distance$, $direction$, $turn$, and fwd, but giving priority to $object$, $distance$, and $direction$ over $turn$, and to $turn$ over fwd in this minimization.

5.6.5 Layer -1's Assumptions

We add the circumscription formula

$$Circ[Layer_{-1}; go\_turn < fast\_halt\_robot < halt\_robot < go\_fwd; \overrightarrow{Z}_{-1}].$$

During each cycle of Layer -1, it receives information from Layer 0 as to the action it needs to take, go forward or turn. It then tries to prove that this action is indeed taken, if there is no need to halt the robot.

It is important to notice that (for this layer and throughout), in general, messages from higher layers may *set the value* of any of the predicates in $L(Layer_{-1})$ and not only affect the minimized predicate. For example, in Layer -1 the predicate ab_avoid is varied together with the rest of $\overrightarrow{Z}_{-1}$, but higher layers (in the current case, Layer 0 or the input layer) may send a message setting ab_avoid to some value, turning off the obstacle avoidance behavior of Layer -1.

5.7 Layers Do Not Always Have a Model of Time

Many proposals can be used to represent the effects of events and the flow of time (e.g., (McCarthy and Hayes, 1969; Reiter, 1991; Gelfond and Lifschitz, 1998; Miller and Shanahan, 1999; Doherty et al., 1998; Thielscher, 1998; Sandewall,

1994)). Typically, these proposals are considered to have a fixed relationship to the world. For example, situation calculus includes a constant symbol $s0$ which refers to a particular situation.

In contrast to these, many of the theories described above for the layers do not include any notion of time. In particular, Layer -1, Layer 0 and Layer 1 include very reactive-like theories. They are not concerned with the flow of time but rather with the current state of the world. For these theories, time is implicit as are situations.

Even Layers 2 and 3, which have a dynamic picture of the world, do not reason about the flow of time before the current situation. At every cycle, they are asked to prove the goal for their respective layer. For that computation, they regard $s0$ as the current situation, and the sensors determine what we know about this situation. In particular, these layers do not include information from earlier times. This limits the amount of reasoning we can do with these layers, representing a tradeoff made by these layers to avoid costly computations.

Future layers (e.g., a Layer 4 or perhaps parallel layers to Layer 2 and 3) can include such information, having different connections between their symbols and the environment. For example, such layers can regard $s0$ as a fixed situation attached to the beginning of the world of the robot (the time when the robot was turned on). This would entail regarding the current situation as the one that resulted from the actions performed by the robot and other events that occurred since the beginning of the world of the robot ($s0$). This may also require some accounting of the time that passed since the beginning of an action or the beginning of time (as used in (Reiter, 1996), for example).

6 Implementation on a Mobile Robot

We have implemented the above architecture using the PTTP theorem prover (Stickel, 1992; Stickel, 1988b; Stickel, 1988a; Stickel, 2003), on a cluster of Sun SuperSparc 1 stations running SWI Prolog or Quintus Prolog as the underlying interpreter for PTTP. The system runs on a Nomad 200 robot.

6.1 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 nonmonotonic reasoning (or at least some approximation of it), and possibly needs to allow for theory resolution (Stickel, 1985). Theories are also likely to require algebraic computations (such as trigonometric functions) for which systematic algorithms are better suited than a theorem prover. Thus, the theorem prover also needs to allow for easy embedding of semantic attachments.

Provers we examined included Otter (McCune, 1994) (a resolution theorem prover), ACL2 (Kaufmann et al., 2000) (an industrial-strength version of the Boyer-Moore theorem prover) and ATP (Farquhar, 1997) (a model elimination theorem prover). The major difficulties we encountered with them (although not all difficulties were encountered with all provers) were the inability to append semantic attachments easily, the complexity of making the theorem prover run on a given platform, the inability to control the inference process easily (via strategies or otherwise), and the lack of documentation. In addition we also examined a few proof checkers including PVS (Rushby et al., 2003), HOL (Gordon and Melham, 1993) and GETFOL (Giunchiglia, 1994), all of which were found unsuitable due to their need for at least some human intervention.

PTTP (Prolog Technology Theorem Prover) is a model-elimination theorem prover 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).

Compared to the other theorem provers we examined, 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 use of semantic attachments. Also, its output at run-time together with its iterative-deepening procedure allow for proof-progression analysis and control. Finally, although PTTP lacked suitable documentation, there was a fair collection of sufficiently illustrative examples. As a result, despite some difficulties incurred by the use of semantic attachments and built-in predicates (such as the algebraic $<$ relation), PTTP gave us relatively little trouble in either installation or use.

6.2 The Software and Dynamics of the System

Our 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 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 com-

munication (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 (is it 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 arrived on a specific port, we reuse the information that arrived previously. This allows the layers to work in 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.

6.3 Offline Experiments with PTTP and Simulated Sensors

	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

Table 1
Proof time (in seconds) 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*.)

We subjected our system to a set of experiments in a simulated office building environment. Table 1 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 (results from a sample online run are shown in Section B [6]).

Layers 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. (Because of space concerns, we have included in Table 1 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 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.

6.4 LiSA's Online Behavior on a Nomad 200 Robot

We ran several experiments on LiSA with goals of traveling to different rooms in the building. Figure 7 presents average total time measurements for each layer during these experiments. This figure shows that the current implementation is adequate for a reactive behavior, especially given that without having to log its behavior and with faster computers we can speed up the system by a factor of about 8. A few improvements can be made to the implementation that would make it faster (e.g., use a faster theorem prover, faster hardware, better route planner in Layer 3, better planner in Layer 2, and a few optimizations of the code), and we hope to use some of those in the future.

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 (see Figure 8). 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 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.

---

[6] 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.

[7] 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

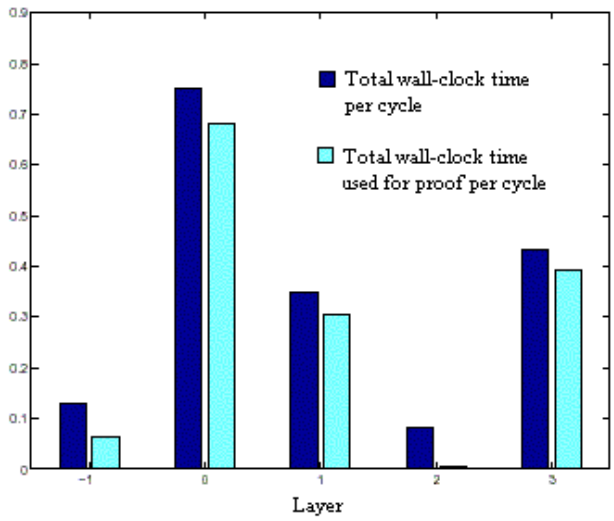

Fig. 7. Average time (in seconds) per cycle for each of the layers [7] (each column corresponds to the time taken by the indicated layer). The bars for each layer correspond to the total wall-clock time per cycle and the total wall-clock time used for the proof in each cycle, in this order from left to right.

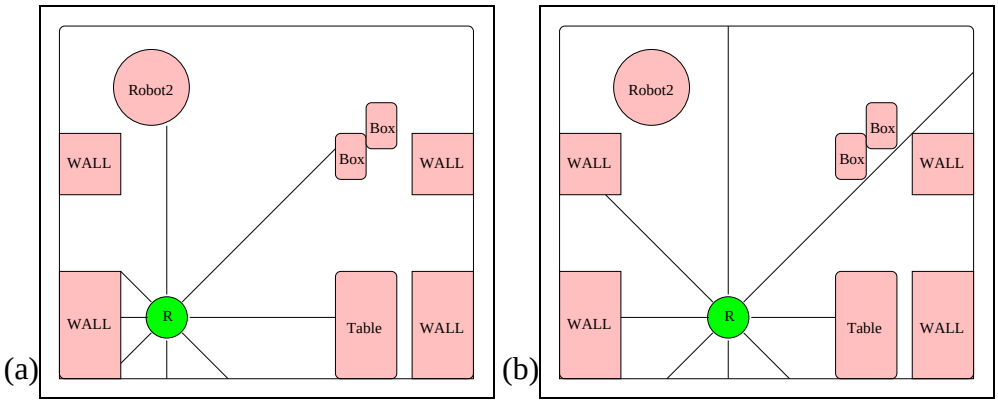

Fig. 8. LiSA's movement using sonars.

Figures 8 through 10 diagrammatically display an execution of LiSA's movement across the lab. Between Figures 8(a) and 8(b) LiSA's Layer 0 pushes it away from obstacles. From Figure 9(a) to Figure 10(b) Layer 1 sends an axiom to Layer 0 telling it that there is another (*virtual*) object that should influence its decision on where to go. In Figure 10(a) Layer 1 sends an axiom that asserts the existence of this object with a changed position (relative to the robot) to account for the fact the the robot did not go straight towards its goal (Figure 10(b)).

In our experiments, LiSA took from 30 seconds to two minutes to move from one landmark to the next, for landmarks approximately 4-10 meters apart and obstacles 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.

runs. These mechanisms typically slow the system down by a factor of about 4.

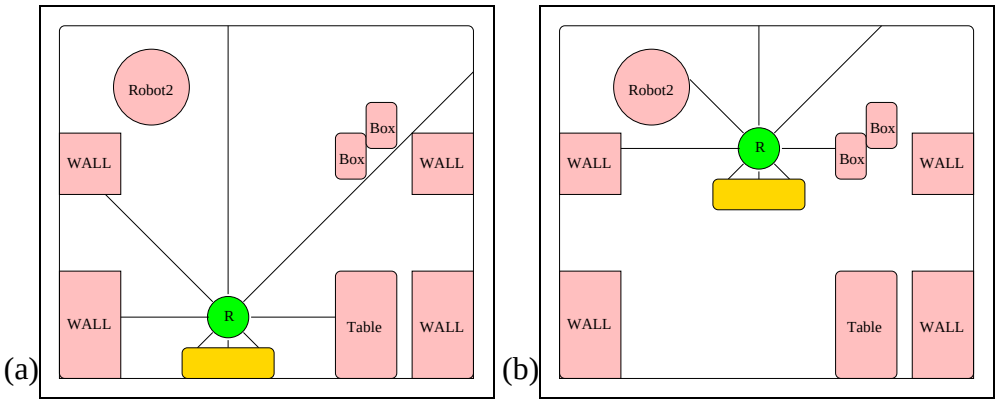

Fig. 9. LiSA's movement using sonars and a *pushing object*.

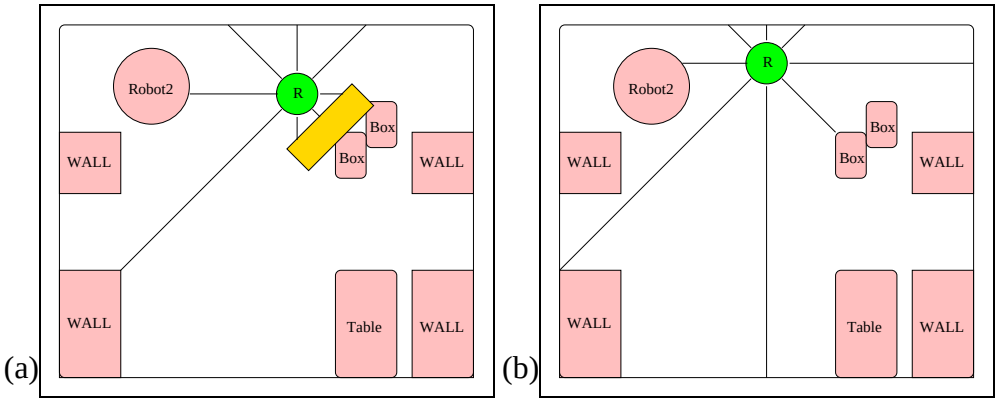

Fig. 10. LiSA's movement using sonars and a *pushing object*.

One of the strengths of our architecture is that it allows us to send more knowledge to the system as it is running. We can send such knowledge to change the behavior of the system or extend it.

For example, 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.

In a more recent experiment a robot was occupying one of the landmarks that LiSA wanted to use. LiSA kept trying to get to that landmark without success. In this case, we were able to use the input layer to send an axiom into LiSA's layers 2,3 telling them that there is a path between two other landmarks. This allowed LiSA to use a different path on its way to the goal location.

These incidents showcase the strength of using the representation-and-reasoning approach in general and the LSA architecture in particular. There are always things that the designer of a robotic system cannot foresee, cases in which some programmers will have to go and recompile (sometimes redesign) the system with a patch that would take care of the problem. When using the approach that we took, one can simply send more knowledge to the system while it is running, thereby changing its behavior in the desired way with no need for recompilation or redesign of the software.

6.5 Tuning the System

Theorem provers are notoriously slow, which is one of the main reasons they are usually 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 also attribute the speed achieved by our system to several optimizations that we describe below.

First, dividing the planning so that Layer 2 executes "local planning" for the elevator domain allows 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 helps prevent complex unifications.

Rather than use the theorem prover to compute the predicate get_force in Layer 0, it is embodied in a C function semantic attachment, significantly speeding up its computation. The function calls Prolog's setof operator to collect all the objects for which existence proofs can be found (applying an implementation of the CWA described below), computes the sum of the forces contributed by each object, then returns the force vector $[Strength, Direction]$.

Furthermore, since every proof in Layer 0 re-proves get_force many times, caching these proofs also improved the performance of Layer 0 significantly (from approximately 10 seconds to 0.1 seconds per proof on a Sun UltraSparc 60).

During our experiments it became clear to us that much of the bottleneck in some layers is 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 implement a rough approximation for defaults. For ex-

ample, 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 in. 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).

7 Related Work: Cognitive Robotics and Subsumption Architectures

Compared to other approaches to robot-control that use logic, the LSA is the only system using general-purpose, full FOL theorem provers for reasoning and control. 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. Other differences between our system and previous work exist and are interesting in the context of future work and combination of the different approaches.

Shanahan (Shanahan, 1996) describes a map-building process using abduction, and then implements his theory in an algorithm that is proved to have an abductive semantics. Later work (Shanahan, 1998; Shanahan and Witkowski, 2000; Shanahan, 2000) uses *abductive logic programming* to solve planning and sensory problems for mobile robots. In particular, some of these implemented systems use hierarchical planning and can deal with some noisy sensory data. The approach described there uses online hierarchical planning, allowing fast planning and execution monitoring. That work chooses to use logic programming tools (especially Prolog) instead of general-purpose theorem provers. In comparison, the work we describe in this paper is based on the expressivity of FOL theorem provers, and makes no assumptions as to the form of the axioms nor does it appeal to restricted expressivity. The form of hierarchical planning used in our work somewhat predates that of (Shanahan, 2000).

Baral and Tran (Baral and Tran, 1998) 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., (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.

Systems based on the GOLOG project (e.g., (Levesque et al., 1997; Giacomo et al.,

1997; Giacomo et al., 1998; Reiter, 1998; Lakemeyer, 1999; Grosskreutz and Lakemeyer, 2000; Boutilier et al., 2000)) have a planner that computes/plans a high-level GOLOG program off-line. This plan is specified using the situation calculus ((McCarthy and Hayes, 1969)). The plan is executed and monitored later by the robot. This family of systems allows plans that are specified on a high level, having nondeterministic actions and actions that require further elaborations. During execution the GOLOG controller uses the plan specification to guide a search in the plan space as needed. This allows the efficient absorption of sensory information, plans that require some limited planning (or re-planning) during execution and the consideration of events that do not depend on the robot (natural events). Logic and situation calculus are used to give semantics for GOLOG programs, keeping the system formally clean.

Another, somewhat earlier, line of work concentrated on compilation of logical descriptions into control languages (e.g., (Kaelbling, 1987; Kaelbling, 1990; Kaelbling and Rosenschein, 1990; Rosenshein and Kaelbling, 1995)). This approach takes a description in logic of the planning problem, the condition-achievement goal, or the condition-maintenance goal. It uses this representation to create a rule-based representation that can be readily used to control an agent in real-time. This approach provides a way to specify agents using logic or a formal model and then to compile this representation to a reactive one.

Compared to all of 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, the ability to send logical-formulae advice to the robot at run-time has not been a property of any system (to our knowledge) since Shakey the robot (Nilsson, 1984).

An interesting application of logic in robotics has emerged in the context of Robotic Soccer. Of particular relevance to our work are (Jung, 1999; Stolzenburg et al., 2000). This research presents layered architectures in which some layers use logic programming languages (e.g., Prolog) to specify predicates and functions that can be used higher layers' reasoning. The lowest layer is reactive (using some Prolog rules) and the higher layers are in charge of choosing one of a set of available scripts to perform (each script depends on the role the robot takes in the game).(Stolzenburg et al., 2000) even goes so far as to propose the use of a general purpose FOL theorem prover, but it does not implement or explore this direction in any detail. The main conceptual difference between our work and these (besides the added expressivity and declarativeness in our system) is in that in our system every layer is autonomous and influence between layers is carried via axioms sent from higher layers to lower ones. This allows our system to be more flexible and reactive.

Compared to subsumption systems for robot control (e.g., (Brooks, 1990; Brooks and Flynn, 1989; Matarić, 1992; Horswill, 1993; Brooks and Stein, 1994)) 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, for better or worse, 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. (Maes, 1989; Nakashima and Noda, 1998) present modifications to the subsumption architecture approach that suggest extensions of our architecture that incorporate voting, but exploring this is outside the scope of the current work.

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.

8 Conclusion

In this paper 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.

Our system combines the virtues of using the represent-and-reason paradigm and the behavioral-decomposition paradigm. It allows multiple goals to be serviced simultaneously and reactively. It also allows high-level tasks and is tolerant to different changes and elaborations of its knowledge in runtime. Finally, it allows us to give more commonsense knowledge to robots. In these characteristics it is the closest system to the Advice Taker portrayed by (McCarthy, 1958) that is known to us.

There are many important avenues for building on our approach. *Memory* and *state* can be added to the system easily, but the logical semantics must be modified to account for them. 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. In this respect, we are exploring the automatic decomposition of tasks into layers that together compose subsumption architectures. We hope to achieve this using principles from (Amir, 2001; Amir and McIlraith, 2003; Amir and Engelhardt, 2003).

In the immediate future we plan to add layers that create maps and layers that

reason about and update explicit beliefs about the world. We are also working on incorporating vision sensory capabilities.

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

Acknowledgments

We wish to thank Mark Stickel for allowing us to use his PTTP source code (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. Finally, the anonymous reviewers of this manuscript contributed greatly to its final form.

This research was supported by an AFOSR grant AF F49620-97-1-0207, and by a National Physical Science Consortium (NPSC) fellowship.

A Proofs

A.1 Theorem 4.4: LSA is Complete for Sensors to Actions

We prove our theorem by induction on the number of layers. By definition 4.1, if T has only one layer ($Layer_0$), then $T \models \varphi$ iff $Circ[Layer_0; \overrightarrow{C}_0; \overrightarrow{Z}_0] \models \varphi$. For $\varphi_0 = \varphi$ and $k = 0$ we get that $Circ[Layer_k; \overrightarrow{C}_k; \overrightarrow{Z}_k] \models \varphi_k$. This proves the theorem for $n = 0$.

Assume that the theorem is correct for n and we prove it for $n+1$. Let T' be the set of layers of T without $Layer_0$. Let $\varphi \in \mathcal{L}(Layer_0)$ such that $T \models \varphi$. By definition, $Circ[Layer_0 \cup Circ[Layer_1 \cup ...; \overrightarrow{C}_1; \overrightarrow{Z}_1]; \overrightarrow{C}_0; \overrightarrow{Z}_0] \models \varphi$.

Let γ be the set of prime implicates of $Circ[Layer_1 \cup ...; \overrightarrow{C}_1; \overrightarrow{Z}_1]$ in $L(T') \cap L(Layer_0)$. From the induction hypothesis we know that there are $k \geq 0$ and $\varphi_1, ...\varphi_k$ such that $\varphi_1 = \gamma$ and $Circ[Layer_k; \overrightarrow{C}_k; \overrightarrow{Z}_k] \models \varphi_k$, and for all i such that $1 \leq i < k$, $\varphi_i \in \mathcal{L}(Goal_i)$ and $Circ[Layer_i \cup \varphi_{i+1}; \overrightarrow{C}_i; \overrightarrow{Z}_i] \models \varphi_i$.

Recall that we assume that for every $i < n$, only predicates of $L(Goal_i)$ can appear

in both $Layer_i, Layer_{i+1}$. From Theorem 4.2,

$$Circ[Layer_0 \cup \gamma; \overrightarrow{C}_0; \overrightarrow{Z}_0] \models \varphi \iff$$
$$Circ[Layer_0 \cup Circ[Layer_1 \cup ...; \overrightarrow{C}_1; \overrightarrow{Z}_1]; \overrightarrow{C}_0; \overrightarrow{Z}_0] \models \varphi$$

Thus, taking $\varphi_0 = \varphi$, and $\varphi_1 = \gamma$ provides the induction step, and $Circ[Layer_0 \cup \gamma; \overrightarrow{C}_0; \overrightarrow{Z}_0] \models \varphi$. ■

A.2 Theorem 4.4: LSA is Sound for Sensors to Actions

We prove our theorem by induction on the number of layers. By definition 4.1, if T has only one layer ($Layer_0$), then $T \models \varphi$ iff $Circ[Layer_0; \overrightarrow{C}_0; \overrightarrow{Z}_0] \models \varphi$. For $k = 0$ we get that $Circ[Layer_k; \overrightarrow{C}_k; \overrightarrow{Z}_k] \models \varphi_k$. This proves the theorem for $n = 0$.

Assume that the theorem is correct for n and we prove it for $n + 1$. Let T' be the set of layers of T without $Layer_0$. By definition, $Circ[Layer_0 \cup Circ[Layer_1 \cup ...; \overrightarrow{C}_1; \overrightarrow{Z}_1]; \overrightarrow{C}_0; \overrightarrow{Z}_0] \models \varphi_0$.

From the induction hypothesis we know that $T' \models \varphi_1$ because φ_1 is the set of prime implicates of $Circ[Layer_1 \cup ...; \overrightarrow{C}_1; \overrightarrow{Z}_1]$ in $L(T') \cap L(Layer_0)$. Recall that we assume that for every $i < n$, only predicates of $L(Goal_i)$ can appear in both $Layer_i, Layer_{i+1}$. From Theorem 4.2,

$$Circ[Layer_0 \cup \varphi_1; \overrightarrow{C}_0; \overrightarrow{Z}_0] \models \varphi_0 \iff$$
$$Circ[Layer_0 \cup Circ[Layer_1 \cup ...; \overrightarrow{C}_1; \overrightarrow{Z}_1]; \overrightarrow{C}_0; \overrightarrow{Z}_0] \models \varphi_0$$

Thus, $Circ[Layer_0 \cup Circ[Layer_1 \cup ...; \overrightarrow{C}_1; \overrightarrow{Z}_1]; \overrightarrow{C}_0; \overrightarrow{Z}_0] \models \varphi_0$, and the induction step is complete. ■

B Complete PTTP Theories

Each layer in our implementation of the LSA loads a theory in its initialization. This theory consists of two main fragments: a control component and a sensory component. The control theories used in the different layers are different typically, but the sensory modules are replicated multiple times. A diagram presenting the configuration of a layer executable with the proper layer theory is presented in Figure B.1. A more detailed view of the architecture (Figure B.2) shows the final allocation of axioms to the different layers (notice that the combination L0,M1 appears twice; layers 0 and -1 use the same axioms but prove different goals).

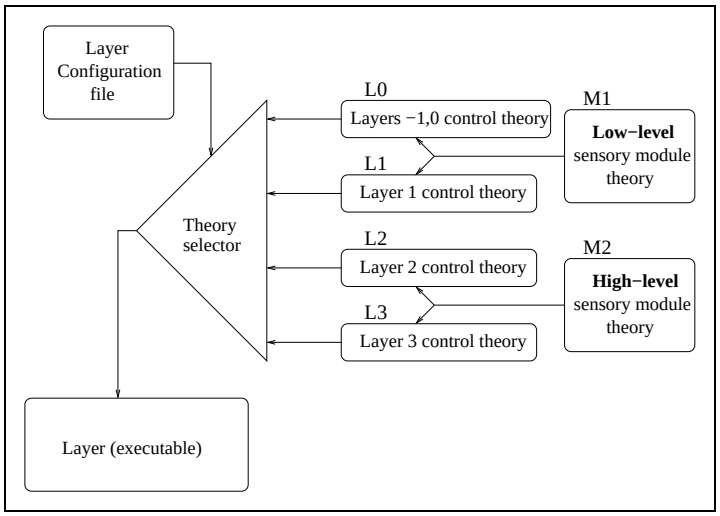

Fig. B.1. The selection of theories for a layer's executable.

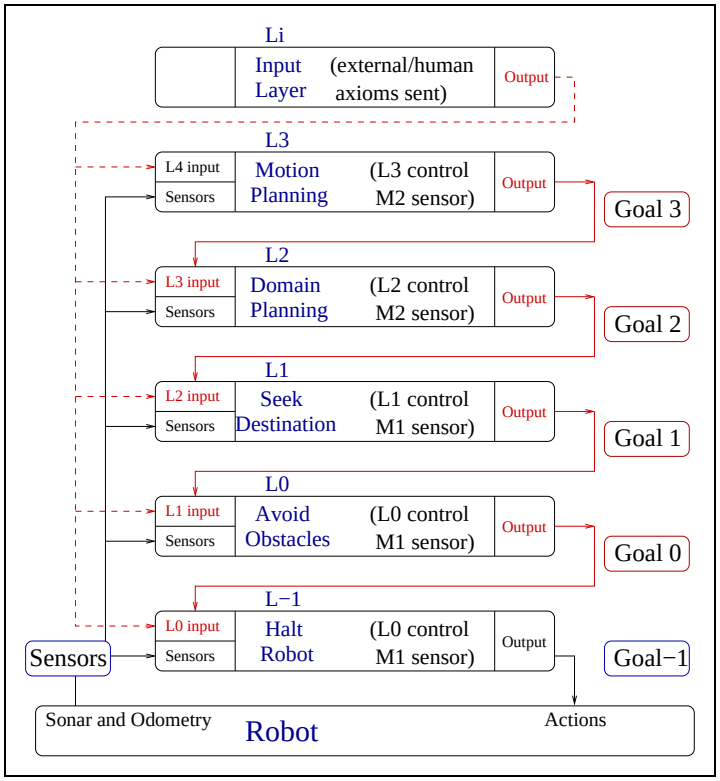

Fig. B.2. Diagrammatic view of an LSA system controlling a robot.

In the following we present the complete theories that are used by a layer executable, sorted into sensory module theories and control layer theories. These detail the different theories presented in Figure B.1. The theories are presented in their original form, a clausal-like form (after Skolemization) suitable as input to PTTP. The following are conventions that are used by the input language of PTTP.

- In PTTP, predicate, function and constant symbols are named using English letter sequences in which the the first letter is in *lower caps*. Variables are named using

English letter sequences in which the the first letter is in *UPPER-CAPS*.

- Variables are implicitly universally quantified and have the scope of the sentence in which they appear.
- Semicolon (";") corresponds to a *logical OR* ("$\vee$").
- Comma (",") corresponds to a *logical AND* ("$\wedge$").
- "$not\_$" preceding a literal corresponds to a *logical negation* ("$\neg$").
- "$:-$" can be used to specify a clause that can be resolved only in a single direction (only literals on the right-hand side can be resolved against other clauses; a resolvent child of this clause can resolve the left-hand side only when that child is a single literal).
- Percentage sign ("%") designates the beginning of a comment that is not interpreted by PTTP. We use these in the body of the axioms to explain axioms and their purpose.
- PTTP handles equality ("$=$") by a unification test. Thus, $=$ means *unifiable*, and $=\backslash=$ means *not unifiable*. It has no other equality predicate, and *paramodulation* or other inference rules for equality are not used. For this reason we attempted to minimize the use of function symbols and equality and to restrict such uses to cases when the unification test is a correct mechanism for testing equality or when equality is tested between arithmetic terms that can be evaluated at the time of equality test. This attempt gives rise to several modeling choices, such as choosing to model the number pi (π) using

$$pi(3.14159), not\_pi(C0) :- CO =\backslash= 3.14159$$

  instead of $pi = 3.14159$.
- The PROLOG predicate $prove(goal, max, min, step)$ is defined by PTTP to try to prove $goal$, and takes the following arguments: $goal$ is the goal to prove, max is the maximum depth of search for a proof (default = large number), min is the minimum depth of the search for a proof (default = 0), and $step$ is the depth bound step size for the iterative deepening search (default = 1).

B.1 High-Level Sensor Module (M2)

This theory represents the sensor module used by layers 2 and 3. It includes the following non-logical symbols:

Predicates • PTTP-internal predicates ($<$, $=$, etc.).

- $pi(\langle PI \rangle)$ – PI is the number π
- $sonar\_reading(0..15, \langle Distance \rangle)$
- $sonar\_reading\_internal(0..15, \langle Distance \rangle)$ – the axioms in our layers refer only to $sonar\_reading\_internal(\cdot)$ and not to $sonar\_reading(\cdot)$ because this way $sonar\_reading$ can be asserted (into Prolog and PTTP) without recompilation of the rest of the axioms; we define similar "$\_internal$" predicates for $curr\_loc$, $offset$, and $curr\_dir$ below.

- $curr\_loc(\langle Xinternal\rangle, \langle Yinternal\rangle)$
- $curr\_loc\_internal(\langle Xinternal\rangle, \langle Yinternal\rangle)$
- $offset(\langle XOff\rangle, \langle YOff\rangle, \langle AngOff\rangle)$
- $offset\_internal(\langle XOff\rangle, \langle YOff\rangle, \langle AngOff\rangle)$
- $curr\_dir(\langle Ainternal\rangle)$
- $curr\_dir\_internal(\langle Ainternal\rangle)$
- $angle\_deg\_rad(\langle DegA\rangle, \langle RadA\rangle)$ – converts angles in degrees to radians and back (required for some of the semantic attachments that we use).
- $cartesian(\langle Logical\_position\rangle, \langle [X, Y]\rangle)$
- $vConnected(\langle Logical\_position1\rangle, \langle Logical\_position2\rangle)$ – the two logical positions are visually connected (there is a line-of-sight between them)
- $inCorridor(\langle Logical\_position\rangle, \langle Corridor\rangle)$
- $room(\langle Logical\_position\rangle)$
- $short\_distance(\langle [X1, Y1]\rangle, \langle [X2, Y2]\rangle)$
- $distance\_threshold(X)$
- $current\_landmark(\langle Landmark\rangle)$
- $at(\langle Agent\rangle, \langle Logical\_position\rangle, \langle Situation\rangle))$

Functions • PTTP-internal functions ($-$, abs, etc.)
- $front(\langle Logical\_position\rangle)$

Constant Symbols • Numerical constants
- $pedrito\_office$, $hector\_office$, $corridor\_entrance$, etc. – logical positions in the world.
- $c1Alink31$, $c1A1$, etc. – corridors
- r – robot
- $s0$ – initial situation (the current state of the world)

$curr\_loc$ and $curr\_dir$ are the robots location and direction with respect to the robot's coordinate system. When the robot is initialized it is put in a physical initial position and these are set to $0, 0$. As the robot proceeds in the world, the odometry measurements change and are integrated into the current location and direction. $offset$ specifies the transformation parameters from a global coordinate system to the robot's (in case we did not put the robot physically in position $0, 0$ at initialization or we chose to introduce some displacement on top of the odometry measurements.

Distances are in 1/10-in, and angles used by the rest of the layer are in radians in the range $[-\pi, \pi]$. Angles reported by the robot's body are in 1/10-deg in the range $[0, 3600]$ and need to be converted (which we do in the axiom defining $curr\_dir\_internal$).

is_sensor_theory is a Prolog predicate that is defined to hold for the theory described henceforth. This allows us to ask different queries from the theory and to join it with other axioms, as demonstrated at the end of the following script.

```
is_sensor_theory((
```

```
%%% Parameters %%%
        pi(3.14159), (not_pi(C0) :- C0=\=3.14159),

%%% The following is an example of sensory input received
%%% by a layer:

%   nsonars(16),
%   sonar_reading(0, 40),   sonar_reading(1, 40),  % front
%   sonar_reading(2, 40),   sonar_reading(3, 1000),
%   sonar_reading(4, 1000), sonar_reading(5, 1000), % left
%   sonar_reading(6, 1000), sonar_reading(7, 1000),
%   sonar_reading(8, 1000), sonar_reading(9, 1000), % rear
%   sonar_reading(10, 1000),sonar_reading(11, 1000),
%   sonar_reading(12, 1000),sonar_reading(13, 1000),%right
%   sonar_reading(14, 1000),sonar_reading(15, 1000),
%   curr_loc(280,230),      curr_dir(3.14159/2+0.40),
%   offset(0,0,0),
%   current_landmark(front(elev(floor(2))))

% If we are dealing with multiple floors then the cartesian
% location above may fit two places in the two different
% floors, so another axis must be added to the position.

%%% Sensory Axioms %%%

% We add *_internal so that it is easy to assert and
% retract sensory input without compilation (saves time for
% sensory assertion).

    (sonar_reading_internal(Snum, DistSonar):-
                    sonar_reading(Snum, DistSonar)),

    (curr_loc_internal(Xinternal, Yinternal):- % subtract
           curr_loc(Xhere, Yhere),        %  offset
           offset_internal(XOff6, YOff6, AngOff6),
           Xinternal is Xhere - XOff6,
           Yinternal is Yhere - YOff6),

    (curr_dir_internal(Ainternal):-  % convert to rad and
           curr_dir(Ang),            % subtract offset
           offset_internal(XOff6, YOff6, AngOff6),
           angle_deg_rad(Ang - AngOff6, Ainternal)),

        % The first translates [0,3600] to [-PI,PI]
    (angle_deg_rad(DegA, RadA):-
           var(RadA), pi(PI),
           RadA is ((((integer(DegA)+1800) mod 3600)-1800)
                                        /3600)*(2*PI)),
        % The second translates [-PI,PI] to [-1800,1800]
    (angle_deg_rad(DegA, RadA):-
           var(DegA), pi(PI),
           DegA is integer((RadA/(2*PI))*3600)),

    (offset_internal(XOff,YOff,AngOff):-
           offset(XOff,YOff,AngOff)),

%%% Map Axioms %%%

% Robotics lab at Stanford Gates building

    cartesian(zero_pt,[0,0]),
    cartesian(corridor_cross,[805,-300]),
    cartesian(mid_lab,[2129,-945]),
    cartesian(among_friends,[2080,60]),
    cartesian(corridor2_cross,[3357,-638]),
    cartesian(corridor_entrance,[397,519]),
    cartesian(front(hector_office),[334,2301]),
    cartesian(front(lise_office),[394,3896]),
```

```
    cartesian(front(uri_office),[522,5939]),
    cartesian(corridor_turn,[601,6929]),
    cartesian(front(pedrito_office),[-594,7047]),
    cartesian(pedrito_office,[-1136,7683]),
    cartesian(mid_corridor3,[2515,5829]),
    cartesian(front(chris_room),[4421,5740]),
    cartesian(front(daphne_room),[4318,3838]),
    cartesian(corridor2_entrance,[4524,1532]),
    cartesian(mid_corridor2,[4817,-704]),

    vConnected(corridor_cross,zero_pt),
    vConnected(mid_lab,corridor_cross),
    vConnected(among_friends,mid_lab),
    vConnected(among_friends,corridor_cross),
    vConnected(among_friends,corridor2_cross),
    vConnected(mid_lab,corridor2_cross),
    vConnected(zero_pt,corridor_entrance),
    vConnected(corridor_cross,corridor_entrance),

    inCorridor(corridor_entrance,c1A1),
    inCorridor(front(hector_office),c1A1),
    inCorridor(front(lise_office),c1A1),
    inCorridor(front(uri_office),c1A1),
    inCorridor(corridor_turn,c1A1),
    inCorridor(corridor_turn,c1Alink31),
    inCorridor(front(pedrito_office),c1Alink31),

    room(pedrito_office), room(hector_office),
    room(lise_office),    room(uri_office),

% Interesting axioms: 1. Front of room is displaced from
% the room according to its direction.  2. rooms on the
% same corridor have identical x (or y) coordinates.

%%% Current Landmark %%%

% predicates: curr_loc, cartesian, current_landmark

    (not_curr_loc(X,Y); not_cartesian(Place, Cartesian2);
         not_short_distance([X,Y], Cartesian2);
         current_landmark(Place)),

    (short_distance([X1,Y1],[X2,Y2]):-
         distance_threshold(Dist), abs(X1-X2)<Dist,
         abs(Y1-Y2)<Dist),
    (distance_threshold(100)),

    (not_current_landmark(CurrPlace); at(r, CurrPlace, s0))

% The above assumes that the current location is one of
% the cartesian pairs in the map above. In general, we want
% to allow arbitrary locations and compute either the closest
% landmark or which room the robot is in. This is taken care
% of in the theory for layer 3 where we in fact need it to
% allow acting when we are in transit between landmarks.
)).
%%%%%%%%%%%%%%%%%% END OF LOGICAL THEORY %%%%%%%%%%%%%%%%%%


sense :- is_sensor_theory(ThS), pttp(ThS).

% The following tests the sensor module in isolation.

testS :- sense, prove((curr_loc(X,Y))),
         print('current location = ('), print(X),
         print(','), print(Y), print(')'), nl.
```

The sentences outside the main is_sensor_theory declaration defining $sense$ and $testS$ perform sensing (loading the axioms above into PTTP) and a single query answering test (trying to prove that the current location is some X, Y, thus returning the values of X, Y that satisfy the query) given some sensory input from the robot (such as those commented out at the beginning of the script).

B.2 Layer 3 Control Theory (L3)

The following file contains the theory of the LSA for Layer 3. This theory together with the high-level sensory theory (M2 in Section B.1) can translate logical locations into Cartesian positions and vice-versa, and can find plans that reach a certain goal location from the current location. The axiomatization is that of high-level actions, describing the entities for spatial reasoning as rooms, corridors, and room fronts.

Our implementation of Layer 3 can be seen as taking a sentence describing a goal situation (typically, this one is sent from the Input Layer) and producing a logical location which is sent to Layer 2 and serves there as a goal location.

It includes the following nonlogical symbols:

Predicates • PTTP-internal predicates ($<$, $=<$, etc.).

- $room(\langle Logical\_position \rangle)$
- $corridor(\langle Corridor \rangle)$
- $inCorridor(\langle Logical\_position \rangle, \langle Corridor \rangle)$
- $vConnected(\langle Logical\_position1 \rangle, \langle Logical\_position2 \rangle)$ – the two logical positions are visually connected (there is a line-of-sight between them)
- $current\_landmark(\langle Landmark \rangle)$
- $curr\_loc(\langle Xinternal \rangle, \langle Yinternal \rangle)$
- $cartesian(\langle Logical\_position \rangle, \langle [X, Y] \rangle)$
- $pos\_between([X1, Y1], [X, Y], [X2, Y2])$ – Cartesian position $[X, Y]$ is in between Cartesian positions $[X1, Y1], [X2, Y2]$. This holds for $[X, Y]$ if it inside the rectangle defined by the line connecting two points $[X1, Y1], [X2, Y2]$ and the distance satisfying the $max\_dist\_from\_line$ predicate (see Figure B.3).
- $max\_dist\_from\_line(\langle Max\_dist\_from\_line \rangle)$ – see $pos\_between$ for explanation
- $intersect(\langle Corridor1 \rangle, \langle Corridor2 \rangle)$ – corridors that intersect
- $at\_static(\langle Item \rangle, \langle Logical\_position \rangle)$ – $Item$ is always at $Logical\_position$ ($Item$ can be a logical position as well).
- $at(\langle Item \rangle, \langle Logical\_position \rangle, \langle Situation \rangle)$
- $doorTo(\langle Door \rangle, \langle Room \rangle)$
- $vLinked(\langle Logical\_position1 \rangle, \langle Logical\_position2 \rangle, \langle S \rangle)$ – locations that are sometimes visually connected and sometimes not (e.g., elevators)

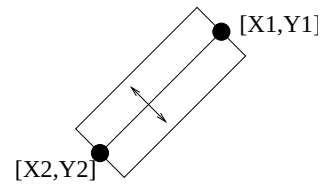

Fig. B.3. The rectangle defined by $[X1, Y1], [X2, Y2], max\_dist\_from\_line$

- $eq(\langle V\rangle, \langle W\rangle)$ – equality predicate that is sometimes used instead of the built-in $=$
- $roomLevelPlace(\langle Logical\_position\rangle)$ – is this logical position a room-type place?
- $atgoal(\langle Item\rangle, \langle Situation\rangle)$ – item is at the goal location in this situation
- $goal\_location(\langle Logical\_position\rangle)$
- $firstSit(\langle S\rangle, \langle S1\rangle)$ – the situation ($S1$) that is immediately after $s0$ in the definition of situation S
- $action(\langle S\rangle, \langle A\rangle)$ – the last action A in a sequence defining a situation S

Functions • PTTP-internal functions ($-$, abs, etc.)

- $between(\langle Logical\_position1, Logical\_position2\rangle)$ – a logical position which corresponds to the region between $Logical\_position1, Logical\_position2$ (see Figure B.3). $Logical\_position1, Logical\_position2$ are expected to correspond to ovals specified by a center Cartesian position and a (fixed) radius.
- $floor(\langle Number\rangle)$ – takes a number and returns a logical location (the floor indexed by this number)
- $front(\langle Logical\_position\rangle)$
- $result(\langle A\rangle, \langle S\rangle)$ – the situation resulting from executing action A in situation S
- $elev(\langle Floor\rangle)$ – the logical location of the elevators at floor $Floor$ (whether an elevator is on that floor or not)

Constant Symbols • Numerical constants

- r – robot
- $rm218$, $rm132$, etc. – rooms
- $c1Aelev$, $c1A1$, etc. – corridors
- $s0$ – initial (current) situation

```
is_theory3((

%%% Spatial Reasoning Domain Theory %%%
%%% Spatial Representation of the floors
% Floor 2 -- rooms
    room(rm218),    room(rm208),
    room(rm202),    room(rm201),
    room(copy2A),   room(kitchen2A),
    room(library2A),
% Floor 1 -- rooms
    room(rm132),    room(rm104),
    room(rm134),
    room(c1A1entrance), %the entrance to the corridor 1 of 1A.
    room(c1A2entrance), %the entrance to the corridor 2 of 1A.
% Floor 2 -- corridors
    corridor(c2A1),    corridor(c2A2),
    corridor(c2Alink1),corridor(c2Alink2),
```

```
    corridor(c2Aelev), corridor(c2Aksl),
% Floor 1 -- corridors
    corridor(c1A1),    corridor(c1A2),
    corridor(c1Alink1),corridor(c1Alink2),
    corridor(c1Aelev), corridor(c1Arobotics),
    corridor(c1Alink31),corridor(c1Alink32),
% Floor 2 -- rooms in corridors
    inCorridor(front(rm218), c2A1),
    inCorridor(front(rm208), c2A1),
    inCorridor(front(rm208), c2Alink1),
    inCorridor(front(rm202), c2Alink1),
    inCorridor(front(rm201), c2Alink1),
    inCorridor(front(rm201), c2A2),
    inCorridor(front(kitchen2A), c2A2),
    inCorridor(front(copy2A), c2A1),
    inCorridor(front(library2A), c2A1),
    inCorridor(front(library2A), c2Aelev),
% Floor 1 -- rooms in corridors
    inCorridor(front(rm104), c1Aelev),
    inCorridor(front(rm132), c1Alink31),
    inCorridor(front(rm132), c1A1),
    inCorridor(front(rm134), c1Alink31),
    inCorridor(front(kitchen1A), c1A2),
    inCorridor(front(copy1A), c1A2),

%%% Spatial Representation of intermediate positions. %%%
%% This is important for planning what to do when the
%% robot is in an intermediate position between landmarks.

    (vConnected(between(Pos1,Pos2),Pos1) :-
        vConnected(Pos1,Pos2)),
    (vConnected(between(Pos1,Pos2),Pos2) :-
        vConnected(Pos1,Pos2)),

    (current_landmark(between(Pos1,Pos2)) :-
        vConnected(Pos1,Pos2),
        curr_loc(X,Y),
        cartesian(Pos1,C1), cartesian(Pos2,C2), C1\=C2,
        pos_between(C1, [X,Y], C2)),

    max_dist_from_line(150),
    (not_max_dist_from_line(Dist1):- Dist1\=150),

    (pos_between([Xa,Ya], [X,Y], [Xb,Yb]) :-
        max_dist_from_line(Dist),
        Dist > (abs( (Ya-Yb)*X + (Xb-Xa)*Y +Yb*Xa -Ya*Xb )
                  / sqrt( (Ya-Yb)**2 + (Xb-Xa)**2 )),
        (Xa < Xb; (Xa >= X, X >= Xb)),
        (Xa > Xb; (Xa =< X, X =< Xb)),
        (Ya < Yb; (Ya >= Y, Y >= Yb)),
        (Ya > Yb; (Ya =< Y, Y =< Yb))),


% The last formula is the result of computing the line
% formula for [Xa,Ya],[Xb,Yb] and then computing the distance
% of the point [X,Y] to this line. The line formula is:
%   Ax+By+C=0  for   A=Ya-Yb   B=Xb-Xa   C = Yb*Xa - Ya*Xb
%     The distance between a point [X,Y] and a line is
%     |A*X + B*Y + C| / sqrt( A^2 + B^2 )
% We make sure that [X,Y] is indeed in the range between
% the two other points, i.e., that it is in a box defined
% by the two endpoints.

    intersect(c2A2,c2Aelev), % the corridors intersect

%%% Spatial domain for using the elevator

    at_static(rm218,floor(2)),   at_static(rm132,floor(1)),
```

```
    at_static(c1Aelev,floor(1)), at_static(c2Aelev,floor(2)),

    (not_at_static(X,L) ; at(X,L,S)),
% constant "at" is universal for all situations
    (not_at(L1,L2,S); not_at(L2,L3,S); at(L1,L3,S)),
                                        % Transitivity
% Entrance to 1A wing.
    doorTo(c1A1entrance,c1Aelev),  doorTo(c1A1entrance,c1A1),
    doorTo(c1A2entrance,c1Aelev),  doorTo(c1A2entrance,c1A2),
    (not_doorTo(L,C) ; inCorridor(door(L,C),C)),

% Notice: layer 3 has an abstract "elevator of floor X",
% whereas layer 2 considers two elevators that "implement"
% this abstract elevator.
    room(elev(floor(1))),
    room(elev(floor(2))),
% linking elevator entrances and corridors
    inCorridor(front(elev(floor(1))),c1Aelev),
    inCorridor(front(elev(floor(2))),c2Aelev),
    vLinked(elev(floor(1)),elev(floor(2))),

% Axioms about visual links between places.
    (not_doorTo(L,C); vLinked(door(L,C),C)), % doors to rooms
    (not_room(L); vLinked(L,front(L))), % rooms and room fronts

    (not_corridor(C) ; not_inCorridor(L1,C) ;
        not_inCorridor(L2,C) ; vConnected(L1,L2)),
    (not_vLinked(L1,L2) ; vConnected(L1,L2)),

    (not_vLinked(L1,L2) ; vLinked(L2,L1)),
    (not_vConnected(L1,L2) ; vConnected(L2,L1)),

    (not_intersect(C1,C2); inCorridor(intersection(C1,C2),C1)),
    (not_intersect(C1,C2); inCorridor(intersection(C1,C2),C2)),
              % corridor intersections

%%% Situation-dependent axioms: Theory of Action
    (not_current_landmark(CurrPlace1); at(r, CurrPlace1, s0)),
    at(eyal,rm218,s0),
    at(pedrito,rm132,s0),

%%% Effect Axioms

    % Simple move
    (not_at(r,L0,S) ; not_vConnected(L0,L) ;
          at(r,L,result(moveto(L),S))),

%%% Domain Constraints
% Notice the use of equality which is in fact equality + UNA:
    (not_at(X,L1,S) ; not_at(X,L2,S) ; not_roomLevelPlace(L1);
        not_roomLevelPlace(L2) ; eq(L1,L2)),
    (not_at(X,L1,S) ; not_at(X,L2,S) ; not_floor(L1) ;
        not_floor(L2) ; eq(L1,L2)),

% Defining what is a place that is a room.
    (roomLevelPlace(L1):- (room(L1);corridor(L1);
        (eq(L1,front(L)), room(L)))),
    (roomLevelPlace(L1) ; (not_room(L1), not_corridor(L1),
            (not_eq(L1,front(L)) ; not_room(L)))),

%%% GOALS %%%
%%% *** -> perhaps coming from the level above:
%        goal(sg), at(r,floor(1),sg),
    (atgoal(Someone, Somesituation):-
        goal_location(GoalLocation),
        at(Someone, GoalLocation, Somesituation)),

%%% Plan management
```

```
    (firstSit(result(A,S),S1):- firstSit(S,S1)),
    firstSit(result(A,s0),result(A,s0)),
    action(result(A,S),A),

%%% Equality
    (eq(L1,L2):-nonvar(L1),nonvar(L2),L1=L2),
    (not_eq(L1,L2):-L1\=L2)
)).
%%%%%%%%%%%%%% END OF LOGICAL THEORY %%%%%%%%%%%%%%%%%%%%%%%

% instead the following could be '(goal_location(rm208)))'.
subsuming_thy(no_subs_thy).

is_theory((Th3,ThS,ThSub)) :- is_theory3(Th3),
    is_sensor_theory(ThS), subsuming_thy(ThSub).

lay3load :- is_theory(Th), pttp(Th).

add_proof_landmark(Lnd) :-
    latch_clauses(Cl), pttp_latch((Cl,at(r,Lnd,s0))),!.
add_proof_landmark(_).

before_proof:-
    print('starting to prove the robot location'),
    prove(at(r,CurrLandmark,s0),12),
    print('proved the robot is in '), print(CurrLandmark),
    ((CurrLandmark = between(P1,P2),
        add_proof_landmark(CurrLandmark)) ;  true), !.

% The following is the PROLOG goal called by Layer 2's
% executable at every cycle ('goal_layer3(X)'):
goal_layer3(target_landmark(TargetLandmark)) :-
    before_proof,
    print('starting to prove the plan for the robot'),
    prove(atgoal(r,S),20),
    print('proved the plan is '), print(S),
    print('starting to find the target landmark'),
    ((S\=s0, prove(firstSit(S,S1),50),
             prove(at(r,TargetLandmark,S1),20));
    % true ->
             prove(at(r,TargetLandmark,s0),20)),
    print('found the landmark '), print(TargetLandmark),
    save_result(go_to(TargetLandmark)).
goal_layer3(target_landmark(Target)) :-
    print('failed to prove.'), go_to(Target).
goal_layer3(failed_proof_layer3).

save_result(Pred) :-
    retractall(Pred), assert(Pred).

% Test code for Layer 3:
out3:- lay3, prove(at(r,rm104,S)), prove(firstSit(S,S1)),
    prove(action(S1,A)),
    prove(at(r,Landmark,S1)),
    print('Plan = "'), print(S), print('"'), nl,
    print('Landmark = "'), print(Landmark),
    print('" using the action "'), print(A), print('"').
```

A sample proof with this theory is the following. We begin by listing the axioms as read into the PTTP theorem prover (we omit most of the axioms that are not used in the sample proof). PTTP numbers the axioms with consecutive numbers, and uses this numbering to present the proof. [8]

[8] The Prolog output here and in the rest of the appendix was set to fit the page width, and

```
PTTP input formulas:
  1  room(rm218).
  ...
 45  current_landmark(between(_G483, _G486)):-
         vConnected(_G483, _G486), curr_loc(_G526, _G527),
         cartesian(_G483, _G533), cartesian(_G486, _G539),
         _G533\=_G539,pos_between(_G533,[_G526,_G527],_G539).
 ...
 71  not_vConnected(_G787, _G788);vConnected(_G788, _G787).
 72  not_intersect(_G533, _G539);
         inCorridor(intersection(_G533, _G539), _G533).
 73  not_intersect(_G533, _G539);
         inCorridor(intersection(_G533, _G539), _G539).
 74  not_current_landmark(_G1020);at(r, _G1020, s0).
 ...
 77  not_at(r, _G1047, _G779);not_vConnected(_G1047, _G775);
         at(r, _G775, result(moveto(_G775), _G779)).
 78  not_at(_G526, _G787, _G779);not_at(_G526, _G788, _G779);
         not_roomLevelPlace(_G787);not_roomLevelPlace(_G788);
         eq(_G787, _G788).
 79  not_at(_G526, _G787, _G779);not_at(_G526, _G788, _G779);
         not_floor(_G787);not_floor(_G788);eq(_G787, _G788).
 80  roomLevelPlace(_G787):-room(_G787);corridor(_G787);
         eq(_G787, front(_G775)), room(_G775).
 81  roomLevelPlace(_G787);
         not_room(_G787),not_corridor(_G787),
              (not_eq(_G787, front(_G775)); not_room(_G775)).
 82  atgoal(_G1187, _G1188):-goal_location(_G1193),
         at(_G1187, _G1193, _G1188).
 83  firstSit(result(_G1208, _G779), _G1206):-
         firstSit(_G779, _G1206).
 84  firstSit(result(_G1208, s0), result(_G1208, s0)).
 85  action(result(_G1208, _G779), _G1208).
 ...
 96  cartesian(zero_pt, [0, 0]).
 ...
112  cartesian(mid_corridor2, [4817, -704]).
113  vConnected(corridor_cross, zero_pt).
114  vConnected(mid_lab, corridor_cross).
...
118  vConnected(mid_lab, corridor2_cross).
...
132  not_curr_loc(_G1794, _G1795);
         not_cartesian(_G1800, _G1801);
         not_short_distance([_G1794, _G1795], _G1801);
         current_landmark(_G1800).
133  short_distance([_G1826, _G1829], [_G1832, _G1835]):-
         distance_threshold(_G1841),abs(_G1826-_G1832)<_G1841,
         abs(_G1829-_G1835)<_G1841.
134  distance_threshold(100).
135  not_current_landmark(_G1870);at(r, _G1870, s0).
136  no_subs_thy.

PTTP to Prolog translation time: 1.04 seconds,
including printing
Prolog compilation time: 0.28 seconds, including printing
Start cycle 1
.....
PTTP input formulas:
137  sonar_reading(0, 219).
 ...
152  sonar_reading(15, 142).
153  curr_loc(0, 0).
154  curr_dir(3419).
```

---

also shortened by removing duplicate and unnecessary items such as proof-search progress indicators. We put "..." where such parts are omitted.

```
155  offset(0, 0, 0).
156  goal_location(corridor2_cross).

PTTP to Prolog translation into latch time: 0.03 seconds,
 including printing
Assserting into Prolog time: 0.02 seconds, including printing
Start cycle 3
'starting to prove the robot location'
Proof time: 73 inferences in 0.01 seconds, including printing
Proof:
Goal#  Wff#  Wff Instance
-----  ----  ------------
  [0]    0   query :- [1].
  [1]   74      at(r, zero_pt, s0) :- [2].
  [2]  132         current_landmark(zero_pt) :- [3],[4],[5].
  [3]  153            curr_loc(0, 0).
  [4]   96            cartesian(zero_pt, [0, 0]).
  [5]  133            short_distance([0, 0], [0, 0]) :- [6].
  [6]  134               distance_threshold(100).
'proved the robot is in 'zero_pt'
starting to prove the plan for the robot'
Proof time: 175491 inferences in 16.25 seconds,
 including printing
Proof:
Goal#  Wff#  Wff Instance
-----  ----  ------------
  [0]    0   query :- [1].
  [1]   82      atgoal(r, result(moveto(corridor2_cross), result(moveto(mid_lab),
                      result(moveto(corridor_cross), s0)))) :- [2] , [3].
  [2]  156         goal_location(corridor2_cross).
  [3]   77         at(r, corridor2_cross, result(moveto(corridor2_cross),
                         result(moveto(mid_lab), result(moveto(corridor_cross),
                         s0)))) :- [4] , [16].
  [4]   77            at(r, mid_lab, result(moveto(mid_lab),
                            result(moveto(corridor_cross), s0))) :- [5] , [14].
  [5]   77               at(r, corridor_cross, result(moveto(corridor_cross),
                                  s0)) :- [6] , [12].
  [6]   74                  at(r, zero_pt, s0) :- [7].
  [7]  132                     current_landmark(zero_pt):- [8],[9],[10].
  [8]  153                        curr_loc(0, 0).
  [9]   96                        cartesian(zero_pt, [0,0]).
 [10]  133                        short_distance([0,0], [0,0]):- [11].
 [11]  134                           distance_threshold(100).
 [12]   71                  vConnected(zero_pt, corridor_cross) :- [13].
 [13]  113                     vConnected(corridor_cross, zero_pt).
 [14]   71               vConnected(corridor_cross, mid_lab)  :- [15].
 [15]  114                  vConnected(mid_lab, corridor_cross).
 [16]  118            vConnected(mid_lab, corridor2_cross).
'proved the plan is 'result(moveto(corridor2_cross), result(moveto(mid_lab),
    result(moveto(corridor_cross), s0)))'

 starting to find the target landmark'
Proof time: 4 inferences in 0.01 seconds, including printing
Proof:
Goal#  Wff#  Wff Instance
-----  ----  ------------
  [0]    0   query :- [1].
  [1]   83      firstSit(result(moveto(corridor2_cross), result(moveto(mid_lab),
                      result(moveto(corridor_cross),s0))),
                      result(moveto(corridor_cross),s0)) :- [2].
  [2]   83         firstSit(result(moveto(mid_lab),
                         result(moveto(corridor_cross),s0)),
                         result(moveto(corridor_cross),s0)) :- [3].
  [3]   84            firstSit(result(moveto(corridor_cross),s0),
                            result(moveto(corridor_cross),s0)).

Proof time: 418 inferences in 0.03 seconds, including printing
Proof:
```

```
Goal#  Wff#  Wff Instance
-----  ----  ------------
  [0]    0   query :- [1].
  [1]   77      at(r, corridor_cross, result(moveto(corridor_cross), s0)) :-
                      [2], [8].
  [2]   74         at(r, zero_pt, s0) :- [3].
  [3]  132            current_landmark(zero_pt):-[4],[5],[6].
  [4]  153               curr_loc(0, 0).
  [5]   96               cartesian(zero_pt, [0, 0]).
  [6]  133               short_distance([0,0], [0,0]) :- [7].
  [7]  134                  distance_threshold(100).
  [8]   71         vConnected(zero_pt, corridor_cross):- [9].
  [9]  113            vConnected(corridor_cross, zero_pt).
'found the landmark 'corridor_cross
```

B.3 Layer 2 Control Theory (L2)

The following file contains the theory of the LSA for Layer 2. This theory together with the high-level sensory theory (M2 in Section B.1) translates logical locations into Cartesian positions and vice-versa, and performs mid-level action planning, such as reasoning about using the elevator.

Our implementation of Layer 2 can be seen as taking a sentence describing a goal situation from Layer 3 and producing a Cartesian position which is sent to Layer 1 and serves there as a target location.

It includes the following non-logical symbols:

Predicates • PTTP-internal predicates ($\backslash=$, $=$, etc.).

- $move\_cmd(\langle X\rangle, \langle Y\rangle)$
- $target\_landmark(\langle Logical\_position\rangle)$
- $elevator\_related(\langle Logical\_position\rangle)$ – the location $Logical\_position$ is related to running the elevators
- $cartesian(\langle Logical\_position\rangle, \langle [X, Y]\rangle)$
- $current\_landmark(\langle Landmark\rangle)$
- $elevator(\langle Logical\_position\rangle)$
- $at(\langle Agent\rangle, \langle Logical\_position\rangle, \langle Situation\rangle)$
- $firstSit(\langle S\rangle, \langle S1\rangle)$ – the situation ($S1$) that is immediately after $s0$ in the definition of situation S
- $action(\langle S\rangle, \langle A\rangle)$ – the last action A in a sequence defining a situation S
- $vLinked(\langle Logical\_position1\rangle, \langle Logical\_position2\rangle, \langle S\rangle)$ – locations that are sometimes visually connected and sometimes not (e.g., elevators)

Functions • PTTP-internal functions ($-$, abs, etc.)

- $floor(\langle Number\rangle)$ – takes a number and returns a logical location (the floor indexed by this number)
- $elev(\langle Floor\rangle)$ – the logical location of the elevators at floor $Floor$ (whether an elevator is on that floor or not)
- $front(\langle Logical\_position\rangle)$

- $result(\langle A\rangle, \langle S\rangle)$ – the situation resulting from executing action A in situation S
- $elevEntr(\langle E\rangle, \langle Floor\rangle)$ – the logical location of the elevator entrance on a given floor (E may be any of the elevators)
- $moveto(L)$, $orderElev(Floor)$ – action schemas

Constant Symbols • Numerical constants

- r – robot
- $elev1$, $elev2$ – elevators
- $callElev$, $wait$ – actions

```
is_theory2((

%%% Lower-Level Spatial Reasoning Domain Theory %%%
% Currently, we can execute only motion commands, so we
% distinguish those from other commands (such as ordering
% the elevator), and only the former get executed (get
% sent to layer 1).
    (move_cmd(X,Y):- target_landmark(Logical),
          not_elevator_related(Logical),
          cartesian(Logical, [X,Y])),

    elevator_related(elev(floor(Floor))),
    (elevator_related(front(elev(floor(Floor2)))) :-
          current_landmark(elev(floor(F3)))),
    (elevator_related(Logical) :- elevator(Logical)),

    (not_elevator_related(Logical):-
          Logical \= elev(floor(Floor)),
          (Logical \= front(elev(floor(Floor2)));
              not_current_landmark(elev(floor(F3)))),
          not_elevator(Logical)),

% Notice that layer 3 has an abstract "elevator of floor
% X", whereas layer 2 considers two elevators that
% "implement" this abstract elevator.

    (move_cmd(X,Y):- target_landmark(Landmark),
          at(r,Landmark,Sl1), firstSit(Sl1,Sfirst),
          at(r,Inter,Sfirst), cartesian(Inter, [X,Y])),

%%% Plan management

    (firstSit(result(Af1,Sf1),Sf2):- firstSit(Sf1,Sf2)),
    firstSit(result(Af2,s0),result(Af2,s0)),
    action(result(Af3,Sf3),Af3),

%%%%  Elevator specific axioms %%%%

    (not_target_landmark(elev(floor(F5)));
     (target_landmark(elev1), target_landmark(elev2))),
% translate from layer 3's elevator terminology
% to layer 2's more specific terminology

    elevator(elev1),
    elevator(elev2),
    (not_elevator(Elev):- Elev\=elev1, Elev\=elev2),
    (not_elevator(E); at(elevEntr(E,floor(X)),floor(X),S2)),

    vLinked(elevEntr(E,floor(Floor3)),
            front(elev(floor(Floor3))),S3),
% linking elevator entrances and elevator fronts.

% Axioms about (dynamic) visual links between places.
```

```
    (not_elevator(E) ; not_at(E,floor(F),S2) ;
         vLinked(E,elevEntr(E,floor(F)),S2)),
% linking elevators and elevator entrances
    (not_vLinked(L1,L2,S1) ; vLinked(L2,L1,S1)),

%%%% Effect Axioms %%%%

% Simple move:
% Moving to a location that is visually linked to the
% robot's current location results in the robot being
% at the target location.
    (not_at(r,L0,S8) ; not_vLinked(L,L0,S8) ;
         at(r,L,result(moveto(L),S8))),

% Calling the elevator:
% The robot stays put at the result of calling the
% elevator and after some time an elevator must come.
    (not_at(r,front(elev(floor(F1))),S4) ;
      (at(r,front(elev(floor(F1))),result(callElev,S4)),
       (at(elev1,floor(F1), result(wait,result(callElev,S4)));
        at(elev2,floor(F1), result(wait,result(callElev,S4))))))),

% Waiting: the robot does not move (but elevators might).
    (not_at(r,X,S); at(r,X,result(wait,S))),

% Command the elevator:
% If the robot is in the elevator, then ordering the
% elevator to move to a different floor results in the
% elevator moving to that floor (after some waiting period)
% and the robot is still at that elevator.
    (not_at(r,E,S5) ; not_elev(E);
       (at(r,E,result(orderElev(floor(F2)),S5)),
        at(E,floor(F2),
           result(wait,result(orderElev(floor(F2)),S5))))))

%%%% Frame Axioms that were omitted in the running system:
%
%      (not_at(elev1,floor(X),S) ;
%             at(elev1,floor(X),result(moveto(L),S))),
%         % The elevator does not move by itself.
%      (at(r,L,result(callElev,S)); not_at(r,L,S)),
%      (not_at(r,L,S) ; at(r,L,result(orderElev(F),S))),
%         % The elevator does not move by itself.
%      (eq(L1,L2):-nonvar(L1),nonvar(L2),L1=L2),
%      (not_eq(L1,L2):-L1\=L2)
%         % Equality
%
%%%% These frame axioms make the theorem prover go balistic
%%%% with respect to time spent proving. All of the sudden,
%%%% it takes it much longer than previously. They are not
%%%% needed for our purposes, so we omitted them.
)).
%%%%%%%%%%%%%%%%%% END OF LOGICAL THEORY %%%%%%%%%%%%%%%%%%

% Put control theory (L2) and sensory theory (M2) together:
is_theory((Th2,ThS)):-is_theory2(Th2),is_sensor_theory(ThS).

% Loading a theory into PTTP
lay2load :- is_theory(Th), pttp(Th).

% The following is the goal that is called by Layer 2's
% executable at every cycle ('goal_layer2(X)') (Right
% now subsumption in this layer is done by negation as
% failure (the parameter "15" below indicates steps limit)):
goal_layer2(destination(GoalX,GoalY)) :-
        prove((move_cmd(GoalX,GoalY)), 15),
        print('proof succeeded. move to coordinates ('),
        print(GoalX), print(','), print(GoalY), print(')').
```

```
goal_layer2(failed_proof_layer2):-print('failed proof. ').
```

This layer proves at every cycle that the current destination of the robot is specified by some Cartesian coordinates $GoalX, GoalY$. When Layer 3 introduces a ground sentence of the form $target\_landmark(\langle Logical\_position \rangle)$ and $Logical\_sentence$ is instantiated with some landmark term, then subsumption is used to remove previous sentences of that form (sent from Layer 3), and it is also used to conclude that the new set of target landmarks includes only the given position (and possibly others that the layer may have as fixed target landmarks [9]).

A sample proof with this theory is the following. Again, we begin by listing the axioms as read into the PTTP theorem prover and omit most of the axioms that are not used in the sample proof. PTTP numbers the axioms with consecutive numbers, and uses this numbering to present the proof. We put "..." where parts are omitted.

```
 1  move_cmd(_G221, _G222):-target_landmark(_G227),
         not_elevator_related(_G227),
         cartesian(_G227, [_G221, _G222]).
...
 5  not_elevator_related(_G227):-_G227\=elev(floor(_G250)),
         (_G227\=front(elev(floor(_G264)));
                not_current_landmark(elev(floor(_G270)))),
         not_elevator(_G227).
...
11  elevator(elev1).
12  elevator(elev2).
13  not_elevator(_G437):-_G437\=elev1, _G437\=elev2.
...
18  not_at(r,_G532,_G533); not_vLinked(_G538,_G532,_G533);
         at(r, _G538, result(moveto(_G538), _G533)).
19  not_at(r, front(elev(floor(_G565))), _G559);
      at(r, front(elev(floor(_G565))), result(callElev, _G559)),
      (at(elev1, floor(_G565), result(wait, result(callElev, _G559)));
       at(elev2, floor(_G565), result(wait, result(callElev, _G559)))).
20  not_at(r,_G221,_G618);at(r,_G221, result(wait,_G618)).
21  not_at(r,_G454,_G632);not_elev(_G454);
      at(r, _G454, result(orderElev(floor(_G651)),_G632)),
      at(_G454, floor(_G651),
       result(wait, result(orderElev(floor(_G651)),_G632))).
...
31  cartesian(corridor_cross, [805, -300]).
...
68  distance_threshold(100).
69  not_current_landmark(_G1273);at(r, _G1273, s0).

PTTP to Prolog translation time: 0.56 seconds,
 including printing
Prolog compilation time: 0.18 seconds, including printing
Start cycle 1
 ...
PTTP input formulas:
 70  sonar_reading(0, 140).
 ...
 85  sonar_reading(15, 77).
 86  curr_loc(38, -103).
 87  curr_dir(2962).
```

[9] In case more than one target landmark is specified, the engineer providing the axioms is expected to supply some precedence between them, or the theorem prover may find a proof that one of the landmarks is achieved.

```
 88  offset(0, 0, 0).
 89  target_landmark(corridor_cross).
 90  goal_location(corridor2_cross).

PTTP to Prolog translation into latch time: 0.02 seconds,
 including printing
Assserting into Prolog time: 0.04 seconds,
 including printing
Start cycle 994
Proof time: 5 inferences in 0 seconds, including printing
Proof:
Goal#  Wff#  Wff Instance
-----  ----  ------------
  [0]    0   query :- [1].
  [1]    1      move_cmd(805, -300) :- [2] , [3] , [5].
  [2]   89         target_landmark(corridor_cross).
  [3]    5         not_elevator_related(corridor_cross):-[4].
  [4]   13            not_elevator(corridor_cross).
  [5]   31         cartesian(corridor_cross, [805, -300]).
'proof succeeded. move to coordinates ('805,-300')'
```

B.4 Low-Level Sensor Module (M1)

This theory represents the sensor module used by layers 1, 0, and -1. It uses and makes available values for the following predicates:

- $pi(3.14159)$ – the number π to five decimal digits of precision; ideally, should be a constant.
- $sonar\_reading(0..15, \langle Distance \rangle)$, $sonar\_reading\_internal(0..15, \langle Distance \rangle)$ – $sonar\_reading$ returns the distance from the robot of the object observed by the given sensor. Ideally, should be a function. The axioms in our layers refer only to $sonar\_reading\_internal$ and not to $sonar\_reading$ because this way $sonar\_reading$ can be asserted (into Prolog and PTTP) without recompilation of the rest of the axioms; M1 simply translates $sonar\_reading$ assertions into the equivalent $sonar\_reading\_internal$ predicates. By the same token, we also define "$\_internal$" predicates for $curr\_loc$, $offset$, and $curr\_dir$ described below. For similar reasons, we define "$\_external$" predicates for $destination$, $object$, $distance$, $direction$, go_fwd, and go_turn predicates described below to facilitate passing them between layers.
- $curr\_loc(\langle X \rangle, \langle Y \rangle)$, $curr\_loc\_internal(\langle X \rangle, \langle Y \rangle)$ – the robot's location with respect to the robot's coordinate system; ideally, should be a constant.
- $curr\_dir(\langle Angle \rangle)$, $curr\_dir\_internal(\langle Angle \rangle)$ – the robot's orientation with respect to the robot's coordinate system; ideally, should be a constant.
- $offset(\langle XOff \rangle, \langle YOff \rangle, \langle AngOff \rangle)$, $offset\_internal(\langle XOff \rangle, \langle YOff \rangle, \langle AngOff \rangle)$ – the transformation parameters from the global coordinate system to the robot's.
- $angle\_deg\_rad(\langle DegA \rangle, \langle RadA \rangle)$ – converts angles in degrees to radians and back (required for some of the semantic attachments that we use).
- $destination(\langle X \rangle, \langle Y \rangle)$, $external\_destination(\langle X \rangle, \langle Y \rangle)$ – the goal location input by Layer 2 into Layer 1; M1 translates the $external\_destination$ input from Layer 2 into the $destination$ used by Layer 1.

- $object(\langle New\_Obj \rangle)$, $external\_object(\langle New\_Obj \rangle)$ – a new object input by Layer 1 into Layer 0; M1 translates the $external\_object$ input from Layer 1 into the $object$ used by Layer 0.
- $distance(\langle New\_Obj \rangle, \langle NO\_Dist \rangle)$, $external\_distance(\langle New\_Obj \rangle, \langle NO\_Dist \rangle)$ – distance of new object from robot; M1 translates the $external\_distance$ input from Layer 1 into the $distance$ used by Layer 0.
- $direction(\langle New\_Obj \rangle, \langle NO\_Dir \rangle)$, $external\_direction(\langle New\_Obj \rangle, \langle NO\_Dir \rangle)$ – direction of new object from robot; M1 translates the $external\_direction$ input from Layer 1 into the $direction$ used by Layer 0.
- $go\_fwd(\langle Speed \rangle)$, $external\_fwd(\langle Speed \rangle)$ – speed at which robot should move forward; ideally, should be a constant. M1 translates the $external\_fwd$ input from Layer 0 into the go_fwd used by Layer -1.
- $go\_turn(\langle Angle \rangle)$, $external\_turn(\langle Angle \rangle)$ – angle robot should turn; ideally, should be a constant. M1 translates the $external\_turn$ input from Layer 0 into the go_turn used by Layer -1.

Note that the robot's coordinate system does *not* move with the robot. Distances are in 1/10-in, angles are in radians in the range $[-\pi, \pi]$. (Note: input angles are in 1/10-deg in the range $[0, 3600]$ and, thus, need to be converted.)

```
is_sensor_theory((

%%% Parameters %%%

% Note that, for practicality, all of the constants and functions
% described in this module are represented as predicates.

    pi(3.14159), (not_pi(C0) :- C0=\=3.14159),


%%% Sensor Input %%%

    % We add *_internal so that it is easy to assert and retract
    % sensory input without compilation (saves time for sensory
    % assertion).
    (sonar_reading_internal(Snum, DistSonar):-
      sonar_reading(Snum, DistSonar)),

    (curr_loc_internal(Xinternal, Yinternal):- % subtract offset
      curr_loc(Xhere, Yhere),
      offset_internal(XOff6, YOff6, AngOff6),
      Xinternal is Xhere - XOff6,
      Yinternal is Yhere - YOff6),

    (curr_dir_internal(Ainternal):-  % convert to rad & subtract offset
      curr_dir(Ang),
      offset_internal(XOff6, YOff6, AngOff6),
      angle_deg_rad(Ang - AngOff6, Ainternal)),

    % Translate angles from [0,3600] to [-PI,PI].
    (angle_deg_rad(DegA, RadA):-
      pi(PI),
      var(RadA),
      RadA is ((((integer(DegA)+1800) mod 3600)-1800)/3600)*(2*PI)),

    % Translate angles from [-PI,PI] to [-1800,1800].
    (angle_deg_rad(DegA, RadA):-
      pi(PI),
```

```
      var(DegA),
      DegA is integer((RadA/(2*PI))*3600)),

    (offset_internal(XOff,YOff,AngOff):-
      offset(XOff,YOff,AngOff)),


    % The following is needed to incorporate L2's inputs into L1.
    (destination(DestX,DestY):- external_destination(DestX,DestY)),

    % The following are needed to incorporate L1's inputs into L0.
    (object(New_Obj):- external_object(New_Obj)),
    (distance(New_Obj, NO_Dist):- external_distance(New_Obj, NO_Dist)),
    (direction(New_Obj, NO_Dir):- external_direction(New_Obj, NO_Dir)),

    % The following are needed to incorporate L0's inputs into L-1.
    (go_fwd(Sp):- external_fwd(Sp)),
    (go_turn(An):- external_turn(An))

)).


% Load theory into PTTP.
sense :- is_sensor_theory(ThS), pttp(ThS).

% Test the sensor module in isolation.
testS :- sense, prove((curr_loc(X,Y))), print('current location = ('),
     print(X), print(','), print(Y), print(')'), nl.
```

B.5 Layer 1 Control Theory (L1)

This file contains the theory of the LSA for Layer 1. The theory implements destination seeking. It receives a position from Layer 2 in Cartesian coordinates and greedily tries to move the robot in that direction. It does so by taking advantage of Layer 0's obstacle-avoidance approach; it passes to Layer 0 a "virtual pushing object" whose "location" is close to the robot in the quadrant opposite the goal location so that Layer 0, in attempting to avoid obstacles, will tend to move the robot away from the pushing object and in the desired direction. Layer 1 uses the robot's current location and orientation provided by the low-level sensory theory (M1 in Section B.4) to compute the specific quadrant for this pushing object.

The theory includes the following predicate symbols:

- PTTP-internal predicates (le, $= \backslash =$, etc.)
- pi, $object$, $distance$, and $direction$ are as described for the low-level sensor module (M1).
- $curr\_loc\_internal$ and $curr\_dir\_internal$ are input from M1 and are as described there.
- $destination(\langle X \rangle, \langle Y \rangle)$ – $external\_destination(\langle X \rangle, \langle Y \rangle)$ input from Layer 2, translated by the sensor layer to $destination(\langle X \rangle, \langle Y \rangle)$; $[X, Y]$ are the coordinates of the goal destination.
- $margin(50)$ – maximum distance (along both axes) of robot from destination

for us to consider the robot to be at the destination; ideally, should be a constant.
- $marginal\_distance(\langle X1\rangle, \langle Y1\rangle, \langle X2\rangle, \langle Y2\rangle)$ – true iff $[X1, Y1]$ is within the margin of $[X2, Y2]$.
- $nquads(8)$ – number of quadrants space around robot is divided into; ideally, should be a constant.
- $quadrant(\langle X\rangle, \langle Y\rangle, \langle Quad\rangle)$ – determines the quadrant in which $[X, Y]$ falls; ideally, should be a function.
- $quad\_angle(\langle Quad\rangle, \langle Ang\rangle)$ – determines the angle of the specified quadrant relative to the robot; ideally, should be a function.
- $angle\_world\_vs\_robot(\langle WorldAng\rangle, \langle LocalAng\rangle)$ – converts an angle in the global coordinate system into the robot's local coordinate system; ideally, should be a function.
- $between\_minus\_and\_plus(PI, \langle Ang1\rangle, \langle Ang2\rangle)$ – $Ang2$ is the renormalization of $Ang1$ to be between $-PI$ and PI where PI is π.
- $push\_object(\langle PUSH\_OBJECT\rangle)$
- $push\_object\_dist(20)$ – default distance for the pushing object; ideally, should be a constant.
- $has\_push\_object(\langle Quad\rangle)$

Function symbols used include the PTTP-internal functions ($-$, abs, etc.). Constant symbols used include numerical constants and z, the virtual pushing object.

The theory follows:

```
is_theory1((

%%%% INPUT %%%

% constants: curr_loc_internal, curr_dir_internal, destination

% Note that, for practicality, these and all other constants and
% functions described in this layer are represented as predicates.


%%% PARAMETERS %%%

    pi(3.14159), (not_pi(C0) :- C0=\=3.14159),
    margin(50), (not_margin(MARG) :- MARG=\=50),
                                          % Margin to destination is 50

    push_object(z),

    (not_push_object(PUSH_OBJECT);
     not_curr_loc_internal(Xhere, Yhere);
     not_destination(Xthere, Ythere);
     marginal_distance(Xhere, Yhere, Xthere, Ythere);
     object(PUSH_OBJECT)),
    % We consider adding an object only if we are too far from
    % the destination.

    (marginal_distance(X1,Y1,X2,Y2);
     not_margin(MARGIN);
     le(MARGIN,abs(X1-X2));
     le(MARGIN,abs(Y1-Y2))),

    (not_marginal_distance(X1,Y1,X2,Y2);
```

```
     not_margin(MARGIN);
     (ls(abs(X1-X2),MARGIN), ls(abs(Y1-Y2),MARGIN))),


    push_obj_dist(20),         % default push object distance of 20.
    (not_push_obj_dist(C2) :- C2=\=20),

    nquads(8), (not_nquads(C1) :- C1=\=8),

    (le(X0, Y0); ls(Y0, 0); quadrant(X0, Y0, 0)),
    (ls(Y1, X1); le(X1, 0); quadrant(X1, Y1, 1)),
    (le(Y2, 0); ls(0, X2); le(Y2, abs(X2)); quadrant(X2, Y2, 2)),
    (le(Y3, 0); le(0, X3); ls(abs(X3), Y3); quadrant(X3, Y3, 3)),
    (le(X4, 0); le(0, Y4); ls(X4, abs(Y4)); quadrant(X4, Y4, -1)),
    (ls(X5, 0); le(0, Y5); le(abs(Y5), X5); quadrant(X5, Y5, -2)),
    (le(0, X6); ls(X6, Y6); quadrant(X6, Y6, -3)),
    (ls(0, Y7); le(Y7, X7); quadrant(X7, Y7, -4)),


%%% Axioms %%%


% The following axioms create a virtual object to be placed
% PUSH_OBJ_DIST from the robot in the middle of the quadrant opposite
% the desired direction. Note that the direction function is wrt the
% robot's coordinate system.

    (not_curr_loc_internal(Xhere, Yhere);
     not_destination(Xthere, Ythere);
     not_quadrant(Xhere-Xthere, Yhere-Ythere, Quad0);
     has_push_object(Quad0)),

    (not_push_object(PUSH_OBJECT);
     not_push_obj_dist(PUSH_OBJ_DIST);
     not_has_push_object(Quad1);
     not_quad_angle(Quad1,WorldAng);
     not_angle_world_vs_robot(WorldAng, LocalAng);
     (distance(PUSH_OBJECT, PUSH_OBJ_DIST), direction(PUSH_OBJECT, LocalAng))),

    (quad_angle(Quad,LocalAng):-
     pi(PI), nquads(NQUADS), LocalAng is (Quad+0.5)*2*PI/NQUADS),

    (angle_world_vs_robot(WorldAng, LocalAng) :-
     curr_dir_internal(RobAng), pi(PI),
     between_minus_and_plus(PI, WorldAng-RobAng, LocalAng)),

    (between_minus_and_plus(PI,Ang1,Ang1) :- Ang1 =< PI, Ang1 >= -PI),
    (between_minus_and_plus(PI,Ang1,Ang1+(2*PI)) :- Ang1 < -PI),
    (between_minus_and_plus(PI,Ang1,Ang1-(2*PI)) :- Ang1 > PI)

)).
%%%%%%%%%%%%%%%%%% END OF LOGICAL THEORY %%%%%%%%%%%%%%%%%%%%%%%%

% Assemble control theory (L1) and sensor theory (M2) together:
is_theory((Th0,ThS)) :- is_theory1(Th0), is_sensor_theory(ThS).

% Load theory into PTTP.
lay1load :- is_theory(Th), pttp(Th).


% The following is the goal that is called by Layer 1's executable at
% every cycle ('goal_layer1(X)'). (Right now subsumption in this layer
% is done by negation as failure that's why the "20" steps limit is
% there).
goal_layer1([external_object(PUSH_OBJECT),
    external_distance(PUSH_OBJECT, Dist_po),
    external_direction(PUSH_OBJECT, Dir_calc)]) :-
    time_stamp_layer,
```

```
    prove((object(PUSH_OBJECT), push_object(PUSH_OBJECT)), 20),
    print_time_stamp,
    prove((distance(PUSH_OBJECT, Dist_po), direction(PUSH_OBJECT, Dir_po)), 50),
    Dir_calc is Dir_po,
    print_time_stamp,
    write('push object distance = '), write(Dist_po), nl,
    write('push object direction = '), write(Dir_calc).

goal_layer1(failed_proof_layer1):-print('failed proof. '), print_time_stamp.

% Negation-as-failure will produce "not_external_object(PUSH_OBJECT)"
% in the receiving layer, but we should not produce it ourselves. We
% only tell the receiving layer that we failed.
```

The following is a sample proof:

```
PTTP input formulas:
  1  pi(3.14159).
  2  not_pi(_G680):-_G680=\=3.14159.
  3  margin(50).
  4  not_margin(_G700):-_G700=\=50.
  5  push_object(z).
  6  not_push_object(_G716);not_curr_loc_internal(_G721, _G722);
        not_destination(_G727, _G728);marginal_distance(_G721, _G722, _G727,
        _G728);object(_G716).
  7  marginal_distance(_G746, _G747, _G748, _G749);not_margin(_G754);le(_G754,
        abs(_G746-_G748));le(_G754, abs(_G747-_G749)).
  8  not_marginal_distance(_G746, _G747, _G748, _G749);not_margin(_G754);
        ls(abs(_G746-_G748), _G754), ls(abs(_G747-_G749), _G754).
  9  push_obj_dist(20).
 10  not_push_obj_dist(_G821):-_G821=\=20.
 11  nquads(8).
 12  not_nquads(_G837):-_G837=\=8.
 13  le(_G848, _G849);ls(_G849, 0);quadrant(_G848, _G849, 0).
 14  ls(_G747, _G746);le(_G746, 0);quadrant(_G746, _G747, 1).
 15  le(_G749, 0);ls(0, _G748);le(_G749, abs(_G748));quadrant(_G748, _G749, 2).
 16  le(_G913, 0);le(0, _G920);ls(abs(_G920), _G913);quadrant(_G920, _G913, 3).
 17  le(_G940, 0);le(0, _G947);ls(_G940, abs(_G947));quadrant(_G940, _G947, -1).
 18  ls(_G967, 0);le(0, _G974);le(abs(_G974), _G967);quadrant(_G967, _G974, -2).
 19  le(0, _G995);ls(_G995, _G1001);quadrant(_G995, _G1001, -3).
 20  ls(0, _G1014);le(_G1014, _G1020);quadrant(_G1020, _G1014, -4).
 21  not_curr_loc_internal(_G721, _G722);not_destination(_G727, _G728);
        not_quadrant(_G721-_G727, _G722-_G728, _G1046);has_push_object(_G1046).
 22  not_push_object(_G716);not_push_obj_dist(_G1067);
        not_has_push_object(_G1072);not_quad_angle(_G1072, _G1078);
        not_angle_world_vs_robot(_G1078, _G1084);distance(_G716, _G1067),
        direction(_G716, _G1084).
 23  quad_angle(_G1101, _G1084):-pi(_G1107), nquads(_G1112), _G1084 is
        (_G1101+0.5)*2*_G1107/_G1112.
 24  angle_world_vs_robot(_G1078, _G1084):-curr_dir_internal(_G1145), pi(_G1107),
        between_minus_and_plus(_G1107, _G1078-_G1145, _G1084).
 25  between_minus_and_plus(_G1107, _G1166, _G1166):-_G1166=<_G1107,
        _G1166>= -_G1107.
 26  between_minus_and_plus(_G1107, _G1166, _G1166+2*_G1107):-_G1166< -_G1107.
 27  between_minus_and_plus(_G1107, _G1166, _G1166-2*_G1107):-_G1166>_G1107.
 28  pi(3.14159).
 29  not_pi(_G1232):-_G1232=\=3.14159.
 30  sonar_reading_internal(_G1247, _G1248):-sonar_reading(_G1247, _G1248).
 31  curr_loc_internal(_G1259, _G1260):-curr_loc(_G1265, _G1266),
        offset_internal(_G1271, _G1272, _G1273), _G1259 is _G1265-_G1271,
        _G1260 is _G1266-_G1272.
 32  curr_dir_internal(_G1296):-curr_dir(_G1301), offset_internal(_G1271, _G1272,
        _G1273), angle_deg_rad(_G1301-_G1273, _G1296).
 33  angle_deg_rad(_G1322, _G1323):-var(_G1323), pi(_G1333), _G1323 is
        ((integer(_G1322)+1800)mod 3600-1800)/3600* (2*_G1333).
 34  angle_deg_rad(_G1322, _G1323):-var(_G1322), pi(_G1333), _G1322 is
```

```
       integer(_G1323/ (2*_G1333)*3600).
 35  offset_internal(_G1397, _G1398, _G1399):-offset(_G1397, _G1398, _G1399).
 36  destination(_G1411, _G1412):-external_destination(_G1411, _G1412).
 37  object(_G1423):-external_object(_G1423).
 38  distance(_G1423, _G1434):-external_distance(_G1423, _G1434).
 39  direction(_G1423, _G1446):-external_direction(_G1423, _G1446).
 40  go_fwd(_G1457):-external_fwd(_G1457).
 41  go_turn(_G1464):-external_turn(_G1464).

PTTP to Prolog translation time: 0.25 seconds, including printing

Prolog compilation time: 0.1 seconds, including printing
Start cycle 1
 ...
Start cycle 1044

137 inferences so far.  'failed proof. '
PTTP input formulas:
 42  sonar_reading(0, 140).
 ...
 57  sonar_reading(15, 77).
 58  curr_loc(38, -103).
 59  curr_dir(2962).
 60  offset(0, 0, 0).
 61  destination(805, -300).
 62  goal_location(corridor2_cross).

PTTP to Prolog translation into latch time: 0.01 seconds, including printing

Assserting into Prolog time: 0.02 seconds, including printing
Start cycle 1045

Proof time: 31 inferences in 0 seconds, including printing
Proof:
Goal#  Wff#  Wff Instance
-----  ----  ------------
  [0]    0   query :- [1] , [10].
  [1]    6      object(z) :- [2] , [3] , [7] , [8].
  [2]    5         push_object(z).
  [3]   31         curr_loc_internal(38, -103) :- [4] , [5].
  [4]   58            curr_loc(38, -103).
  [5]   35            offset_internal(0, 0, 0) :- [6].
  [6]   60               offset(0, 0, 0).
  [7]   61         destination(805, -300).
  [8]    8         not_marginal_distance(38, -103, 805, -300) :- [9].
  [9]    3            margin(50).
 [10]    5      push_object(z).

Proof time: 9170 inferences in 0.46 seconds, including printing
Proof:

Goal#  Wff#  Wff Instance
-----  ----  ------------
  [0]    0   query :- [1] , [23].
  [1]   22      distance(z, 20) :- [2] , [3] , [4] , [11] , [14].
  [2]    5         push_object(z).
  [3]    9         push_obj_dist(20).
  [4]   21         has_push_object(3) :- [5] , [9] , [10].
  [5]   31            curr_loc_internal(38, -103) :- [6] , [7].
  [6]   58               curr_loc(38, -103).
  [7]   35               offset_internal(0, 0, 0) :- [8].
  [8]   60                  offset(0, 0, 0).
  [9]   61            destination(805, -300).
 [10]   16            quadrant(38-805, -103- -300, 3).
 [11]   23         quad_angle(3, 2.74889) :- [12] , [13].
 [12]    1            pi(3.14159).
 [13]   11            nquads(8).
 [14]   24         angle_world_vs_robot(2.74889, 2.74889- -1.11352-2*3.14159) :-
```

```
                        [15] , [21] , [22].
 [15]   32            curr_dir_internal(-1.11352) :- [16] , [17] , [19].
 [16]   59               curr_dir(2962).
 [17]   35               offset_internal(0, 0, 0) :- [18].
 [18]   60                  offset(0, 0, 0).
 [19]   33               angle_deg_rad(2962-0, -1.11352) :- [20].
 [20]    1                  pi(3.14159).
 [21]    1            pi(3.14159).
 [22]   27            between_minus_and_plus(3.14159, 2.74889- -1.11352,
                           2.74889- -1.11352-2*3.14159).
 [23]   22      direction(z, 2.74889- -1.11352-2*3.14159) :-
                        [24] , [25] , [26] , [33] , [36].
 [24]    5         push_object(z).
 [25]    9         push_obj_dist(20).
 [26]   21         has_push_object(3) :- [27] , [31] , [32].
 [27]   31            curr_loc_internal(38, -103) :- [28] , [29].
 [28]   58               curr_loc(38, -103).
 [29]   35               offset_internal(0, 0, 0) :- [30].
 [30]   60                  offset(0, 0, 0).
 [31]   61            destination(805, -300).
 [32]   16            quadrant(38-805, -103- -300, 3).
 [33]   23         quad_angle(3, 2.74889) :- [34] , [35].
 [34]    1            pi(3.14159).
 [35]   11            nquads(8).
 [36]   24         angle_world_vs_robot(2.74889, 2.74889- -1.11352-2*3.14159) :-
                        [37] , [43] , [44].
 [37]   32            curr_dir_internal(-1.11352) :- [38] , [39] , [41].
 [38]   59               curr_dir(2962).
 [39]   35               offset_internal(0, 0, 0) :- [40].
 [40]   60                  offset(0, 0, 0).
 [41]   33               angle_deg_rad(2962-0, -1.11352) :- [42].
 [42]    1                  pi(3.14159).
 [43]    1            pi(3.14159).
 [44]   27            between_minus_and_plus(3.14159, 2.74889- -1.11352,
                           2.74889- -1.11352-2*3.14159).
push object distance = 20
push object direction = -2.42077
```

B.6 Layers -1 and 0 Control Theory

The following contains the theory for Layer 0 which implements obstacle avoidance. It takes as input the sensor data provided by the low-level sensor module (M1 in Section B.4) and any virtual pushing objects provided by Layer 1 and computes the angle and forward speed needed to maximally avoid these objects.

The theory is also used for Layer -1 which determines whether the robot should halt since halt detection depends on a subset of the obstacle avoidance axioms. It takes the $turn$ and fwd inputs from Layer 0 and uses these to determine whether the robot should turn, go forward, or halt. Because its default behavior is to halt, it keeps the robot protected from (non-moving) obstacles while Layer 0 deliberates. The extraneous Layer 0 axioms do not significantly affect Layer -1's performance in practice.

Unless otherwise specified, distances are in 1/10-in and angles are in radians.

The theory includes the following predicate symbols:

- PTTP-internal predicates (le, $=\backslash=$, etc.)
- pi is as described in the the low-level sensor theory (M1).
- $sonar\_reading\_internal$ is input from M1 and is as described there.
- $object(\langle Object \rangle)$ – true if $Object$ is an object. $external\_object(z)$ is input from Layer 1 into Layer 0 and translated by M1 into $object(z)$, the virtual pushing object.
- $distance(\langle Object \rangle, \langle Distance \rangle)$ – the distance of $Object$ from the robot. We assume each object is a point and sonars sense distinct objects, so that $distance$ should be a function ideally. Layer 1 inputs $external\_distance(z, \langle PUSHING\_OBJ\_DIST \rangle)$ into Layer 0 and M1 translates it into $distance(z, \langle PUSHING\_OBJ\_DIST \rangle)$, the distance of the virtual pushing object.
- $direction(\langle Object \rangle, \langle Direction \rangle)$ – the direction of $Object$ with respect to the robot. Our assumption that each object is a point implies $direction$ should also be a function ideally (assuming we limit its range to $[0, 2\pi]$). Layer 1 inputs $external\_direction(z, \langle Pushing\_Object\_Dir \rangle)$ into Layer 0 and M1 translates it into $direction(z, \langle Pushing\_Object\_Dir \rangle)$, the direction of the virtual pushing object.
- ab_avoid – abnormal predicate which higher layers can set to true to override obstacle avoidance behavior. For example, this makes it possible to make the robot push an object it would otherwise avoid. We currently do not use this predicate, so it is fixed to false.
- $nsonars(16)$ – number of sonar sensors; ideally, should be a constant.
- $min\_dist(30)$ – minimum distance that an object is allowed to be in front of the robot before it halts; ideally, should be a constant.
- $min\_angle(0.3)$ – minimum turning angle; ideally, should be a constant.
- $min\_speed(10)$ – minimum forward speed; ideally, should be a constant.
- $sonar(\langle Sonar \rangle)$ – true if argument is an integer corresponding to a sonar.
- $sonar\_direction(\langle Sonar \rangle, \langle Dir \rangle)$ – direction of corresponding sonar with respect to robot; ideally, should be a function with range $[0, 2\pi]$.
- $correct\_dist(\langle Sonar \rangle, \langle Orig\_dist \rangle, \langle Coarse\_dist \rangle)$ – a coarse filter for the distance $Orig\_dist$ returned by the sonar corresponding to $Sonar$; $Coarse\_dist$ is a weighted average of the distances returned by the sensor and its two immediate neighbors. This is based on the assumption that objects observed by nearby sensors tend to be nearby. Ideally, $correct\_distance$ should be a function.
- $adjacent\_right\_sonar(\langle SonarL \rangle, \langle SonarR \rangle)$ – true if the corresponding sonars are adjacent.
- $fast\_halt\_robot$ – made true if an unprocessed sonar reading indicates that there may be an object too close in front of the robot.
- $object\_ahead$ – true if an object has been detected too close in front of the robot.
- $halt\_robot$ – command for robot to halt; made true iff $object\_ahead$.
- $get\_force([\langle ForceMag \rangle, \langle ForceDir \rangle])$ – the magnitude and direction of the "force" exerted by the surrounding objects on the robot. Ideally, should be a constant.
- $get\_move\_speed(\langle ForceMag \rangle, \langle MoveSpeed \rangle)$ – converts the force magnitude

argument into a movement speed for the robot. Ideally, should be a function.

- $get\_move\_dir(\langle ForceDir\rangle, \langle MoveDir\rangle)$ – converts the force direction into a movement direction for the robot. Ideally, should be a function (restricted to the range $[-\pi, \pi]$).
- $heading\_angle(\langle Angle\rangle)$ – the robot's direction of forward movement; ideally, should be a constant.
- $heading\_speed(\langle Speed\rangle)$ – the robot's speed of forward movement; ideally, should be a constant.
- $angle\_deg\_rad(\langle DegAng\rangle, \langle RadAng\rangle)$ – relates angles in degrees and radians.
- $need\_turn(\langle Angle\rangle)$ – true if $Angle$ is above the turning angle threshold.
- $turn(\langle Angle\rangle)$ – indicates the angle (in degrees) the robot should turn; non-zero if heading angle is above threshold. Ideally, should be a constant.
- $go\_turn(\langle Angle\rangle)$ – angle input into Layer -1 from Layer 0, translated from $external\_turn$ by M1; ideally, should be a constant.
- $need\_fwd(\langle Speed\rangle)$ – true if $Speed$ is above the speed threshold.
- $fwd(\langle Speed\rangle)$ – indicates the speed at which the robot should move forward; non-zero if robot does not need to turn and heading speed is above the speed threshold. Ideally, should be a constant.
- $go\_fwd(\langle Speed\rangle)$ – speed input into Layer -1 from Layer 0, translated from $external\_fwd$ by M1; ideally, should be a constant.

Function symbols used include PTTP-internal functions ($-$, abs, etc.). Constant symbols used include numerical constants.

The theory follows:

```
is_theory0((
% This can be used to control communication with higher layers.
    not_ab_avoid,
%    ab_avoid, % !!!! For the experiment 1a(scenario 2) only.


%%%% Layer -1 INPUT %%%
%
% functions: sonar_reeding_internal, go_turn, go_fwd

%%%% Layer 0 INPUT %%%
%
% predicates: object
% functions: sonar_reading_internal, distance, direction

% Note that, for practicality, the functions above and all other
% functions and constants described in this layer are represented as
% predicates, except where noted otherwise.


%%% PARAMETERS %%%

    pi(3.14159), (not_pi(C0) :- C0=\=3.14159),

    nsonars(16), (not_nsonars(C1) :- C1=\=16),
    min_dist(30), (not_min_dist(C2) :- C2=\=30), % halting distance.
    min_angle(0.3), (not_min_angle(C3) :- C3=\=0.3),
    min_speed(10), (not_min_speed(C4) :- C4=\=10),
```

```
    % 0 <= Sonar_number < NSONARS
    (not_nsonars(NSONARS) ;
     (not_sonar_reading_internal(Sonar_number0, Dist0) ; le(0,Sonar_number0)),
     (not_sonar_reading_internal(Sonar_number0, Dist0) ;
      ls(Sonar_number0, NSONARS))),

    (not_pi(PI) ; not_nsonars(NSONARS) ; sonar_direction(N, N*(2*PI/NSONARS))),


%%% Axioms %%%

%% Module: Sonar %%
%%--------------%%


    (not_pi(PI) ; not_sonar_reading_internal(Sonar_number1, Dist2) ;
     not_sonar_direction(Sonar_number1, Dir2) ;
     ls(Dist2,0) ; ls(Dir2,0) ; ls(PI*2,Dir2) ;
     not_correct_dist(Sonar_number1, Dist2, Dist2_new) ;
     (object(obj_sk1(Dist2_new, Dir2)),
      distance(obj_sk1(Dist2_new, Dir2), Dist2_new) ,
      direction(obj_sk1(Dist2_new, Dir2), Dir2))),

% "correct_dist" is a coarse filter for sonar values using neighboring sonars.

    (correct_dist(Sonar_n1, D2, (DistL+DistR+4*D2)//6) ;
      not_adjacent_right_sonar(Sonar_n1, RightS) ;
      not_adjacent_right_sonar(LeftS, Sonar_n1) ;
      not_sonar_reading_internal(LeftS, DistL) ;
      not_sonar_reading_internal(RightS, DistR) ;
      (ls((DistL+DistR),D2))),

    (correct_dist(Sonar_n1, D2, (DistL+DistR+2*D2)//4) ;
      not_adjacent_right_sonar(Sonar_n1, RightS) ;
      not_adjacent_right_sonar(LeftS, Sonar_n1) ;
      not_sonar_reading_internal(LeftS, DistL) ;
      not_sonar_reading_internal(RightS, DistR) ;
      le(D2,(DistL+DistR))),


    (adjacent_right_sonar(N, N1):- nonvar(N), ls(0,N), (N1 is N-1)),
    (adjacent_right_sonar(N2, N3):- nonvar(N3), ls(N3,15), (N2 is N3+1)),
    (adjacent_right_sonar(0, N4):- nsonars(NSONARS1), (N4 is NSONARS1 - 1)),


%% Module: Collide %%
%%----------------%%


% For proving fast_halt_robot

    sonar(0), sonar(1), sonar(2), sonar(3), sonar(4), sonar(5),
    sonar(6), sonar(7), sonar(8), sonar(9), sonar(10), sonar(11),
    sonar(12), sonar(13), sonar(14), sonar(15),

    (fast_halt_robot :-
       pi(PI), sonar(Sonar_number1),
       sonar_direction(Sonar_number1, Dir5),
       (ls(Dir5,PI/3); ls((2*PI)-PI/3,Dir5)),
       sonar_reading_internal(Sonar_number1, Dist5),
       min_dist(MIN_DIST), le(Dist5,MIN_DIST)),

% For proving halt_robot:

    (not_object_ahead; halt_robot),
    (object_ahead; not_halt_robot),

    (not_pi(PI); not_min_dist(MIN_DIST);
     not_object(Obj5); not_distance(Obj5, Dist3); le(MIN_DIST,Dist3);
```

```
     not_direction(Obj5, Dir4);
     (le(PI/3, Dir4), le(Dir4, (2*PI)-PI/3));
      object_ahead),

    (not_pi(PI); not_min_dist(MIN_DIST);
     not_object_ahead;
     (object(obj_sk2), distance(obj_sk2, dist_sk1), ls(dist_sk1,
      MIN_DIST), direction(obj_sk2, dir_sk1), ((ls(2*PI - PI/3,
      dir_sk1)); (ls(dir_sk1,PI/3)))))),


%% Module: Feelforce %%
%%------------------%%

% This module is implemented by a semantic attachment (a c program).


%% Module: Runaway %%
%%-----------------%%

% get_force returns a [magnitude, direction] pair. Magnitude is a
% positive number proportional to the gravitational force of the
% objects on the robot. Direction is the orientation of the force, in
% the range [-PI,PI].

    (ab_avoid; not_get_force([ForceMag0,ForceDir0]);
     not_get_move_speed(ForceMag0,MoveSpeed0);
     not_get_move_dir(ForceDir0, MoveDir0);
     (heading_angle(MoveDir0), heading_speed(MoveSpeed0))),

% We set the robot to move away from the objects with a speed
% proportional to the objects' gravitational force. To do so, we
% transform the force direction from the range [-PI,PI] to
% [0,2PI] by adding 2PI, then "modding" by 2PI. We then subtract PI to
% make the robot move in the opposite direction.
%
% (Note: We must add 2PI before "modding" because prolog's "mod"
% returns a negative integer for a negative dividend. Also, we do the
% computation in 1/10-radians because "mod" requires integer
% arguments. Finally, we use precomputed multiples of PI for
% simplicity.)

    (get_move_speed(ForceMag1, MoveSpeed1) :- (MoveSpeed1 is ForceMag1)),

    (get_move_dir(ForceDir1, MoveDir1) :-
     (MoveDir1 is ((((integer(ForceDir1*100) + 628) mod 628)/100) - 3.14))),

% NOTE: If there are no objects, the robot may spin indefinitely.


%% Module: Turn %%
%%-------------%%

    (not_heading_angle(Angle1); not_need_turn(Angle1);
     not_heading_speed(Speed3); not_need_fwd(Speed3);
     not_angle_deg_rad(DegAngle,Angle1);
     turn(DegAngle)),

    (not_min_angle(MIN_ANGLE);
     not_need_turn(Angle2);
     (heading_angle(Angle2), ls(MIN_ANGLE, abs(Angle2)))),

% The following computes the need_turn angle.
% Note: Although the "le" predicate is an antecedent in the following axiom, we
% have placed it at the end of the clause. This is to accommodate a
% system-specific restriction requiring all of the variables needed by an "le"
% predicate (and similar built-in predicates) to be instantiated before the
```

```
% prover attempts to evaluate it. By placing the predicate at the end of the
% clause, we force the theorem prover to notice it only after seeing all the
% other literals. No generality is lost.

    (not_min_angle(MIN_ANGLE);
     not_heading_angle(Angle3);
     need_turn(Angle3);
     le(abs(Angle3), MIN_ANGLE)),



%% Module: Forward %%
%%-----------------%%

% Note: If there is only one object in the domain at a distance 1
% directly behind the robot, the robot will head off at a speed of
% 100.

    (not_heading_speed(Speed1); not_heading_angle(Angle4);
     need_turn(Angle4); not_need_fwd(Speed1);
     fwd(Speed1)),


    (not_min_speed(MIN_SPEED);
     not_need_fwd(Speed2);
     ls(MIN_SPEED, Speed2)),
% second direction:
% Note: As in the similar need_turn axiom above, we put the "le" predicate at
% the end of the clause to accommodate system-specific restrictions.
    (not_min_speed(MIN_SPEED);
     need_fwd(Speed2);
     le(Speed2, MIN_SPEED))

)).



% Assemble control theory (L1) and sensor theory (M2) together:
is_theory((Th0,ThS)) :- is_theory0(Th0), is_sensor_theory(ThS).

% Load theory into PTTP.
lay0load :-  is_theory(Th), pttp(Th).



% Layer 0: Subsumption in this layer is done via the domain-closure
% (via negation as failure) of 'object'.

prove_angle(A):-print('starting to prove "turn"'),time_stamp_layer,
    prove((turn(A)),15),
    print('finished proving "turn".'), print_time_stamp.
    prove_angle(0):-print('failed proving "turn"'),print_time_stamp.

prove_speed(S):-print('starting to prove "fwd"'),time_stamp_layer,
    prove((fwd(S)),10).
prove_speed(0):-print('failed proving "fwd"'),print_time_stamp.

goal_layer0([external_fwd(S), external_turn(A)]) :-
   prove_angle(A),prove_speed(S),
   print('turn angle = '), print(A), nl, print('fwd speed = '), print(S).


% Layer -1: Subsumption in this layer is done via negation as failure
% of 'fast_halt_robot' and 'halt_robot'.

prove_turn(A):- print('start prove go_turn(A)'),time_stamp_layer,
    prove(go_turn(A),20).
```

```
prove_turn(0):-print('failed proof. '), print_time_stamp.


prove_fwd(0):- print('start prove fast_halt_robot'),time_stamp_layer,
    prove(fast_halt_robot,10,2,2), print_time_stamp,
    print('start prove halt_robot'),
    time_stamp_layer, prove(halt_robot,50,10,40),
    print('succeeded halt_robot '), print_time_stamp.
prove_fwd(S):- print('proof failed'),print_time_stamp,
    print('start prove go_fwd(S)'),time_stamp_layer,
    prove((go_fwd(S)),5), print_time_stamp.
prove_fwd(0):-print('failed proof. '), print_time_stamp.

goal_layer_1([S, A]) :- prove_turn(A), prove_fwd(S),
    print('turn angle = '), print(A), nl, print('fwd speed = '), print(S).
goal_layer_1([0,0]).
```

Layer 0 sample proof

```
PTTP input formulas:
  1  not_ab_avoid.
  2  pi(3.14159).
  3  not_pi(_G675):-_G675=\=3.14159.
  4  nsonars(16).
  5  not_nsonars(_G695):-_G695=\=16.
  6  min_dist(30).
  7  not_min_dist(_G711):-_G711=\=30.
  8  min_angle(0.3).
  9  not_min_angle(_G731):-_G731=\=0.3.
 10  min_speed(10).
 11  not_min_speed(_G751):-_G751=\=10.
 12  not_nsonars(_G762); (not_sonar_reading_internal(_G770, _G771);le(0, _G770)),
        (not_sonar_reading_internal(_G770, _G771);ls(_G770, _G762)).
 13  not_pi(_G791);not_nsonars(_G762);sonar_direction(_G798,
        _G798* (2*_G791/_G762)).
 14  not_pi(_G791);not_sonar_reading_internal(_G821, _G822);
        not_sonar_direction(_G821, _G828);ls(_G822, 0);ls(_G828, 0);ls(_G791*2,
        _G828);not_correct_dist(_G821, _G822, _G856);object(obj_sk1(_G856,
        _G828)), distance(obj_sk1(_G856, _G828), _G856), direction(obj_sk1(_G856,
        _G828), _G828).
 15  correct_dist(_G887, _G888,  (_G897+_G898+4*_G888)//6);
        not_adjacent_right_sonar(_G887, _G907);not_adjacent_right_sonar(_G912,
        _G887);not_sonar_reading_internal(_G912, _G897);
        not_sonar_reading_internal(_G907, _G898);ls(_G897+_G898, _G888).
 16  correct_dist(_G887, _G888,  (_G897+_G898+2*_G888)//4);
        not_adjacent_right_sonar(_G887, _G907);not_adjacent_right_sonar(_G912,
        _G887);not_sonar_reading_internal(_G912, _G897);
        not_sonar_reading_internal(_G907, _G898);le(_G888, _G897+_G898).
 17  adjacent_right_sonar(_G798, _G992):-nonvar(_G798), ls(0, _G798), _G992 is
        _G798-1.
 18  adjacent_right_sonar(_G1017, _G1018):-nonvar(_G1018), ls(_G1018, 15),
        _G1017 is _G1018+1.
 19  adjacent_right_sonar(0, _G1044):-nsonars(_G1049), _G1044 is _G1049-1.
 20  sonar(0).
 ...
 35  sonar(15).
 36  fast_halt_robot:-pi(_G791), sonar(_G821), sonar_direction(_G821, _G1157),
        (ls(_G1157, _G791/3);ls(2*_G791-_G791/3, _G1157)),
        sonar_reading_internal(_G821, _G1187), min_dist(_G1192),
        le(_G1187, _G1192).
 37  not_object_ahead;halt_robot.
 38  object_ahead;not_halt_robot.
 39 not_pi(_G791);not_min_dist(_G1192);not_object(_G1225);not_distance(_G1225,
       _G1231);le(_G1192, _G1231);not_direction(_G1225, _G1243);le(_G791/3,
       _G1243), le(_G1243, 2*_G791-_G791/3);object_ahead.
```

```
 40 not_pi(_G791);not_min_dist(_G1192);not_object_ahead;object(obj_sk2,
       distance(obj_sk2, dist_sk1), ls(dist_sk1, _G1192),
       direction(obj_sk2, dir_sk1), (ls(2*_G791-_G791/3, dir_sk1);
       ls(dir_sk1, _G791/3)).
 41  ab_avoid;not_get_force([_G1349, _G1352]);not_get_move_speed(_G1349, _G1359);
        not_get_move_dir(_G1352, _G1365);heading_angle(_G1365), heading_speed(_G1359).
 42  get_move_speed(_G1380, _G1381):-_G1381 is _G1380.
 43  get_move_dir(_G1392, _G1393):-_G1393 is
        (integer(_G1392*100)+628)mod 628/100-3.14.
 44 not_heading_angle(_G1425);not_need_turn(_G1425);not_heading_speed(_G1435);
       not_need_fwd(_G1435);not_angle_deg_rad(_G1445, _G1425);turn(_G1445).
 45 not_min_angle(_G1456);not_need_turn(_G1461);heading_angle(_G1461),
       ls(_G1456, abs(_G1461)).
 46 not_min_angle(_G1456);not_heading_angle(_G1484);need_turn(_G1484);
       le(abs(_G1484), _G1456).
 47 not_heading_speed(_G1502);not_heading_angle(_G1507);need_turn(_G1507);
       not_need_fwd(_G1502);fwd(_G1502).
 48  not_min_speed(_G1527);not_need_fwd(_G1532);ls(_G1527, _G1532).
 49  not_min_speed(_G1527);need_fwd(_G1532);le(_G1532, _G1527).
 50  pi(3.14159).
 51  not_pi(_G1565):-_G1565=\=3.14159.
 52  sonar_reading_internal(_G1580, _G1581):-sonar_reading(_G1580, _G1581).
 ...
 59  object(_G1756):-external_object(_G1756).
 60  distance(_G1756, _G1767):-external_distance(_G1756, _G1767).
 61  direction(_G1756, _G1779):-external_direction(_G1756, _G1779).
 62  go_fwd(_G1790):-external_fwd(_G1790).
 63  go_turn(_G1797):-external_turn(_G1797).

PTTP to Prolog translation time: 0.73 seconds, including printing

Prolog compilation time: 0.2 seconds, including printing

PTTP input formulas:
 64  sonar_reading(0, 65).
 ...
 79  sonar_reading(15, 58).
 80  curr_loc(810, -337).
 81  curr_dir(3246).
 82  offset(0, 0, 0).
 83  external_object(z).
 84  external_distance(z, 20).
 85  external_direction(z, -3.08225).
 86  goal_location(corridor2_cross).

PTTP to Prolog translation into latch time: 0.02 seconds, including printing

Assserting into Prolog time: 0.03 seconds, including printing
Start cycle 551
'starting to prove "turn"'
Proof time: 9 inferences in 0 seconds, including printing
Proof:
Goal#  Wff#  Wff Instance
-----  ----  ------------
  [0]    0   query :- [1] , [3] , [5].
  [1]   59      object(z) :- [2].
  [2]   83         external_object(z).
  [3]   60      distance(z, 20) :- [4].
  [4]   84         external_distance(z, 20).
  [5]   61      direction(z, -3.08225) :- [6].
  [6]   85         external_direction(z, -3.08225).
9 inferences so far.
Proof time: 1509 inferences in 0.08 seconds, including printing
Proof:
Goal#  Wff#  Wff Instance
-----  ----  ------------
  [0]    0   query :- [1].
  [1]   14      object(obj_sk1((91+65+4*140)//6, 1* (2*3.14159/16))) :-
```

```
                    [2] , [3] , [5] , [8].
  [2]    2         pi(3.14159).
  [3]   52         sonar_reading_internal(1, 140) :- [4].
  [4]   65            sonar_reading(1, 140).
  [5]   13         sonar_direction(1, 1* (2*3.14159/16)) :- [6] , [7].
  [6]    2            pi(3.14159).
  [7]    4            nsonars(16).
  [8]   15         correct_dist(1, 140,  (91+65+4*140)//6) :-
                        [9] , [10] , [11] , [13].
  [9]   17            adjacent_right_sonar(1, 0).
 [10]   18            adjacent_right_sonar(2, 1).
 [11]   52            sonar_reading_internal(2, 91) :- [12].
 [12]   66               sonar_reading(2, 91).
 [13]   52            sonar_reading_internal(0, 65) :- [14].
 [14]   64               sonar_reading(0, 65).

Proof time: 1529 inferences in 0.09 seconds, including printing

 ... [proof for sonar 1 duplicated]

 ... [proofs (and duplicates) for sonars 0, 2-15]

 ... [proofs (and duplicates) for sonars 0-15 duplicated]

4864 inferences so far.
8021 inferences so far.  'failed proving "turn"'
'starting to prove "fwd"'
Proof time: 57 inferences in 0 seconds, including printing
Proof:
Goal#  Wff#  Wff Instance
-----  ----  ------------
  [0]    0   query :- [1].
  [1]   47      fwd(26) :- [2] , [6] , [10] , [12].
  [2]   41         heading_speed(26) :- [3] , [4] , [5].
  [3]    1            not_ab_avoid.
  [4]   42            get_move_speed(26, 26).
  [5]   43            get_move_dir(2.90998, -0.23).
  [6]   41         heading_angle(-0.23) :- [7] , [8] , [9].
  [7]    1            not_ab_avoid.
  [8]   42            get_move_speed(26, 26).
  [9]   43            get_move_dir(2.90998, -0.23).
 [10]   45         not_need_turn(-0.23) :- [11].
 [11]    8            min_angle(0.3).
 [12]   49         need_fwd(26) :- [13].
 [13]   10            min_speed(10).
'turn angle = '0
'fwd speed = '26
```

Layer -1 sample proof

```
PTTP input formulas:
  1  not_ab_avoid.
 ... [identical to Layer 0 above]
 63  go_turn(_G1797):-external_turn(_G1797).

PTTP to Prolog translation time: 0.73 seconds, including printing

Prolog compilation time: 0.2 seconds, including printing

Start cycle 1
'start prove go_turn(A)'
21 inferences so far.  'failed proof. '
'start prove fast_halt_robot'
465 inferences so far.  'proof failed'
'start prove go_fwd(S)'
```

```
6 inferences so far.  'failed proof. '
'turn angle = '0
'fwd speed = '0

Start cycle 2
 ...
PTTP input formulas:
 64  external_fwd(0).
 65  external_turn(0).

PTTP to Prolog translation into latch time: 0 seconds, including printing

Assserting into Prolog time: 0 seconds, including printing
Start cycle 54
'start prove go_turn(A)'
Proof time: 2 inferences in 0 seconds, including printing
Proof:
Goal#  Wff#  Wff Instance
-----  ----  ------------
  [0]    0   query :- [1].
  [1]   63      go_turn(0) :- [2].
  [2]   65         external_turn(0).
'start prove fast_halt_robot'
465 inferences so far.  'proof failed'
'start prove go_fwd(S)'
Proof time: 2 inferences in 0 seconds, including printing
Proof:
Goal#  Wff#  Wff Instance
-----  ----  ------------
  [0]    0   query :- [1].
  [1]   62      go_fwd(0) :- [2].
  [2]   64         external_fwd(0).
'turn angle = '0
'fwd speed = '0

PTTP input formulas:
 64  external_fwd(0).
 65  external_turn(0).

PTTP to Prolog translation into latch time: 0 seconds, including printing

Assserting into Prolog time: 0 seconds, including printing
Start cycle 55

 ...

PTTP input formulas:
 64  sonar_reading(0, 69).
 ...
 79  sonar_reading(15, 90).
 80  curr_loc(679, -305).
 81  curr_dir(3112).
 82  offset(0, 0, 0).
 83  external_fwd(24).
 84  external_turn(0).
 85  goal_location(corridor2_cross).

PTTP to Prolog translation into latch time: 0.02 seconds, including printing

Assserting into Prolog time: 0.03 seconds, including printing
Start cycle 720
'start prove go_turn(A)'
Proof time: 2 inferences in 0 seconds, including printing
Proof:
Goal#  Wff#  Wff Instance
-----  ----  ------------
  [0]    0   query :- [1].
  [1]   63      go_turn(0) :- [2].
```

```
  [2]   84         external_turn(0).
'start prove fast_halt_robot'
545 inferences so far.  'proof failed'
'start prove go_fwd(S)'
Proof time: 2 inferences in 0 seconds, including printing
Proof:
Goal#  Wff#  Wff Instance
-----  ----  ------------
  [0]    0   query :- [1].
  [1]   62      go_fwd(24) :- [2].
  [2]   83         external_fwd(24).
'turn angle = '0
'fwd speed = '24
```

References

Amir, E. (2001). Efficient approximation for triangulation of minimum treewidth. In *Proc. Seventeenth Conference on Uncertainty in Artificial Intelligence (UAI '01)*, pages 7–15. Morgan Kaufmann.

Amir, E. (2002). Interpolation theorems for nonmonotonic reasoning systems. In *8th European conference on logics in artificial intelligence (JELIA'02)*, pages 233–244. Springer.

Amir, E. and Engelhardt, B. (2003). Factored planning. In *Proc. Eighteenth International Joint Conference on Artificial Intelligence (IJCAI '03)*. Morgan Kaufmann.

Amir, E. and Maynard-Reid II, P. (1998). Logic-based subsumption architecture. In AAAI 1998 Fall Symposium on Cognitive Robotics.

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

Amir, E. and Maynard-Reid II, P. (2001). LiSA: A robot driven by logical subsumption. In working notes of the CommonSense'01 symposium.

Amir, E. and McIlraith, S. (2003). Partition-based logical reasoning for first-order and propositional theories. *Artificial Intelligence*. Accepted for publication.

Arkin, R. C. (1998). *Behavior-based robotics*. MIT Press.

Baral, C. and Tran, S. C. (1998). Relating theories of actions and reactive control. *Electronic Transactions on Artificial Intelligence (http://www.etaij.org)*, 2(3-4):211–271.

Boutilier, C., Reiter, R., Soutchanski, M., and Thrun, S. (2000). Decision-theoretic, high-level agent programming in the situation calculus. In *Proc. National Conference on Artificial Intelligence (AAAI '00)*, pages 355–362. AAAI Press.

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

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

Brooks, R. A. and Flynn, A. M. (1989). Robot beings. In *Proceedings of the*

IEEE/RSJ Int'l Conference on Intelligent Robotics and Systems (IROS-89), pages 2–10.

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

Doherty, P., Gustafsson, J., Karlsson, L., and Kvarnström, J. (1998). Temporal action logics (TAL) language specification and tutorial. *Electronic Transactions on Artificial Intelligence (http://www.etaij.org)*, 3:nr 15.

Farquhar, A. (1997). ATP. Stanford KSL Website. http://www.ksl.stanford.edu/people/axf/reference-manual.html.

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

Gelfond, M. and Lifschitz, V. (1998). Action languages. *Electronic Transactions on Artificial Intelligence (http://www.etaij.org)*, 3:nr 16.

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

Giacomo, G. D., Lespérance, Y., and Levesque, H. J. (1997). Reasoning about concurrent execution, prioritized interrupts, and exogenous actions in the situation calculus. In *Proc. Fifteenth International Joint Conference on Artificial Intelligence (IJCAI '97)*, pages 1221–1226. Morgan Kaufmann.

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

Giunchiglia, E., Kartha, G. N., and Lifschitz, V. (1997). Representing Action: Indeterminacy and Ramifications. *Artificial Intelligence*, 95(2):409–438.

Giunchiglia, F. (1994). GETFOL manual - GETFOL version 2.0. Technical Report DIST-TR-92-0010, DIST - University of Genoa. Available at http://ftp.mrg.dist.unige.it/pub/mrg-ftp/92-0010.ps.gz, and website at http://www-formal.stanford.edu/clt/ARS/Entries/getfol.

Gordon, M. J. and Melham, T., editors (1993). *Introduction to HOL: a theorem proving environment for higher-order logic*. Cambridge University Press. http://www.comlab.ox.ac.uk/archive/formal-methods/hol.html.

Grosskreutz, H. and Lakemeyer, G. (2000). cc-golog: Towards more realistic logic-based robot controllers. In *AAAI/IAAI*, pages 476–482.

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

Jung, C. G. (1999). Layered and resource-adapting agents in the robocup simulation. In Asada, M. and Kitano, H., editors, *RoboCup-98: Robot Soccer WorldCup II*, volume 1604 of *LNAI*, pages 207–220. Springer.

Kaelbling, L. P. (1987). 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.

Kaelbling, L. P. (1990). An architecture for intelligent reactive systems. In Allen,

J., Hendler, J., and Tate, A., editors, *Readings in Planning*, pages 713–728. Morgan Kaufmann publishers Inc.: San Mateo, CA, USA.

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

Kaufmann, M., Manolios, P., and Moore, J. S. (2000). *Computer-Aided Reasoning: An Approach*. Kluwer Academic Publishers. http://www.cs.utexas.edu/users/moore/acl2/acl2-doc.html.

Lakemeyer, G. (1999). On sensing and off-line interpreting in GOLOG. In Levesque, H. and Pirri, F., editors, *Logical Foundations for Cognitive Agents*. Springer.

Levesque, H. and Brachman, R. (1985). A fundamental tradeoff in knowledge representation and reasoning. In Brachman, R. and Levesque, H., editors, *Readings in Knowledge Representation*, pages 41–70. Morgan Kaufmann. Revised from a paper that appeared with the same title in Proc. Fourth Conference of the Canadian Society for Computational Studies of Intelligence.

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

Maes, P. (1989). The dynamics of action selection. In *Proc. Eleventh International Joint Conference on Artificial Intelligence (IJCAI '89)*. Morgan Kaufmann.

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

McCarthy, J. (1958). Programs with common sense. In *Mechanisation of Thought Processes, Proceedings of the Symposium of the National Physics Laboratory*, pages 77–84, London, U.K. Her Majesty's Stationery Office. Reprinted in (McCarthy, 1990).

McCarthy, J. (1986). Applications of Circumscription to Formalizing Common Sense Knowledge. *Artificial Intelligence*, 28:89–116.

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

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

McCune, W. W. (1994). *OTTER 3.0 Reference Manual and Guide*. Argonne National Laboratory/IL, USA. http://www-unix.mcs.anl.gov/AR/otter/.

Miller, R. and Shanahan, M. (1999). The event calculus in classical logic – alternative axiomatizations. *Electronic Transactions on Artificial Intelligence (http://www.etaij.org)*, 4:nr 16. under review.

Minsky, M. (1985). *The Society of Mind*. Simon and Schuster.

Nakashima, H. and Noda, I. (1998). Dynamic subsumption architecture for programming intelligent agents. In *Proc. of International Conf. on Multi-Agent Systems 98*, pages 190–197. AAAI Press.

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

Reiter, R. (1991). The frame problem in the situation calculus: A simple solution

(sometimes) and a completeness result for goal regression. In Lifschitz, V., editor, *Artificial Intelligence and Mathematical Theory of Computation (Papers in Honor of John McCarthy)*, pages 359–380. Academic Press.

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

Reiter, R. (1998). Sequential, temporal GOLOG. In *Principles of Knowledge Representation and Reasoning: Proc. Sixth Int'l Conference (KR '98)*. Morgan Kaumann.

Rosenshein, S. J. and Kaelbling, L. P. (1995). A situated view of representation and control. *Artificial Intelligence*, 73(1–2):149–173.

Rushby, J., Shankar, N., Ruess, H., Owre, S., Tiwari, A., Demoura, L., Saidi, H., and Dutertre, B. (1994-2003). The PVS specification and verification system. SRI CSL Website. http://www.csl.sri.com/pvs.html.

Russell, S. J. and Wefald, E. (1989). Principles of metareasoning. In Brachman, R. J., Levesque, H. J., and Reiter, R., editors, *Proc. First International Conference on Principles of Knowledge Representation and Reasoning (KR '89)*, pages 400–411. Morgan Kaufmann, San Mateo, California.

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

Shanahan, M. (1998). A logical account of the common sense informatic situation for a mobile robot. *Electronic Transactions on Artificial Intelligence (http://www.etaij.org)*, 2:69–104.

Shanahan, M. (2000). Reinventing shakey. In Minker, J., editor, *Logic-Based Artificial Intelligence*. Kluwer.

Shanahan, M. and Witkowski, M. (2000). High-level robot control through logic. In Castelfranchi, C. and Lesprance, Y., editors, *Agent Theories, Architectures and Languages: 7th International Workshop (ATAL'00)*. Springer.

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

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

Stickel, M. E. (1985). Automated deduction by theory resolution. *Journal of Automated Reasoning*, 1:333–355.

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

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

Stickel, M. E. (1988-2003). Pttp theorem prover. SRI Website and QPQ Repository for deductive software. http://www.ai.sri.com/ stickel/pttp.html.

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

Stolzenburg, F., Obst, O., Murray, J., , and Bremer, B. (2000). Spatial agents im-

plemented in a logical expressible language. In Veloso, M., Pagello, E., , and Kitano, H., editors, *RoboCup-99: Robot Soccer WorldCup III*, volume 1856 of *LNAI*, pages 481–494. Springer.

Thielscher, M. (1998). Introduction to the fluent calculus. *Electronic Transactions on Artificial Intelligence (http://www.etaij.org)*, 3:nr 14.