Logic-based Subsumption Architecture

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

June 12, 1998

Abstract

In this paper we describe a logic-based AI architecture based on Brooks' Subsumption Architecture. We axiomatize each of the layers of control in his system separately and use independent theorem provers to derive each layer's output actions given its inputs. We implement the subsumption of lower layers by higher layers using circumscription. We give formal semantics to our approach.

1 Introduction

In [?], Brooks proposed a reactive architecture embodying an approach to robot control different on various counts from traditional approaches. He decomposed the problem into layers corresponding to levels of behavior, rather than according to a sequential, functional form. Within this setting he introduced the idea of subsumption, that is, that more complex layers could not only depend on lower, more reactive layers, but could also influence their behavior. The resulting architecture was one that could service simultaneously multiple, potentially conflicting goals in a reactive fashion, giving precedence to high priority goals.

However, because of its realization in hardware, the architecture lacks declarativeness, making it difficult to implement higher-level reasoning. Also, the semantics of the system are unclear. Furthermore, the increasing hardware complexity with new layers introduces scaling problems. On the other hand, logical techniques are strong in these areas, but typically weak where reactivity is concerned.

In this paper we describe an architecture modeled after Brooks' Subsumption Architecture. We describe our representation for the first two layers of Brooks' system, and describe how subsumption and the general operation of the system can be implemented using theorem proving and circumscription techniques. We give formal semantics to our approach. This is a first step towards creating

a general logic-based AI architecture that is efficient, scalable, and supports reactivity, our long-term goal.

2 Background

2.1 Brooks' Subsumption Architecture

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

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

Brooks would decompose the same domain as follows:

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

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

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

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

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

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

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

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

Wander The `wander` layer consists of two modules which, together with the `avoid` layer, cause the robot to move around aimlessly when it is not otherwise occupied. The `wander` module chooses a new random direction every so often and the `avoid` module combines it with the force output of the avoid layer's feelforce module, outputting an overall heading that suppresses the input to the `avoid` layer's turn module. Hence, when active,

wander mode overrides the default heading computed by the `avoid` layer.

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

2.2 Logic & Circumscription

Since the early days of AI, logic held a promise to serve as the main knowledge representation language in the future intelligent machine [McC58]. In the last decade, theorem provers became wide spread as formal verification tools and lately a few robot systems wielded with logic have emerged (e.g., [Sha96],[GLL97]).

In the logical paradigm, McCarthy's Circumscription [McC80] is one of the first major nonmonotonic reasoning tools. Since its debut, the nonmonotonic reasoning line of work has expanded and several textbooks now exist that give a fair view of nonmonotonic reasoning and its uses (e.g., [Bre91], [Ant97], [BDK97], [DG94], [San94], [Sha97]). The motivations for nonmonotonic reasoning vary from formalizing Common Sense reasoning through Elaboration Tolerance and representation of uncertainty to Belief Revision. We do not expand on these motivations further here, and the reader may look at [Sha97],[McC98],[Pea90] and [Ant97] for further details in these directions.

McCarthy's Circumscription formula [McC80]

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

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

Take for example the following simple theory:

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

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

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

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

3 Logic-Based Subsumption Architecture (LSA)

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

3.1 Intuition and Semantics

The *Logic-Based Subsumption Architecture* (LSA) has in its core a logical theory and a theorem prover. On top of those, there are sensors that affect the theory and manipulators that are affected by the theory (and results found by the theorem prover). In the description that follows we assume that the architecture is used to control a cylindrical robot that has sonar sensors on its perimeter and wheels that control its motion. No other sensors or manipulators are assumed.

On a general note, we want a system based on these premises to work in a loop as follows: First, the physical sonars collect their data and assert it in the form of logical axioms, such as $sonar_reading(sonar_number) = dist$. These axioms[1] are added to the theory comprising the system. Then, we ask the theorem prover to find the required actions for the manipulators. After reaching the conclusions, we transmit them to the robot manipulators, and while the robot executes the requested actions the loop starts again.

The Logic-Based Subsumption Architecture should follow this general pattern, making sure that the loop is fast enough so that the world is not too different at the time we get to our conclusions from the world we based these conclusions on (making sure that the robot does not fall off the cliff while planning a way to avoid the cliff edge). We describe how this behavior is achieved in the next section.

3.2 The architecture

3.2.1 Behavioral decomposition

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

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

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

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

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

3.2.2 Circumscription-based Subsumption

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

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

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

predicates to sentences defining predicates that represent module inputs or outputs which may be suppressed or inhibited, respectively. *During each cycle of any particular layer, we first assert in that layer any sentences that higher layers may have inferred (in the respective output language for those higher layers).* We then circumscribe the `ab` predicates in the theory while varying all other predicates and functions in the axiomatization that we wish to supersede. A theorem prover can then obtain the appropriate outputs (for that layer), taking into account (asserted) interference from higher layers.

Putting the last paragraph in more formal terms, let $Layer_i$ be the theory of layer i, ab_i the additional constant symbol, denoting abnormality for layer i, and P the set of predicates and functions that we wish to vary as a result of the inferred sentences that we include from layer $i+1$. Then, we use the circumscription policy

$$Circ[Layer_i; ab_i; P]$$

¿From an implementation point of view, many times this formula can be substituted with a simple (external to the logic) mechanical interference determining the value of ab_i.

3.2.3 Semantics

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

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

3.3 A Model of Brooks' System

In this part we describe the logical theory corresponding to layers 0 and 1 in Brooks' Subsumption Architecture. We divide our theory to conceptually correspond to the layers and the modules mentioned in figure 1.

In the description that follows, notice that the theorem prover is asked to find the required actions for the modules *Turn* and *Forward* (described below). For simplicity, we omit some parts of the description, and refer the reader to appendix A.

3.3.1 Layer 0

Sonar The Sonar module takes the input from the physical sonars, asserted in the form of the axiom schema $sonar_reading(sonar_number) = dist$, and

translates it to a map of objects (the type of each of the symbols is defined in the appendix).

$$\begin{aligned}
&\forall dist, sonar_number.([sonar_reading(sonar_number) = dist \wedge \\
&\qquad dist \geq 0 \wedge sonar_number \geq 0 \wedge sonar_number < NSONARS] \\
&\iff \exists obj.Object(obj) \wedge Direction(obj) = sonar_direction(sonar_number) \\
&\qquad \wedge Distance(obj) = dist)
\end{aligned}$$

$$\begin{aligned}
&\forall obj1, obj2.(Object(obj1) \wedge Object(obj2) \wedge \\
&\qquad Direction(obj1) = Direction(obj2) \Longrightarrow obj1 = obj2)
\end{aligned}$$

$$sonar_direction(sonar_number) = \frac{2\pi}{NSONARS} * sonar_number \tag{1}$$

Notice that we assumed that the robot's 0 radians reference point is straight ahead, the front sonar is numbered 0, and the sonars are numbered consecutively counter-clockwise from 0 to $NSONARS - 1$.

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

$$(\exists obj.Object_Ahead(obj)) \Longrightarrow halt$$

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

Feelforce Feelforce does the dirty work of computing the combined repulsive force from the different detected objects[2].

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

Runaway

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

Turn

$$need_turn(heading_angle) \Longrightarrow turn(heading_angle) \tag{5}$$

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

Forward

$$\begin{array}{l} \neg halt \wedge \neg need_turn(heading_angle) \wedge need_fwd(heading_speed) \\ \Longrightarrow \qquad fwd(heading_speed) \\ \\ halt \vee need_turn(heading_angle) \Longrightarrow fwd(0) \end{array} \tag{6}$$

3.3.2 Layer 1

Wander

$$\begin{array}{l} (clock \ mod \ WANDER_FREQ) \leq \frac{WANDER_FREQ}{2} \Longrightarrow \\ \qquad wander_angle = random(1) * 2\pi \wedge wander_strength = FORCE \end{array} \tag{7}$$

Avoid

$$\begin{array}{l} heading_angle = new_angle(wander_angle, wander_strength, \\ \qquad force_direction, force_strength) \\ \\ heading_speed = new_strength(wander_angle, wander_strength, \\ \qquad force_direction, force_strength) \end{array} \tag{8}$$

3.3.3 Circumscribing the Theory

Finally, we add the circumscription formula

$$Circ[Layer0; ab_{avoid}; heading_angle, heading_speed]$$

4 Discussion and Future Work

In the last 5 years, the logical approach to AI got re-invigorated with positive results on different frontiers, from planning (e.g., the work of Bibel and of Selman and Kautz ([KMS96])) to sensing and AI architectures (e.g., [Sha96], [YLS94], [GLL97]).

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

At the moment, the system is not yet implemented. Besides implementing the system on a mobile robot, our future work plan includes expanding the layers described in this paper to contain planning layers, map-creating layers (e.g., creating a map of the world (possibly following the work of Shanahan)), layers that contain beliefs about the world (e.g., we may want to doubt our conclusion that we are in a certain location if we believe that a moment ago we

were in a distant location and no reasonable change was done to the world to put us in that new location), etc. This project also serves as an experiment in the Elaboration Tolerance of the layering approach.

5 Acknowledgments

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

References

[Ant97] Grigoris Antoniou. *Nonmonotonic Reasoning.* MIT Press, Cambridge, Massachusetts, 1997.

[BDK97] Gerd Brewka, Jürgen Dix, and Kurt Konolige. *Nonmonotonic Reasoning: An Overview*, volume 73 of *CSLI Lecture Notes.* CSLI Publications, 1997.

[Bre91] Gerhard Brewka. *Nonmonotonic Reasoning: Logical Foundations of Common Sense.* Cambridge University Press, 1991.

[DG94] J.A.Robinson D.M. Gabbay, C.J.Hogger, editor. *Handbook of Logic in Artificial Intelligence and Logic Programming, Volume 3: Nonmonotonic Reasoning and Uncertain Reasoning.* Oxford University Press, Great Britain, 1994.

[GLL97] Giuseppe De Giacomo, Yves Lesperance, and Hector J. Levesque. Reasoning about concurrent execution, prioritized interrupts, and exogenous actions in the situation calculus. In M. Pollack, editor, *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97)*, pages 1221–1226. Morgan Kaufmann, 1997.

[KMS96] Henry Kautz, David McAllester, and Bart Selman. Encoding plans in propositional logic. In J. Doyle, editor, *Proceedings of KR'96*, pages 374–384, Cambridge, Massachusetts, November 1996. KR, Morgan Kaufmann.

[McC58] John McCarthy. Programs with Common Sense. In *Mechanisation of Thought Processes, Proceedings of the Symposium of the National Physics Laboratory*, pages 77–84, London, U.K., 1958. Her Majesty's Stationery Office. Reprinted in McC90.

[McC80] John McCarthy. Circumscription—A Form of Non-Monotonic Reasoning. *Artificial Intelligence*, 13:27–39, 1980. Reprinted in [**?**].

[McC98] John McCarthy. Elaboration Tolerance. In Common-Sense '98 and in McCarthy's web page http://www-formal.stanford.edu/jmc/elaboration.html, 1998.

[Pea90] Judea Pearl. Reasoning under uncertainty. *Annual Review of Computer Science*, 4:37–72, 1990.

[San94] E. Sandewall. *Features and Fluents.* Oxford University Press, 1994.

[Sha96] M. P. Shanahan. Robotics and the common sense informatic situation. In *Proceedings ECAI 96*, pages 684–688, 1996.

[Sha97] Murray Shanahan. *Solving the Frame Problem, a mathematical investigation of the common sense law of inertia.* MIT press, Cambridge, MA, 1997.

[YLS94] F. Lin D. Marcu R. Reiter Y. Lesperance, H. J. Levesque and R. B. Scherl. A logical approach to high-level robot programming: A progress report. In B. Kuipers, editor, *Control of the Physical World by Intelligent Systems: Papers from the 1994 AAAI Fall Symposium*, pages 79–85, 1994.

A The Logical Theory

In this appendix we draw the entire logical theory for our version of Brooks' Subsumption Architecture example from [**?**].

A.1 Language

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

object constants $NSONARS, \pi, numbers, MIN_DIST, MIN_SPEED, MIN_ANGLE,$ $force_direction$, $force_strength$, $force_x$, $force_y, ab_{avoid}$, $heading_angle$, $heading_speed, WANDER_FREQ, wander_angle, wander_strength, FORCE.$

variables Real: $dist, dir, dir_1, dir_2$, $angle_1, str_1, angle_2, str_2$.
Objects: $obj, obj1, obj2, obj'$.
Integer: $sonar_number$.

functions $sonar_reading(int), Direction(object), sonar_direction(int), Distance(object),$ $sum_x(int, int), sum_y(int, int), new_angle(real, real, real, real), new_strength(real, real, real, real).$

predicates $halt, Object(object), Object_Ahead(object), Exactly_One_Object(real, real, object),$ $need_turn(real), turn(real), need_fwd(real), fwd(real).$

library functions tan (tangent), sin (sine), cos (cosine), square root, $*, +, /,$ $clock$, $random(int)$.

A.2 Layer 0

Sonar

$$\begin{aligned}
&\forall dist, sonar_number.([sonar_reading(sonar_number) = dist \wedge \\
&\qquad dist \geq 0 \wedge sonar_number \geq 0 \wedge sonar_number < NSONARS] \\
&\iff \exists obj.Object(obj) \wedge Direction(obj) = sonar_direction(sonar_number) \\
&\qquad \wedge Distance(obj) = dist)
\end{aligned}$$

$$\begin{aligned}
&\forall obj1, obj2.(Object(obj1) \wedge Object(obj2) \wedge \\
&\qquad Direction(obj1) = Direction(obj2) \implies obj1 = obj2)
\end{aligned}$$

$$sonar_direction(sonar_number) = \frac{2\pi}{NSONARS} * sonar_number \tag{9}$$

Collide

$$(\exists obj.Object_Ahead(obj)) \implies halt$$

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

Feelforce

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

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

$$\begin{aligned}
&\forall dir_1, dir_2, obj.(Exactly_One_Object(dir_1, dir_2, obj) \implies \\
&\qquad sum_x(dir_1, dir_2) = sin(Direction(obj)) * \tfrac{1}{Distance(obj)^2}) \\
&sum_x(dir_1, dir_2) = sum_x(dir_1, dir_2 - \tfrac{dir_2 - dir_1}{2}) + sum_x(dir_1 + \tfrac{dir_2 - dir_1}{2}, dir_2)
\end{aligned}$$

$$\begin{aligned}
&\forall dir_1, dir_2, obj.(Exactly_One_Object(dir_1, dir_2, obj) \implies \\
&\qquad sum_y(dir_1, dir_2) = cos(dir) * \tfrac{1}{dist^2}) \\
&sum_y(dir_1, dir_2) = sum_y(dir_1, dir_2 - \tfrac{dir_2 - dir_1}{2}) + sum_y(dir_1 + \tfrac{dir_2 - dir_1}{2}, dir_2)
\end{aligned}$$

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

Runaway

$$\begin{array}{l} \neg ab_{avoid} \Longrightarrow heading_angle = (\pi + force_direction) \ mod \ 2\pi \\ \neg ab_{avoid} \Longrightarrow heading_speed = force_strength \end{array} \tag{13}$$

Turn

$$\begin{array}{l} need_turn(heading_angle) \Longrightarrow turn(heading_angle) \\ \\ \forall angle_1.(need_turn(angle_1) \iff angle_1 > MIN_ANGLE) \end{array} \tag{14}$$

Forward

$$\begin{array}{l} \neg halt \wedge \neg need_turn(heading_angle) \wedge need_fwd(heading_speed) \\ \Longrightarrow \qquad fwd(heading_speed) \\ \\ \forall str_1.(need_fwd(str_1) \iff str_1 > MIN_SPEED) \\ \\ halt \vee need_turn(heading_angle) \Longrightarrow fwd(0) \end{array} \tag{15}$$

A.3 Layer 1

Wander

$$\begin{array}{l} (clock \ mod \ WANDER_FREQ) = 0 \Longrightarrow \\ \qquad wander_angle = random(1) * 2\pi \wedge wander_strength = FORCE \end{array} \tag{16}$$

Avoid

$$\begin{array}{l} heading_angle = new_angle(wander_angle, wander_strength, \\ \qquad force_direction, force_strength) \\ \\ new_angle(angle_1, str_1, angle_2, str_2) = tan^{-1}(\frac{cos(angle_1)*str_1 + cos(angle_2)*str_2}{sin(angle_1)*str_1 + sin(angle_2)*str_2}) \\ \\ heading_speed = new_strength(wander_angle, wander_strength, \\ \qquad force_direction, force_strength) \\ \\ new_strength(angle_1, str_1, angle_2, str_2) = \\ \sqrt{(cos(angle_1) * str_1 + cos(angle_2) * str_2)^2 + (sin(angle_1) * str_1 + sin(angle_2) * str_2)^2} \end{array} \tag{17}$$