Logical Robots: Declarative Multi-Agent Programming in Logica

Demonstration Track

Evgeny Skvortsov
Google LLC
Kirkland, WA, USA
evgskv@google.com

Yilin Xia
University of Illinois
Urbana-Champaign, IL, USA
yilinx2@illinois.edu

Ojaswa Garg
Google LLC
Kirkland, WA, USA
ojaswagarg@google.com

Shawn Bowers
Gonzaga University
Spokane, WA, USA
bowers@gonzaga.edu

Bertram Ludäscher
University of Illinois
Urbana-Champaign, IL, USA
ludaesch@illinois.edu

ABSTRACT

We present Logical Robots, an interactive multi-agent simulation platform where autonomous robot behavior is specified declaratively in the logic programming language Logica. Robot behavior is defined by logical predicates that map observations from simulated radar arrays and shared memory to desired motor outputs. This approach allows low-level reactive control and high-level planning to coexist within a single programming environment, providing a coherent framework for exploring multi-agent robot behavior.

KEYWORDS

Logic Programming; Multi-Agent Systems; Declarative Programming; Robot Simulation

ACM Reference Format:
Evgeny Skvortsov, Yilin Xia, Ojaswa Garg, Shawn Bowers, and Bertram Ludäscher. 2026. Logical Robots: Declarative Multi-Agent Programming in Logica: Demonstration Track. In *Proc. of the 25th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2026), Paphos, Cyprus, May 25 – 29, 2026*, IFAAMAS, 3 pages. https://doi.org/10.65109/UKVJ1021

1 INTRODUCTION

Logic programming has a rich history in robot planning and multi-agent systems, from approaches like Shakey and STRIPS [5, 11] through situation calculus [10] and Golog [8], to modern Answer Set Programming [4, 13] and agent languages like Jason [1] and GOAL [6]. While logic programming has been applied to reactive control (IndiGolog [3] and LTL synthesis [7]), surveys of logic-based MAS technologies [2] confirm most approaches operate at high abstraction levels, leaving low-level control to imperative code. Thus, symbolic-subsymbolic integration remains an open frontier.

We show progress on this frontier with Logical Robots[1], an interactive simulation platform where autonomous robots navigate labyrinths, coordinate around hazards, and pursue goals—**with symbolic planning and low-level control unified in declarative Logica programs** [12]. The key enabler is that robotics problems naturally decompose into aggregations over sensor data: from high-level reasoning such as selecting the nearest beacon (`ArgMin` over distances), to low-level control such as computing steering corrections (`WeightedAverage` over radar data). Unlike Answer Set Programming [9] and Prolog [14], which face grounding bottlenecks with large-scale data, Logica compiles to SQL, treating sensor streams as relational tables and performing aggregations at modern database speeds. This enables sensor fusion, reactive control, and symbolic planning within a single coherent framework.

Contributions: This work provides: (1) A demonstration that Logica can help unify symbolic planning and low-level control through declarative aggregations that process large scale sensor data; and (2) An interactive multi-agent simulation platform with ten examples of progressively challenging coordination scenarios.

2 THE LOGICAL ROBOTS PLATFORM

2.1 Simulation Ontology

Logical Robots provides a two-dimensional labyrinth simulation environment (Figures 1 and 2) containing four entity types:

- **Robots** act as autonomous agents (colored squares with radiating sensor lines), each independently executing the same Logica program for movement control. Each robot is identified by a unique name.
- **Beacons** define stationary waypoints (e.g., A, B, Fire Station) serving as navigational aids and area triggers.
- **Areas** are spatial regions marked by colored circles (blue and red) that toggle between accessible and restricted states when robots detect designated beacons.
- **Win Conditions** represent success criteria, requiring robots to reach target zones (e.g., Mining zones) simultaneously.

2.2 Sensing: Sensors and Memory

Each default labyrinth comes with a fixed number of robots with initial positions. The platform provides two input predicates for robots to process and decide their next move:

- `Sensor(robot_name:, sensor:)` provides immediate observations: `sensor.radar` returns an array of sensor rays, each containing `angle`, `distance`, `object` type (beacon/wall/robot/none), and `label` (beacon ID or robot name) values.

[1]Demo website: https://logica.dev/robots; source code: https://github.com/EvgSkv/logica/tree/main/docs/robots; video: https://tinyurl.com/logicalrobots

Proc. of the 25th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2026), C. Amato, L. Dennis, V. Mascardi, J. Thangarajah (eds.), May 25 – 29, 2026, Paphos, Cyprus. © 2026 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). https://doi.org/10.65109/UKVJ1021

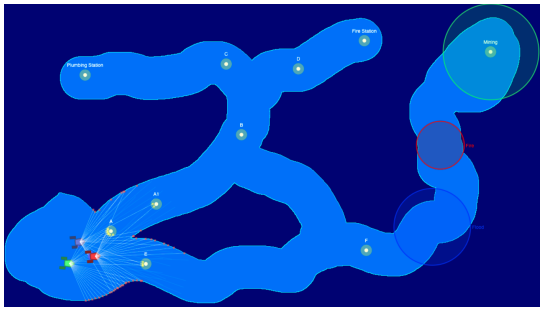

Figure 1: Multi-robot exploration with sensor rays (red endpoints: wall, green endpoints: beacon) and dynamic area toggling triggered by specific beacon detection (e.g., Fire Station).

- `Memory(robot_name:, memory:)` accesses stored information in user-defined data structures (e.g., strings, lists, JSON). By default, each robot accesses only its own memory. However, users can also configure the predicate to allow access to the memory of the other robots in the simulation.

2.3 Reactive Control Logic

Robot behavior is defined by a `Robot` predicate that specifies the **execution model**. The simulation operates in discrete synchronous rounds: each robot independently executes its Logica program once per timestep, reading `Sensor` and `Memory` predicates as inputs and producing `desire` and updated `memory` as outputs. Each robot maintains its own memory store that persists across timesteps until overwritten, with user-configurable cross-robot read access. A complete obstacle-avoiding robot is defined in the following program.

```
1  FreedomMotion(radar) = WeightedAverage {
2    x.distance -> x.angle :- x in radar
3  };
4  Robot(robot_name:, desire:, memory: "I am a robot") :-
5    Sensor(robot_name:, sensor:),
6    freedom = FreedomMotion(sensor.radar),
7    speed = 0.5,
8    desire = {
9      left_engine:  speed - freedom + 0.1,
10     right_engine: speed + freedom
11   };
```

In this simple example, Line 1 defines `FreedomMotion` as a weighted average where each radar ray votes for its angle, weighted by its distance value. Rays detecting distant objects (or no obstacles) cast stronger votes than rays hitting nearby obstacles, which directs the robot to open space. Lines 4–11 define the robot's behavior: it reads sensor data (line 5), computes the preferred direction (line 6), and converts this into differential drive commands (lines 9–10). Positive `freedom` slows the left engine and speeds up the right (turning left), while negative `freedom` does the opposite. The 0.1 asymmetry breaks deadlocks in symmetric configurations.

2.4 Planning With State

While obstacle avoidance is purely reactive, more complex behaviors require state. Consider the distributed pathfinding solution for Figure 2: robots independently observe beacons and store pairwise distances in local memory, while a leader robot aggregates all of the memories of the robots to build the beacon network. Once any robot reaches "Home", the leader computes shortest paths, enabling other robots to navigate via visited beacons.

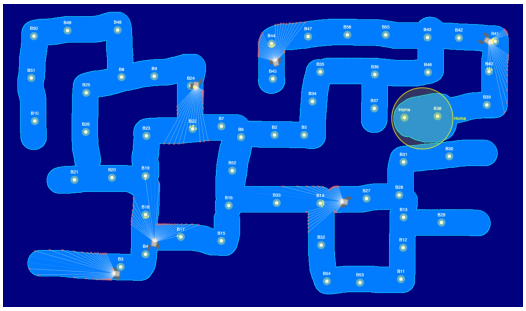

Figure 2: Distributed mapping scenario: robots collaboratively explore a beacon-filled labyrinth, building a shared navigation graph to find their way "Home."

The following Logica rule computes, as part of the Bellman-Ford pathfinding algorithm, the `PosteriorHomeDistance`, which iteratively updates each beacon's shortest distance to "Home". The `Min=` aggregation evaluates three cases and selects the minimum: (1) keep the previous distance `HomeDistance(beacon)`; (2) set distance to zero if this is the "Home" beacon; or (3) compute distance via a neighbor as `HomeDistance(neighbor) + D(neighbor,beacon)`, where `D` represents edge distances between beacons (computed from the radar observations of the robots). Over multiple timesteps, the shortest-path information is recursively propagated through the beacon network.

```
1  PosteriorHomeDistance(beacon) Min= d :-
2    d == HomeDistance(beacon) |
3    d == 0, beacon == "Home"  |
4    d == HomeDistance(neighbor) + D(neighbour, beacon);
```

3 DEMONSTRATION PLAN

We will first introduce the platform interface and demonstrate the core functionalities of Logical Robot: loading labyrinths, writing robot control programs in Logica, executing simulations, utilizing the debug mode and customizing labyrinths with live editor. Next, we will demonstrate three of the ten example scenarios in detail to highlight the platform's capabilities:

(1) **Level 7: Station Management.** Robots have different goals, where some maintain contact with station beacons to disable hazards, while others navigate to the Mining area (Figure 1).
(2) **Level 8: Formation Navigation.** Robots read the memory of encountered robots to follow them and eventually reach the home area together as a group.
(3) **Level 10: Distributed Mapping.** Robots collaboratively discover beacon positions. A leader robot maintains shared memory aggregating these discoveries, enabling distributed Bellman-Ford pathfinding where robots follow computed paths through the beacon network.

The platform's increasingly complex example levels, and the ability to create new levels, positions Logica Robot as a novel educational tool for learning logic-based programming techniques and their use in declaratively building multi-agent systems.

REFERENCES

[1] Rafael H. Bordini, Jürgen F. Hübner, and Michael Wooldridge. 2007. *Programming Multi-Agent Systems in AgentSpeak using Jason.* Wiley.

[2] Roberta Calegari, Giovanni Ciatto, Viviana Mascardi, and Andrea Omicini. 2021. Logic-based technologies for multi-agent systems: a systematic literature review. *Autonomous Agents and Multi-Agent Systems* 35, 1 (2021), 1.

[3] Giuseppe De Giacomo, Yves Lespérance, Hector J Levesque, and Sebastian Sardina. 2009. IndiGolog: A high-level programming language for embedded reasoning agents. In *Multi-Agent Programming: Languages, Tools and Applications*. Springer, 31–72.

[4] Esra Erdem and Volkan Patoglu. 2018. Applications of ASP in Robotics. *KI – Künstliche Intelligenz* 32, 2-3 (2018), 143–149.

[5] Richard E. Fikes and Nils J. Nilsson. 1971. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence* 2, 3-4 (1971), 189–208.

[6] Koen V. Hindriks. 2009. Programming Rational Agents in GOAL. In *Multi-Agent Programming*. Springer, 119–157.

[7] Hadas Kress-Gazit, Georgios E Fainekos, and George J Pappas. 2009. Temporal-logic-based reactive mission and motion planning. *IEEE transactions on robotics* 25, 6 (2009), 1370–1381.

[8] Hector J. Levesque, Raymond Reiter, Yves Lespérance, Fangzhen Lin, and Richard B. Scherl. 1997. GOLOG: A Logic Programming Language for Dynamic Domains. *Journal of Logic Programming* 31, 1-3 (1997), 59–83.

[9] Vladimir Lifschitz. 2019. *Answer set programming*. Vol. 3. Springer Cham.

[10] John McCarthy and Patrick J. Hayes. 1969. Some Philosophical Problems from the Standpoint of Artificial Intelligence. *Machine Intelligence* 4 (1969), 463–502.

[11] Nils J. Nilsson. 1984. *Shakey the Robot.* Technical Report 323. AI Center, SRI International.

[12] Evgeny Skvortsov, Yilin Xia, and Bertram Ludäscher. 2024. Logica: Declarative Data Science for Mere Mortals. In *Proceedings of the 27th International Conference on Extending Database Technology (EDBT)*. 842–845.

[13] Tran Cao Son, Enrico Pontelli, Michael Balduccini, and Torsten Schaub. 2023. Answer Set Planning: A Survey. *Theory and Practice of Logic Programming* 23, 1 (2023), 226–298.

[14] Jan Wielemaker, Tom Schrijvers, Markus Triska, and Torbjörn Lager. 2012. Swi-prolog. *Theory and Practice of Logic Programming* 12, 1-2 (2012), 67–96.