Designing Logic-based Robots [*]

Felipe W. Trevizan, Leliane N. de Barros, Flávio S. Corrêa da Silva

Institute of Mathematics, University of São Paulo
Rua do Matão, 1010, Cidade Universitária
CEP 05508-090 São Paulo, SP, Brasil
{trevisan, leliane, fcs}@ime.usp.br

Abstract

A rational agent exploring a complex and dynamic environment with incomplete information needs cognitive capabilities, e.g. *planning*, in addition to its perception and reaction for basic functionalities. However, mere planning, i.e., reasoning about sequences of actions, is not sufficient to solve problems in such complex environment. This is because (i) agents need to execute actions while they plan, (ii) they must gather and interpret sensor information, (iii) revise their world model, and (iv) adapt their own goals during a task.

The *knowledge representation and non monotonic reasoning* area has shown the advantage of using logical formalisms to specify rational agents for complex robot applications, also called *cognitive robotics* applications. An example of such formalism is the Golog language and its different dialects. What those areas are still missing is implementational testbeds to evaluate existing theories. Since practical experimentation highlights the need for the improvement of existing formal theories at the ontological level.

This work presents a logic-based implementation of an agent for the Wumpus World domain, which can be envisaged as a simplified model of an agent that reasons logically about its actions and sensor information in the presence of incomplete knowledge. Through a complete implementation of an agent in a real robot, the Lego® MindStorms^{TM} robot, we show some experiments to verify that the Wumpus World can be an implementational testbed used to identify difficulties in transforming theory into operational solutions.

Keywords: Cognitive Robotics, Planning, Lego® MindStorms^{TM}.

1 Introduction

Works on logically reasoning agents have made significant progress, specially through the cognitive robotics (CR) [13, 7, 12] – the research field at the confluence of logics and robotics for reason about actions. The aim of CR is to program robotic agents using explicitly only high-level actions and relations among actions characterized as formal logical statements. To archive this goal, much recent research in the area has been devoted to the development of robot high-level languages, such as Golog [13].

Golog is a language based on the situation calculus [14], implemented as a Prolog meta-interpreter. This CR language has been used on experiments with extended logical theories, such as temporal or probabilistic logics [7, 16], but it has rarely seen comparison results between different approaches. In fact, there is not a reliable set of testbed examples, adopted by this community, to be used to compare different approaches.

In previous work [23, 24, 25] we have proposed

[*]This article is an extended version of [23] with minor modifications

the Wumpus World domain as an interesting challenge for the cognitive robotics area. Although this is a domain usually adopted to teach classes on knowledge representation in introductory courses on AI, the design, programming and implementation of a complete program for an agent is not a trivial task. This can be corroborated by the fact that the Wumpus World has been proposed as a special theme in the Sixth Workshop on Nonmonotonic Reasoning, Actions, and Change at IJCAI 2005 [1] and Eleventh Nonmonotonic Reasoning at KR 2006 [2].

This work presents a logic-based implementation of an agent for the Wumpus World domain, which can be envisaged as a simplified model of an agent that reasons logically about its actions and sensor information in the presence of incomplete knowledge. Through a complete implementation of an agent in a real robot, the Lego® MindStorms™ robot, we show some experiments to: (i) demonstrate how to interleave on-line execution of the robot low-level actions with the Golog logical projection of the high-level control program and (ii) measure the agent time spent on reasoning for a simple evaluation. We also give some guarantees about the behavior of this agent.

This paper is organized as follow. In Section 2 we briefly describe the Golog language and the situation calculus. In Section 3 we introduce the Lego® MindStorms™ robots , as well as the language Legolog. Section 4 presents the development phases of a Legolog agent for the Wumpus World, that are: (1) the agent's action description in situation calculus formalism; (2) the agent Golog implementation; (3) the proof of some desirable features; (4) the implementation of the low-level actions for the Lego® MindStorms™ robot; and (5) how we have constructed a real setup for the Wumpus World. Section 5 gives the results of some experiments of the Legolog agent. Section 6 makes a discussion about few other works for the Wumpus World.

2 Golog: A Language for Cognitive Robotics

Planning can be defined as the problem of finding a sequence of actions to achieve a desired state of the world (*goal state*) or behaviors (*goal task*). This usually amounts to computationally intractable problems, since the search space is proportional to $n_a^{|\text{plan}|}$, in which n_a is the number of possible actions and $\|\text{plan}\|$ is the length of the smallest sequence of actions that archives the goal state from the initial state (i.e. a solution plan). In order to make this search space smaller, classical planning algorithms employ:

- conflict resolution techniques for actions, which typically transform state space search into plan space search [10];
- heuristic methods [4, 9] to guide the state space search; or
- compound tasks, which define constraints on actions compositions, through task networks, also called *hierarchical task network planning* (HTN planning) [8].

Golog [13] is a programming language for intelligent agents through which we can specify constraints on actions compositions, thus pruning the search space. The constraints are specified in a *high-level program*, which can be defined as a program comprised by (1) primitive instructions, which are the actions an agent can execute in the environment, described as situation calculus statements [14]; (2) tests, which make use of domain dependent predicates that are affected by actions; (3) procedures, which correspond to compound actions as in HTN planning [3]; and (4) non-deterministic choices, which allow lookahead in sets of actions to select what to add to the solution plan.

Therefore, instead of looking for a sequence of actions to achieve a goal, a Golog agent looks for a sequence of actions to generate a valid execution of its high-level program, i.e., a valid decomposition of the high level program that implements the agent, resulting in its desired behavior. This is very similar to an HTN planning agent implementation.

2.1 The situation calculus

Golog is based on the situation calculus [14]: a logical formalism based on First Order Predicate Logics (FOPL) that allows the logical projection of world properties. Its ontology includes *situations*, which are snapshots of the world; *fluents*, which represent world properties; and *actions*, which are capable of altering the truth value of fluents. In situation calculus, the constant s_0 denotes the *initial situation*; the function $do(\alpha, \sigma)$ denotes the *resulting situation* after performing the action α in situation σ; the predicate $poss(\alpha, \sigma)$ represents that action α can be executed in situation σ; and the predicate $holds(\phi, \sigma)$ represents that fluent ϕ is true in situation σ. Section 4.2 shows an example of situation calculus axioms for the Wumpus World.

Given a specification of a planning domain as a situation calculus axiomatization, the solution plan can be found through theorem proving in FOPL. Let $\mathcal{A}$ be the set of axioms that characterize the actions of an agent, $\mathcal{I}$ the set of axioms that characterize the initial situation and $\mathcal{G}$ a logical statement that characterizes the agent's goal. The constructive proof of

$$\mathcal{A} \wedge \mathcal{I} \models (\exists S).legal(S) \wedge \mathcal{G}(S), \quad \text{where} \\ legal(S) \equiv poss(\alpha_1, s_0) \wedge \cdots \wedge \\ poss(\alpha_n, do(\alpha_{n-1}, do(\ldots, do(\alpha_1, s_0))) \ldots),$$

generates an instance of the variable S as the term $do(\alpha_n, do(\ldots, do(\alpha_1, s_0)) \ldots)$, which corresponds to the sequence of actions $\langle \alpha_1, \ldots, \alpha_n \rangle$, that when executed by the agent from the initial situation s_0, takes it to the goal situation.

2.2 The Golog Meta-interpreter

Golog programs are executed by a specialized theorem prover (Table 1) in FOPL [13]. The user must provide a situation calculus axiomatization $\mathcal{A}$, describing the actions of an agent (*declarative knowledge*), plus a control program c, specifying the desired behavior of the agent (*procedural knowledge*). The execution of the Golog program corresponds to the proof that $\mathcal{A} \models exec(c, s_0, \sigma)$, where $exec(c, s_0, \sigma) \equiv (\exists \sigma).\sigma \wedge legal(\sigma)$ and $\sigma = do(\alpha_n, do(\ldots, do(\alpha_1, s_0)) \ldots)$ is a decomposition of c.

Another characteristic of Golog is that it performs *off-line* planning, that is, Golog searches for a sequence of actions that is a valid execution of a high-level program *before* any action has been actually executed by the agent. In order to solve problems that require the execution of actions *during* planning, namely *on-line* planning, the language IndiGolog [11] has been created. Using IndiGolog we can specify programs that perform sensing and execution of actions, while search for a solution plan. However, IndiGolog is not capable to combine the *off-line* and *on-line* planning, and a solution to this problem is partially done in section 4.3.3.

```
:- op(950,xfy,[&]).  % sequence
:- op(500,xfy,[?]).  % test (temporal projection)
:- op(960,xfy,[|]).  % non-deterministic choice
:- op(960,xfy,[~]).  % negation as failure
exec(A1 & A2,S1,S3) :-
        exec(A1,S1,S2), exec(A2,S2,S3).
exec(P?,S,S):- holds(P,S).
exec(A1 | A2,S1,S2) :-
        exec(A1,S1,S2); exec(A2,S1,S2).
exec(if(P,A1,A2),S1,S2) :-
        exec(P? & A1 | ~P? & A2,S1,S2).
exec(star(E),S1,S2) :-
        S1=S2; copy(E,E1), exec(E & star(E1),S1,S2).
exec(while(P,A),S1,S2) :-
        copy(P,P1), exec(star(P? & A) & ~P1?,S1,S2).
exec(A,S1,S2) :- proc(A,A1), exec(A1,S1,S2).
exec(A,S,do(A,S)) :- prim(A), poss(A,S).
holds(A=A,_).
holds(~P,S) :- not holds(P,S).
% frame axiom
holds(P,do(A,S)) :- holds(P,S), not affects(A,P).
```

Table 1. A simplified implementation of the Golog as a Prolog meta-interpreter.

3 Legolog: a Golog for the Lego® MindStorms™ Robot

Usually, researchers in cognitive robotics make use of simulations to test their theories, due to the difficulty to find affordable robots that are simple to program, assemble and use. The Lego® MindStorms™ robots present all these features: they are affordable and simple to program; they also allow the implementation of reasoning capabilities without requiring skills about the robot's hardware.

The main component of the Lego® MindStorms™ robot is the **RCX brick** (RCX stands for *Robotic Commander Explorer*). It contains a Hitachi H8/3297 16 bits microprocessor, capable of controlling up to three actuators and three sensors simultaneously. The actuators are motors with possible selection of five rotation speeds for both directions, and the sensors can be light sensors, and touch sensors. The RCX also has an infrared port that can communicate with an *infrared tower*, which can be connected to a desktop's serial port, enabling the communication between the robot and the computer. All of these features can be programmed by the robot designer, e.g., in the language NQC (*not-quite C*), which was used in this work.

3.1 Legolog

Legolog [12] is a software package that includes the IndiGolog meta-interpreter, and the implementation of a communication protocol between the RCX brick and the desktop. This protocol enables the exchange of messages during the execution of a program stored in the RCX brick.

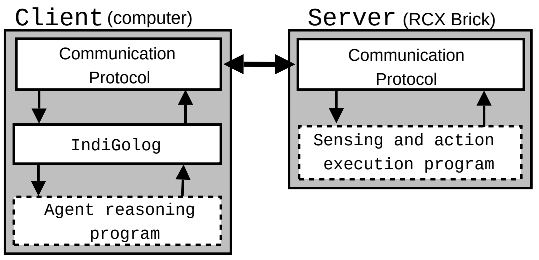

Figure 1. Client/server model for the Legolog package, in which dashed rectangles are defined by the programmer.

Legolog can be used to model client-server applications, in which the RCX is the server of actuators and sensors, and the desktop (executing the IndiGolog meta-interpreter) is the client. In Figure 1 we show how a client/server architecture breaks a Legolog agent in modules, where (1) **Client** is the module executed in the desktop. It contains the IndiGolog meta-interpreter and the *agent reasoning program*; (2) **Server** is the module executed in the RCX brick that contains the *actions execution program*.

Thus, a program to control a robot in Legolog is comprised by two main parts (dashed rectangles in Figure 1): an *agent reasoning program* and an *sensing and action execution program*.

Agent reasoning program. When executed by the IndiGolog meta-interpreter, in the desktop, it performs incremental *on-line* generation of plans composed of primitive actions, taking into account the robot perceptions.

Sensing and action execution program. Implemented in NQC and stored in the RCX. It specifies how primitive actions and perceptions are executed in the robot.

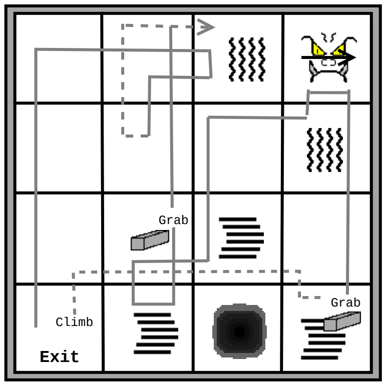

Figure 2. A solved instance of the Wumpus World.

Using this model, the robot designer can define the appropriate level of abstraction of actions delegated to the RCX, i.e. the degree of autonomy of the robot. The RCX can be programmed simply to control inputs and outputs; or it can be programmed to implement more complex actions, e.g. follow a line, find an object, or even to perform a more complex task detailed in Section 4.5.

4 Case Study: A Legolog Agent for the Wumpus World

In [12] an implementation of Legolog was done for a simple task: make the robot to follow a line while it recognizes and reacts to its perceptions (marks over the line). Although for this simple task there are still some challenges on how to program a Legolog agent with such behaviour [12], this task does not involve complex reasoning. In this paper, the main motivation is to propose a more complex testbed domain for CR agents: the Wumpus World, which requires an agent with higher level cognitive actions. The following sections present the main steps of building a Legolog agent for the Wumpus World.

4.1 The Wumpus World: an agent searches for a treasure in a hostile environment

The Wumpus World problem contains an agent that must explore a square grid, having information only about the neighborhood of the square

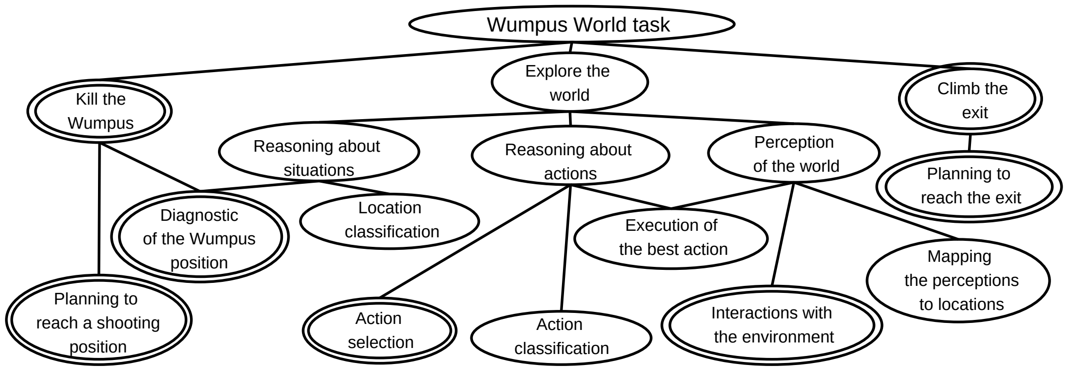
Wumpus World task
Kill the Wumpus
Explore the world
Climb the exit
Reasoning about situations
Reasoning about actions
Perception of the world
Planning to reach the exit
Diagnostic of the Wumpus position
Location classification
Execution of the best action
Mapping the perceptions to locations
Planning to reach a shooting position
Action selection
Action classification
Interactions with the environment

Figure 3. Task decomposition for an agent in the Wumpus World.

in which it is located. The goal of the agent is to explore the grid, which is surrounded by walls, collecting the highest possible score on the way. The agent can increase its score by collecting gold bars, spread in various squares, plus performing the least number of movements. The agent must also avoid pits, and a wandering agent-devouring monster called Wumpus.

In Figure 2 we show a 4×4 instance of the Wumpus World in which the grid has already been successfully explored. The solid line represents the path to explore the environment and the dashed lines represent planned paths to kill the Wumpus or to get out of the grid. Squares containing horizontal straight lines represent *breeze*, and vertical curved lines represent the *smell* of Wumpus. This map characterizes an instance of the problem with high difficulty for the agent, requiring 41 actions to reach the solution state. The arrow indicates that the Wumpus has been killed.

An agent for the Wumpus World has incomplete information about the world, since it can only sense the Wumpus (by sensing its smell) or a pit (by sensing a breeze) when it is in a neighboring square to the Wumpus or a pit. Therefore, the agent must perform a hypothetical reasoning about the world while exploring the environment, in order to classify the squares as safe or dangerous. The agent can also kill the Wumpus using an arrow.

The Wumpus World domain has been used in introductory courses of Artificial Intelligence. Nevertheless, the design, programming and implementation of a complete program for an agent in the Wumpus World can not be found in the literature. In fact, the Wumpus World problem is not completely solved in introductory courses. In [17], for instance, only a few suggestions are presented on how to model the set of tasks that the agent must execute (ovals in Figure 3). Moreover, that book only hints that the logical specification of the solution of the Wumpus World problem can be implemented as a Prolog program. The construction of such program, however, can be quite complicated without resorting to a language such Golog or IndiGolog.

4.2 Logical specification of the agent

The specification in situation calculus of the agent for the Wumpus World was based initially on [17]. Figure 3 shows the task decomposition for the agent implemented in this world. Double ovals represent sub-tasks modelled and implemented specifically in this project, while simple ovals are tasks usually discussed in AI books.

The fluents in the Wumpus World are: **smelly(L)** or **breeze(L)**, which mean that position L is smelly and breezy; **atAgent(L)**, means that the agent is at position L; **agentDirection(D)**, means that D is the current direction of the agent; **visited(L)** and **secure(L)**, which denote that position L has been visited or is safe; and **holding(O)**, denotes that the agent is holding object O.

Using these fluents, one can write a set of axioms in the situation calculus to specify the agent. These axioms are divided into: (1) *initial state axioms*, which describe the initial state of the world; (2) *successor state axioms*, which represent how the fluents are changed or remain unchanged after the actions. For example, the successor state

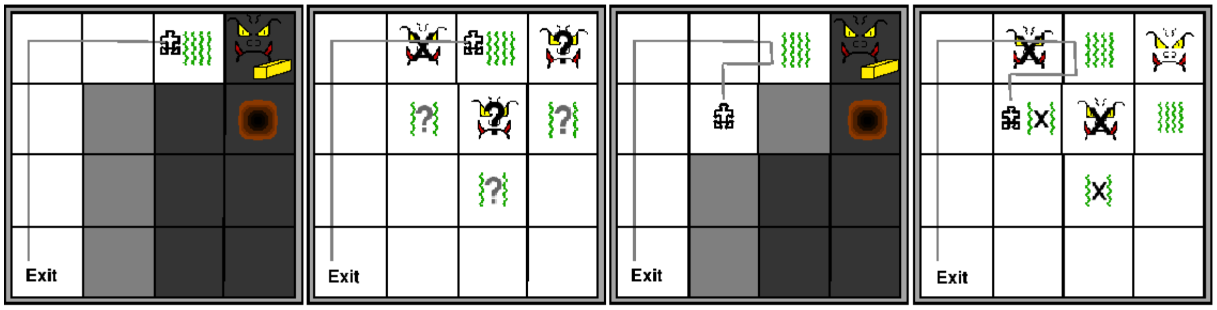

Figure 4. Four maps representing two situations in the Wumpus World. The first and the third maps show what the agent already knows about the environment: white squares represent visited positions; black squares represent not visited positions; and grey squares represent not visited positions which were inferred to be safe. The second map shows the agent's hypothetical reasoning about the first map, and the fourth map shows the same about the third map. Squares labelled with "?" denote hypothesis; squares labelled with "*X*" denote a false hypothesis.

axiom for the fluent **smelly(L)** is:

```
holds(smelly(L), do(A, S)) :-
    holds(smelly(L), S);
      A==forward, stench(do(A, S)),
      holds(atAgent(L), do(A, S)).
```

It defines a position **L** as *smelly* in situation **do(A,S)** if **L** was already *smelly* in situation **S** or if the agent sensed the smell of Wumpus when reaching it due to the action **A**. Similarly, additional axioms must be defined describing the effects of all remaining agent's actions: **turn**, **forward**, **grab**, **shoot**, and **climb**.

4.3 Procedures in IndiGolog: the agent reasoning programming

Besides the above axioms, we must also specify the following compound tasks:

4.3.1 Actions classification

The set of (primitive or composite) actions that can be executed from a given situation are classified as **Great**, **Good**, **Medium**, **Risky** and **Deadly**. This classification is implemented by procedures in IndiGolog and its modification implies in different behaviors of the agent. For instance:

```
proc(greatAction,
  if(holding(gold),
     planning([0,0], [climb | []]),
     [sense(glitter), grab] # [tryToKillWumpus])).
```

This defines an agent whose action with highest priority (**greatAction**) is to plan the way out of the grid if it has already found gold bars. Otherwise, the agent must collect the gold whenever it senses its shine. Finally, if none of the previous actions are possible, the agent must try to kill the Wumpus.

4.3.2 Diagnosis of the Wumpus position

An interesting feature of the Wumpus World problem is that it requires the determination of the Wumpus position based on incomplete information. This task, illustrated in Figure 4, is named diagnosis since the identification of the Wumpus position can *explain* the observations (smelly positions) of the agent in previous situations. The following procedure:

```
proc(tryToKillWumpus,
  [?(holding(arrow)), consultKB(smellyPos(SmellyP)),
   startSet(WumpusP), findWumpus(SmellyP, WumpusP),
     planKillWumpus(WumpusP)]).
```

specifies that in order to kill the Wumpus the agent must consult its knowledge base (**consultKB**) and retrieve a list of positions in which the smell of the Wumpus was sensed. Then the procedure **findWumpus** is triggered to generate and discriminate a list of hypotheses about the possible positions of the Wumpus (**WumpusP**). If the list contains a single position, then the procedure **planKillWumpus(WumpusP)** leads the agent to a plan to reach an adequate position to shoot an arrow and kill the Wumpus. The agent has only one arrow, and should not risk to waste it before knowing for sure where the Wumpus is.

The Golog procedure **findWumpus** below shows the generation and discrimination of hypotheses about the Wumpus position.

```
proc(findWumpus([[SmeX, SmeY] | SmePos], Wpos),
  [abductWumpusAt([SmeX, SmeY+1], south, Wpos),
   abductWumpusAt([SmeX, SmeY-1], north, Wpos),
   abductWumpusAt([SmeX+1, SmeY], west, Wpos),
   abductWumpusAt([SmeX-1, SmeY], east, Wpos),
   if(SmePos = [],
     [cut(findWumpus(SmePos, Wpos))],
     [endSet(Wpos)])]).
```

In this procedure, for each position [**SmeX, SmeY**] in which the smell of the Wumpus was sensed, the four adjacent positions are considered hypotheses of localizations of the Wumpus, since the Wumpus can be sensed only when it is adjacent the agent. The generated hypotheses are then discriminated by the procedure **abductWumpusAt**:

```
proc(
 abductWumpusAt([WumX, WumY], IgnDir, Wpos),
  if(secure([WumX, WumY]), [],
    if(wall([WumX, WumY]), [],
    [possibleWumpusPos([WumX, WumY], IgnDir),
      addToSet(Wpos, [WumX,WumY])]
     # [?(inCave)]))]).
```

Initially, it is checked whether the position [**WumpusX, WumpusY**] has already been classified as safe or wall, otherwise the procedure **possibleWumpusPos** analyzes the three adjacent positions, excluding the position in the already visited direction *IgnDir*. This is done by verifying the hypothesis of the Wumpus being at position [**WumX, WumY**] through the analysis of the past perceptions in order to detect conflicts, i.e., if an adjacent position to the previously position [**WumX, WumY**] was visited and has not been labelled as *smelly*.

4.3.3 Planning to find the way out and the Wumpus

The agent must, in certain situations, plan to reach a goal state crossing only safe positions, without sensing the world (*off-line* planning). This is an example of an agent which adapts its own goals during the performance of a task. In the Wumpus World, this task occurs in three occasions: when the agent decides to (1) **kill the Wumpus**; (2) to **climb the exit**; and (3) **explore a safe position** not in its adjacency. For (1) it must find the closest position in the direction of the Wumpus, and for decision (2) and (3) it must find the shortest path to the exit or safe position respectively.

The planning algorithm was implemented in Prolog and performs an iterative deepening search in the state space [15] when the Prolog query "`plan(S),exec(S),holds(agentAt[Goal_Pos],S)`" is done, where **plan(S)** and **exec(S)** are defined as:

```
exec(s0).
exec(do(A,S)) :- poss(A,S), exec(S).
plan(s0).
plan(do(A,S)) :- plan(S).
```

This algorithm is used with the axioms of the situation calculus, to infer that $(\exists s).plan(s) \wedge exec(s) \wedge G$, in which G is a goal state. It is interesting to notice that this algorithm, despite its simplicity, is the most concise way to find a solution plan s, based on the situation calculus axioms.

4.4 Description of the representation of the Wumpus World

In order to build a physical model, we had to find a way to represent the environment considering the sensors available for the Lego® MindStorms™. In our case, we only used a light sensor. The squares in the grid were identified by tags with different light emitting properties (opaque dark tags, shining silver tags, and so on).

In Figure 5 the dashed lines delimit each position in the Wumpus World, and the solid lines represent the possible paths for the robot to move.

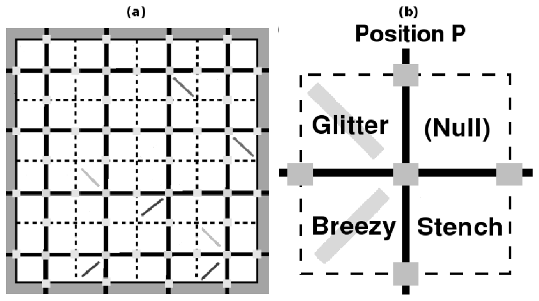

Figure 5. (a) Physical representation of Figure 2 and (b) zoom of the position (4,4).

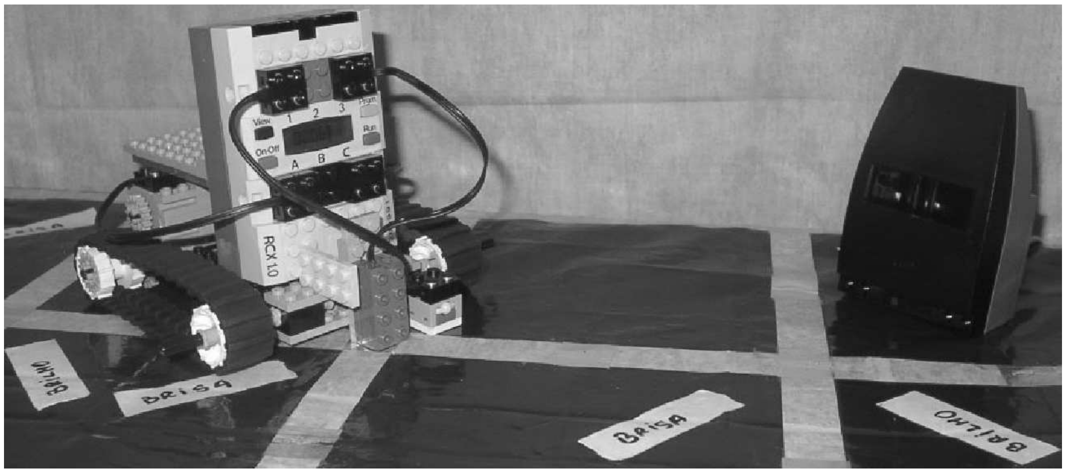

Figure 6. Lego® MindStorms™ robot and the physical model of the Wumpus World.

The **perception tags** were created to represent the three possible perceptions for the Wumpus World: **glitter**, **stench** and **breezy**. All positions in the environment were divided in quadrants. Each quadrant is used to represent one type of perception (Figure 5). Another type of tag is the **rotation tag**, used by the perception algorithm to collect perception in the quadrants in P. A physical model to represent the Wumpus World can be seen in Figure 6.

4.5 Primitive actions for the RCX brick

In the decomposition that implements the Wumpus World in Legolog, the perceptions in a position (**glitter**, **stench** and **breezy**), with no previously specified order, is implemented by the primitive action **percept**. The perception of the agent is therefore performed autonomously by the RCX. In order to sense a position, the robot must perform a 360° rotation collecting all perceptions in all quadrants and then return to the original orientation. Hence, the action **percept** is implemented using four iterations of the function **get_boolean_sense** (Table 2), returning a vector containing four *booleans* to the computer, encoded as an integer.

The robot can reach a position coming from four alternative directions (north, south, east or west), hence the reasoning program of the agent uses the present orientation of the robot to determine the amount of circular *shifts* that must be applied to the vector to recover the correct perception.

```
void(get_boolean_sense) {
  int light_dif;
  //Restarting global variable
  boolean_sense = FALSE;
  //Rotating till coming out of the rotation tag
  turnToExitMark(bg_dif_min, bg_dif_max, 1);
  light_dif = light_sensor - base_value;
  //Checking whether agent found the perception
  if(light_dif >= sense_dif_min &&
    light_dif <= sense_dif_max) {
      //Perception found
      boolean_sense = TRUE;
      PlaySound(SOUND_UP);
      //Leaving perception tag
      turnToFindMark(bg_dif_min, bg_dif_max, 1);
      //Looking for guide line or rotation tag
      turnToFindMark(line_dif_min, mark_dif_max, 1);
  } //If tag not found, the robot is already on line
}
```

Table 2. NQC program to perform a 90° rotation looking for perceptions.

The following primitive actions were also implemented in the RCX block besides the action **percept**: **turn_clockwise**, **turn_anti_clockwise**, **forward**, **grab**, **shoot** and **climb**. From these, only the first three can move the robot; i.e., turning 90° clockwise, counter-clockwise, or advancing one square, respectively. For the other actions (**grab**, **shoot** and **climb**), the robot emits different sounds indicating that they have been executed and waiting for a manual update of the environment, that is: grab the gold, remove the Wumpus, and remove the agent from the exit position.

Instance number	1	2	3
Wumpus position	[3,3]	[0,2]	[3,1]
Coin position	[1,1]	[1,2]	[3,0]
Pits positions	[2,0]	[3,1], [3,3]	
Number of steps of the executed plan	22	25	18
Average time for *on-line* action selection	0.03s	0.42s	0.05s
Standard deviation of time for *on-line* action selection	0.01s	1.68s	0.03s
Average time for plan generation (*off-line*)	0.03s	1.35s	0.13s
Standard deviation time for plan generation (*off-line*)	0.01s	1.30s	0.10s
Total time spent by the agent reasoning program	0.38s	13.25s	9.16s

Table 3. Statistic results for 3 instances of the Wumpus World with 1 coin position and 0.2 **probability of pits.**

4.6 Evaluating the designed agent

One of the goals in CR is to prove that the designed agent has the expected behavior. Since Golog is complete, i.e. it searches the entire plan space, we only have to show that the agent high-level program prunes the undesirable plans and not a candidate solution. In proposition 1 we prove that our agent does not die in the cave.

Proposition 1. The Golog agent for the Wumpus World does not die while it is in the cave.

Proof. Since our agent only moves to positions labelled as secure, it is enough to prove that this label mechanism is correct. A position is defined as secure through the following situation calculus axiom:

```
holds(secure(X), do(A, S)) :-
1 | holds(secure(X), S) ; (
2 |   percept([Ste, Bre, _, _, _], S),
3 |   holds(atAgent([Ax, Ay]), S), X=[Px, Py],
4 |   Ste==none , Bre==none, (
5 |    (Px=Ax, (Py is Ay+1 ; Py is Ay-1)) ;
6 |    (Py=Ay, (Px is Ax+1 ; Px is Ax-1)))).
```

Due to the static environment, the recurrence in line 1 is correct and its base is the initial position which is secure by the definition of the Wumpus World domain. Lines 2 and 3 only collect information about the current state and lines 4, 5, 6 define that the position X is secure if the agent did not percept the smell of Wumpus nor breeze of the pit and X is adjacent to the agent's position in situation S. Since each hostile entity for an agent in this domain, i.e. the Wumpus and the pits, produces perceptions in all positions of their adjacency, this definition is correct. Therefore there is no risk for the agent to move into positions labelled as secure. Because of the negation as failure implemented in Prolog, while there is no evidence that a position is safe, it will not be labelled as secure, however, just one evidence is sufficient to conclude that it is secure. □

Other important feature of this agent is the optimality (in terms of numbers of `forward` actions in the plan) when moving through secure positions to: (1) exit the cave; (2) kill the Wumpus; and (3) explore the nearest secure position. This feature is due to the *off-line* optimal planning procedure `plan` that searches only the subset of the state space in which all positions are safe. As commented in Section 4.3.3, this procedure performs an iterative deepening search.

5 Experimental Results

Table 3 shows three instances of the Wumpus problem, with the Wumpus position, coin position and pits positions specified in lines 2, 3 and 4, respectively. Those experiments require reasoning about incomplete information. In all experiments the robot has correctly detected safe and dangerous positions and also found optimal planning solutions to determine the approximation of the Wumpus and the exit. The experiments show that for a medium size plan (22, 25 and 18 steps), the average time spent for *on-line* action selection was less then 0.5 seconds, while the average time for *off-line* planning was less then 1.4 seconds.

The times on Table 3 can be considered low since they are the results of the actual on-line execution of Golog plans by a real robot, i.e., a robot capable of handling low-level actions sent by the high-level controller.

6 Other solutions for the Wumpus World

Traditionally, it is a hard task to compare different formalisms for reasoning about actions, specially with incomplete information. However, this is an important activity that can help one to identify the strengths and weakness of the various approaches. When this comparison is based on a testbed application, it is possible to make some comparisons, nevertheless it is still very difficult to establish good metrics. For instance, the score of the Wumpus World agent can be used has a metric but: (i) two different agents should be compared on the same set of problem instances, and (ii) the time spent to solve the same problems should be measured accordingly. Another difficulty about comparisons issues is with respect to the attempt to justify a better performance. That is because the implementation of a cognitive robotic agent has always a domain dependent knowledge layer which causes its performance to rely in part on the expertise of the knowledge designers and in part on the logical formalism adopted which should be the main research interest of the non monotonic community.

Next, we make some brief discussion about other works that have recently tried to solve the Wumpus World problem. A planning approach for the Wumpus World was made by Blai Bonnet in [6] and consists on modelling the problem by a partial-observable Markov decision process (POMDP) [5] with deterministic actions. This approach only reasons about sequences of actions, i.e., the solution is completely deliberated before executing any action. As a result, the planner agent has a performance worse than others dedicated solutions.

The work described in [18] also gives a solution using IndiGolog. In this implementation, they proposed some modifications in the IndiGolog to deal with incomplete information. This is done by extending the ontology of the situation calculus as follow: fluents can have *possible values* at a situation; *known* is a new predicate to say that a fluent has only one possible value at a specific situation.

Another logic-based agent implementation for the Wumpus World is given in [22]. This approach uses FLUX, a constraint logic programming method for the design of agents based on fluent calculus [21]. The fluent calculus is an extension to the situation calculus in which *states* are represented by the set of fluents that are true in it. This feature combined with constraint programming makes FLUX an interesting and efficient framework for cognitive robotics. By doing experiments with the FLUX agent we have verified that the problem instance illustrated in Figure 4 can not be completely solved, i.e. the agent does not kill the Wumpus to grab the gold.[1] However, in [22] the authors suggest some future improvements to the agent that seems to solve this problem.

Finally, the implementation proposed in [20], uses the SNePS framework [19] and different from the others, it explicitly aims to model general human-level intelligence instead of maximize the use of computing power to optimize problem solving. For instance, they have a implemented an agent for the Wumpus World that has mental states, such as bored.

7 Conclusions

In cognitive robotics is essential: (1) to show how it is possible to interleave high-level procedures execution with low-level robot sensing and actions execution; and (2) to have testbed domains to show how a logic-based agent can be used for high-level control and facilitate comparisons between different approaches.

In this paper we show how to implement the high-level behavior required by an agent in the Wumpus World using *Legolog*, a Golog dialect which contains a communication protocol for a real robot: the Lego® MindStorms™ robot. The main goal of the experiments was to verify the interleaving of on-line execution and the Golog logical projection. Besides that, this paper has the aim of motivating the CR community to use the Wumpus World as a challenging testbed domain for comparisons between logic-based robotic approaches.

Acknowledgments

This work has been supported by *Fundação de Amparo a Pesquisa do Estado de São Paulo* (FAPESP), grant number 03/08311-3, and by Conselho Nacional de Pesquisa (CNPq), grant number 308530/03-9.

[1]We used the implementation available at `http://www.fluxagent.org/demos.htm`

References

[1] *Sixth Workshop on Nonmonotonic Reasoning, Action, and Change*, Edinburgh, UK, August 2005.

[2] *International Workshop on Non-Monotonic Reasoning*, Lakes District, England, June 2006.

[3] L. N. Barros and E. Iamamoto. Planejamento de tarefas em golog. In *SBAI*, 2003.

[4] B. Bonet and H. Geffner. HSP: Heuristic Search Planner. In *Proc. of AIPS.*, 1998.

[5] B. Bonet and H. Geffner. Planning with incomplete information as heuristic search in belief space. In S. Chien, S. Kambhampati, and C. Knoblock, editors, *Proc. 6th International Conf. on Artificial Intelligence Planning and Scheduling*, pages 52–61, Breckenridge, CO, 2000. AAAI Press.

[6] B. Bonet and H. Geffner. GPT: Planning with Uncertainty and Partial Information, 2002. http://www.tecn.upf.es/∼hpalacio/model2/dos/slides2.pdf (visited in march 2006).

[7] C. Boutilier, R. Reiter, M. Soutchanski, and S. Thrun. Decision-theoretic, high-level agent programming in the situation calculus. In *Workshop on Decision-Theoretic Planning, Proc. KR-00*, Apr 2000.

[8] K. Erol, J. A. Hendler, and D. S. Nau. UMCP: A sound and complete procedure for hierarchical task-network planning. In *Proc. AIPS*, pages 249–254, 1994.

[9] J. Hoffmann and B. Nebel. The FF planning system: Fast plan generation through heuristic search. *J. of Artificial Intelligence Research*, 14:253–302, 2001.

[10] S. Kambhampati, C. A. Knoblock, and Q. Yang. Planning as refinement search: A unified framework for evaluating design tradeoffs in partial-order planning. *Artificial Intelligence*, 76:167–238, 1995.

[11] Y. Lespérance and H. Ng. Integrating planning into reactive high-level robot programs. In *Proc. of the 2nd International Cognitive Robotics Workshop*, August 2000.

[12] H. Levesque and M. Pagnucco. Legolog: Inexpensive experiments in cognitive robotics. In *Proc. of the 2nd International Cognitive Robotics Workshop*, August 2000.

[13] H. Levesque, R. Reiter, Y. Lespérance, F. Lin, and R. Scherl. GOLOG: A logic programming language for dynamic domains. *JLP*, 31:59–84, 1997.

[14] J. Mccarthy. *Situations, actions and causal laws.* MIT Press, 1963.

[15] S. L. Pereira and L. N. Barros. Formalizing planning algorithms: a logical framework for the research on extending the classical planning approach. In *Proc. of the ICAPS Workshop: Connecting Planning Theory with Practice*, 2004.

[16] R. Reiter. Sequential, temporal golog. In *Principles of Knowledge Representation and Reasoning: Proc. KR-98*, pages 547–556, 1998.

[17] S. Russel and P. Norvig. *Artificial Intelligence: A Modern Approach.* Prentice-Hall, Inc., 2nd. edition, 2003.

[18] S. Sardina and S. Vassos. The Wumpus World in IndiGolog: A preliminary report. In *Workshop on Nonmonotonic Reasoning, Action, and Change (NRAC'05), IJCAI*, 2005.

[19] S. C. Shapiro. The cassie projects: An approach to natural language compentence. In J. P. Martins and E. M. Morgado, editors, *EPIA 89: Proc. of the 4th Portuguese Conference on Artificial Intelligence*, pages 362–380. Springer, Berlin, Heidelberg, 1989.

[20] S. C. Shapiro and M. Kandefer. A SNePS Approach to The Wumpus World Agent or Cassie Meets the Wumpus. In *Workshop on Nonmonotonic Reasoning, Action, and Change (NRAC'05), IJCAI*, 2005.

[21] M. Thielscher. The fluent calculus: A specification language for robots with sensors in nondeterministic. Technical Report CL-2000-01, Artificial Intelligence Institute, Department of Computer Science, Dresden University of Technology, 2000.

[22] M. Thielscher. A FLUX agent for the Wumpus World. In *Workshop on Nonmonotonic Reasoning, Action, and Change (NRAC'05), IJCAI*, 2005.

[23] F. W. Trevizan, L. N. Barros, and F. S. Correa da Silva. Low cost experiments in Cognitive Robotics for planning in hostile environments with incomplete information. In *Proceedings of the XI Conference of the Spanish Association for Artificial Intelligence (CAEPIA)*, Santiago de Compostela, Galicia, 2005.

[24] F. W. Trevizan and L. N. de Barros. Robótica cognitiva. Relatório técnico de cniciação científica FAPESP 308530/03-9-1, Universidade de São Paulo, Departamento de Ciência da Computação, Março 2004.

[25] F. W. Trevizan and L. N. de Barros. Robótica cognitiva. Relatório técnico de cniciação científica FAPESP 308530/03-9-2, Universidade de São Paulo, Departamento de Ciência da Computação, Setembro 2004.