$\exists a \exists s[Snapped(IronMan, Result(a, s))] \implies Report$

Hesham Morgan; Tarek Badreldin; Omar Osama
Team 2
German University in Cairo - Introduction to Artificial Intelligence (CSEN 901)

Abstract

The logical agent reasons about the world given the facts and rules in the knowledge base. In this report, the logical agent based on *prolog* inference rules and successor axioms is observed. The discussion is based on a simple problem to show the capabilities of such a system.

1 Introduction

Description The problem comprises of some 2D grid where some collectables are placed in cells in the grid, A cell is a position in the grid. The agent starts in a given cell initially and tries to reach some goal. Moreover, the agent can move in four directions relative to its position, each move transitions the agent from the cell it was in to an adjacent one. An adjacent cell is a cell that is directly next to the cell in either of the following directions, $directions = \{up, right, down, left\}$. The agent can move in any of the allowed directions as the action of some state, a state is the description of the world at a specific time step. A time step is an indicator which refers to some state of the world based on some action done in some previous state. An action is the operation done by the agent to change some aspect of the world as defined by the effects of the action if and only if the preconditions of the actions are satisfied in the give state. In the environment there is one type of enemy, which is the villain of the problem stated. The goal of the agent is to collect all the collectables to eliminate the villain of the problem. All of the objects in the environment are stationary and cannot move from their initial position. The villain can be eliminated if all the stones are collected and the agent is in the villains cell with damage units less than *one hundred*. Therefore the actions the agent can perform are $actions = \{up, right, down, left, collect, end\}$, where end is the action discussed to eliminate the villain of the problem.

Specification The problem is themed after the Marvel Cinematic Universe latest movies, Infinity Wars & End Game. Therefore more specifications to the described problem will be explained for the context which would be discussed in later sections. After the event of Infinity Wars, Iron Man, the protagonist of the story, goes to freeze Thanos, the villain of the story who eliminated half of the civilization on earth. Iron Man then goes on a journey to retrieves all of the infinity stones scattered in the universe to be able to defeat Thanos. There are *four stones* scattered throughout the universe, which Iron Man must collect. After Iron Man attain the *four stones*, Iron Man goes to Thanos, unfreezes him, and eliminate him by snapping his fingers while wearing the *Infinity gauntlet*, which attains him a wish while saying ***"I am Iron Man"*** and sacrificing himself to save humanity.

Motivation A logical agent reasons about the world to reach a solution to the problem. A solution is a set of situations that describe the agent's actions from the initial state. The problem defined was thought as a practical approach to increase the understanding of a simple logical system based on successor state axioms introduced in the lectures.

2 Methodology

In this section the semantics of the action terms and predicate symbols for the problem stated is presented. The aim of the agent is to reason about the world to reach the goal state. The agent generates all the possible outcomes based on the successor axioms and check for their correctness till reaching the goal.

Problem Definition Each predicate that is used to define this problem, defines a different aspect of the world. The predicates used to define the $EndGame$ problem are defines as follows

Predicate. $grid(m, n)$ is the representation of the grid dimensions of the problem where,

- m is the number of rows in the grid,
- n is the number of columns in the grid,

The grid dimensions is an important aspect to the problem that should be represented for the problem specifications for the agent's reasoning.

Predicate. $ironman(x, y, s)$ is the representation of the agent's location in the world where,

- x is the location of the agent in the grid in respect to the rows,
- y is the location of the agent in the grid in respect to the columns,
- s is a set of situations which led the agent to that location or action.

The reasoning of the agent is represented as the sequence of the action which led the agent to state that agent is at in a given situation.

Predicate. $thanos(x, y)$ is the representation of the villain's location in the grid where,

- x is the location of the villain in the grid in respect to the rows,
- y is the location of the villain in the grid in respect to the columns.

The villain's location for this problem is an essential information for the agent's reasoning, as it represents the the goal state under some conditions.

Predicate. $stone(x, y, s)$ is the representation of the stones scattered in the world where,

- x is the location of the stone in the grid in respect to the rows,
- y is the location of the stone in the grid in respect to the columns,
- s is a set of situations which the stone is existing at.

The stones represent the sub goals the agent must aim for to reach the situation that contains the solution.

Action An action is the operation done by the agent to change some aspect of the world as defined by the effects of the action if and only if the preconditions of the actions are satisfied in the given state. In the problem defined the action set $\{up, right, down, left, collect, kill, snap\}$ are as follows.

- $\forall x, y, s[[ironman(x, y, s) \land x > 0] \iff [ironman(x_{shifted}, y, Result(up, s)) \land (x_{shifted} = x - 1)]]$
- $\forall x, y, s[[ironman(x, y, s) \land grid(m, n) \land x < m] \iff [ironman(x_{shifted}, y, Result(down, s)) \land (x_{shifted} = x + 1)]]$
- $\forall x, y, s[[ironman(x, y, s) \land y > 0] \iff [ironman(x, y_{shifted}, Result(left, s)) \land (y_{shifted} = y - 1)]]$
- $\forall x, y, s[[ironman(x, y, s) \land grid(m, n) \land y < n] \iff [ironman(x, y_{shifted}, Result(left, s)) \land (y_{shifted} = y + 1)]]$
- $\forall x, y, s[[ironman(x, y, s) \land stone(x, y, s)] \iff [ironman(x, y, Result(collect, s)) \land \neg stone(x, y, Result(collect, s))]]$
- $\forall x, y, s[[ironman(x, y, s) \land thanos(x, y) \land \forall i, j[\neg stone(i, j, s)]] \iff [ironman(x, y, Result(snap, s))]]$

The actions listed are not accurate as there are some constraints that should be applied, but would be confusing, so a reduced model was defined. For the movement actions, the move action allows Iron Man to transition to the cell adjacent to the one he is in if it was not out of the grid border. For simplification of the movement axiom in this section, the condition for not being able to move to $Thanos$' cell till all stones are collected was not mentioned. Meanwhile, the collect action allows Iron Man to collect a stone if it existed in the state s. As for snap, the action is applicable if Iron Man collected all the stones in the grid at situation s, and Iron Man was in $Thanos$' cell at situation s. As a result, Iron Man would snaps which indicates a goal state for the problem.

Predicate A predicate describes the state of the world at some state described by the situations that occurred in the world, the successor state axioms for this problem are defined as follows.

- $\forall x, y, a, s[ironman(x, y, Result(a, s)) \iff (a = up \land ironman(x_{original}, y, s) \land (x_{original} = x + 1)) \lor (a = down \land ironman(x_{original}, y, s) \land (x_{original} = x - 1)) \lor (a = left \land ironman(x, y_{original}, s) \land (y_{original} = y + 1)) \lor (a = right \land ironman(x, y_{original}, s) \land (y_{original} = y - 1)) \lor [ironman(x, y, s) \land ((a = up \implies (x = 0)) \lor (a = down \implies (grid(m, n) \land x = m)) \lor (a = left \implies (y = 0)) \lor (a = right \implies (grid(m, n) \land y = n)) \lor (a = collect \implies stone(x, y, s)) \lor (a = snap \implies (\neg stone(x, y, s) \land thanos(x, y))))]]$
- $\forall x, y, a, s[stone(x, y, Result(a, s)) \iff \exists i, j[a = up \land ironman(i, j, s)] \lor \exists i, j[a = down \land ironman(i, j, s)] \lor \exists i, j[a = right \land ironman(i, j, s)] \lor \exists i, j[a = left \land ironman(i, j, s)] \lor [(a = collect \implies \neg ironman(x, y, s))]]$

The $ironman$ successor axiom state that the agent would be in some cell at a given situation if the agent either transitioned to that cell through a movement action or it was at the borders of the world grid, so the agent cannot surpass the grid borders. As for $stone$ successor axiom state a stone exists in the world at some given situation if it was not collected by the agent.

3 Implementation

In this section, the implementation of the logical agent and the encoding of the problem are presented. The implementation is presented in the intuitive way of constructing the logical agent. The program is divided into 2 sections, the first being the knowledge base rules representation generator, secondly being the logical agent implemented in *Prolog*. Note that the code shown in this section is for explanation purposes, and some of the structure might be different.

Knowledge Base The knowledge base represents the facts and rules of that represents the world defined by the problem. In this problem the program takes *"M, N; Ix, Iy; Tx, Ty; Sx1, Sy1, Sx2, Sy2, Sx3, Sy3, Sx4, Sy4"* as an input, where M and N stand for the rows and columns, I stands for Iron Man, T stands for Thanos, and S stands for stones. A *Java* program was written that takes the input string of the grid configuration, and converts it into *Prolog*'s facts form. The *Java* program splits on semicolons to get the different sections from the input, then splits the stones sections on commas to get the information of each stone. The information extracted from the input string is converted into an output file in the following form.

- $grid(M, N)$. is the dimensions of the grid,
- $thanos(X, Y)$. is the representation of Thanos' location in the grid,
- $ironman(X, Y, R, S)$ is the representation of the Iron Man's location in the grid.

In contrary to the discussed model of problem in the ***Problem Definitions Paragraph 2***, a different approach was taken from the implementation. The implementation defines the stones existing at some given state in the world are represented in a list R in the $ironman$ predicate, where each stone in the list R is of the form $stone(X, Y)$ to represent the stone location; further clarification is discussed in ***Agent Paragraph 3***.

Agent The agent take the facts in the knowledge base, and reasons about the world with the rules defined for the problem. The $Endgame$ problem defined can be defined in *Prolog* by the following axiom.

Listing 1: Iron Man Axiom

```
ironman(X1,Y1,R1,result(A,S)):-
    ironman(X,Y,R,S),
    grid(M,N),
    (
        (
            (
                (A = collect, member(stone(X,Y), R), delete(R, stone(X,Y), R1), X1 is X, Y1 is Y);
                (
                    (A = left, Y > 0, Y1 is Y - 1, X1 is X, R1 = R);
                    (A = right, X1 is X, Y1 is Y + 1, Y1 < N, R1 = R);
                    (A = up, X > 0, X1 is X - 1, Y1 is Y, R1 = R);
                    (A = down, Y1 is Y, X1 is X + 1, X1 < M, R1 = R)
                )
            ),
            \+ thanos(X1,Y1)
        );
        (
            (
                (A = snap, thanos(X,Y), R1 = R);
                (
                    (A = left, Y > 0, Y1 is Y - 1, X1 is X, R1 = R);
                    (A = right, X1 is X, Y1 is Y + 1, Y1 < N, R1 = R);
                    (A = up, X > 0, X1 is X - 1, Y1 is Y, R1 = R);
                    (A = down, Y1 is Y, X1 is X + 1, X1 < M, R1 = R)
                )
            ),
            length(R,0)
        )
    ).
```

Initially, the $EndGame$ sub goal is to collect all stones to be able to reach a solution. In that state, Iron Man cannot enter *Thanos*' cell. Therefore the rule is be divided into two sections, the first being before collecting all the stones, and the second being after collecting all the stones in the grid and trying to reach *Thanos*' cell to execute the snap action and reach a solution. *Prolog* operates on a generate and test paradigm, where all possible permutations are generated then their validity is checked. *Prolog* also operates in a *DFS* manner in the generating and testing process, hence the predicate recurses to the base state first before checking for the action in line 2. The movement clauses are common in both sections, where the movement actions are implemented as discussed in ***Actions Paragraph 2***. The movement actions defines the new location of Iron Man in $X1$ and $Y1$ based on Iron Man's location in the previous state. The first section of the code defines the allowed actions before collecting all stones in the grid, where the agent can either collect a stone or move to an adjacent cell, where *Thanos* is not in. The collect clause checks if there was a stone in the stone array in the given situation that is in the same cell as Iron Man, if that was satisfied then the collect action is applicable, and the stone is removed from the array. Implementing the destructive array for the stones in the implementation improved the running time as it needed less checks as the sequence goes on. A destructive array is one that removes the facts that are elements of it when the elements are false after the execution of some action. As for the second section, for this clause to be satisfied all stones in the world must have been collected as stated by checking on the stone array's length. So the agent can move to any cell in the grid, and if the agent was at some situation at *Thanos*' cell then the snap action is applicable.

Listing 2: Query

```
snapped(S):-
    S = result(snap,_),
    ironman(_,_,_,S).

query_with_depth(S,N,L):-
    (
    call_with_depth_limit(snapped(S),N,L),
    \+ L = depth_limit_exceeded
    );
    (
    F is N+1,
    query_with_depth(S,F,L)
    ).


query(S):-
    query_with_depth(S,0,_).
```

The goal is queried on in the $snapped$ predicate, where the situation wanted is the one that has the action *snap* at the head of the situational statement in the $ironman$ predicate. However as *Prolog* operates in a *DFS* manner, an *IDS* approach was sought after in the $query$ predicate where a limit to the depth of each iteration is defined and if the query did not reach a solution that was sought then the depth is incremented. The reasoning behind that is evident if the second line in the Iron Man axiom was moved to the end of the program, as that would lead for the *Prolog*'s interpreter to generate the first action infinitely before testing for its validity. Therefore an *IDS* approach for the query would be required so it would check for the validity of each generated sequence at some cutoff depth. However as that problem is handled by returning to the base state for each sequence to check for validity, this approach is not the best as its slower than *DFS* as the combinations to a solutions in the restrictions applied to the problem are finite.

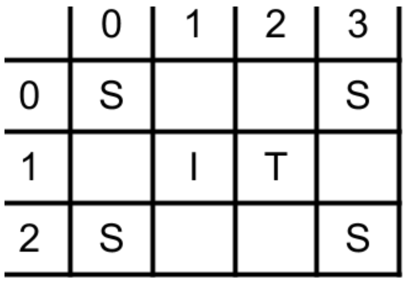
4 Evaluation

In this section two running examples are shown to present the form of the solution resulted from the agent's reasoning on the problem. The limitations put on the problem are discussed from the observations done throughout the implementation of the agent.

Results In this paragraph, the two examples for the $EndGame$ problem are shown. First, the specifications of the machine used for testing is laid out. The computer used for testing had Intel Core i7-8565U at 1.8GHz / 4.6GHz (Base / Turbo), 16 GB LPDDR3 RAM, and the results were as following.

Input Grid : 3,4; 1,1; 1,2; 0,0, 2,0, 0,3, 2,3

Output :

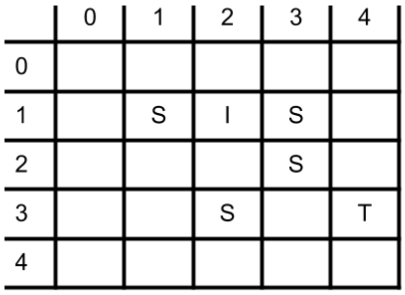


	0	1	2	3
0	S			S
1		I	T	
2	S			S

Figure 1: **3x4 Grid**- 4 Stones

```
S = result(snap, result(down, result(left, result(collect, result(↩
  up, result(up, result(collect, result(right, result(right, result(↩
  right, result(collect, result(down, result(down, result(collect,↩
   result(up, result(left, s0))))))))))))))))
```

Input Grid : 5,5; 1,2; 3,4; 1,1, 1,3, 3,2, 2,3



	0	1	2	3	4
0					
1		S	I	S	
2				S	
3			S		T
4					

Figure 2: **5x5 Grid**- 4 Stones

Output :

```
S = result(snap, result(right, result(right, result(collect, result↩
  (down, result(left, result(collect, result(down, result(collect,↩
   result(right, result(right, result(collect, result(left, s0)))))))↩
  ))))))
```

Discussion The limitations that are put on the problem are a result from *Prolog*'s generate and test paradigm, for which tries to generate all possible permutations of the sequence which is NP-Complete and would require large amount of computational time. Therefore the relaxation states in the problem defined that decreased the amount of stones in the grid and decreased the grid size was required for testing. Furthermore, in the 5x5 grids, if the four stones were placed in the grid's corners, it would require a lot of computational time which is why such a run was not shown in the examples presented in this section.