A Formal And Executable Model For Path Finding

N. Maragos, D.N. Kleftouris, C. Ziogou

Dept of Information Technology
Technological Educational Institute of Thessaloniki
Thessaloniki 546 06, Greece
Email : { nmarag, klefturi, ziochr }@it.teithe.gr

Abstract:
One of the topics of research in the area of robotics is path finding i.e. to determine the discrete robot's motion steps towards the accomplishment of a particular goal. The objective of the current project is to investigate the behavior and the operation of a hypothetical robot using the formal methodology of Coloured Petri Nets (CP-Nets). To this purpose a CP-Net model of the way the robot moves in a presumptive environment where a pilot-scale labyrinth exists is constructed. In this environment, the hypothetical robot is able to move, to change its direction, to avoid obstacles until it finds a preset goal. Results of the CP-Net model execution show how the prediction of robot's movements in its attempt to reach the goal in the environment with obstacles is performed and reveal a number of problems that are not intuitively obvious from the structure of the model.

1 Introduction

Formal methods have been proven to be a powerful tool for studying and verifying discrete event systems, such as robot movement planning. The main advantage of using formal methods in modeling systems is that they result in concrete specifications suitable to computer-aided verification.

Coloured Petri Nets (CP-Nets)[8] is a formal technique for the specification, design, simulation and verification of discrete event systems. CP-Nets are a combination of Petri-Nets used to model systems and a programming language used to create parameterizable characterists of the models.

In daily life we come across to problems, in which we want to find a sequence of actions, which, from an initial state can lead us to a certain goal. Such problems are called planning problems. The sequence of actions which provides us with a solution to our problem is called a plan. Such problems are highly complicated and usually require much power to be computed.

A planning problem can be defined with the following three descriptions :

- The description of the initial world. This description may not be complete.
- The description of the goals that have to be achieved.
- The description of available actions, that must be accomplished in order to achieve the goals.

The problem's solution results in a set of actions, which can be applied to strike the goals. The model mostly used for the definition of planning problems is the STRIPS (Stanford Research Institute Planning System) model[10]. The principles established in the STRIPS model are adopted in our work as well.

Robot activity planning deals with obtaining the necessary steps to reach to a defined goal. To achieve this a plan must be created to find the appropriate path between a start and a destination in the environment. The plan, which derives from the planning process, will be the adequate sequence of steps and actions that will allow the accomplishment of the task. Depending on the goal, robot planning can be interpreted and applied with different ways. In general, a plan must be considered as a set of particular features of the robot movement that are performed according to some restrictions avoiding any possible collision with obstacles and barriers in the environment.

A quite adequate option to achieve the plan is the use of artificial intelligence techniques (AI)[5]. In such a technique the main elements of the system are states and actions that are used in order to solve a general problem. Various systems are based on the concept of states and actions such as STRIPS systems.

In most robot motion planning methods, the input is a low-level definition of robot's movements and the solution is related to the robot and the local environment features. However this scheme is not adopted in real environments. When the actual working environment is involved a high-level definition is required. This means that the development of the plan should be dependent not only on the robot and the local environment features but also on the factory's global environment[4]. When this happens the whole planning process is enhanced with re-planning, decomposition and prediction. In the current project we assume that the robot plan is dependent only on on local environment.

In this work we apply CP-Nets and the supporting Design/CPN[6, 7, 9] computer tool to construct an executable model for finding the movements of a robot in searching a goal in a pilot-scaled labyrinth. The contribution is a concise modeling approach towards developing a formal robot path finding specification which executes the actions that the robot can do while walking. Then the determined path may consist a plan for the robot to follow in reaching the goal. The remainder of the paper is organized as follows. The next section reviews in brief the CP-Nets, their operation and the capabilities of the integrated software tool Design/CPN. In section 3 the pilot scale environment is described, a formal definition of the robot's states is given as well as an algorithmic definition of robot's actions. In section 4 the CP-Net model of robot's path finding is constructed. In section 5 the CP-Net model is executed to prove its validity on one hand and to produce experimental results on the other. The last section mainly refers to further directions for research.

2 Coloured Petri Nets

Coloured Petri nets (CP-Nets), is a graphical oriented language for design, simulation, and analysis of discrete event systems like communication protocols, distributed, concurrent and parallel systems. CP-Nets carry tokens from a given color-type and hence tokens are distinguished from each other. CP-Nets consist of two different kinds of nodes, places and transitions, and arcs. Each one of these components has a number of properties. Places have name, color type and initial marking. The name of a place can be a string, the colour type shows the kind of tokens the place can store and the initial marking shows the number of tokens of the same colour the place has at the beginning of the execution. Arcs have on them variables or constants of the colour type they carry. Finally transitions have a name and guards. A guard is an expression that must be evaluated to true for a transition to occur.

The formal definition of a Coloured Petri Net is the following:

$N = [P, T, F, C, G, V, m_0]$ is a CP-Net iff

1. [P, T, F] is a Petri net.
2. C is a function from $P \cup T$ into nonempty sets of colours.
3. G is a guard function defined from T into conditional expressions of type C.
4. V is a function which associates with each arc (p, t) or (t, p) in F a function from C(t) to multisets of C(p).
5. m_0 is a function to each $p \in P$ a multiset on C(p). It represents the initial marking of the net.

Places can be joined to fusion places. Fusion places marked with a FG tag, create more instances of the same place, and all these different places work as one. When a token is removed from one of them, this token will be removed from all. Fusion places connect them in different pages and contribute to a more readable model, with fewer arcs. Another useful facility of CP-Nets is the substitution transitions. These transitions marked with a HS tag, represent a compound action, and have an associated CP-Net, which models this action in detail. When one such transition is to occur, the run flow goes into the model it represents, and occurs when the model is entirely executed. They are like the functions in programming languages.

Arcs can only connect nodes of different kind, and take the colour type of the place they reach. The basic idea in Coloured Petri Nets is that when a transition occurs, tokens from its input places are removed and new ones are added to its output places according to corresponding arc expressions. A transition occurs or

fires only when all its input arcs have at least one token carried on them, and the guard, if exists, evaluates to true.

Design/CPN is an interactive computer tool to build and execute CP-Nets. It requires X-Windows or Linux platforms, and it can provide us with an editor to build our models, a simulator to execute them and, other facilities for model checking and validation.

3 Robot Path Finding

In our system we have a robot that moves in a discretized space where at fixed locations there are obstacles. The robot's mission is to avoid the obstacles and when it reaches the goal, to stop "walking". The robot "knows" its current position, its current direction and is able to "see" only the place that is next to it, in the same direction of course.

The form of data that constitute robot's initial and other states, defined in a formal manner are the following [10]:
Robot_at(x,y) where x,y are integers and denote coordinates
Direction(D) where D is one of the four directions (North, South, East, West)
Choice(D) where D is one of the four directions
Obstacle_at(x,y) where x,y are integer numbers
Goal_at(x,y) where x,y are integer numbers

To complete the description of the robot's "world", the actions that it is able to perform should be defined using the directives that describe its state. These actions are defined in an algorithmic way and are the following:

```
detect_goal:
if robot_at(x,y) and goal_at(x,y)
then stop.
move_west:
if robot_at(x,y) and direction(W)
then robot_at(x-1, y).
move_east:
if robot_at(x,y) and direction(E)
then robot_at(x+1, y).
move_south:
if robot_at(x,y) and direction(S)
then robot_at(x, y-1).
move_north:
if robot_at(x,y) and direction(N)
then robot_at(x, y+1).
avoid_obstacle_west:
if robot_at(x,y) and direction(W) and obstacle(x+1,y)
then choice(D).
avoid_obstacle_east:
if robot_at(x,y) and direction(E) and obstacle(x-1,y)
then choice(D).
avoid_obstacle_north:
if robot_at(x,y) and direction(N) and obstacle(x,y+1)
then choice(D).
avoid_obstacle_south:
if robot_at(x,y) and direction(W) and obstacle(x,y-1)
then choice(D).
```

Now a complete "world" has been defined in which the robot "lives" and "walks" until it finds the goal. It has to be mentioned that with directive Choice(D) one of the four directions is selected entirely by chance.

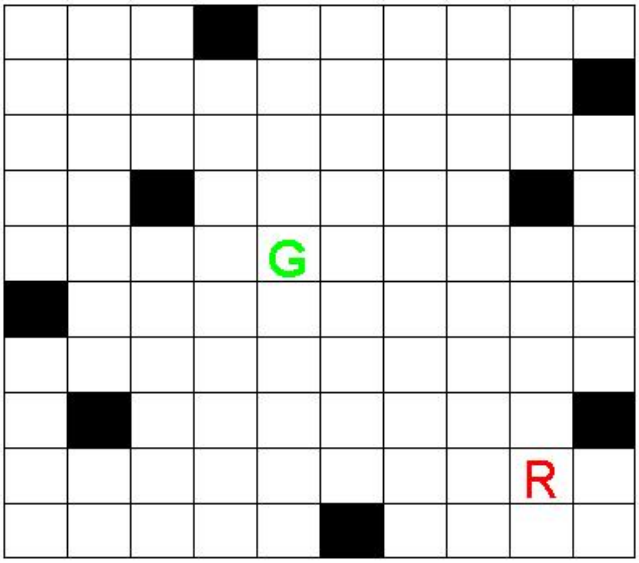

Fig. 1: Robot's environment

Fig. 1 presents a discretized environment sized 10 x 10 with obstacles at fixed positions where the robot is located at position 9,2 and must follow a path in order to reach the goal located at position 5,6. Obviously the path followed by the robot is not a unique one but it depends on its direction. In Table 1 one such a path is described in detail using the state directives and the actions already defined.

Table 1: Robot's movement path

Number of steps	**Robot's condition**	**Rule that fires**
1	robot_at(9,2) direction(n) choice(e) choice(w) choice(s) obstacle_at(6,1) obstacle_at(2,3) … goal_at(5, 6)	move_north
2	robot_at(9,3) direction(n) …	move_north
3	robot_at(9,4) direction(n) …	move_north
4	robot_at(9,5) direction(n) …	move_north
5	robot_at(9,6) direction(n) …	move_north
6	robot_at(9,6) direction(w) …	avoid_obstacle_north move_west

Number of steps	Robot's condition	Rule that fires
7	robot_at(8,6) direction(w) …	move_west
8	robot_at(7,6) direction(w) …	move_west
9	robot_at(6,6) direction(w) …	move_west
10	robot_at(5,6) direction(w) …	detect_object

In the sequel an executable CP-Net model with the above robot's actions as transitions, current and next robot's positions as parameters in guard expressions and robot's states as places is developed and its execution which corresponds to the problem of robot path finding is explained.

4 The CP-Net Model

Hierarchical CP-Nets and the Design/CPN tool were used for the construction of the overall CP-Net model that represents the moves of the robot. The main idea behind this model is the following. The robot exists in a pilot scale labyrinth where some barriers and a goal are present. Two coloursets of integer type, named X and Y, are defined representing coordinates. Moreover, a colourset named Position which is the Cartesian Product of coloursets X and Y, representing a position in the labyrinth is defined. All positions in the labyrinth are numbered like a two dimensional array and are represented as tokens from the colourset Position. Also an enumerated colourset named Direction which contains the tokens East, West, North and South is defined. At any position the robot "knows" its location and its direction (East, West, South, North). The goal is located at a fixed position, where eventually the robot must go to. Initially, the robot checks if it stands in the goal's location. If this is true, it stops "walking" or else, the robot moves to a new position. These are the basic actions that occur in the main model of the compound hierarchical model.

Table 2: The colorsets for modeling the robot

```
color X = int;
color Y = int;
color Boolean = with Yes | No;
color Position = product X * Y;
color Direction = with East | West | South | North;
var x: X;
var y: Y;
var pos1, pos2: Position;
var direction, oldDir, newDir: Direction;
var boolean: Boolean;
```