Analysis of Path Planning Algorithms : a Formal Verification-based Approach

Arash Khabbaz Saberi[1], Jan Friso Groote[1] and Sarmen Keshishzadeh[1]

[1]Eindhoven University of Technology, Eindhoven, The Netherlands
a.khabbaz.saberi@student.tue.nl, j.f.groote@tue.nl, s.keshishzadeh@tue.nl

Abstract

With the emergence of multi-robot systems in the field of robotics there is a need for new approaches for modeling and investigating the behavior of robotic systems. Formal verification is a well-known mathematical method which has been used for decades in order to expose potential design faults in industrial systems. In this paper we introduce the application of formal verification techniques in the context of multi-robot systems. Applying verification techniques, we aim to prove that the collective behavior of a group of robots satisfies certain desired properties. We illustrate our approach using a simple path planning algorithm which conducts a set of robots from their initial positions to their destinations on a planar surface.

Introduction

Multi-robot systems have the potential to solve complicated problems by their collective behavior. In recent years multi-robot systems have attracted the attention of many researchers. The challenge is to control the collective behavior of a number of homogeneous and simple robots in order to accomplish sophisticated tasks. Robustness and versatility are the main characteristics offered by multi-robot systems compared with single robot systems.

The path planning problem for multi-robot systems is of great importance. The objective is to find, or construct a trajectory for each robot to start moving from its initial position toward its destination while avoiding collisions. Path planning for mobile robots has been mainly addressed in the literature from two different aspects. The first approach is to construct trajectories for robots and move them along the predefined trajectories (Laumond, 1998; Luigi Biagiotti, 2009; Saska et al., 2006). The second approach is the algorithmic approach, in which robots use an algorithm to find a way toward their destinations (Jain et al., 2010; min Han, 2007; Pamosoaji and Hong, 2011). Furthermore, existing analysis approaches for robotic systems are mostly focused on the behavioral analysis of individual robots. However, with the increasing popularity of multi-robot systems new approaches are required to analyze the collective behavior of these systems.

In the context of the algorithmic approach to path planning, we propose a new method for behavioral analysis of multi-robot systems. In this approach, a well-known formal verification technique called model checking (Baier and Katoen, 2008) is applied to formally analyze the behavior of a robotic system. Model checking is an algorithmic approach which verifies the validity of a desired system property against a high-level model of the system. Model checking algorithms exhaustively explore all possible behaviors specified by the system model and check whether this model meets the given requirement. This technique is now applied to industrial software/hardware systems in order to verify the correctness of their behavior against a set of requirements. In this paper, we apply it for the analysis of a multi-robot system. The advantage of our approach is that it provides a systematic methodology for modeling and analyzing the collective behavior of a group of robots.

Fig. 1 depicts an overview of our approach. Given an informal specification of a path planning algorithm and the way robots realize this algorithm, we specify a formal model of the robotic system as a set of communicating processes in the mCRL2 language (Groote et al., 2009). Moreover, desired properties of the system are formalized in a mathematical language. In particular, we use the modal μ-calculus (Groote and Mateescu, 1999) for property specification. The verification procedure verifies the formalized property against the system model. If the property is satisfied by the model, the verification procedure will respond with a "Yes". Otherwise, a "No" response is returned. In this case, a counterexample is also generated which describes a situation that the property is violated by the model. We apply the mCRL2 toolset (Cranen et al., 2013) for verification of our case study.

Related work The application of formal methods in the context of robotic systems has been studied in different research works. Most of these works focus on constructing paths in a robot's workspace that satisfy properties specified in a mathematical language. Given an initial position of a robot in its workspace and a requirement specified in linear temporal logic (LTL), (Fainekos et al., 2005) apply a

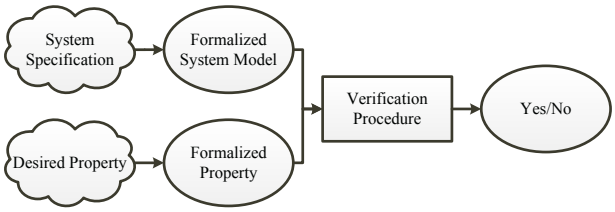

Figure 1: Overview of the Approach

discretization technique to generate a continuous trajectory that satisfies the LTL formula. In this method model checking tools are used for constructing trajectories. The work in (Kress-Gazit et al., 2009) describes an approach for generating a robot controller that guarantees the satisfaction of a given LTL property in all execution scenarios in certain execution environments. It is assumed that the behavior of robot's environment is specified as another LTL formula.

(Fainekos, 2011) proposes a diagnostic approach. First, they define a notion of closeness for properties specified in LTL. For an LTL property violated by the specification of the system they aim to find minimal changes to the formula that makes it satisfiable.

In (Quottrup et al., 2004) a high-level description of a timed automata-based approach for modeling and verification of a multi-robot system is provided. Due to a high level of abstraction only movements of robots are considered in the analyses. Moreover, they do not analyze concrete path planning algorithms. They apply this high level approach to evaluate certain characteristics of a robotic system (e.g., shortest time required by a robot moving in arbitrary directions to arrive at its destination) and construct paths to a destination using model checking evidence.

However, in this paper we include information about the sensing mechanisms of robots and their movements in our analysis. Hence, compared to the timed automata-based approach we investigate a lower level of abstraction. Our approach also offers a high-level language for specifying robotic systems. Moreover, the main objective of this approach is to prove that certain properties are valid/falsified in a system and provide efficient feedback to the designer of a robotic system.

Overview In section *System Specification* we specify the main characteristics of the robotic system that we consider as a case study in this paper. In section *Validation Properties* we discuss the types of properties that we aim to validate. Main features of our analysis approach are explained in section *mCRL2 & the Modal μ-calculus*. We apply this approach in sections *Modeling a Multi-Robot System* and *Verification* to model and verify our case study. *Conclusion* includes some concluding remarks and suggestions for future research.

System Specification

We introduce a simple multi-robot system which we use as a case study throughout the paper. In particular, the physical environment in which robots perform their tasks, main features of the robots (regarding their moving and sensing abilities), and the algorithm that guides robots movements are discussed in this section.

Workspace We assume that robots move on an unbounded planar surface. There are no static obstacles on the surface and robots are the only dynamic obstacles in this setting.

Robots Robot platforms designed for multi-robot applications such as (Bristeau et al., 2011; Mondada et al., 2003) assume that robots need limited computational resources to process their path planning algorithm. On the other hand, these robots are equipped with advanced hardware, e.g., sensors, and processors, to communicate with their environment or perform complicated tasks, e.g., transport heavy objects.

In our case study we consider an algorithm that can be realized by robots using limited resources. We assume that robots are capable of performing translation movements to move toward their destinations. Each robot can also perform in-place rotation movements (e.g., when facing an obstacle) in order to change its direction of movement. Moreover, each robot is only aware of the position that it currently occupies and the location of its destination. Every robot has an embedded sensing device which is capable of detecting obstacles in its range of sense.

Algorithm A robot starts from an initial position and tries to move toward its destination while actively scanning its range of sense for potential obstacles. Whenever a robot senses the presence of an obstacle in its sensing range, it performs an in-place 90-degree counter-clockwise rotation and scans its surroundings along the new direction. This mechanism is repeated until a safe direction is found.

It should be noted that the reactions of the robots are not affected by the actual positions of obstacles. When a robot finds a safe direction, it moves along that direction, steers itself to the destination, and repeats the path finding procedure until it arrives at its destination. In our analyses we do not make any assumptions about the relative processing rate of the robots. In other words, robots can perform their tasks with different processing rates. The same approach can be adapted to systems consisting of robots with identical processing rates.

Validation Properties

In this section we describe several kinds of desired properties for a path planning algorithm. We aim to check for absence of deadlock, absence of collision, and reachability of robots to their destinations for a given path planning algorithm.

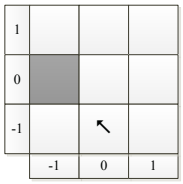

Figure 2: Sensing range for a specific position and movement direction

In order to perform such checks in a robotic system it is required to include information about timing, details of movement trajectories, accuracy of sensors, and shapes of the robots. To make our analyses feasible, we try to keep these details to a minimum. In our analyses we consider a specific abstraction of the robots workspace and robots behavior introduced in *System Specification*.

We assume that the robots workspace is a two dimensional grid which is uniformly partitioned into disjoint cells. Thus each cell in this setting can be characterized by a pair (x, y). Assuming a specific position as the origin of the grid, for each robot the workspace is the following:

$$W = \{(x, y) | x, y \in \mathbb{Z}\}$$

Robots are modeled as objects without any specific shape. Each robot occupies a specific cell on the grid at each moment in time. We assume that robots can perform translation movements along vectors from the following set:

$$D = \{(i, j) | i, j \in \{-1, 0, 1\} \wedge (i, j) \neq (0, 0)\}$$

Translation movements to a neighboring cell are assumed to be atomic events without any time duration. Moreover, performing an in-place rotation by a robot does not change the configuration of the system.

We abstract from the complexities of the sensing mechanism and the way robots realize it. We assume that sensing is an atomic action. Performing this action, robots check for presence of obstacles in their range of sense. The sensing range is assumed to be one cell ahead from a robot's current position along the robot's next movement direction. In Fig. 2 the gray cell depicts the sensing range for a robot in the cell $(0, -1)$ which intends to move along $(-1, 1)$. Sensors are assumed to be accurate and provide correct information.

Finally, we assume that sensing and moving are the only actions robots perform to realize a path planning algorithm. These actions are performed by robots in an infinite loop. Robots that arrive at their destinations can still sense their surroundings. In what follows we explain several properties that can be useful in the context of a path planning algorithm.

Deadlock-freeness Checking for absence of deadlock is one of the general checks that can be performed. Performing this check we can detect problematic situations in the system where no further action can be performed by robots. As mentioned earlier, in our sketched system robots can always sense their surroundings. Thus, in this case deadlock freedom easily follows from the system specification. However, this check could be useful in general.

Collision-freeness For a given path planning algorithm, it is desirable to check that for all trajectories calculated by the algorithm, robots will never collide with an obstacle (which could be another robot). Considering our abstraction of the workspace and robots movements, this means that robots should not share a cell on the grid with another object.

Reachability The ultimate goal of a path planning algorithm is to guide robots to their destinations. We check whether robots can reach their desired destinations in a finite number of steps. It is also possible to define a limit on the number of steps that a robot takes before reaching its goal.

As each robot can prevent other robots from reaching their destinations, we are also investigating reachability while reaching the goal can be prevented by other robots for infinitely many times.

mCRL2 & the Modal μ-calculus

In this section we explain the details of our analysis approach depicted in Fig. 1. In section *Labeled Transition Systems* we describe labeled transition systems as a means for modeling and analyzing the behavior of systems. In section *The mCRL2 language* we introduce mCRL2 as a high-level language for specifying labeled transition systems in a compositional manner. We use this language to capture the behavior of multi-robot systems by specifying and composing the behavior of its components. The modal μ-calculus is the property specification language used in our approach which is briefly explained in section *The modal μ-calculus*.

Labeled Transition Systems

Modeling and analyzing the behavior of systems with labeled transition systems (LTS) is a common technique. A labeled transition system is a directed graph with a set of states, i.e., graph nodes, and a set of transitions, i.e., graph edges. States of an LTS correspond to system states. Transitions are labeled by actions and show the evolution of the system when executing a specific action. Labeled transition systems have an initial state which is depicted by an incoming arrow.

Fig. 3 is a simple LTS which specifies the following behavior for a system. Starting in the initial state an *a* action is performed. Then the system non-deterministically chooses to perform an *a* or *b*. Performing an *a* will result in a state where no further action can be performed by the system. Performing a *b* means that the left branch is executed. After the *b* action is done the system will perform *a* forever.

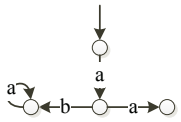

Figure 3: A simple labeled transition system

The mCRL2 language

Modeling systems behavior with LTSs does not scale to complex systems. High-level languages can be used to describe complex LTSs. We use the mCRL2 language for this purpose. In this language LTSs are described as processes. Data specification and process specification are the main aspects of the language.

Data specification The data specification subset of the language allows the user to use the built-in data types (e.g., booleans, natural numbers) or to define and manipulate her own system-specific data types. For instance, in our case study we need two data types to specify the position of each robot and movement vectors. In the mCRL2 syntax we have:

```
sort Pnt = struct P(X:Int,Y:Int);
     Dir = struct D(I:Int,J:Int);          (1)
```

Each variable of type `Pnt` corresponds to a cell on the grid. Instances of `Pnt` can be constructed by applying the constructor `P`. For an instance `p` of type `Pnt` the first and second element of the pair can be accessed through the application of the projection functions `X` and `Y`, respectively. The same argument is applicable to `Dir`. In our modeling we always use instances `d` of `Dir` such that `I(d),J(d)` $\in \{-1, 0, 1\}$.

Useful operations on data types can be defined as functions. Each function is specified by its name, types of arguments and return type, and a set of equations describing the relation between the input(s) and output of the function. We illustrate this with an example:

```
map NextPos:Pnt # Dir -> Pnt;
var p:Pnt, d:Dir;                          (2)
eqn NextPos(p,d)= P(X(p)+I(d), Y(p)+J(d));
```

Given the current position of a robot and its next movement direction, `NextPos` computes its next position. In this example, we have one equation in our equation system (preceded by `eqn`). The `var` block preceding the equation system defines the variables used in the equation. These variables are used for pattern matching. The single equation specifies that for a given position `p` and direction `d`, the return value is an instance of `Pnt`. The `X` attribute of the next position is the sum of `X(p)` and `I(d)`. The `Y` attribute is computed in a similar way.

Process specification The process specification aspect of the language provides a compositional way for system behavior specification. Actions are the main building blocks of processes. We assume that actions are atomic events without any time duration. Actions can also carry data parameters. For the specification of a robotic system, we can define an action which mimics a single step movement of a robot. The parameter of this action specifies the movement vector.

```
act move:Dir;
```

In order to achieve complex behaviors or communication patterns we can combine actions sequentially, in parallel, or make a non-deterministic choice between a (possibly infinite) set of actions. We can also let data values affect the behavior of a process by adding conditional expressions to processes.

The non-deterministic choice between processes P and Q is denoted by $P + Q$. The process $P + Q$ will behave as P or Q. The sequential composition of P and Q is denoted by $P.Q$ and the resulting process first performs the behavior of P and then behaves as Q. The notation $P||Q$ represents the parallel execution of P and Q. It is also possible to enforce communication between actions executed by different processes. Using this facility together with the operator ||, one can force communication between P and Q for specific actions. The rest of the actions will be interleaved. Assuming that c is a boolean expression, the process $c \rightarrow P \diamond Q$ will behave as P if c is satisfied and will behave as Q otherwise.

Using the mCRL2 language, we can specify the movements of a robot as a process. For example, a robot that takes a sequence of non-deterministically chosen right and left steps can be specified as follows:

```
proc Robot(rp:Pnt) =
  move(D(1,0)).Robot(NextPos(rp,D(1,0))) +
  move(D(-1,0)).Robot(NextPos(rp,D(-1,0)));
```

The process carries a data parameter which represents the current position of `Robot`. After each movement along `D(1,0)` or `D(-1,0)` the robot behaves as the `Robot` process with a new parameter calculated by `NextPos`.

One can easily influence the choices of `Robot` by another process, e.g., `Environment`. In our context this process can mimic the behavior of the environment by collecting information about the workspace and enforcing certain constraints on movements. We can establish a communication between the two processes to enforce a movement in a specific direction. To realize this communication in this example, `Environment` should perform an action, e.g., `en_move`, which carries data parameters of the same type as `move`. We describe `Environment` as follows:

```
proc Environment =
    en_move(D(1,0)).en_move(D(-1,0)).Environment;
```

By enforcing the communication between `move` and `en_move` and executing `Robot` and `Environment` in parallel, the required controlling mechanism will be achieved. In this way,

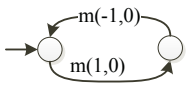

Figure 4: LTS for `Robot||Environment` with enforced communication

`move` and `en_move` are only executed in a synchronized manner both carrying equal data parameters. In our simplified example `Environment` only affects the movement direction of `Robot` by enforcing it to move to its right and left alternatingly. Putting `Robot` and `Environment` in parallel will result in the LTS of Fig. 4 where action `m` represents the communication of `move` and `en_move`.

The modal μ-calculus

We use the modal μ-calculus for specifying system properties. The following grammar gives the basic form of modal μ-calculus formulae where a is an action.

$$\begin{aligned} \phi \quad ::= \quad & \mathit{true} \mid \mathit{false} \mid \neg\phi \mid \phi \wedge \phi \mid \phi \vee \phi \mid \\ & \langle a \rangle \phi \mid [a]\phi \mid \mu X.\phi \mid \nu X.\phi \end{aligned} \tag{3}$$

The operators $\wedge$, $\vee$, and $\neg$ have their usual meanings. The formula $\langle a \rangle \phi$ is valid in a state of an LTS when an action a can be performed such that ϕ is valid after this action a has been done. The formula $[a]\phi$ is valid in a state if all possible executions of action a lead to a state where ϕ holds.

The formulae $\mu X.\phi$ and $\nu X.\phi$ are the minimal and maximal fixed points, respectively. In both cases, X usually occurs in ϕ. The property $\mu X.\phi$ is valid for all the states in the smallest set X that satisfies '$X = \phi_s$'. Similarly, $\nu X.\phi$ is valid for all the states in the largest set X that satisfies '$X = \phi_s$' where ϕ_s represents all states where ϕ is valid. Other capital letters (e.g., Y) can also be used instead of X.

As an example consider a setting with parameterless actions $\{move, sense\}$. The property $\mu X.([sense]X)$ is valid if $sense$ is executed for a finite number of times. In other words, $move$ is unavoidable (unless a deadlock occurs). The property $\nu X \mu Y.([move]X \vee [sense]Y)$ specifies that any subsequence of consecutive $sense$ actions should be finite.

It is often useful to use a variant of the grammar in Eqn. 3 which allows the occurrence of multiple actions or a sequence of actions in a modality. For this purpose regular formulas are used within modalities. The formula $true$ represents the set of all actions and $\cup$ (union), $\cap$ (intersection), and $\neg$ (complement with respect to the set of all actions) can be used to specify action formulas. Regular formulas extend action formulas to allow the use of sequences of actions in modalities. For instance for a subset of actions α, α^* denotes any sequence of actions from α.

Since actions can carry data parameters, we need ways to refer to data values in formulae. As an example consider the set of actions $\{move, sense\}$ both carrying one data parameter of type `Dir`. Data can be introduced to modal formulae referring to $move$ and $sense$ using existential or universal quantifiers. Quantification can be used within modalities. For instance $[\forall d : Dir : (\neg move(d))^*]\phi$ says that as long as there is no move in any direction, ϕ should hold. Quantifiers can also be used outside the scope of modalities with their standard meaning. For instance $\forall d : Dir.[move(d)]\phi$ is true if ϕ holds after performing $move$ in any direction. We can also store and process data values in fixed points. Using this feature it is possible for instance to specify constraints on the number of certain events. Consider the following formula:

$$\mu X(p : Pnt = P0).(\forall d : Dir[move(d)]X(NextPos(p,d)) \vee (X(p) > Y(p)))$$

Here p records the current position of the robot and is initially set to the $P0$ (the initial position). The property says that after finite number of movements the robot should be in a certain part of the grid where $X(p) > Y(p)$ holds.

We refer the interested reader to (Groote and Mateescu, 1999) for a more detailed explanation of the modal μ-calculus and its semantics.

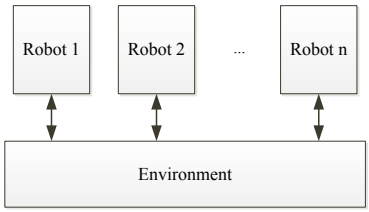

Figure 5: Robot and Environment processes

Modeling a Multi-Robot System

In this section we elaborate on the simple modeling scheme we introduced in the previous section to formalize the multi-robot system described in section *System Specification*. Fig. 5 depicts a schematic view of our modeling approach. For a system consisting of n robots we specify $n + 1$ processes, i.e., n robot processes and 1 environment process, in mCRL2 and put them in parallel. This scheme conforms to our description in *System Specification*, i.e., robots communicate with the environment but they do not have direct communication among themselves.

Each process carries and manipulates certain data parameters. Every robot process carries parameters which indicate its current position and the (potential) direction of the next move. We assign a unique identifier to each robot process so the environment process can distinguish them when performing communications. The environment process carries data parameters to record the current position of all the robots. Since the destinations of the robots are not affected

by the dynamics of the system we model them as global parameters. For instance for Robot1 from Fig. 5 we can specify the destination as the position $(2, 3)$ as follows:

```
map PD1:Pnt;
eqn X(PD1)=2;Y(PD1)=3;
```
(4)

In our case study we assume that the system consists of identical robots. Thus, the processes we use to describe their behavior only differ in the unique parameter that is used for the identification of the robots. This approach can be adapted to systems consisting of robots with nonidentical path planning algorithms. For the sake of simplicity we study a system with two robots in this paper. The following definition specifies a data type with two unique instances which we use for robots identification.

```
sort ID = struct id1|id2;
```

In what follows we explain the mCRL2 processes that we use to describe the robots and the environment.

Robots In our abstraction of the robotic system, each robot scans its surroundings and performs movements in an infinite loop. Thus, robot processes can be specified in terms of actions rs (sense) and rm (move).

We enforce a communication on rs with the environment to collect information about the presence of obstacles. A robot performs rs providing its unique identifier and the next movement direction. Performing rs a robot should be able to react to both outcomes of the performed check, i.e. presence or absence of obstacles. We specify rs as follows:

```
act rs:ID # Dir # Bool;
```
(5)

Scanning the range of sense along vector d by a robot identified by id1 can be modeled as a nondeterministic choice between a "Yes" or "No" response for presence of obstacles, i.e., rs(id1,d,true)+rs(id1,d,false).

To establish a communication on rs, the environment process should perform an action, e.g., es, with identical parameter values. Given the movement direction of the robot, if the environment does not find an obstacle in the range of sense it will perform es(id1,d,false). Next, the robot will perform a move action along the direction vector. Otherwise, the environment will perform es(id1,d,true) and the robot should not move.

A robot identified by id1 performs rm(id1,d) to declare its movement along the vector d. We enforce a communication between the robots and the environment on this action. In this way the environment can calculate the new position of the moving robot and update its information. The following process describes the behavior of a robot identified by id1. The process Robot2 can be specified in a similar way.

```
proc Robot1(p:Pnt,d:Dir) =
  sum b:bool.  rs (id1,d,b).b ->
    Robot1(p,NextDir(id1,d,p,b)) <>
    rm(id1,d).Robot1(NextPos(p,d),
                NextDir(id1,d,NextPos(p,d),b));
```

The syntax sum b: bool. rs (id1,d,b) is a shorthand for rs(id1,d,true)+rs(id1,d,false). The conditional statement in Robot1 indicates that after performing rm the same behavior is repeated with new position and direction parameters calculated by NextPos (see Eqn. (2)) and NextDir functions, respectively. On the other hand, the presence of obstacles only causes an in-place change of direction. The following expressions partially specify NextDir:

```
map NextDir:ID # Dir # Pnt # Bool -> Dir;
var cp:Pnt, cd:Dir, b:Bool;
eqn (cp!=PD1) -> NextDir(id1,cd,cp,false) =
          D(sgn(X(PD1)-X(cp)),sgn(Y(PD1)-Y(cp)));
    (cp!=PD1) -> NextDir(id1,cd,cp,true) =
          D(-Y(cd), X(cd));
    (cp==PD1) -> NextDir(id1,cd,cp,b) = D(0,0);
```

In this specification sgn is a function that extracts the sign of its argument and PD1 is Robot1's destination (Eqn. (4)). The notations == and != denote data equality and inequality. Two instances p1,p2 of Pnt are equal if and only if X(p1)=X(p2) and Y(p1)=Y(p2). The first and second rule of the equation system determine the next movement direction for the first robot when it is not at its destination. Absence of obstacles in the last scan activates the first rule and the next direction is calculated in order to guide the robot closer to its destination. Presence of an obstacle in the last scan activates the second rule which mimics a 90-degree counterclockwise rotation. The last rule sets the direction to $(0, 0)$ when the robot arrives at its destination. The complete specification can be derived by describing a similar behavior for id2.

Fig. 6 depicts the LTS described by Robot1 where each state is labeled by the data parameters carried by the process in that state.

Environment The Environment process records the position of the robots and performs the actions es and em in order to establish the required communications with the robot processes. The following mCRL2 syntax describes this process:

```
proc Environment(p1:Pnt,p2:Pnt) =
 sum id:ID, d:Dir.es(id,d,Sense(id,p1,p2,d)).
  Sense(id,p1,p2,d)->
    Environment(p1,p2)<>
      (id==id1) ->
         em(id1,d).Environment(NextPos(p1,d),p2) +
      (id==id2) ->
         em(id2,d).Environment(p1,NextPos(p2,d));
```

The summations over the data types ID and Dir indicate that this process can establish a communication with any robot on the action es to perform a check for obstacles along direction d. The function Sense performs this check. If a movement is possible, Environment will update its information with the new position. Otherwise it will repeat the same

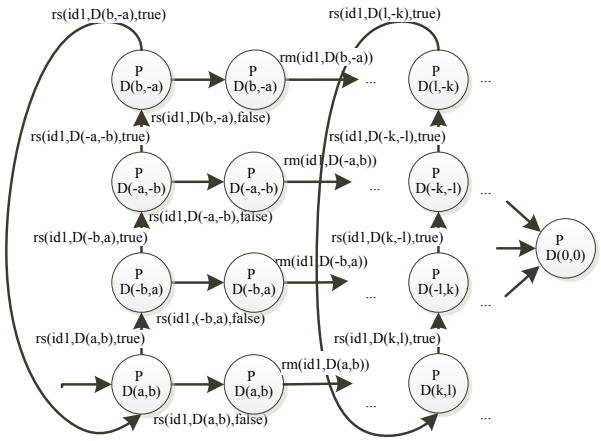

Figure 6: LTS description for `Robot1`

behavior. A partial specification of `Sense` is as follows:

```
map Sense:ID # Pnt # Pnt # Dir -> Bool;
var cp1,cp2:Pnt, cd:Dir;
eqn (PD1!=cp1) -> Sense(id1,cp1,cp2,cd) =
                  (cp2==NextPos(cp1,cd));
    (PD1==cp1) -> Sense(id1,cp1,cp2,cd) = false;
```

The first rule checks for presence of obstacles when the first robot is not in its destination. It simply checks whether a movement in the specified direction will cause a collision with the second robot (see Fig. 2). Since robots can only move along $(0, 0)$ after arriving at their destinations, the second rule always declares absence of obstacles when `Robot1` arrives at its destination. The complete specification can be derived by describing a similar behavior for `id2`.

Finally, we initialize the specified processes and put them in parallel to achieve a model for the system (Fig. 5). To this end we use the following mCRL2 syntax:

```
init
 allow({m,s},comm({rm|em -> m, rs|es -> s},
    Robot1(P01,D01) || Robot2(P02,D02) ||
    Environment(P01,P02)));
```
(6)

The `comm` operator is used to establish communication between `rs` and `es` and renames this communication to a single action (`s`). The `allow` operators enforces the specified communications. In other words it blocks non-synchronous execution of `rs` and `es`. The actions `rm` and `em` are treated in the same way. The initial position and direction parameters, e.g., `P01`, can be specified similar to Eqn. (4).

Verification

In what follows we first formalize the properties from *Validation Properties* in the modal μ-calculus. We report on the results and observations we achieved on verifying these properties against the specification discussed in the previous section. We applied the mCRL2 toolset for verification.

Deadlock-freeness In any reachable state of the system it is possible to perform an action:

$$[true^*]\langle true\rangle true \tag{7}$$

Collision-freeness Trajectories calculated by the algorithm will not cause a collision for two robots initially at $P01$ and $P02$, i.e., robots will not share a cell on the grid:

$$\begin{aligned}&\nu X(p_1 : Pnt = P01, p_2 : Pnt = P02).\\ &\quad(([\forall\, id : ID, d : Dir.\neg m(id, d)]X(p_1, p_2)) \wedge\\ &\quad(\forall\, d : Dir.[m(id1, d)]X(NextPos(p_1, d), p_2)) \wedge\\ &\quad(\forall\, d : Dir.[m(id2, d)]X(p_1, NextPos(p_2, d))) \wedge\\ &\quad(p_1 != p_2))\end{aligned} \tag{8}$$

Reachability `Robot1` (initially at $P01$) should reach its destination (*PD1*) with a finite number of movements:

$$\begin{aligned}&\mu X(p : Pnt = P01).\nu Y.(([\nexists d : Dir.m(id1, d)]Y \wedge\\ &\quad(\forall d : Dir.[m(id1, d)]X(NextPos(p, d)))) \vee (p == \mathit{PD1}))\end{aligned} \tag{9}$$

Applying the mCRL2 toolset we verified these properties against the model of the system. The properties (7) and (8) hold for any combination of different initial and destination positions for the robots chosen from the following set:

$$TestPoints = \{(x, y)|x, y \in \{0, \ldots, 5\}\} \tag{10}$$

However, reachability does not hold in general. For instance, we identified the counterexample in Fig. 7. Movements of the first and second robot are depicted by filled and dashed arrows, respectively. The numbers denote the order of the performed moves. Initial and destination cells are marked by circles and flags, respectively. In this case the second robot moves relatively slowly compared to the first robot and it stops at the destination of the first robot. This causes a livelock in the first robot's behavior, i.e, it will perform the same sequence of actions without making any progress.

In an attempt to characterize the occurring problem we introduce a new notion of reachability where infinite movement steps are allowed when at least one of the robots is "close" to the destination of the other robot. The following formula formalizes this property:

$$\begin{aligned}&\mu X(p_1 : Pnt = P01, p_2 : Pnt = P02).\\ &\quad\nu Y(p'_1 : Pnt = p_1, p'_2 : Pnt = p_2).\\ &\quad(((\forall d : Dir.[m(id1, d)]\\ &\qquad(((Near(p'_2, PD1) \vee Near(NextPos(p'_1, d), PD2)) \wedge\\ &\qquad\quad Y(NextPos(p'_1, d), p'_2))\\ &\qquad\vee((!Near(p'_2, PD1) \wedge !Near(NextPos(p'_1, d), PD2)) \wedge\\ &\qquad\quad X(NextPos(p'_1, d), p'_2))))\wedge\\ &\quad(\forall d : Dir.[m(id2, d)]Y(p'_1, NextPos(p'_2, d)))\wedge\\ &\quad([\forall id : ID, d : Dir.!m(id, d)]Y(p'_1, p'_2)))\vee\\ &\quad(p_1 == \mathit{PD1}))\end{aligned}$$

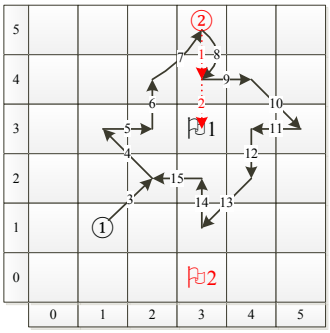

Figure 7: A counterexample for reachability

The function `Near` is defined as follows where `abs` is the built-in absolute value function:

```
map Near:Pnt # Pnt -> Bool;
var p1,p2:Pnt;
eqn Near(p1,p2) = (abs(X(p1)-X(p2)) <= 2 &&
                   abs(Y(p1)-Y(p2)) <= 2 );
```

This property is satisfied by the system described in the previous section for any reasonable combination of initial and destination positions from Eqn. (10).

Conclusion

In the context of high level algorithmic approach to the path planning problem we studied ways to analyze multi-robot systems in a systematic way. The final goal is to verify a set of desired properties against a high-level model of a system and provide efficient feedback to the designer of the system.

We have introduced an approach based on process algebras for modeling and analyzing path planning algorithms. We have used the mCRL2 language and the modal μ-calculus for describing a multi-robot system and its properties. The mCRL2 toolset has been used for the verification of these properties. We have applied this approach to investigate the correctness of a set of useful properties in a simple multi-robot system. Our observations show that for a simple path planning algorithm useful properties can be verified with the proposed approach efficiently (about one minute for each property on a standard desktop computer).

We envisage applying our approach to systems with more sophisticated path planning algorithms. Applying this approach to systems consisting of a large number of robots can also be considered as future work.

References

Baier, C. and Katoen, J.-P. (2008). *Principles of Model Checking*. MIT Press.

Bristeau, P. J., Callou, P. J., Vissiere, D., and Petit, N. (2011). The navigation and control technology inside the AR.Drone micro UAV. In *Proceedings of IFAC'11*, pages 1477–1484.

Cranen, S., Groote, J. F., Keiren, J. J., Stappers, F. P., de Vink, E. P., Wesselink, W., and Willemse, T. A. (2013). An overview of the mCRL2 toolset and its recent advances. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 199–213. Springer.

Fainekos, G. (2011). Revising temporal logic specifications for motion planning. In *Proceedings of ICRA'11*, pages 40 –45.

Fainekos, G., Kress-Gazit, H., and Pappas, G. (2005). Temporal logic motion planning for mobile robots. In *Proceedings of ICRA'05*, pages 2020 – 2025.

Groote, J. and Mateescu, R. (1999). Verification of temporal properties of processes in a setting with data. *Algebraic Methodology and Software Technology*, pages 74–90.

Groote, J., Mathijssen, A., Reniers, M. A., Usenko, Y., and van Weerdenburg, M. (2009). Analysis of distributed systems with mCRL2. *Process Algebra for Parallel and Distributed Processing*.

Jain, S., Sawlani, M., and Chandwani, V. (2010). Ad-hoc swarm robotics optimization in grid based navigation. In *Proceedings of ICARCV'10*, pages 1553–1558.

Kress-Gazit, H., Fainekos, G., and Pappas, G. (2009). Temporal-logic-based reactive mission and motion planning. *IEEE Transactions on Robotics*, 25(6):1370 –1381.

Laumond, J. P. (1998). *Robot Motion Planning and Control*. Springer-Verlag.

Luigi Biagiotti, C. M. (2009). *Trajectory Planning for Automatic Machines and Robots*. Springer.

min Han, K. (2007). Collision free path planning algorithm for robot navigation problem. Master's thesis, University of Missouri-Columbia.

Mondada, F., Guignard, A., Bonani, M., Bar, D., Lauria, M., and Floreano, D. (2003). SWARM-BOT: from concept to implementation. In *Proceedings of IROS'03*, volume 2, pages 1626 – 1631.

Pamosoaji, A. and Hong, K.-S. (2011). Collision-free path and trajectory planning algorithm for multiple-vehicle systems. In *Proceedings of RAM'11*, pages 67–72.

Quottrup, M., Bak, T., and Zamanabadi, R. (2004). Multi-robot planning : a timed automata approach. In *Proceedings of ICRA'04*, volume 5, pages 4417 – 4422.

Saska, M., Macas, M., Preucil, L., and Lhotska, L. (2006). Robot path planning using particle swarm optimization of Ferguson splines. In *Proceedings of ETFA'06*, pages 833–839.