Formal Verification of an Autonomous Wheel Loader by Model Checking

Rong Gu, Raluca Marinescu,
Cristina Seceleanu, Kristina Lundqvist

MÄLARDALEN UNIVERSITY SWEDEN

6TH INTERNATIONAL CONFERENCE ON FORMAL METHODS IN SOFTWARE ENGINEERING
FormaliSE June, 2018

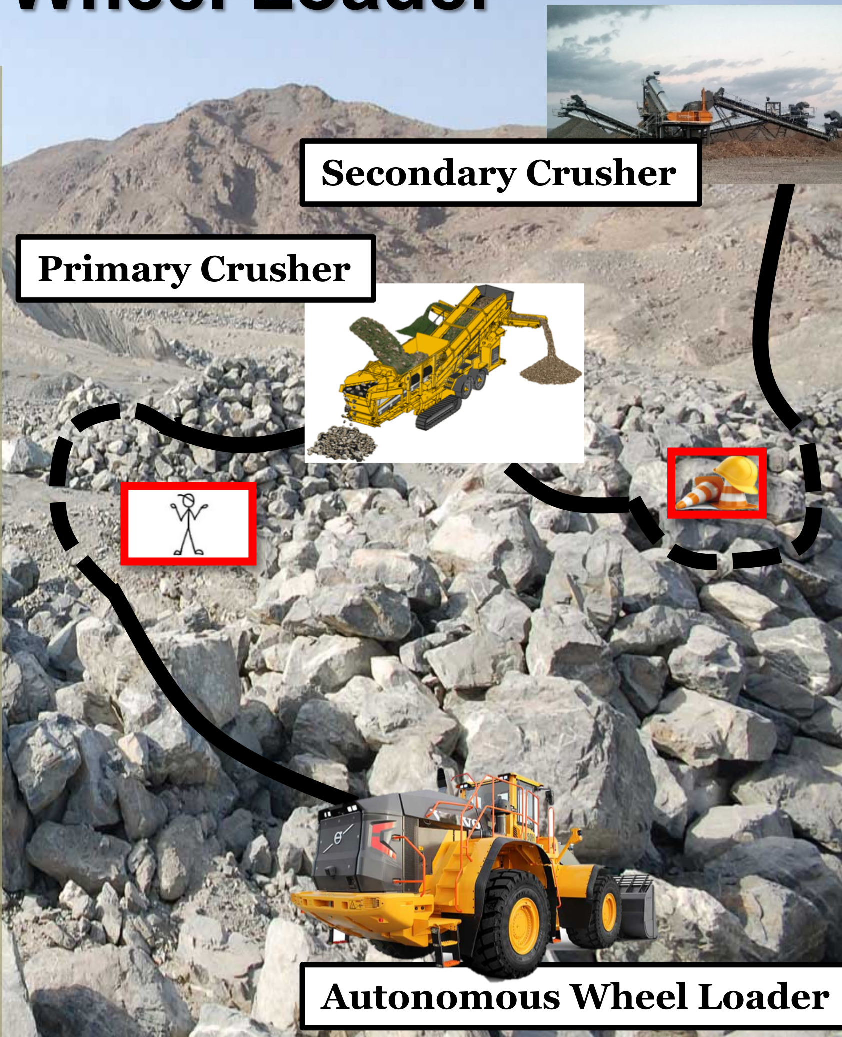
Use case: Autonomous Wheel Loader

Autonomous Wheel Loader (AWL)

(a) A heavy construction vehicle
(b) Transports material, loads and unloads at crushers
(c) No human operator on-board
(d) Works under any condition, e.g., dusty, raining, foggy, and dark environment
(e) Existing prototype has no intelligence (e.g. collision avoidance) and no dependability guarantees
(f) Path planning and replanning for autonomous path following and collision avoidance

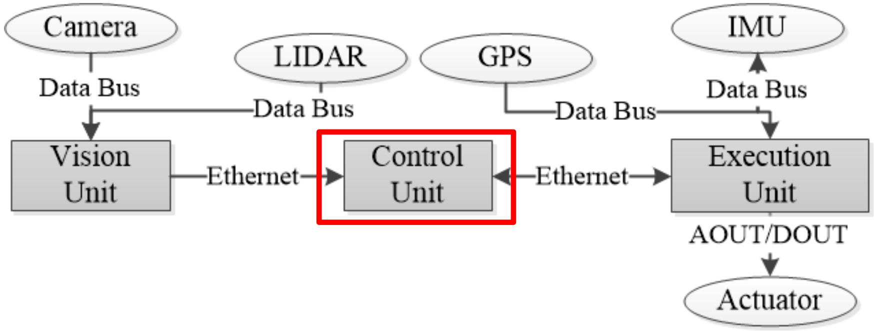
Requirement

a) An AWL must calculate the initial path before it starts to move and avoid all kinds of obstacles dynamically as it moves.
b) Follow the planned path autonomously.
c) React to errors in the control system timely and correctly .

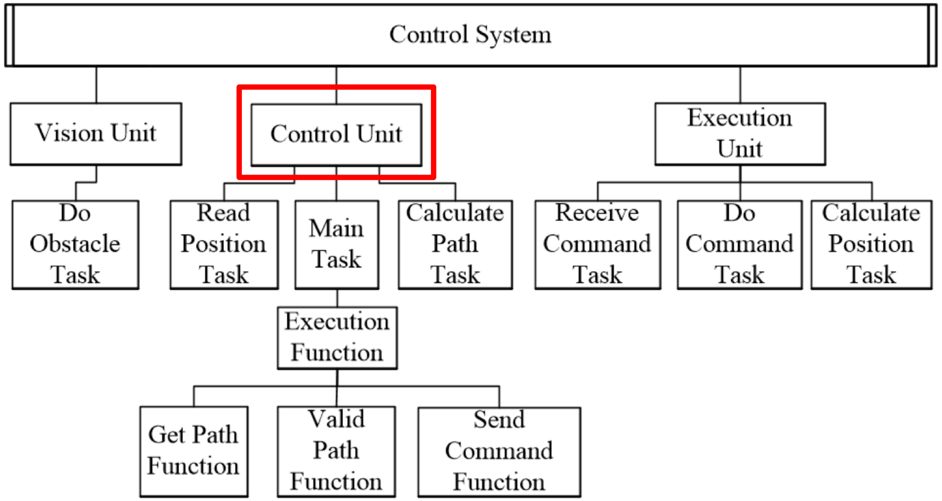


Use case: Autonomous Wheel Loader

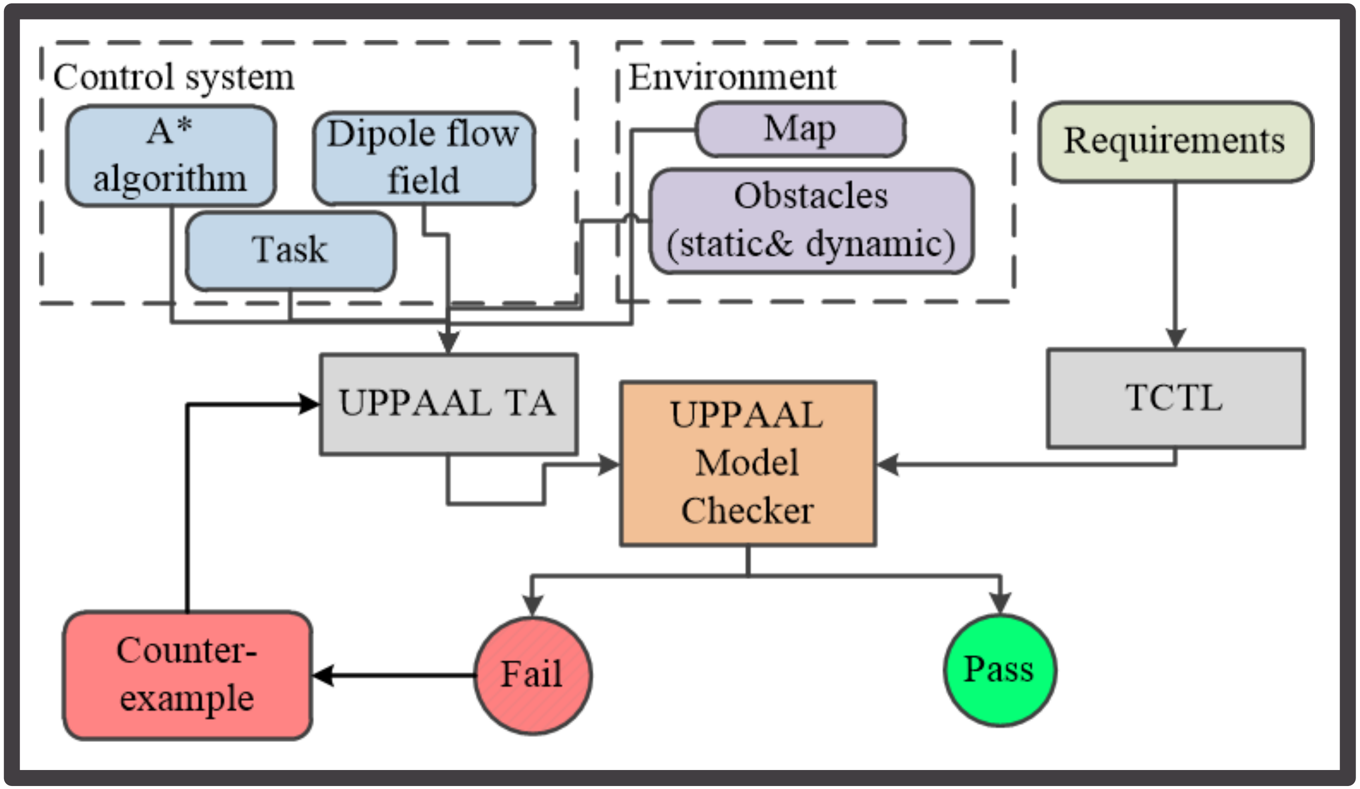
- The architecture of the AWL's control system



- Task allocation in the control system



Method: Formal Modelling and Verification



- **UPPAAL TA**: UPPAAL Timed Automata
- **TCTL**: Timed Computation Tree Logic

Preliminaries: Path-planning algorithm – A* algorithm

- A widely used algorithm for path finding and graph traversal.
- A* algorithm works in a grid.
- 2-dimensional array (int map[N][N]), 1: walkable, 0:"not walkable".
- A* algorithm is an extension of Dijkstra's algorthim, finding the shortest path from A to B.

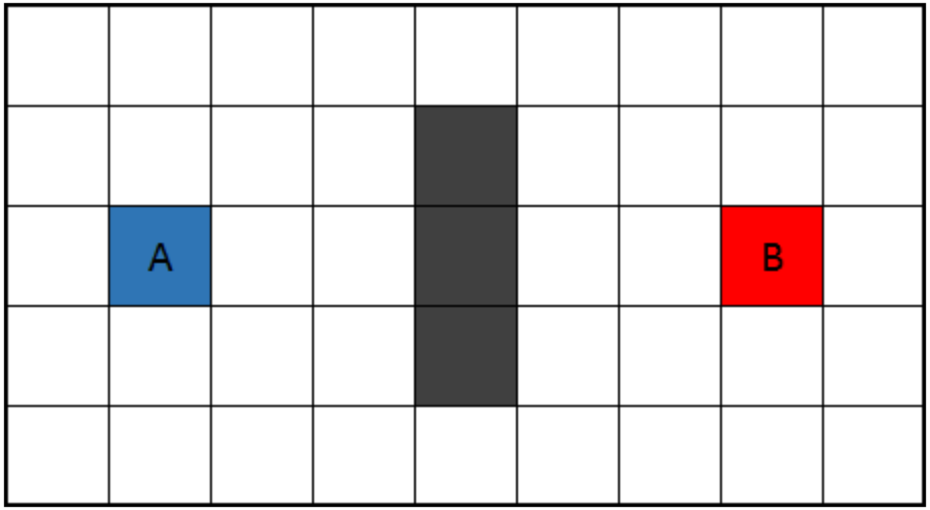


Figure 2. A* algorithm works in grid.

Preliminaries: Path-planning algorithm – A* algorithm

- F = G + H,
  - G: cost from start to current cell
  - H: estimated cost from current cell to destination
- Manhattan Distance: The simple sum of the horizontal and vertical distance ignoring the "unreachable" cells.

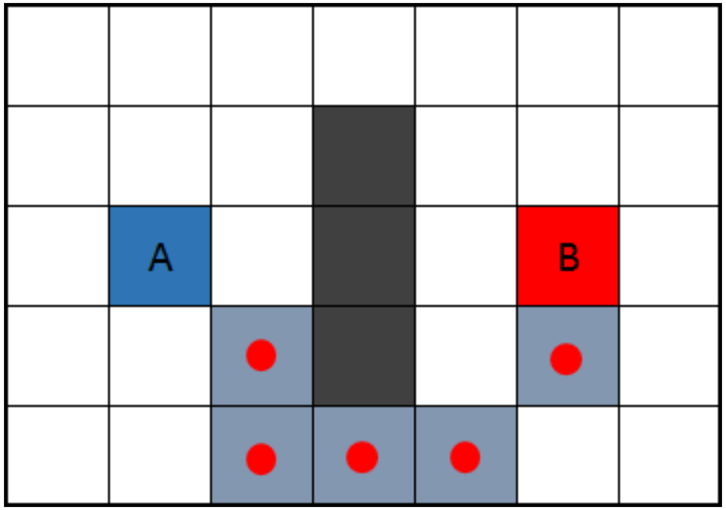
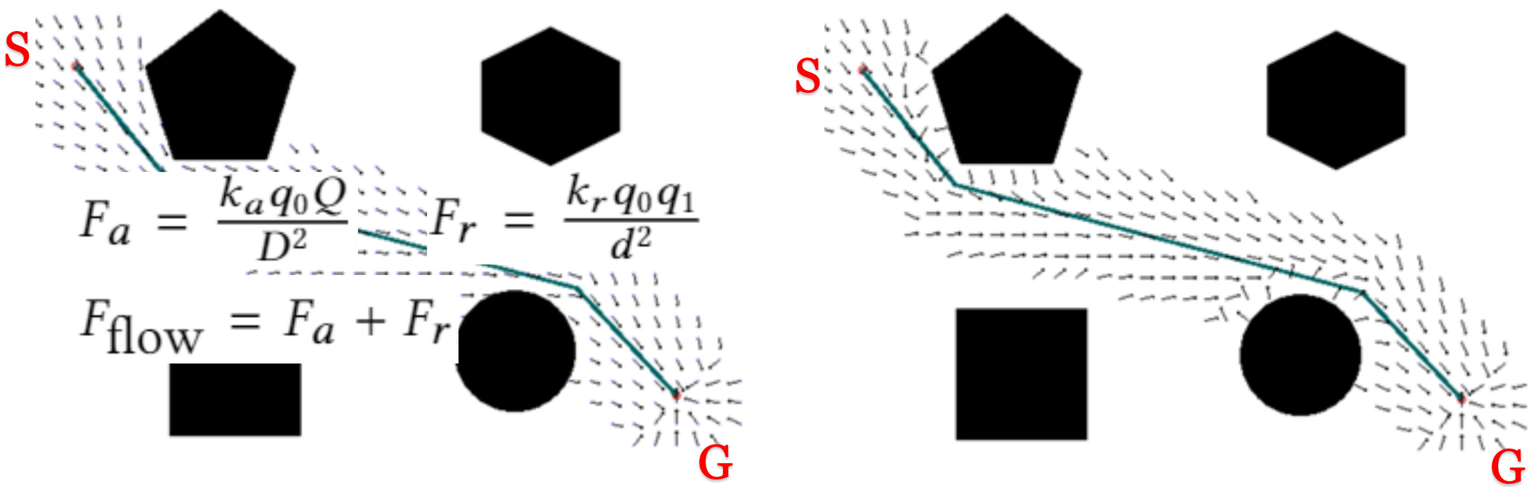


Figure 3. A* algorithm finds the shortest path.

Preliminaries: Collision-avoidance algorithm

Dipole Flow Field: Static Flow Field – avoid static obstacles



(A) (B)

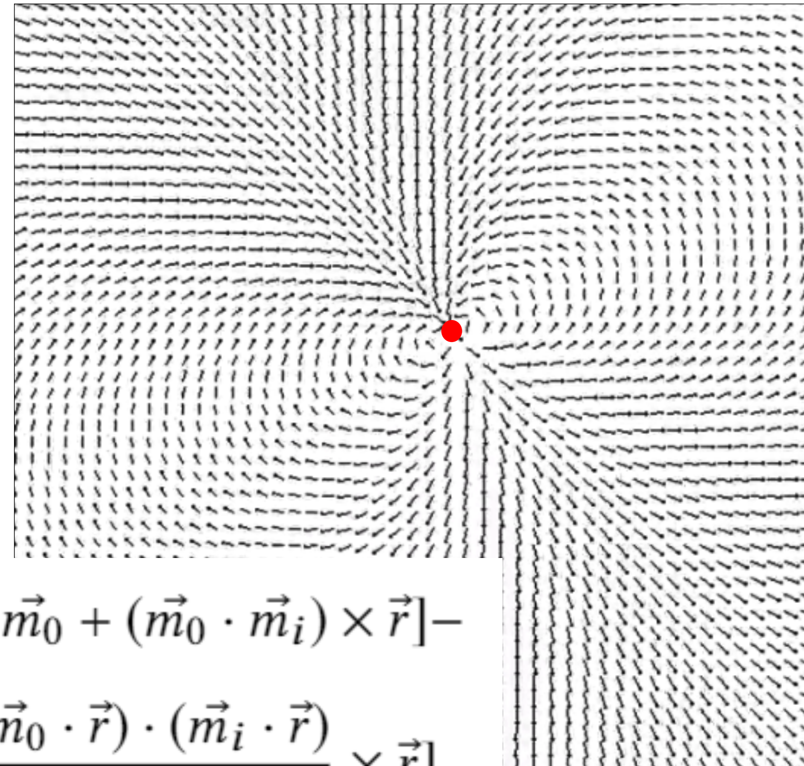
Figure 6. The representation of the static flow field (unity vectors), (A) the initial path with the configured static attractive field, (B) the static flow field with added repulsive force to the obstacles [1].

[1] LanAnh Trinh, Mikael Ekström, and Baran Çürüklü. 2017. Dipole Flow Field for Dependable Path Planning of Multiple Agents. In IEEE/RSJ International Conference on Intelligent Robots and Systems.

Preliminaries: Collision-avoidance algorithm

Dipole Flow Field: Dynamic Dipole Field – avoid dynamic obstacles

- Every object is assumed to be a source of magnetic dipole field.
- The magnitude of the magnetic moment is proportional to the velocity.
- Two moving objects repulse each other when they are close enough.



a moving object[1].

$$\vec{m} = k_m \vec{v}$$

$$\vec{F}_d = \frac{k_d}{d^5}[(\vec{m}_0 \cdot \vec{r}) \times \vec{m}_i + (\vec{m}_i \cdot \vec{r}) \times \vec{m}_0 + (\vec{m}_0 \cdot \vec{m}_i) \times \vec{r}] - \frac{5 \cdot (\vec{m}_0 \cdot \vec{r}) \cdot (\vec{m}_i \cdot \vec{r})}{d^2} \times \vec{r}],$$

[1] LanAnh Trinh, Mikael Ekström, and Baran Çürüklü. 2017. Dipole Flow Field for Dependable Path Planning of Multiple Agents. In IEEE/RSJ International Conference on Intelligent Robots and Systems. http://www.es.mdh.se/publications/ 4883-

Preliminaries: Timed automata and UPPAAL

- Timed automata (TA): finite state machines with real-valued clocks
- UPPAAL: A TA-based toolbox for validation and verification of real-time systems.

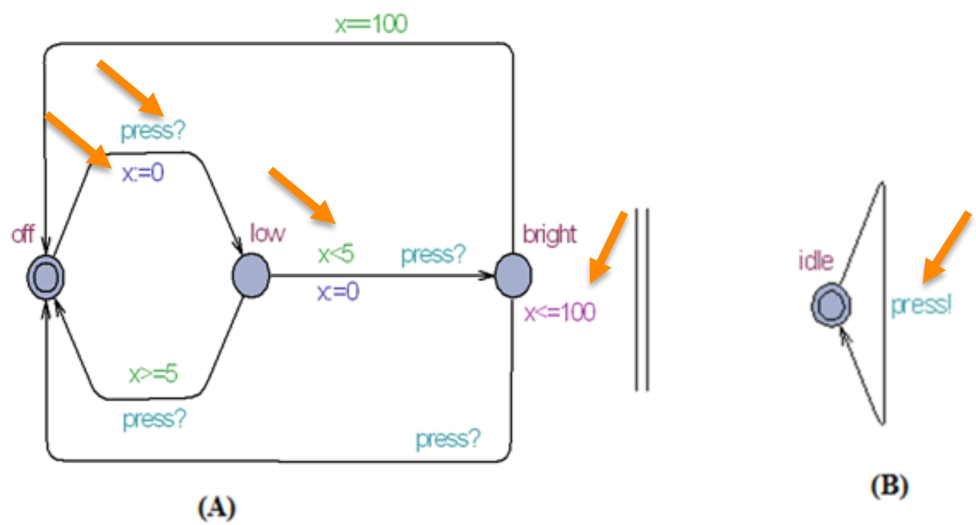


Figure 8. A lamp example of a network of UPPAAL TA

Preliminaries: Timed Computation Tree Logic

Formalize the natural-language requirements to (Timed) Computation Tree Logic (TCTL) queries, which are in the form:

E◊ p	There exists a path where p eventually holds
	The shortest/fastest path to the state holding p
A□ p	For all paths, p always holds
A◊ p	For all paths, p will eventually hold
$p \rightarrow_{<=t} q$	For all paths, if p holds then q will eventually hold within T time units.

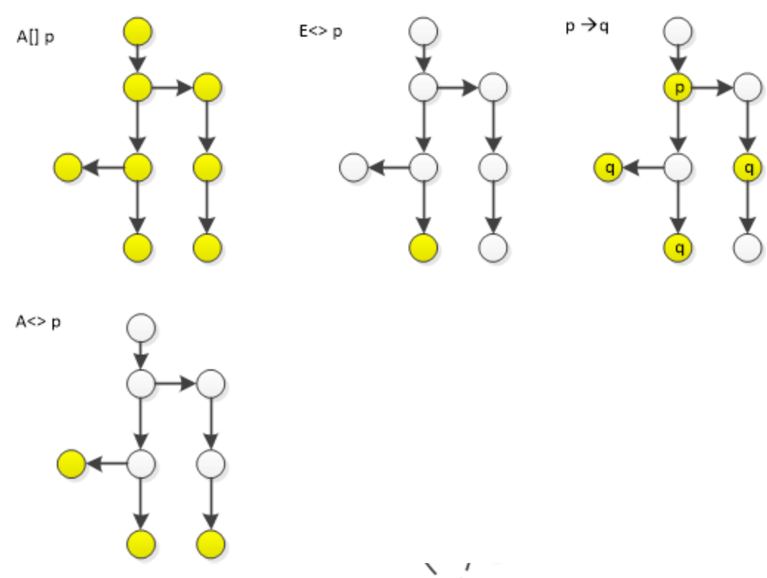
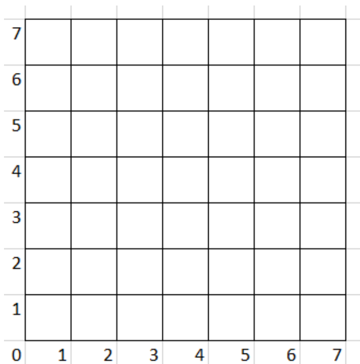


Figure 9: Different types of TCTL queries and their expressions in UPPAAL

Abstraction of map and movement

- Map Abstraction



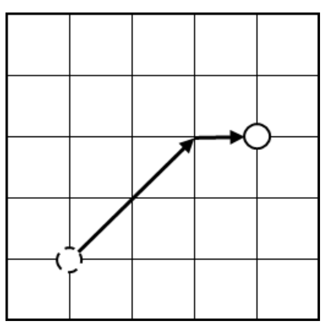
2-D Cartesian grid

$$f_1 : \mathbb{R}_+^2 \rightarrow \mathbb{Z}_+^2 \quad f(x, y) = (z_x, z_y)$$

$$x - \frac{\epsilon}{2} \leqslant z_x \leqslant x + \frac{\epsilon}{2}, \text{ and } y - \frac{\epsilon}{2} \leqslant z_y \leqslant y + \frac{\epsilon}{2}$$

Map Equation

- Movements Abstraction



Movements of AWL and moving obstacles

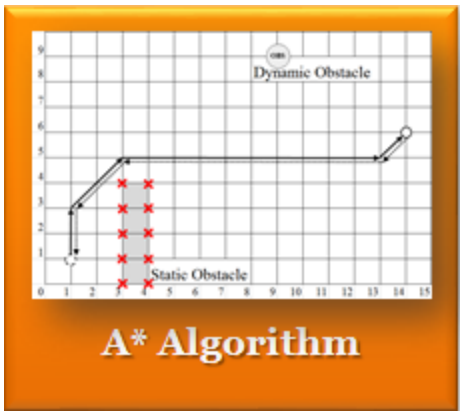
$$p = (z_{x_0}, z_{y_0})(z_{x_1}, z_{y_1}) \cdots (z_{x_{n-1}}, z_{y_{n-1}})(z_{x_n}, z_{y_n})$$

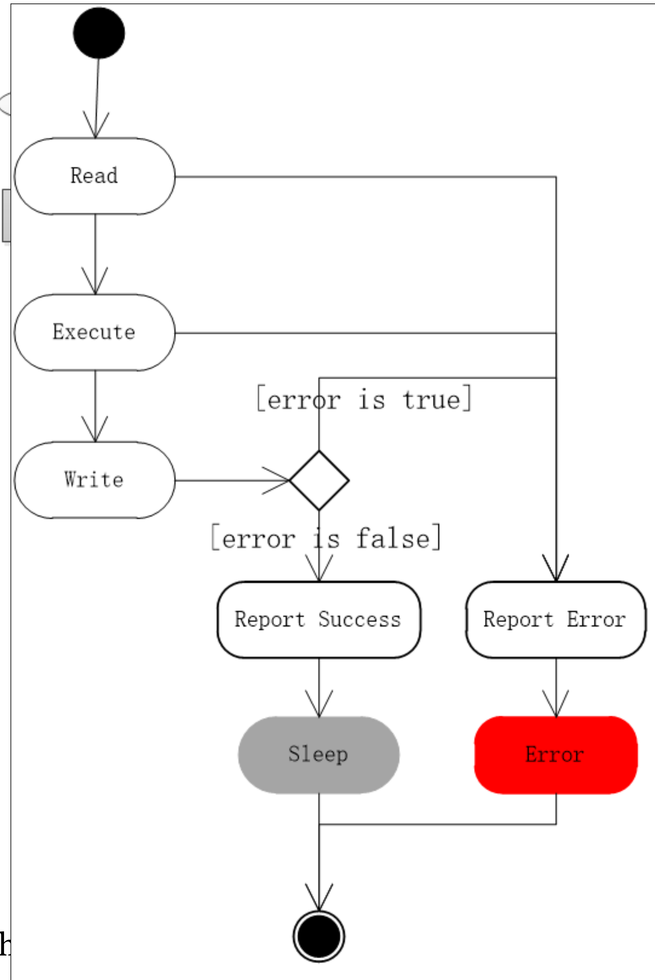
$$\begin{cases} z_{x_i} = z_{x_{i-1}} \pm v, \text{ where } x_i \geq 1 \\ z_{y_i} = z_{y_{i-1}} \pm v, \text{ where } y_i \geq 1 \end{cases}$$

Three types of forbidden movements

Movements Equation

Formal model of tasks and algorithms





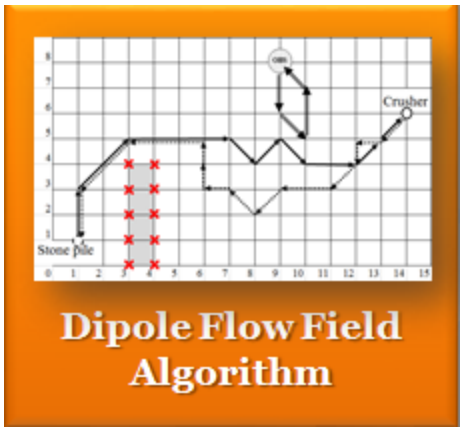
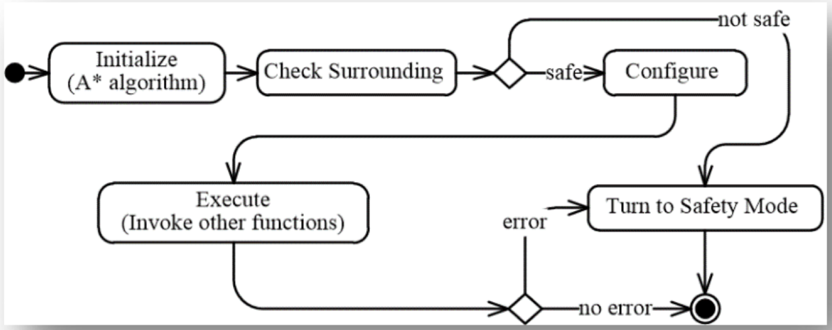
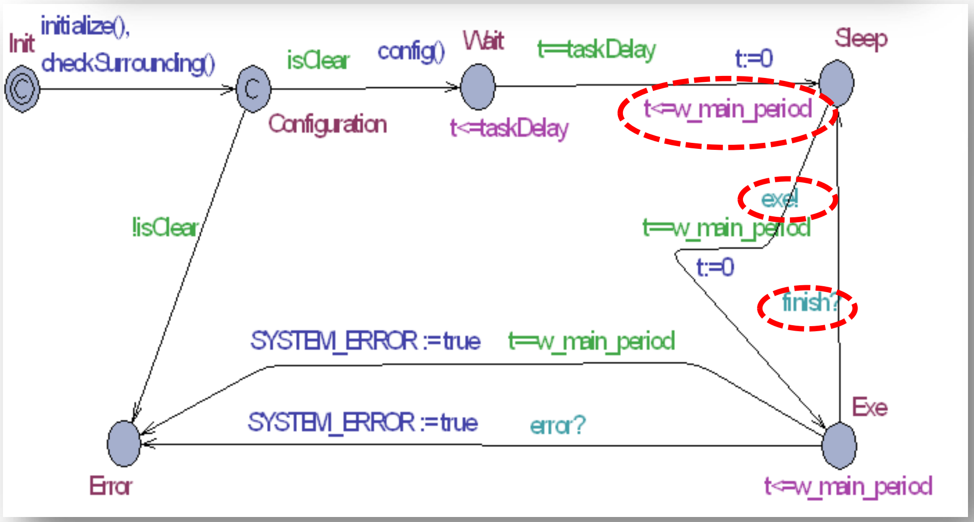


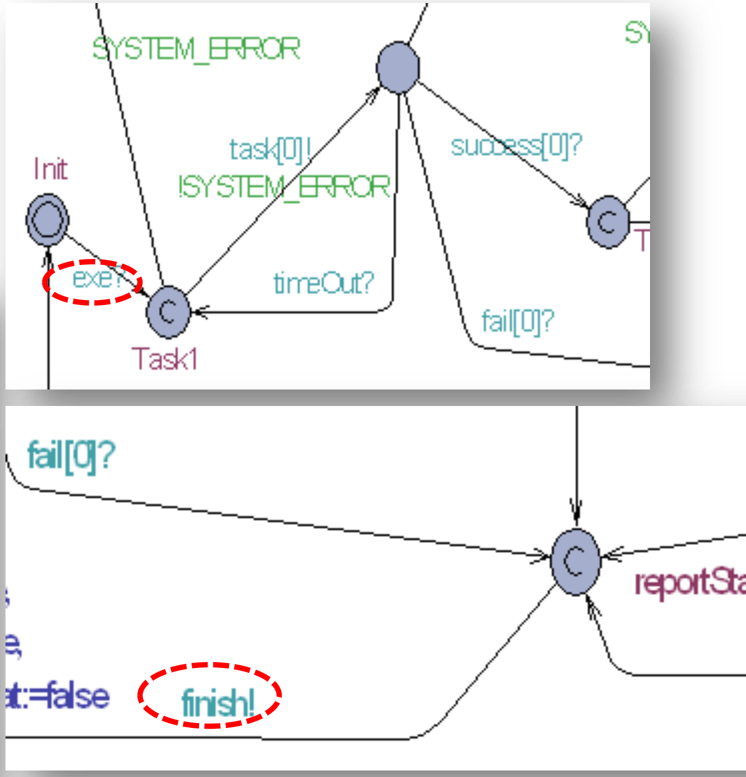
Figure: Model th ns in TA

Mapping activity diagrams to TA



Activity Diagrams





(A) TA of Main Task

(B) TA of Execution Function

Timed Automata (TA) in UPPAAL

Overview of the system model

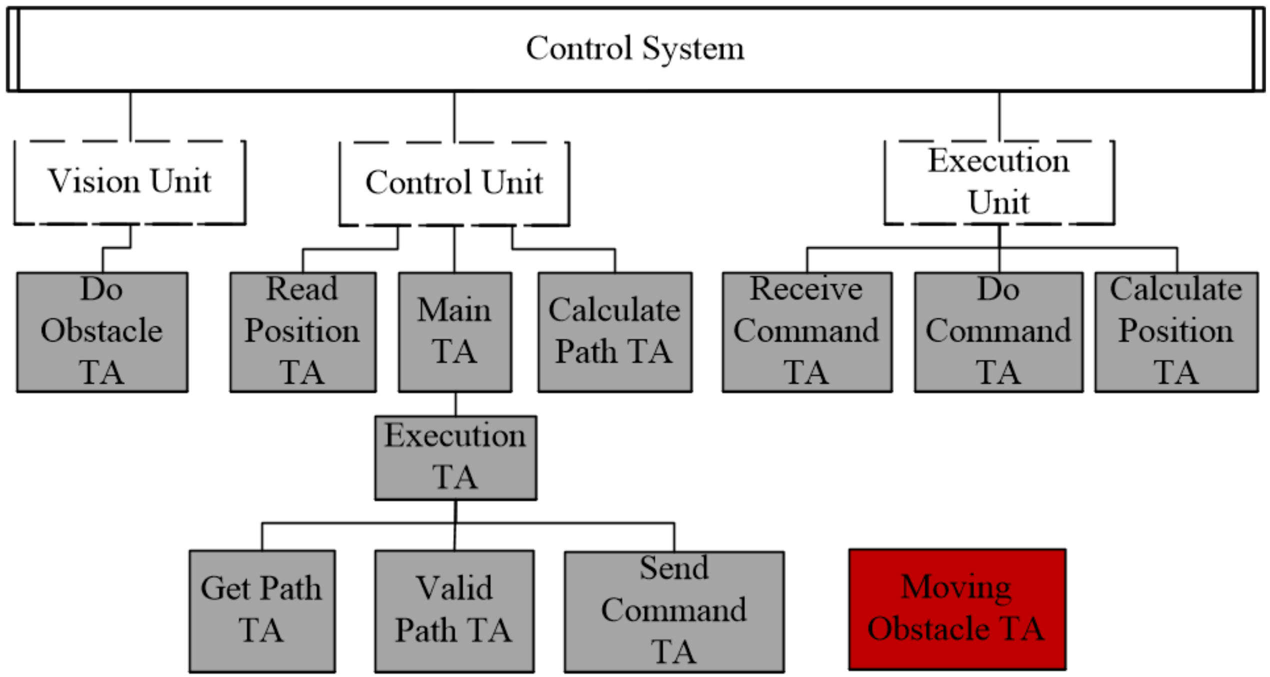
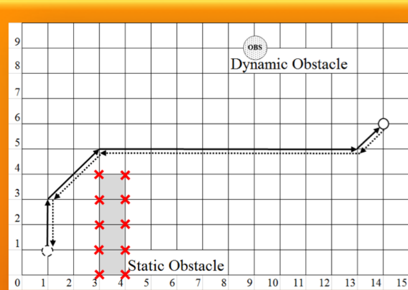


Figure 11: Tasks and functions of system
Figure 12: TA allocation in the control system, TA for moving obstacle

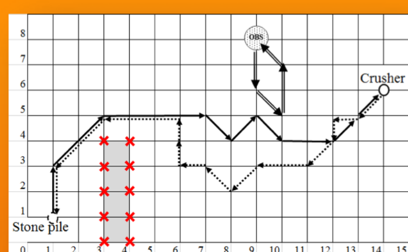
Modeling the path-planning and collision-avoidance algorithms



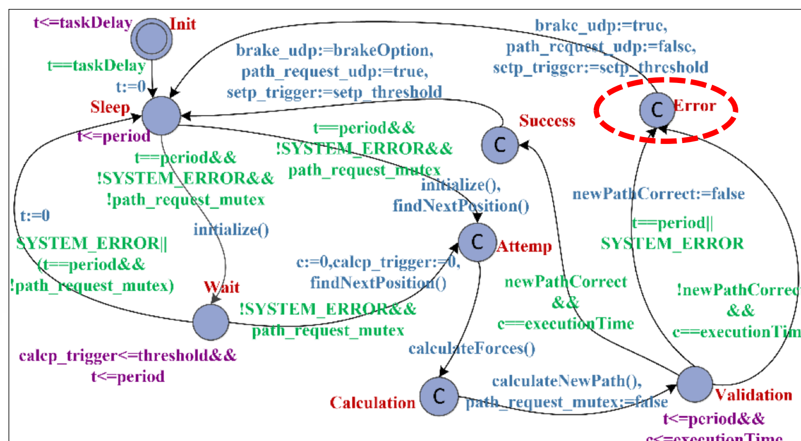
A* Algorithm

```
void AStart()
{
    Point ts, ps;
    int i, j;
    bool findEnd=false;

    insert(open, start);

    while ((open.listLen != 0) && (!findEnd))
```

C-code functions



Dipole Flow Field Algorithm



TA

Model for data communication

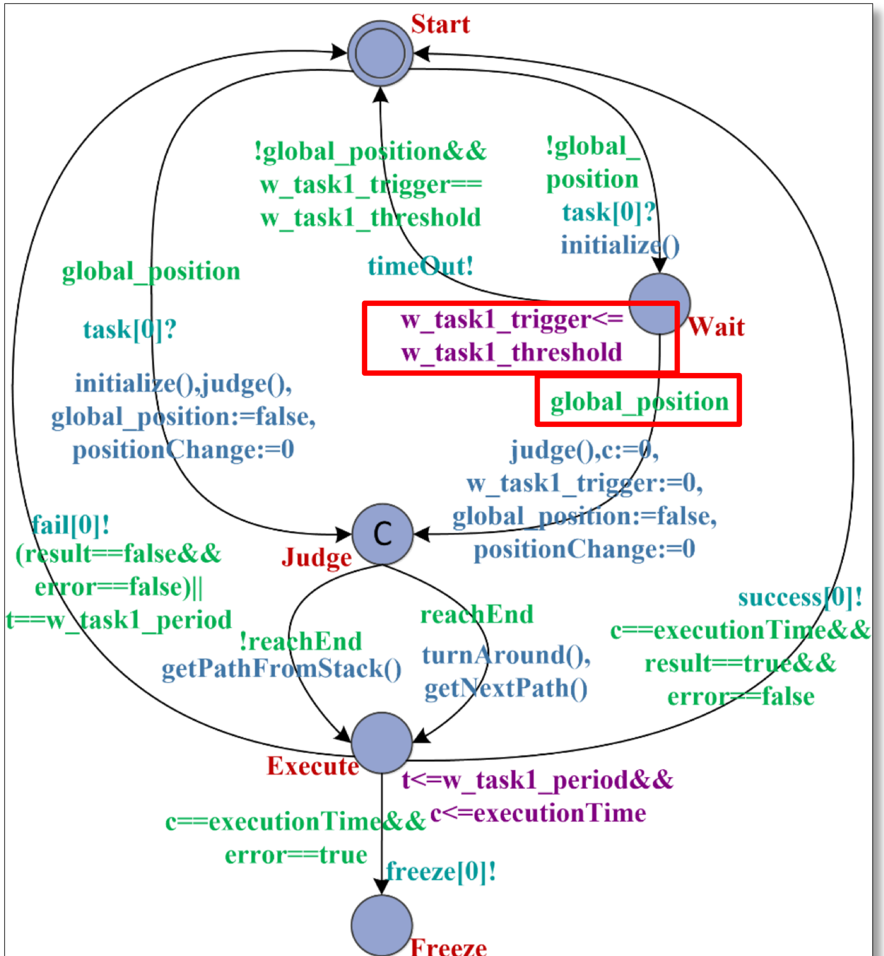


Figure 13: a TA of a task waiting for data “wheel loader’s position” from another task

Data communication between tasks: global variables, clocks/channels:

a) Crucial signals: channels, e.g., **freeze, fail**, etc.

b) Asynchronous signals: clocks, e.g., **w_task1_trigger <= w_task1_threshold**.

Formal Verification - Initial Path Computation

Initial Path Computation: during initialization, an AWL must compute an initial path to the destination, which ought to avoid all the static obstacles identified in the quarry

Formalize requirement

Query
Q1.0: E<> mainTask.Wait
Q1.1: A<> mainTask.Wait imply lenOfPathStack > 0
Q1.2: E<> currentPosition == pile and destination == crusher
Q1.3: (currentPosition == pile and destination == crusher) −> currentPosition == crusher
Q1.4: E<> currentPosition == crusher and destination == pile
Q1.5: (currentPosition == crusher and destination == pile) −> currentPosition == pile
Q1.6: A[] forall(i:int[0,9]) currentPosition != staticObstacle[i]

Formal Verification - Obstacle Avoidance

Obstacle Avoidance: AWLs must avoid static and dynamic objects around them in due time before returning to the initial path

Formalize requirement

Q2.0: A[] currentPosition != currentObstacle
Q1.3: (currentPosition == pile and destination == crusher) –> currentPosition == crusher
Q1.4: E<> currentPosition == crusher and destination == pile
Q1.5: (currentPosition == crusher and destination == pile) –> currentPosition == pile

Formal Verification – Reaction to Errors

Mode Switch Mode A: if the information of obstacles cannot be reported to the control unit, which is very dangerous, AWL must freeze its motion within 20 time units.

Formalize requirement

Q3.1: E<> errorStart == true
Q3.2: error_start==true –> (SYSTEM_ERROR==true and reaction_time<=20)

Formal Verification - End-to-end Deadline

End-to-end Deadline: to guarantee a certain productivity, AWLs must finish one cruise within 2200 time units.

Formalize requirement

```
Q4.0: (currentPosition==pile and destination==crusher) --> (currentPosition==pile and destination==pile and gClock <= 2200)
```

Formal Verification - results

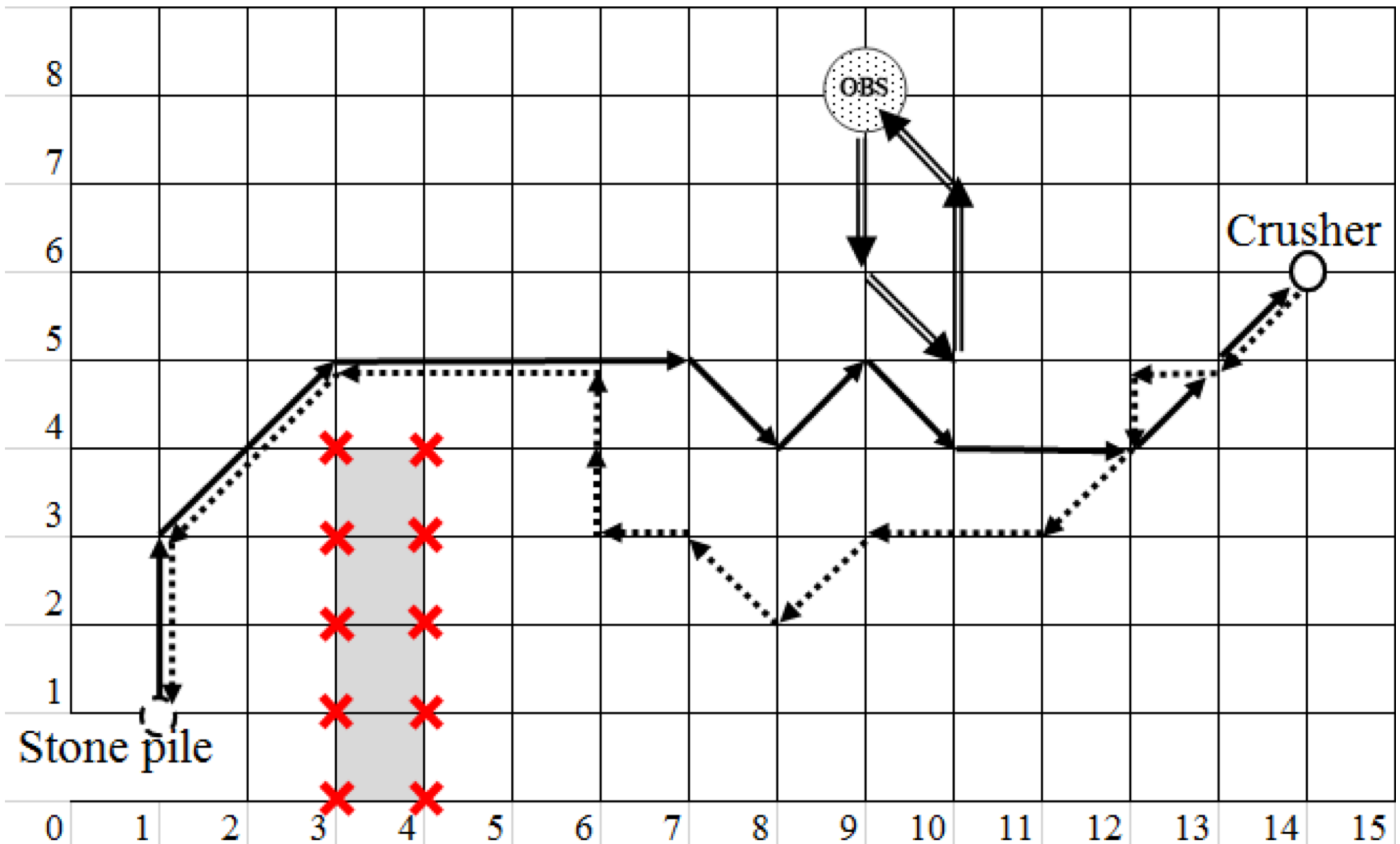
Table 1: Verification queries and results

Requirement	Query	Result	States explored	Time
Initial path computation	Q1.0: E<> mainTask.Wait	Pass	2	110 ms
	Q1.1: A<> mainTask.Wait imply lenOfPathStack > 0	Pass	8780	484 ms
	Q1.2: E<> currentPosition == pile and destination == crusher	Pass	1	0 ms
	Q1.3: (currentPosition == pile and destination == crusher) –> currentPosition == crusher	Pass	14191	1125 ms
	Q1.4: E<> currentPosition == crusher and destination == pile	Pass	2339	297 ms
	Q1.5: (currentPosition == crusher and destination == pile) –> currentPosition == pile	Pass	14204	782 ms
	Q1.6: A[] forall(i:int[0,9]) currentPosition != staticObstacle[i]	Pass	8780	485 ms
Obstacle avoidance	Q2.0: A[] currentPosition != currentObstacle	Pass	125941	6297 ms
	Q1.3: (currentPosition == pile and destination == crusher) –> currentPosition == crusher	Pass	227646	13969 ms
	Q1.4: E<> currentPosition == crusher and destination == pile	Pass	2678	375 ms
	Q1.5: (currentPosition == crusher and destination == pile) –> currentPosition == pile	Pass	192406	10656 ms
Mode switch: error A	Q3.1: E<> errorStart == true	Pass	30	234 ms
	Q3.2: error_start==true –> (SYSTEM_ERROR==true and reaction_time<=20)	Pass	91	250 ms
Mode switch: error B	Q3.1: E<> errorStart == true	Pass	29	234 ms
	Q3.2: error_start==true –> (SYSTEM_ERROR==true and reaction_time<=15)	Pass	320	266 ms
End-to-end deadline	Q4.0: (currentPosition==pile and destination==crusher) –> (currentPosition==pile and destination==pile and gClock <= 2200)	Pass	590326	36641 ms

PASS

36641 ms

What else did we observe?



Conclusion

- We have created a formal model of an industrial prototype of an AWL and its working environment
  - a discrete map and a TA for a moving obstacle
  - 11 TA for algorithms and tasks in the control system
  - encoded computations in the C-code functions of TA
- We have verified the system model and the algorithms against AWL's requirements
  - functional requirements
  - timing requirements
- Counter-examples found by exhaustive verification are helpful for future optimization of system design and algorithms.

Lessons learned and future work

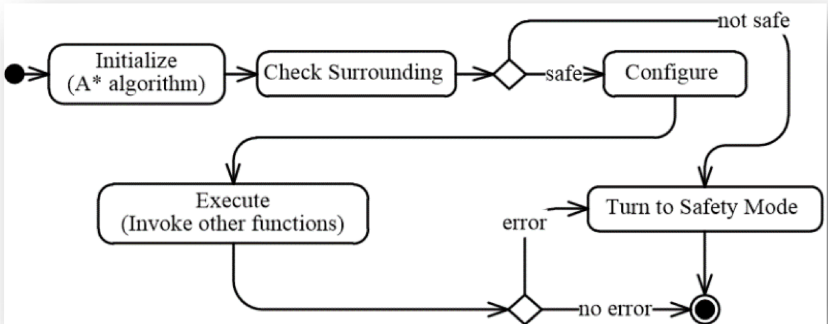
- Lack of floating-point value support in UPPAAL
  - More accurate path-planning and collision-avoidance algorithms need real numbers
  - UPPAAL model only supports integers
- Limitations of Dipole Flow Field algorithm applied in collision avoidance
- Hierarchical verification model/method is needed for more complex system model
  - Discrete model and exhaustive verification: decision-making component
  - Continuous model and statistical verification: real-valued map and dynamics, any-angle path, etc.

Thank you for listening!

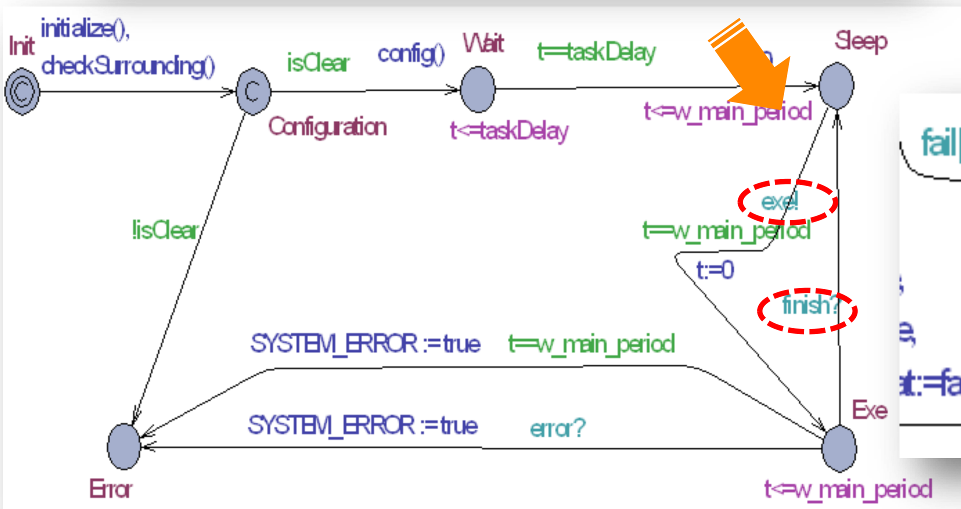
Rong Gu (rong.gu@mdh.se)

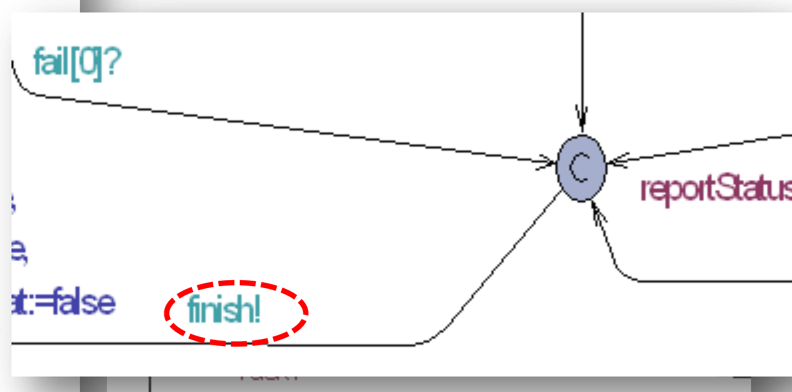
MÄLARDALEN UNIVERSITY SWEDEN

Mapping activity diagrams to TA



Activity Diagrams





(A) TA of Main Task

(B) TA of Execution Function

Timed Automata (TA) in UPPAAL