Bachelor Thesis in Computer science DVA331

MODEL CHECKED REINFORCEMENT LEARNING FOR MULTI-AGENT PLANNING

Erik Wetterholm
ewm20004@student.mdu.se

Examiner: Cristina Seceleanu
Mälardalen University, Västerås, Sweden

Supervisor: Rong Gu
Mälardalen University, Västerås, Sweden

2023-08-30
Abstract

Autonomous systems, or agents as they sometimes are called can be anything from drones, self-driving cars, or autonomous construction equipment. The systems are often given tasks of accomplishing missions in a group or more. This may require that they can work within the same area without colliding or disturbing other agents’ tasks. There are several tools for planning and designing such systems, one of them being UPPAAL STRATEGO. Multi-agent planning (MAP) is about planning actions in optimal ways such that the agents can accomplish their mission efficiently. A method of doing this named MCRL, utilizes Q learning as the algorithm for finding an optimal plan. These plans then need to be verified to ensure that they can accomplish what a user intended within the allowed time, something that UPPAAL STRATEGO can do. This is because a Q-learning algorithm does not have a correctness guarantee. Using this method alleviates the state-explosion problem that exists with an increasing number of agents. Using UPPAAL STRATEGO it is also possible to acquire the best and worst-case execution time (BCET and WCET) and their corresponding traces. This thesis aims to obtain the BCET and WCET and their corresponding traces in the model.
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td>2</td>
</tr>
<tr>
<td>Contents</td>
<td>3</td>
</tr>
<tr>
<td>1. Introduction</td>
<td>4</td>
</tr>
<tr>
<td>2. Background</td>
<td>6</td>
</tr>
<tr>
<td>2.1 UPPAAL and Timed Automata</td>
<td>6</td>
</tr>
<tr>
<td>2.2 UPPAAL STRATEGO and Timed Games</td>
<td>6</td>
</tr>
<tr>
<td>2.3 Q-learning</td>
<td>6</td>
</tr>
<tr>
<td>2.4 MALTA</td>
<td>7</td>
</tr>
<tr>
<td>3. Research Problem</td>
<td>8</td>
</tr>
<tr>
<td>3.1 Purpose</td>
<td>8</td>
</tr>
<tr>
<td>3.2 Problem formulation</td>
<td>8</td>
</tr>
<tr>
<td>3.2 Motivation and Goals</td>
<td>8</td>
</tr>
<tr>
<td>4. Methods</td>
<td>9</td>
</tr>
<tr>
<td>4.1 Research methods</td>
<td>9</td>
</tr>
<tr>
<td>4.2 WCET and BCET</td>
<td>10</td>
</tr>
<tr>
<td>4.3 Generating the trace</td>
<td>10</td>
</tr>
<tr>
<td>4.4 Creating the parser and parsing traces</td>
<td>10</td>
</tr>
<tr>
<td>5. Result</td>
<td>11</td>
</tr>
<tr>
<td>5.1 WCET and BCET</td>
<td>11</td>
</tr>
<tr>
<td>5.2 Generating and parsing the trace</td>
<td>12</td>
</tr>
<tr>
<td>6. Discussion</td>
<td>13</td>
</tr>
<tr>
<td>6.1 WCET and BCET</td>
<td>13</td>
</tr>
<tr>
<td>6.1 Parser</td>
<td>13</td>
</tr>
<tr>
<td>8.1 Previous works</td>
<td>13</td>
</tr>
<tr>
<td>7. Related Work</td>
<td>14</td>
</tr>
<tr>
<td>8. Conclusions</td>
<td>15</td>
</tr>
<tr>
<td>8.1 Future work</td>
<td>15</td>
</tr>
<tr>
<td>References</td>
<td>16</td>
</tr>
</tbody>
</table>
1. **Introduction**

A system containing autonomous systems requires planning to ensure the safety of both the equipment and personnel that might work close together. Autonomous systems can be described as a system that can function and collaborate with others without the need for human intervention [1]. Autonomous systems are used to achieve goals despite a changing environment. An example of a changing environment where autonomous systems might be used is in factories where robots can move objects around while humans operate in the same area. One group of these robots might have the task of moving some items to another area where the next group of robots then needs to photograph them. Since these robots depend on each other it is important that a mission plan can guarantee that they will complete all of these tasks within their given time frame. It is for this reason that the execution time for robots and their different objectives are defined to be between the worst execution time WCET and the best execution time BCET. If a robot for some reason would not be able to complete their objective before another robot wants its place, then it is important that the mission planner is able to adapt to that and redirect the latter if that is the most efficient thing to do, which is something the value for WCET can help with. This value can however not just be the sum of all WCET of tasks from each of the agent's objectives since this might not always be a realistic value. The reason for this is the changing environment that might impact traveling times for example. It is for this reason that the WCET for the entire mission is valuable since that will include possible changes that an agent might need to make. The value for BCET is also very useful since that can help with enabling and increasing the amount of collaborations and help users configure the environment such that it achieves the BCET. But for a user to know if the BCET has been achieved or how it could be done they first need the trace for the BCET.

Planning the routes and actions of autonomous systems can be done in several ways. A new method proposed in [15] utilizes Q learning [8] for MAP (Multi-agent planning). A plan for multiple agents collaborating to accomplish the same goal. This new method then uses UPPAAL STRATEGO [3] for verifying plans. One of the existing tools for MAP is MALTA [2]. MALTA is a toolchain for the mission planning of autonomous systems. It consists of three components: front-end, middleware, and back-end. The front end is a GUI with a programming interface where the user can create missions for autonomous systems. The mission area that can be configured within the front end might include forbidden areas, where an agent is forbidden to move or tasks that need to be completed. It will also include the positions of the agents within the system. The mission will then be sent to the middleware where it calculates the paths and these paths are then sent to the backend to calculate the mission plan, including the task schedule and path plan. After the computation, the generated mission is visualized in the GUI. Each autonomous system can be given requirements of what it must possess, cameras or other sensors. The middleware is for path planning and model generation. The middleware can then make use of the back end to check the models and run task scheduling with the help of UPPAAL STRATEGO [3]. What this thesis and the research behind it contributes with is a way to acquire the WCET and BCET for a multi-agent plan and an easy and efficient way to convert the BCET and WCET traces that would result from that into something that the tool MALTA easily can use.

The contribution of this thesis is to continue the work on MCRL by creating a parser that can convert the output from UPPAAL STRATEGO into something that MALTA can read and algorithms for retrieving values for WCET and BCET that can help users develop and test models that will be used with MALTA. The questions it aims to answer are:

1. What algorithm can be used to obtain the BCET and WCET (best-case and worst-case execution time)?
2. How can the execution traces of BCET and WCET be extracted from the model?
3. How can the resulting trace be converted into a format that can be used in MALTA?

UPPAAL STRATEGO’s interface enables a user to send queries to a loaded model. The functions that can obtain the values for BCET and WCET will be created with the help of the available documentation that exists for UPPAAL STRATEGO and its query language. Values for WCET might not be possible to create in the beginning since that requires that there is an endpoint where the clocks stop or else the value for WCET will be infinite. If the models that are being used do not include this then experiments will need to be made on them to
find the changes that will fix this. Step two is then to get the trace from the resulting values of BCET and WCET. This is once again done with the help of UPPAAL STRATEGO and the ability to send specific queries with certain instructions, in this case instructions to get the fastest path to the goal that is a completed mission. This will then generate a trace that can be viewed in UPPAAL but also exported as an XTR file. XTR stands for XTrkCAD Demo file [29], which was introduced in UPPAAL 3.0 to store the traces. To visualize the trace in MALTA then requires that the resulting XTR file is converted into an XML formatted in a way that can be read by MALTA. A parser was for this reason developed in this thesis and serves as the bridge between UPPAAL STRATEGO and MALTA.
2. Background

2.1 UPPAAL and Timed Automata

Timed automata [10] (TA) is an extension of finite automata that includes clock variables. These clocks are then able to show the amount of elapsed time since they began. During transitions, the clock variables can be compared to integer values that can then be used as guards for the transition. The value of the clock variables all increase with the same amount at the same speed. UPPAAL [3] is a tool developed as a collaboration between the Department of Information at Uppsala University [11] in Sweden and the Department of Computer Science at Aalborg University in Denmark [12]. It is a tool created for modeling, simulation, and verifying real-time systems. UPPAAL uses a modeling formalism called UPPAAL timed automata (UTA), an extension of TA. The different parts of UPPAAL can help with creating, validating, and verifying real-time systems. UPPAAL can be used to create a network of TA, which can then be tested with the help of UPPAAL'S built-in ability to give queries to models. To better show what a timed automata in UPPAAL might look like we can consider the following example taken from UPPAAL’S tutorial [13]. The example below aims to mimic the function of a lamp that has two light settings. The two states where the lamp is turned on are “low” and “bright”. The initial location, visualized by a double ring, is for the lamp “off” and for the user “idle”. The model also includes a clock variable “y”, that is used as a guard. On the edge between “off” and “low” the value of “y” is reset to zero. shown by “y:= 0”. it is also shown that there is a synchronized edge shown with “press?” and will only be taken when the user takes the edge with “press!” If the user were to take the edge with “press!” before the next five seconds the lamp would move to location “bright” since the edge going back to “off” has a guard that forbids movement when “y” is below five, and the opposite goes for the edge from “low” to “bright”.

![Diagram of Lamp and User](image)

*Fig 1. Synchronized model of a lamp and a user in UPPAAL [13]*

2.2 UPPAAL STRATEGO and Timed Games

UPPAAL STRATEGO [3] is a variant of UPPAAL that is built on games and strategies. A game is a model that consists of several players that compete against each other with different and independent objectives. A game is won when a player manages to reach the goal. A strategy is the list of a player's actions at any possible state that has the end goal of reaching the player's goal. Timed games are an extension of timed automata by categorizing actions into two classes, controllable and uncontrollable actions.

2.3 Q-learning

Q-learning (QL) [8] is a form of model-free reinforcement learning (RL) [9]. RL and QL differ from other algorithms like supervised learning [9] in the way that it does not know what are good actions and what are bad actions. It instead works with rewards, or reinforcement when it reaches a goal. An example of this would be learning how to fly an airplane and there is no input on what actions to take and when. The only output to the algorithm is reinforcement when it manages to fly for a specified amount of time or distance. The algorithm will
then work towards achieving the highest amount of reinforcement. In the case for UPPAL Stratego [3] it uses it with the help of the Monte Carlo method [27], a way of estimating the result of an uncertain environment. This is then improved upon with Q-learning until a result is found.

In Fig.2 the goal of the plane is to reach the “Goal” without hitting any mountains. For each tile that plane flies it loses one point and crashing into a mountain will result in fifty points being lost and the game ending. But reaching fuel will earn it five points and if the plane reaches the goal it will gain fifty points. The Q-table is then used to find the best path with the most points associated with it, since that is what drives a Q-learning algorithm.

2.4 MALTA

MALTA [2] is a tool developed by Rong Gu, Afshin Ameri from Mälardalens University, and Eduard Baranov from Catholic University of Leuven. It is a collection of tools for planning, visualizing, and validating plans for autonomous systems. It consists of three parts, front-end, middleware, and back-end. The front-end is the graphical interface along with a programming interface named MMT. The middleware of MALTA is for path planning and generating models, for this MALTA uses TAMAA-DALi(Timed-Automata based Mission planner for Autonomous Agents-DALi) [6]. DALi [28] is an algorithm for planning paths and considers limitations that the environment might pose. DALi consists of two steps, planning long paths and planning short paths. The long planning creates the path before the path is taken by agents and is compared to the short planner, relatively slow to create. The short planner is then used during the navigation and helps with finding the shortest path for avoiding obstacles that might appear. TAMAA-DALi receives and to some extent processes information from MMT. The backend is for scheduling tasks which include verifying a reachable goal. For this MALTA utilizes TAMAA [9], a timed automata-based mission planner for autonomous agents. It does this with the help of UPPAAL [3].
Fig 3. The different parts that MALTA contains (this figure is adapted from the literature [2])
3. Research Problem

Autonomous systems are often designed to work in a group to solve a variety of problems. This could be one set of autonomous machines grabbing something off the shelf and delivering it to a drop-off point where another group of machines picks it up and loads it into a truck. This requires the robot to move in its designated area while completing its goals while also ensuring that all of the safety criteria are met. MAP (Multi-agent planning) is the planning and design of such systems. For this, there exist a couple of tools. For this thesis the relevant ones are MALTA [1] and UPPAAL [5]. UPPAAL is used to create the systems and from that get the BCET (best-case execution time) and WCET (worst-case execution time). It is from these values that a trace can be acquired, the most relevant being BCET. The trace then needs to be parsed before it is sent to MALTA where it is shown to the user.

3.1 Purpose

The purpose of figuring out the BCET and WCET is so the user can be sure that the model they have created follows eventual restrictions. For this, it is also necessary to show how any eventual paths/traces look. For this, a parser is used to parse the trace that is outputted from UPPAAL into something that MALTA can read where it is later shown to the user.

3.2 Problem formulation

1. What algorithm can be used to obtain the BCET and WCET (best-case and worst-case execution time)?
2. How can the execution traces of BCET and WCET be extracted from the model?
3. How can the resulting trace be converted into a format that can be used in MALTA?

3.2 Motivation and Goals

Development and testing of autonomous systems is something that is more relevant than ever with the increasing amount of robots and systems on worksites. Knowledge about the values for WCET and BCET can help with development of autonomous systems and help alleviate bottlenecks that might be slowing the system down. Since Q-learning does not have a correctness guarantee it will be useful for users to see that the robots finish on time and thereby ensures a suratin level of productivity.
4. Methods

4.1 Research methods

![Research process diagram](image)

**Fig 4. Research process**

In Fig. 4 the process that this thesis takes is shown. Step 1 consists of creating and defining all the research questions that would be relevant to achieving the final goal. This goal was created to help with the development of MALTA. Step 2 was primarily knowledge review and setting up and becoming familiar with all the necessary things that were already known. In this case that was UPPAAL STRATEGO since that is what MALTA would use and since this was built for MALTA that was the only option. Step 3 is where the method and type of method are chosen. In this case, experiments are chosen to answer question 1 and 2. Question 3 is solved by creating a parser so that only needs implementation. Step 4 is where execution of the methods are done and where code for the parser is created and implemented. Step 5 consists of testing executing all of the steps until an output from UPPAAL could be used in MALTA. If that did not create the required result then step 4 is used again to fix possible problems. The final step is to evaluate the results to see if they are sufficient and will be able to perform the task that they are supposed to.

To answer the research questions connected to this thesis a pre-existing model for autonomous systems is used [20]. The first part of the work is to acquire a value for WCET and BCET and all of the work for that is done in UPPAAL STRATEGO 4.1.20 Stratego 8 [21]. Following that, a parser is developed that could take the trace generated by UPPAAL and turn it into something that could be used by MALTA.

The model used in this thesis named “game0.xml” [20] can be found in the github associated with this thesis. It consists of several systems and a declaration where values such as number of agents and milestones are defined along with some functions that help with testing if the game is won.

![Game0.xml diagram](image)

**Fig 5. “game0.xml”, “Referee”**
The “Referee” shown in Fig. 5 is used in the model to tell when the goal has been achieved and when the times of BCET and WCET should be measured. The model uses a function called isGameWon() to do this which tests if the goal has been achieved.

### 4.2 WCET and BCET

Obtaining values for WCET and BCET first requires using a built-in function in the model, “game0” in this case that creates a strategy for that instance and model. This is done each time the model is changed since that would render the policy obsolete and out-of-date. Then the first part is to create the functions that could retrieve the WCET and BCET if possible. This might not be the case since it requires that the model is built for it. Creating the functions is done with the help of documentation on UPPAAL [26]. The second part is to ensure that the model could generate the correct values. For this experiments are done on it until realistic values are achieved.

### 4.3 Generating the trace

Generating a trace first required a function that could reach the goal-state since what we want is a game that is won. When a function is created the option to create the fastest trace needs to be selected before running the function. This would create the new trace in the “Symbolic Simulator” tab. This could then be saved as an XTR file [22]. Now a parser needed to be created and used to convert the XTR file to an XML file that could be used in MALTA. The parser will need an intermediate file that can translate names and such from the XTR file. For this, the built-in tool “Veriftya” is included with UPPAAL. Running the command “UPPAAL_COMPILE_ONLY = 1 ./verifyta name.xtr -> name.if” when running the console in the following folder “/uppaal-4.1.20-stratego-8-linux64/bin” with the trace file located in the same folder.

### 4.4 Creating the parser and parsing traces

There is an existing parser for UPPAAL [23] that uses the XTR and IF to parse traces. The parser for MALTA was then built on an existing one, “tracer.cpp”. For printing to an XML file, the open-source tool TinyXML [24] was used to ensure that a correct XML file was created.

The algorithms for obtaining the WCET and BCET were then run on similar models with different amounts of agents, milestones, tasks, and devices. This was done to see if there were any significant differences in time consumption between the different models. This was then done for the parsing of traces to see if there were any of the attributes that had a significant impact on the executing time. These tests were run five times for each of the different models to get an average value.

The input for the parser is then the trace file created and the intermediate file “.if” and from that a “.xtr” file is created with the correct format and time it took for the parser to parse the trace.
5. Result

5.1 WCET and BCET

The algorithm that can be used to get the WCET for the model is “sup{isGameWon()}:timeConsumption under policy”. It requires that, when the game is won, the clock timeConsumption is stopped.

The algorithm that can be used to get the BCET for the model is “inf{isGameWon()}:timeConsumption under policy”. “policy” is in this case the name of the strategy that is required by UPPAAL STRATEGO. The strategy was created by using the query:

```
"strategy policy = maxE(var*50 - timeConsumption)[<=TOTALTIME]/{
Rong.location,
m_stone0Secondary0_TK0.location, m_secondary0Stone0_TK0.location,
m_stone1Secondary0_TK0.location, m_secondary0Stone1_TK0.location,
m_primary0Secondary0_TK0.location, m_secondary0Primary0_TK0.location,
t_digging_WL0.location, t_unloading_WL0.location,
t_digging_WL1.location, t_unloading_WL1.location,
t_loadingFromWL_TK0.location, t_loadingFromPrimary_TK0.location, t_unloadingToSecond_TK0.location,
var,
agents[0].a_position, agents[0].a_monitors[0], agents[0].a_task.t_id, agents[0].a_task.t_deviceUse.d_id,
agents[0].a_task.t_deviceUse.d_position, agents[0].a_status[ID_DIG_WHEELLOADER],
agents[0].a_status[ID_UNLOAD_TO_TRUCKS_WHEELLOADER],
agents[1].a_position, agents[1].a_monitors[0], agents[1].a_task.t_id, agents[1].a_task.t_deviceUse.d_id,
agents[1].a_task.t_deviceUse.d_position, agents[1].a_status[ID_DIG_WHEELLOADER],
agents[1].a_status[ID_UNLOAD_TO_TRUCKS_WHEELLOADER],
agents[2].a_position, agents[2].a_monitors[0], agents[2].a_task.t_id, agents[2].a_task.t_deviceUse.d_id,
agents[2].a_task.t_deviceUse.d_position, agents[2].a_status[ID_DIG_WHEELLOADER],
agents[2].a_status[ID_UNLOAD_TO_TRUCKS_WHEELLOADER]}
} -> {<: Rong.Win || Rong.Lose}
```

Both the queries for BCET and WCET require that there is an existing policy and that timeConsumption only stops when the game is won and is never changed after that since that would change the resulting time. Experimenting with the model showed that one solution is to make the referee's final states urgent since that will disable the ability to pass time within the model.

---

![Diagram](image.png)

Fig 6. "game0.xml", "Referee" with the win node changed to urgent.
5.2 Generating and parsing the trace

The reachability query constrained by a given policy that can be used to generate a trace for the fastest won game is “E<> isGameWon() under policy”. To get the fastest time, the option for the fastest diagnostic trace needs to be enabled. This will then generate the fastest trace in the symbolic simulator tab.

The parser can be found at [20].

The following table shows five times and the average time for creating the policy, running the query and parsing with the corresponding values for agents and milestones. I also includes the number of iterations for each agent and the number of lines in the output file.

<table>
<thead>
<tr>
<th></th>
<th>game0</th>
<th>game1</th>
<th>game4</th>
<th>game6</th>
<th>game9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agents</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Milestones</td>
<td>4</td>
<td>6</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Creating policy time/s</td>
<td>9,922</td>
<td>20,031</td>
<td>15,859</td>
<td>22,094</td>
<td>15,063</td>
</tr>
<tr>
<td></td>
<td>10,407</td>
<td>19,735</td>
<td>16,672</td>
<td>23,5</td>
<td>17,39</td>
</tr>
<tr>
<td></td>
<td>10,515</td>
<td>24,5</td>
<td>15,438</td>
<td>24,875</td>
<td>16,422</td>
</tr>
<tr>
<td></td>
<td>10,422</td>
<td>23,718</td>
<td>15,891</td>
<td>24,843</td>
<td>14,328</td>
</tr>
<tr>
<td></td>
<td>10,141</td>
<td>22,545</td>
<td>17,453</td>
<td>21,507</td>
<td>15,625</td>
</tr>
<tr>
<td>Average time/s</td>
<td>10,2814</td>
<td>22,1058</td>
<td>16,2626</td>
<td>23,3638</td>
<td>15,7656</td>
</tr>
<tr>
<td>Running query time/s</td>
<td>0,016</td>
<td>0,015</td>
<td>0,016</td>
<td>0,032</td>
<td>0,016</td>
</tr>
<tr>
<td></td>
<td>0,016</td>
<td>0,015</td>
<td>0,016</td>
<td>0,015</td>
<td>0,015</td>
</tr>
<tr>
<td></td>
<td>0,017</td>
<td>0,014</td>
<td>0,015</td>
<td>0,015</td>
<td>0,016</td>
</tr>
<tr>
<td></td>
<td>0,015</td>
<td>0,016</td>
<td>0,015</td>
<td>0,016</td>
<td>0,016</td>
</tr>
<tr>
<td></td>
<td>0,016</td>
<td>0,015</td>
<td>0,016</td>
<td>0,016</td>
<td>0,017</td>
</tr>
<tr>
<td>Average time/s</td>
<td>0,016</td>
<td>0,0156</td>
<td>0,0158</td>
<td>0,0188</td>
<td>0,0162</td>
</tr>
<tr>
<td>Parsing time/s</td>
<td>5,642</td>
<td>7,557</td>
<td>6,933</td>
<td>8,331</td>
<td>4,553</td>
</tr>
<tr>
<td></td>
<td>6,065</td>
<td>7,561</td>
<td>6,686</td>
<td>6,679</td>
<td>5,751</td>
</tr>
<tr>
<td></td>
<td>4,976</td>
<td>6,443</td>
<td>6,266</td>
<td>7,629</td>
<td>5,097</td>
</tr>
<tr>
<td></td>
<td>8,631</td>
<td>6,979</td>
<td>5,933</td>
<td>6,109</td>
<td>4,894</td>
</tr>
<tr>
<td></td>
<td>4,498</td>
<td>5,077</td>
<td>6,162</td>
<td>7,191</td>
<td>5,031</td>
</tr>
<tr>
<td>Average time/s</td>
<td>5,9624</td>
<td>6,7414</td>
<td>6,396</td>
<td>7,1878</td>
<td>5,0652</td>
</tr>
<tr>
<td>Iterations per agent</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Agent 0</td>
<td>5</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Agent 1</td>
<td>0</td>
<td>10</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Agent 2</td>
<td>0</td>
<td>-</td>
<td>0</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>Number of lines in output file</td>
<td>1394</td>
<td>2596</td>
<td>1265</td>
<td>1476</td>
<td>1311</td>
</tr>
</tbody>
</table>

*Fig 7. Execution time for each game with corresponding attributes.*
6. Discussion

Looking at the time it takes for the creation of policies there is a clear difference in time, even compared to other games with the same number of milestones and agents. This is also reflected in the average parsing time. There could be plenty of reasons for this, but of the main ones is the amount of iterations, number of times an agent completes all of their goals, in each trace. Another value that can have an impact on the time is the number of lines in the output file since that shows the number of steps that were taken in the trace.

6.1 WCET and BCET

The values for WCET could only be generated when there was a final state where the clock stops. In the model used in the thesis, this could be done by using the already existing “Referee” since it would go to a milestone that is either “Lose” or “Win” depending on the time and or if the game is won. According to the documentation for UPPAAL [25], it might also be possible to achieve the same result with the use of stopwatches. This would then be implemented in a similar way where the stopwatch is stopped in the same locations that were previously set to urgent.

6.1 Parser

With UPPAAL there existed several tools and parsers that took the “.xtr” file that UPPAAL would output when exporting traces and the intermediate file from “Verifyta” and visualized them in different ways. One of these would output all the information from the trace into the console. It was from this that the parser was built which saved on having to create functions to read the “.xtr” and “.if” files and translating them into something that a human could read. This was important since the way these files represented the traces was either just numbers or a combination of number and dots, not human readable. The amount of the output from the previous output that could be used was close to zero and the main work for the new parser was therefore to manage and adapt the output into something usable by MALTA. One noteworthy thing about the previous parser is that it did not handle time and timings in the trace and this had the consequences that the new parser, developed in this thesis also is unable to do that since they work on the same output. This is quite a significant restriction since being able to show the time is a great tool that is valuable when developing an autonomous system. The time is however not necessary for showing the trace since the order in which the agents do their specific tasks are still done in order and will therefore still show the correct trace.

8.1 Previous works

This thesis builds on a previou paper [15] that proposed a new method for planning of autonomous systems named MCRL (model checked reinforcement learning). In that paper they use UPPAAL STRATEGO to implement their method and their own tool MALTA to generate the models. The parser developed in this paper has the purpose of parsing the output from UPPAAL STRATEGO into an input that MALTA can handle.
7. Related Work

There are a lot of studies and developments around multi-agent systems and model-checking [7, 15, 16, 17, 19]. Many of them focus on either using existing methods to improve existing systems or develop new ones. One of these studies is “Strategy Synthesis for Autonomous Driving in a Moving Block Railway system with UPPAAL STRATEGO” [19]. Using the same tool, UPPAAL STRATEGO that this thesis uses, they are able to create an “abstraction of a moving block signaling system with autonomous driving as a stochastic priced timed game. [19]”. Compared to that study this thesis, despite working within the same subject and with the same tools differs greatly in what the contributions are. Mainly in that this thesis works towards creating a new tool that with the help of UPPAAL can streamline processes of developing new systems and not developing a new system. Since new systems can be created with his new streamlined process.

Another paper [16] puts its focus on the correctness of mission plans for groups of autonomous systems. This includes completing all the tasks until a final goal is achieved, making sure each of the tasks is performed at their corresponding milestone, that each task is executed in the correct order, and finally that each of the tasks are completed within the given time frame. The aim is then to include all of these in one tool that automatically can create mission plans for autonomous systems. The paper does only aim at developing a proof-of-concept for the new way of planning autonomous systems. The goal is to use TAMAA to ensure the correctness of a mission plan. Then to find the best possible ways, the shortest route between two milestones they propose using Theta*. To avoid the increased time that a system with a larger amount of milestones might suffer from, combine this with a Quad-Tree data structure. While this paper in many ways differ from this thesis the correctness of plans is still highly relevant and used with Q-learning the also help with

One paper that focuses on TAMAA is “TAMAA:UPPAAL-based mission planning” [17]. This is highly relevant since this is something that the tool MALTA uses. While development in MALTA is beyond the scope of this thesis, it is still something that is used later in the pipeline when the parsed traces have been sent to MALTA.

Another one [15], more related to this thesis, is a paper proposing a new approach to the already existing MCRL method, that along with mode checking utilizes reinforcement learning to increase the ability to scale models and increase the number of agents. For this, they use UPPAAL STRATEGO which also enables them to ensure that the models are correct by post-verification. Along with this, they use their own tool MALTA for model generation. What this aims to add to this is a way to convert the output from the tool UPPAAL STRATEGO into the input for MALTA, streamlining the process of creating new multi-agent autonomous systems. Doing this will help complete the toolchain found in Fig.1.
8. Conclusions

What this thesis aims to answer was how to acquire values for BCET and WCET of a model. The queries for this are in the following form:

BCET: \( \inf\{isGameWon()\}:timeConsumption \) under policy

WCET: \( \sup\{isGameWon()\}:timeConsumption \) under policy

inf means infimum, sup means suprema. isGameWon() is the condition that needs to be true. The variable timeConsumption is the clock on that inf and sup uses. under policy means using that specific strategy, named policy.

To achieve the value for WCET the model had to be changed such that the win state could stop the “timeConsumption” clock that was used in the model “game0.xml”. This was done by setting the win node of the “Referee” to urgent since, according to UPPAALS documentation, a clock is unable to pass time when a system is in an urgent state. The following question was how a trace could be extracted from these values. This did require a new query, “E <> isGameWin()” under policy” that then could generate the trace. This trace could then, with the help of UPPAAL’s ability to export traces be exported as an “xtr” file. The final, and most important question this thesis aimed to answer was how this resulting trace could be converted into something that MALTA could use. The parser that was created for is able to transform the file into an “.xml” that MALTA can use, but not without drawbacks. The main one is that it is unable to handle time which is something that could be useful in MALTA. Usage of this parser will still help with development and testing of autonomous systems since there now exists an easy way to convert UPPAAL’s output into MALTA’s input.

8.1 Future work

The current parser is as previously stated unable to handle times, which is the biggest downside with it. Future work with a parser for UPPAAL and MALTA would benefit from a complete redesign which could have the added benefit of increasing performance. The main difficulty with adding the ability to read time is that it would mean completely changing the way the “.if” file and “.xtr” file are read.
References


