Bachelor thesis
Computer Science

Radboud University

Statistical Model Checking of a Digital Hydraulic Power Management System

Author:
Rob ten Berge
0740357

First supervisor/assessor:
Prof. dr., Frits Vaandrager
f.vaandrager@cs.ru.nl

Second assessor:
Dr., D. N. Jansen
dnjansen@cs.ru.nl

April 16, 2015

Abstract

This thesis extends on a Finnish case study on a Digital Hydraulic Power Management System, which is an example of a cyber-physical system. Such systems may have strict bounds on properties, which can be proven using modeling techniques and verification queries. Scaling of models to arbitrary size is under ongoing research, which this paper explores using the example of the Digital Hydraulic Power Management System with the tool UPPAAL. Statistical Model Checking sidesteps the state space explosion problem, and this paper contains a practical example of using UPPAAL-SMC to visualize results efficiently. Experiences with UPPAAL-SMC are also discussed.

Contents

Chapter 1

Introduction

This thesis will extend on a Finnish case study by Boström et al on Digital Hydraulic Pomp Motor Systems (DHPMS). They describe a DPHMS as a "universal flow source for hydraulic systems" [4]. As seen in figure 1.1, an arbitrary number of pistons are connected to a rotating shaft powered by an electric engine. Each piston is connected to possibly several outlets and each outlet can have a variable desired amount of pressure. The flow from and to the cylinders is controlled by on/off valves. It is these valves that must be triggered by software at a precise time. If the timing of the valve triggers is off then the system may be damaged over time and eventually cause the hydraulic system to fail. It is therefore crucial for the software to have strict bounds on the worst case response times.

Cyber-physical systems such as a DHPMS may have requirements that dictate strict bounds on properties. Modeling the entire physical system as well as the software may likely result in models that are far too large and thus abstractions and higher level reasoning are required to make statements on the system as a whole.

The original case study was carried out using the tool TIMES [2] and concluded that their approach did not scale. The goal of our research was to re-create the original results regarding response times using UPPAAL, investigate scaling beyond the original results, and then attempt to apply UPPAAL-SMC to the model to calculate averages and maxima using many more cylinders than is calculable with traditional model checking.

The original paper mentioned but did not attempt to apply Statistical Model Checking. It claimed that the SMC extension of UPPAAL only supports uniform and exponential distributions, which is false [5]. Likewise, the TIMES tool makes use of unique priorities among tasks with equal priorities, leading to technically inaccurate results which were barely adressed at all.

This paper extends on the original case study by adding the following results:

- A theoretical analysis of WCRT for DHPMS with arbitrary number of cylinders, found in chapter 2.
- An UPPAAL model that is equivalent but significantly simpler than the TIMES model, explained in chapter 3.
- WCRT analysis scaling to five cylinders, deadlock testing scaling to six cylinders, detailed in section 3.5.
- Extension to UPPAAL-SMC with results based on raw data by Boström, shown in chapter 4.

Our contribution to the field of SMC lies in the practical application of UPPAAL-SMC on a simplified model of a real-life system. Lesssons learned and suggestions for improvements can be found in chapter 5, and final comments on the capabilities and stability of UPPAAL-SMC can be found in chapter 6.

1.1 The DHPMS

The physical details of the DHPMS are explained in a clear way in the paper by Boström et al:
"For simplicity, only one pressure outlet of the DHPMS is considered, i.e, a digital pump-motor is studied. The overall hydraulic architecture of the digital pump motor is presented in Fig. [1.1]. Let N_v be the total number of on/off valves in the DHPMS. Each valve should be opened and closed exactly once in one turn of the rotating shaft. Therefore the software of the DHPMS needs to initiate $2N_v$ triggering events within one period. Let ej denote an event of opening or closing of a valve $j \in [0, 2N_v - 1]$, t_e^j and θ_e^j denote the desired time and angle of the rotating shaft for the event ejrespectively. The angle θ_e^j and all other angles are given relative to a fixed position on the axle. The calculation of θ_e^j depends on factors such as desired hydraulic fluid flow direction (mode), the angular velocity of the shaft ω, the size of the hydraulic chamber and compressibility of the fluid. The details of the calculation of θ_e^j can be found elsewhere [...]. Only ω and t_e^j need to be calculated by the DHPMS, all other parameters can be considered as input data arriving from the rest of the hydraulic system. The angular velocity ω is defined as $\omega = \frac{d\theta}{dt}$,where θ denotes the current angle of the rotating shaft. If ω is constant, the time delay before triggering of the valve t_e^j can be estimated as $t_e^j = \frac{\theta_e^j - \theta}{\omega}$. Hence, estimation of both ω and t_e^j requires measurement of the actual angle of the rotating shaft. This task is solved with an incremental rotary encoder [...], which is commonly used for angular velocity measurement. The measurement requires the following two sensors:
- Zero sensor that produces an impulse when $\theta = 0°$.
- Tooth sensor that outputs an impulse when it meets with one of teeth

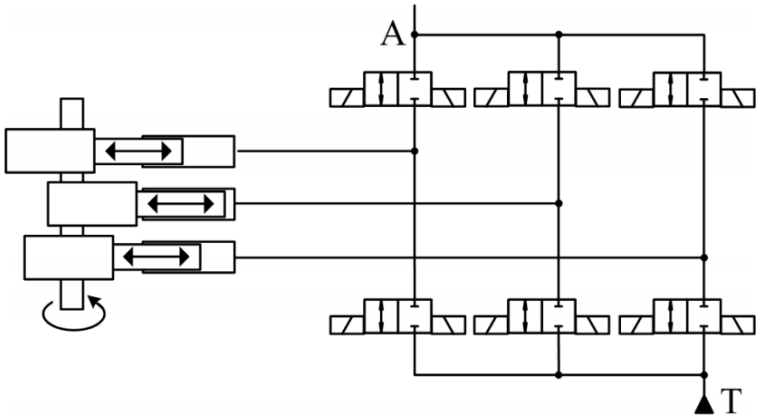


Figure 1.1: The hydraulic architecture of a digital pump motor. The figure shows three pistons connected to a rotating shaft to the left. Each piston is connected via valves to a high pressure line A and a tank line T.

placed at flxed positions on the rotating shaft.
There is some angle difference between starting point of angle measurement and teeth [..]. In practice this difference cannot be eliminated due to technological reasons. Therefore impulses from zero sensor and tooth sensor never arrive simultaneously. In the system the shaft is attached to an electric motor, which runs unloaded with $\omega_{max} = 25\pi$s-1 (12.5Hz or 750rpm).

We assume that all teeth are placed absolutely uniformly at the shaft. The discrete character of sensor signals implies that θ can be known precisely only at the moment of arrival of a signal from either zero or tooth sensor. During the rest of the time, θ can be estimated with an existing kinematic models, e.g., model of a rotating shaft with constant angle velocity. We used the following deflnitions for discretisation:
- N - total number of teeth placed on the rotating shaft. The studied system has 144 teeth;
- i - index of the current tooth met by the tooth sensor, $i \in [0..N-1]$;
- $\theta 0$ - angle difference between positions of the zero sensor and the flrst tooth presented in Fig. 2;
- t_i - time of arrival of impulse from the tooth sensor at position i.
[. . .]
The DHPMS in the case study of the paper has three cylinders and each cylinder has two valves, one for the pressure line and one for the tank line as shown in Fig. [1.1]. Let $\theta_{b,k}$ denote the angle position of the shaft that corresponds to the position with maximum extraction (bottom-dead-centre) of piston of cylinder k. The angle of the shaft is counted from the bottom position of piston of the flrst cylinder, so that $\theta_{b,1} = \theta_0 = 0°$. Positions of pistons of other cylinders are shifted by 120° with respect to each other,

therefore $\theta_{b,2} = 120°$ and $\theta_{b,3} = 240°$. The construction of the DHPMS imposes constraints on opening and closing of all valves as follows:
- The pressure line can be opened if $\theta \in [\theta_{b,k}, \theta_{b,k} + 90°)$ and closed if $\theta \in [\theta_{b,k} + 90°, \theta_{b,k} + 180°)$.
- The tank line can be opened if $\theta \in [\theta_{b,k} + 180°, \theta_{b,k} + 270°)$ and closed if $\theta \in [\theta_{b,k} + 270°, \theta_{b,k} + 360°)$.

The proposed architecture of the DHPMS software is presented [...]. This architecture is based on interrupt handling and use of timer-triggered (delayed) tasks. There are two main interrupt handlers:
- The zero sensor handler is needed to determine the moment for starting of a new period of rotating shaft. This allows to define reference points of the time t_e^j and to reset tooth index (counter) i.
- The tooth sensor handler increments i, estimates θ_i and $\hat{\omega}_i$ [...] , and calculates both angles θ_e^j and correspondent time delays t_e^j for each triggering event e_j.
The tooth sensor handler enables delay-based triggering of valve triggering tasks. The task for event e_j is triggered if θ_e^j is expected to be between current and next tooth of the rotating shaft: $\theta_i \leq \theta_e^j \wedge \theta_e^j < \theta_{i+1}$."

1.2 Scheduling Theory

This thesis makes use of a few basic definitions of scheduling theory. In a scheduling system, the response time of a task is the time between the initial scheduling of the task and the moment the task concludes execution. This response time may vary due to pre-emption, for example, and the highest response time is appropriately named the worst case response time. While for some systems it is possible to calculate the worst case response times through the use of formulae and algorithms instead of models, these are usually higher upper bounds than is realistically attainable in models [6].
The DHPMS software uses fixed-priority scheduling with pre-emption. Each task has a fixed priority value, and when a task with a higher priority is scheduled, it is placed ahead of tasks with lower priority including the currently running task.

Chapter 2

Theoretical Approach

This chapter presents a theoretical approach to calculating the Worst Case Response Time (WCRT) of the DHPMS. Any mention of time units corresponds to microseconds in the physical system.

2.1 Assumptions

The physical system runs at 12.5 rotations per second unloaded and contains 144 teeth. This results in a rotation time of 80 milliseconds, and a delay between teeth of $555.\bar{5}$ microseconds. We round this number to 555 microseconds for simplicity, and allow the number of teeth to be any number divisible by 4 and the number of cylinders. We abstract from a variable rotation speed because the given variance of ± 5 microseconds is only a fraction of the distance between teeth and is not useful unless timing errors are also modeled. There are six tasks to be scheduled (see table 2.1). While it is possible to give each cylinder its own valve tasks, this is unnecessary due to symmetry leading to each cylinder having the same WCRT for each Valve task labeled the same. The Zero and Tooth tasks are periodic, while the four Valve tasks are considered sporadic but required to be scheduled exactly once in a fixed interval. A single-core processor is available as resource, and the scheduling uses fixed-priority with pre-emption.

2.2 A single cylinder

A single cylinder in the hydraulic system rotates through four quadrants in which a specific valve can be triggered. The order is: Open Pressure Valve(P_o), Close Pressure Valve (P_c), Open Tank Valve (T_o), Close Tank Valve (T_c). In figure 2.1, the Pressure Valve corresponds with Valve1 and the Tank Valve with Valve2. A key problem of WCRT in the DHPMS is that Valve tasks from a previous quadrant can be scheduled at the very end

Task	Duration (μs)	Priority
Zero	10	8
Tooth	20	7
Valve1Open	10	6
Valve1Close	10	5
Valve2Open	10	6
Valve2Close	10	5

Table 2.1: Assumed worst case execution times and priority per task

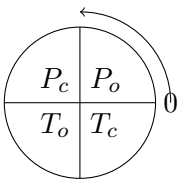


Figure 2.1: A single cylinder

of that quadrant, spilling over into the next. If the Valve for the following quadrant is scheduled instantly, one of the two Valve tasks will be delayed.

2.3 Multiple cylinders

The original paper contains a lemma regarding worst-case response times. (Lemma 1) The worst-case response times will occur after tooth events on a multiple of 30 degrees.
This lemma is derived from a reasoning that the greatest common divisor of 120 (360 divided by three cylinders) and 90 (360 divided into four quadrants) is 30. We can be more precise though. Let the number of cylinders be C. During a tooth event, at most $\gcd(4, C)$ cylinders will change quadrant. This means either 0, 1, 2 or 4 extra valve events can occur on the edge of a quadrant. This is most apparent when looking at graphs such as 2.2 for arbitrary cylinder counts and comparing how many cylinders line up to eachother given the greatest common divisor of 4 and C. Examples up to a cylinder count of 11 can be found in the appendix in figure A.1.
If C is relative prime to 4, then at most one cylinder will change quadrant on a tooth event, which is the least and thus preferred for cylinder configurations.
The Finnish case study investigated three and four cylinder configurations and concluded having a task for each valve event was not better than using a polling method with 50 microsecond intervals. The above result shows that

for example a five cylinder system would in fact have a better WCRT. It remains an open question whether average response times would be better as well, because this is dependent on (realistic) probability distributions of Valve tasks.

The time between teeth is signficantly larger than the amount of time spent processing tasks as long as C is small. In the physical system, the time between teeth is approximately 555 microseconds. There is an assumed delay of 200 microseconds between the Zero event and the first tooth event, leaving a delay of roughly 355 microseconds between the last tooth event and the Zero event. Thus, to even have the Zero task influence the WCRT, the system would need to schedule 20 Valve tasks, and it would have to be a system of 20 or more cylinders.

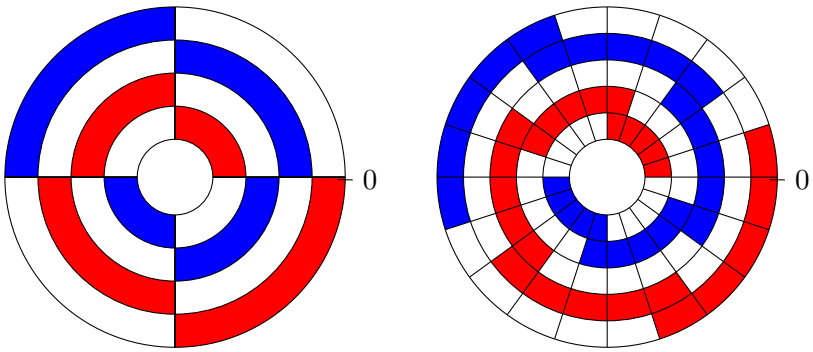


Figure 2.2: Four cylinders and five cylinders

Figure 2.2 depicts how the quadrants of different cylinders overlap. The first cylinder is the inmost circle, the second cylinder the second inmost circle, and so on. Each solid line depicts the start or end of a quadrant of a cylinder. The red surface denotes the quadrant for Valve1Open or Open Pressure Valve. The blue surface denotes the quadrant for Valve2Open or Open Tank line. The quadrants for Valve1Close and Valve2Close are not colored to avoid clutter, as the image is entirely symmetric in regards to the colored areas belong to Valve$x$Open or Valve$x$Close (where $x \in \{1, 2\}$).

2.4 Calculating WCRT

The WCRT for closing valves can be calculated through summing of task execution times. Each cylinder in the system can potentially schedule a Valve task on the exact same moment a cylinder changes quadrant, leading to C regular Valve tasks scheduled. We know that dependent on C, there are one to four extra Valve tasks outstanding exactly when a quadrant changes and

with a tooth event on that same moment. The Zero task is too far from the very first tooth event to have an influence on the WCRT while C is smaller than 20, as reasoned above. For tasks to close valves, this leads to a formula of: $\text{WCRT} = 20 + 10C + 10 \cdot \gcd(4, C)$, with $C < 20$.
For opening valves, one only has to consider half the Valve tasks because closing valves has a lower priority than opening valves and thus cannot cause pre-emption. The formula for these simplifies to:
$\text{WCRT} = 20 + \frac{10C+10 \cdot \gcd(4,C)}{2}$, with $C < 20$.

To conclude, both formulae are linear in the number of cylinders leading to worst case response times (far) higher than the 50 microsecond baseline of polling as seen in table 2.4. As the number of cylinders increases, even the factor for the number of outstanding Valve tasks becomes relatively small. On the other hand, these results are entirely pessimistic and should have a very low chance of ever occurring. This is something that can be tested using Statistical Model Checking.

Cylinders	WCRT Open Valve	WCRT Close Valve
3	40	60
4	60	100
5	50	80
6	60	100
7	60	100
8	80	140
9	70	120
10	80	140
11	80	140
12	100	180
13	90	160

Table 2.2: Theoretical worst case response times per number of cylinders in the DHPMS

Chapter 3

From TIMES to UPPAAL

In order to convert the TIMES model to an UPPAAL model, we first describe the TIMES model in detail. Then we describe how an UPPAAL model was based on it, and how it was simplified. It is worth noting that as the optimization and usage of UPPAAL features progressed queries ran slower for reasons we do not quite understand. In the TIMES and UPPAAL models, 1 time unit corresponds to 1 microsecond in the physical system.

3.1 Understanding the TIMES model

> TIMES is a modelling and schedulability analysis tool for embedded real-time systems, developed at Uppsala University in 2001. It is appropriate for systems that can be described as a set of preemptive or non-preemptive tasks which are triggered periodically or sporadically by time or external events. It provides a graphical interface for editing and simulation, and an engine for schedulability analysis. [2]

The TIMES model abstracts from realistic slowing/accelerating of the shaft, simplifies the number of teeth to 12 which is the minimum required for three and four cylinders, uses a uniform distribution for the release of the Valve tasks, and keeps the release of Valve tasks independent from eachother. The physical system runs at 12.5 rotations per second unloaded and contains 144 teeth. This results in a rotation time of 80 milliseconds, and a delay between teeth of $555.\bar{5}$ microseconds. The TIMES model uses a minimum teeth delay (variable `Tmin`) of 550 time units and a maximum (variable `Tmax`) of 560. This means the model runs at approximately 12 times the speed of the real life system because it assumes one-twelfth of the number of teeth, but maintains the realistic delay between teeth from the physical system.

For each task the behaviour (B), priority (Pr), and duration (D) are defined. While the tasks in the DHPMS are considered sporadic and TIMES

does support sporadic tasks, they are labeled as 'C' for controlled because sporadic tasks in TIMES may only have a minimum distance between arrival times (column T), while the tasks in the DHPMS must be triggered in a specific interval. Controlled tasks have their arrival times defined by timed automaton templates. As the deadline requirements for the tasks in this system are complex or irrelevant, the values in the Deadline column (D) are 'inf' for infinite.

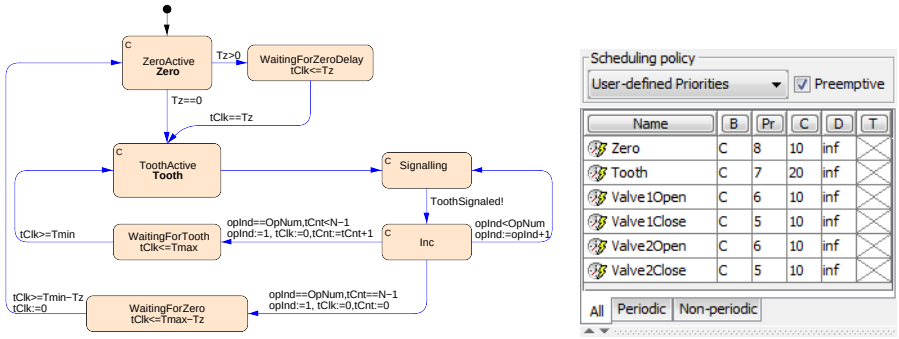


Figure 3.1: RotatingShaft template of the TIMES model and a screenshot of the global settings for the DHPMS project in TIMES

The RotatingShaft template handles the release of the Zero and Tooth tasks, as well as signalling of the tooth to the Operation template. As there is a non-zero delay between the Zero sensor and the first tooth, it first goes into a delay location. As a note, the transition with '`tZ == 0`' is never taken, because `tZ` is a constant and not zero in the model. The guard from `WaitingForZeroDelay` is for an unknown reason an equality instead of a greater-or-equals, but this does not appear to affect the results. Because synchronization in TIMES cannot broadcast to multiple instances at once, there is a loop which goes over the cylinder indices between the `Signalling` and `Inc` locations. Then, the tooth number counter (`tCnt`) is incremented and the template moves to one of two delay locations, depending on whether it is on the last (normally twelfth) tooth or not.

The Operation template appears large and unwieldy, but the basic idea is straightforward. Transitions are either activated by `ToothSignaled` signals from RotatingShaft, or there is a delay, or a Valve task is scheduled. There are four locations per quadrant, for example `WaitForTooth1`, `WaitForTooth12`, `WaitForDelay1`, and `OperationActive` for the first quadrant of a cylinder.
In `WaitForTooth1` the instance waits for the tooth counter to hit the initial value given through the parameter `tInd`. Then it non-deterministically chooses to wait for an unknown number of tooth signals (but smaller or

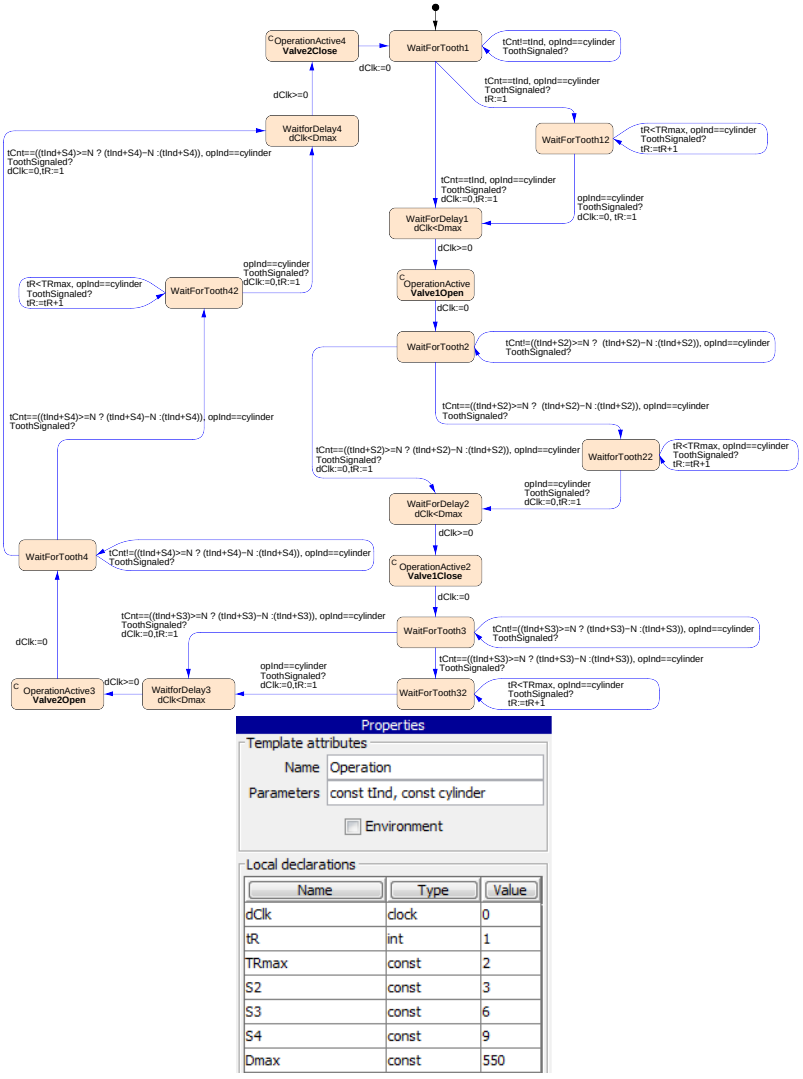


Figure 3.2: Operation TIMES template with properties window

equal to the constant `TRmax`), or go to the delay location. At the location `WaitForTeeth12`, it chooses non-deterministically for up to two teeth signals whether to stay in the `WaitForTeeth12` location, or go to the delay location. This has the result that 50% of the time, valve events are scheduled between the first and second tooth, 25% between the second and third, and the remaining 25% between the third and fourth tooth. If the number of teeth were to be increased, it would become increasingly likely for a Valve task to be scheduled at the start of the quadrant. This does not affect standard UPPAAL queries, because the state space includes all possible transitions, but it would be unsuitable for SMC queries. TIMES does not appear to have a

function comparable to UPPAAL's select statement to non-deterministically choose a random number, leading to this kind of construction being required. Once the instance is in the `WaitForDelay` location, it waits up to 550 time units before transitioning to the `OperationActive` location, therefore it can never overshoot the true edge of a quadrant. The Valve task is then set to be scheduled, and the instance moves to the next `WaitForTeeth` location to wait for the start of the next quadrant. A quirk of the Operation template is that the second and third cylinder do not release Valve tasks during the first four teeth and eight teeth respectively of a run, as they wait for the tooth indicator to be equal to the constant variable `tInd`. TIMES also does not have support for the modulo operator, hence the use of a ternary operator in several fields.

3.2 Translation

> The model-checker Uppaal is based on the theory of timed automata and its modelling language offers additional features such as bounded integer variables and urgency. The query language of Uppaal, used to specify properties to be checked, is a subset of CTL (computation tree logic). [3]

The reader is assumed to be familiar with the tool UPPAAL, for more information on it we refer to the tutorial paper by Behrmann et al [3].

The translation from a TIMES model to an UPPAAL model can be made in a straightforward way, because TIMES has a very similar syntax. In fact, TIMES uses an older version of UPPAAL as back-end and has an option to export to .ta format, which is an older format no longer used by UPPAAL 4.0, but can still be read. Because of the large version differences, the exported model contains many syntax errors when opened in UPPAAL 4.0 and 4.1. We decided not to pursue investigating and editing the erronous syntax.

The release of a task was changed to a synchronization channel, as this cannot happen inside a location in UPPAAL because updates and synchronizations are only possible on transitions. It was visible in the broken exported file that TIMES internally adds edges with synchronizations to the template for a location with a task release. While it is now the case that the release of a task happens after the location where it is triggered in TIMES, this is necessary to keep the TIMES template intact as much as possible and avoids adding new locations to the templates.

3.3 Scheduling Framework

To convert the TIMES model to an UPPAAL model we also need to have templates that model scheduling and resource usage. The scheduling frame-

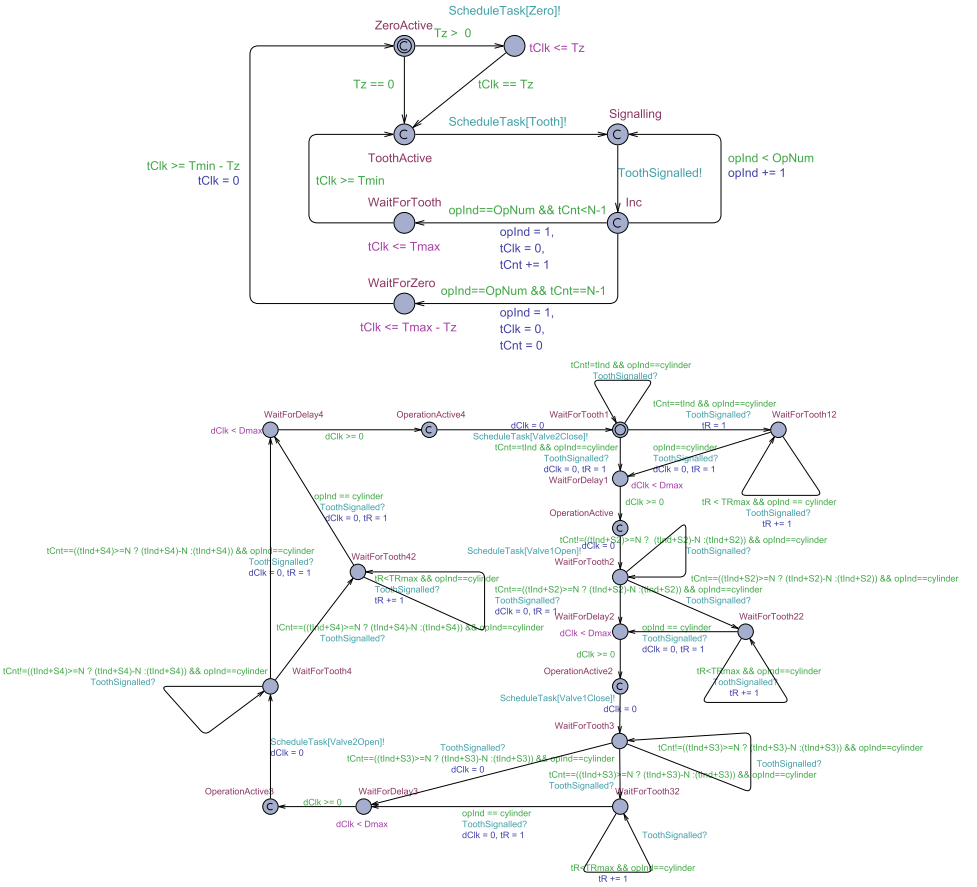


Figure 3.3: First version of the UPPAAL DHPMS model, above is the RotatingShaft template, lower image is the Operation template

work we took as base is also bundled with UPPAAL 4.1 as an example.[7] The framework contains many features that were of no use to this particular case study, and those were removed in the quest to verify the query "`A[] not deadlock`". It also does not support sporadic tasks.

During verification we discovered that the altered templates overestimate response times. For Valve1Open/Valve2Open and Valve1Close/Valve2Close the worst case response times were respectively 50 and 90, instead of the expected 40 and 60. My supervisor provided a different template for scheduling and resources shown in figure 3.4.

Unfortunately, computation times for non-SMC queries increased exponentially at first. The problem turned out to be that the array `completiontime` was defined as an '`int`', which has a default range of $[-32768, 32767]$ as noted in the UPPAAL help file under Types. Properly defining the range for this array is a kind of catch-22, because to calculate the upper bound the set

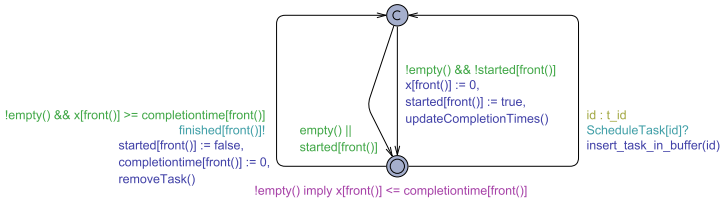


Figure 3.4: Resource template for a single-core CPU with fixed-priority scheduling and pre-emption

range must be at or above the upper bound. The Resource template has no way of knowing this in advance, nor does the user. Fortunately, UPPAAL is able to calculate the exact upper bound with only an imprecise guess of 500 in a short amount of time. Our conjecture is that UPPAAL uses the upper bound on `completiontime` to reduce the searched state space, and when the bound is set too high UPPAAL assumes certain reductions cannot be done leading to a much larger state space searched.

3.4 Simplification

Once we confirmed that the initial conversion to UPPAAL functioned, it was time to simplify using the more advanced features in UPPAAL unavailable in TIMES. The goal of simplification was two-fold: one was to make the templates more understandable from a glance, and the other to hopefully speed up the calculation times. Unfortunately, the latter goal was not met, as calculation times increased.

There was not much to simplify in the RotatingShaft template, as shown in figure 3.5. The unused transition with the guard '`tZ == 0`' was removed and the loop to signal each cylinder seperately was removed in favor of changing the channel to a broadcast channel, which is not available in TIMES. As broadcast channels do not block, it was required to define queries to test if the signal was always received before continuing. This template was already ready for tests with increased number of teeth due to usage of the `N` variable, as well as for increasing the number of cylinders.

The Cylinder (renamed from Operation) template was simplified drastically, as seen in figure 3.6 There were originally four locations per quadrant, which was knocked down to three. The non-uniform behaviour was fixed by uniformly choosing a number of teeth to wait before looping through teeth signals at location `WaitForDelay`.
The variable `quadrant` was added to keep track of which quadrant the instantiation of the Cylinder template is in. Instead of three global variables `S2`, `S3`, `S4`, an array `S` was defined locally in the Cylinder template. The cal-

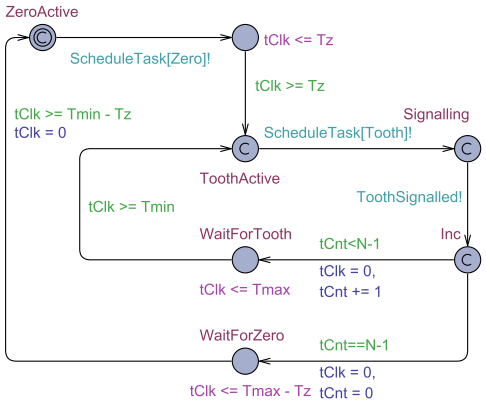


Figure 3.5: Simplified RotatingShaft template

culation of which valve task to release was softcoded to a `Valve()` function.

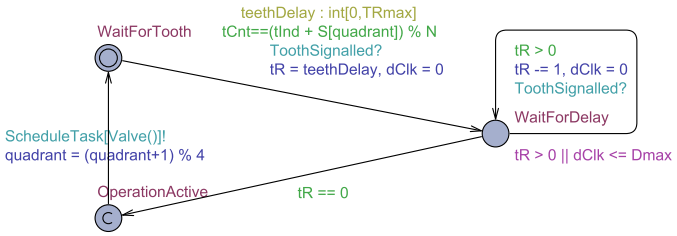


Figure 3.6: Simplified Operation template renamed to Cylinder

UPPAAL has support for total symmetry reduction [12].
We investigated briefly if this feature would be of use to the case study for the Cylinder templates and the tasks. There is no total symmetry in either case, because the functionality of a Cylinder template depends on an ordering of the index of the instantiation and each task is effectively unique in usage even for duplicate tasks. It remains an open question if more advanced forms of symmetry reductions involving more specific sets could allow for reductions in state space for this kind of cyclical system in UPPAAL.
Likewise, UPPAAL has support for progress measures [9]. The idea is that by defining an expression that is weakly monotonic, UPPAAL can determine which states provide the most progress and reduce the state space with this information. Unfortunately, any attempt to define a progress measure for the model resulted in higher computation times. As in the Herschel case study [11], there is no intrinsic measure of progress in this kind of cyclic model,

but while their application of a progress did function for their models, the same pattern did not function for the DHPMS models.

3.5 Verifications

Our goals with verifying the simplified templates are to confirm deadlock freedom and to determine WCRT for as many cylinders as possible. With the current Resource and Measure templates, UPPAAL scales to six cylinders for deadlock testing and up to five cylinders when using `sup` queries for WCRT. Our conjecture is that calculation times increase so severely because in addition to the state space explosion problem UPPAAL has to consider every permutation of task events, which for n events increases on the order of $n!$ or more. Furthermore, due to the method of implementing arbitrary number of cylinders, there are four additional clocks (for four duplicate valve tasks) per four additional cylinders. Increasing the number of clocks in an UPPAAL system increases computation times and memory usage significantly. Due to these factors, even if further optimizations were to be done it would likely be impossible for current computer systems to go through the entire state space of a model with ten or more cylinders.

No. of Cylinders	Computation time	Maximum memory usage
3	00h00m08s	28 MB
4	00h29m40s	1897 MB
5	09h53m12s	7585 MB
6	99h58m36s	77349 MB

Table 3.1: Verification times and maximum memory usage for the query '`A[] not deadlock`'

No. of cylinders	WCRT for Valve1Open & Valve2Open	Valve1Close & Valve2Close	Computation time	Maximum memory usage
3	40	60	000h01m23s	43 MB
4	60	100	032h11m38s	7,279 MB
5	50	80	038h31m13s	14,269 MB
6	-	-	507h00m00s+	46,000+ MB

Table 3.2: Worst Case Response Times ordered by cylinder count

One important distinction between the original results of [4] and the

UPPAAL verification is that Valve1Open and Valve2Open have the same WCRT in table 3.2. While tasks can be defined with equal priority in TIMES, internally tasks with equal priority are given unique priorities based on the order from top to bottom. [2]
For example, in TIMES Valve1Open has a higher internal priority than Valve2Open and therefore Valve2Open can be pre-empted by Valve1Open, leading to a lower WCRT for Valve1Open.
This behaviour was only tersely mentioned by the Finns, but it does lead to an inconsistency between the task definitions and the calculated results in [4]. Tasks with equal priority and duration in this kind of symmetric system must have equal worst case response times. The UPPAAL scheduling template does handle such tasks in a consistent manner. As far as it was calculable, the values in table 3.2 are also consistent with the theorized results found in section 2.4. Note: For four cylinders and above, the WCRT is the maximum of the two duplicate tasks for each Valve. It would have sufficed to run the `sup` query over only the duplicate tasks, as those have the highest WCRT due to only being scheduled if the original task is already present in the resource buffer.
For five cylinders, the number of teeth was set to 20, for all other queries the number of teeth was set to 12.
For three cylinders the query ran was:
‘`sup: Measure(Valve1Open), Measure(Valve1Close), Measure(Valve2Open), Measure(Valve4Close)`‘
For four cylinders and above the query ran was:
‘`sup: Measure(Valve1Open), Measure(Valve1Close), Measure(Valve2Open), Measure(Valve4Close), Measure(Valve1Open+4), Measure(Valve1Close+4), Measure(Valve2Open+4), Measure(Valve4Close+4)`‘
Due to the inclusion of an invariant ‘`x' == 0`’ in the Idle location in the Measure automaton for faster calculations, the queries returned a ‘`Formula MAY be satisfied.`. In the case of `sup` queries this has the result that the listed upper bounds may not be the lowest upper bounds, but using queries such as those below, it was confirmed that the `sup` queries did return the lowest upper bounds. If one explicitly enables over-approximation, the `sup` queries in fact return higher upper bounds. It is important to add that these queries are not the most efficient to run, if we already know the exact upper bound. It is also more efficient to run queries with only one Measure instantiation. As an example, we shall show how to confirm the upper bound of Valve1Close in a system of four cylinders. We add to to the System tab:
‘`Measure_Valve1CloseDup = Measure(Valve1Close+4);`’, replace ‘`Measure`’ with ‘`Measure_Valve1CloseDup`’ in the ‘`system`’ line and run the following two queries:
‘`A[] Measure_Valve1CloseDup.Action imply Measure_Valve1CloseDup.x <= 100`’

'E<> Measure_Valve1CloseDup.Action and Measure_Valve1CloseDup.x == 100'.
The first query determines that 100 is an upper bound, and the second (if satisfied) proves that there exists a path where the value of 100 is reached, therefore 100 is the smallest upper bound. Explicitly using over-approximation is highly efficient for the A[] query with calculation times of 27 seconds, but returns a '*may be satisfied*' for the E<> query. Using standard DBM, the resource usage for the E<> query is in the order of 268 seconds and under one gigabyte of RAM for this example. Even though these numbers are notably less than testing for deadlock, this approach will likely still not scale to ten cylinders or more.

The queries with a maximum memory usage of 2GB or less were run on an AMD Phenom 2 processor at 3.6Ghz using the Windows 32-bit version of UPPAAL 4.1.19. The remaining queries were ran on a cluster node with AMD Opteron processors at 2.3Ghz using the Linux 64-bit version.
As can be seen from the tables 3.1 and 3.2 we run into the state space explosion problem by 6 cylinders. By the time the WCRT query for six cylinders was terminated, there were still millions of states left to go through at a rate of at most hundreds of states per second.

Chapter 4

Moving onto UPPAAL-SMC

Statistical Model Checking applies statistics on automata to determine a probability range for a given query. It uses mathematical sound results from the field of statistics. Instead of searching the state space of an automata to prove statements, SMC is based on runs through the automata with bounds on clocks or steps to guarantee termination. [10] UPPAAL-SMC is the implementation of SMC for UPPAAL. It offers for example more freedom for floating point calculations by including a variable type `double`, a user-friendly interface for creating graphs using SMC-style queries called the Plot Composer, and allowing probabilistic transitions. [5]

UPPAAL-SMC does not allow any non-determinism in the model for its queries. This results in select statements being unuseable except to emit or receive a channel array. Likewise, multiple non-exclusive transitions leaving a location are also not allowed. If possible, it would be preferable to model a system such that both UPPAAL and UPPAAL-SMC both work, as each has possible queries which the other does not have. Being limited to using a subset of UPPAAL and UPPAAL-SMC features limits model freedom, and it becomes tempting to have one standard UPPAAL model and an UPPAAL-SMC model, each for their own queries. Proving that these two models are functionally identical or even semantically identical is not trivial, and not guaranteed to be true at all. Instead of the non-deterministic features of UPPAAL, UPPAAL-SMC introduces probabilistic transitions and the `random()` function returning a double. This results in a problem when a random integer is required. While it is is possible to write a function to convert a floating point number to an integer, this is not ideal. The functions `ceil()` and `floor()` are available, but return a double.

UPPAAL-SMC offers three kinds of queries: simulations, probabilities and expected values.
Simulations queries have the following general syntax:
`simulate N [<=bound] {E1, ... , Ek}`, with `N` and `bound` positive inte-

gers (for Windows UPPAAL version 4.1.19, `bound` is limited to 32 bit signed integer values), and `E1` to `Ek` expressions to be monitored. Each run will have the expressions drawn onto one graph, accessible upon right-clicking the completed query. The graph data remains in memory until UPPAAL is closed, and there are more options visible upon right-clicking the graph. The options available are Areas, Show, Scale, Export and Numeric locale. The tick-able options in Areas appear to be non-functional in this window, they are functional when using the Plot Composer. The options in the section Show such as Title and Legend are tick-able, primarily for customizing a graph to be exported. The options for Scale set the x-axis and/or y-axis to log scale. The graph may be exported to various graphic file formats and to comma-seperated-value text files. The bound may also be for a specific clock (syntax: `simulate N [clock <= bound] {E1, ..., Ek}`) or a number of steps (syntax: `simulate N [# <= bound] {E1, ..., Ek}`. Finally, the query can be extended to a search for a maximum specified number of runs (`N2`) for which a given expression is true, using the following syntax (using the implicit time bound as example): `simulate N [<= bound] {E1, ..., Ek} : N2 : expression`. This can be used for example to specifically display runs which violate deadlines instead of having to find those manually between thousands of runs.

Probability queries have the basic syntax of `Pr[bound](phi)`. The number of runs is based on alpha and epsilon listed in the Statistical Parameters option menu, where alpha is the confidence interval and epsilon the approximation interval. More detailed information can be found in the UPPAAL-SMC Overview paper by Bulychev et al. UPPAAL-SMC is able to calculate (cumulative) probability distributions based on the clock in the bound. By changing a variable to a clock with zero rate and bounding runs using `phi` one can obtain probability distributions of any clock or variable.

Expected values queries use the syntax `E[<=bound;N](min:expr)` for the expected minimum values of `expr` and `E[<=bound;N](max:expr)` for the expected maximum values.

There are four goals to be achieved in the conversion to SMC:

1. Arbitrary delay for each Valve task
2. Arbitrary speed in RotatingShaft that does not affect calculations in Cylinder
3. Arbitrary amount of Cylinders
4. Maintain awareness of true quadrant edges for deadline misses

Arbitrary delay for each Valve task was implemented through a pattern from the tutorial to UPPAAL-SMC. The variable `delay` is a clock with its rate set to zero manually. UPPAAL requires that guards compare clocks with eachother, or a clock to an integer value. Comparing to a const double results in inconsistent behaviour in UPPAAL-SMC where it may give an error that the system is time-locked (as in, deadlocked), while a double

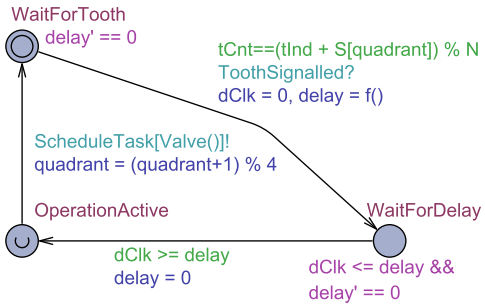


Figure 4.1: Cylinder template modified for UPPAAL-SMC

(functionally a clock with its rate set to zero) would trigger fine-grained discretization.
The function `f()` was setup such that it returned the value of yet another function for ease of changing distribution of Valve tasks.
Because the delay is now a value of an arbitrary function, there is significantly more freedom in defining a probability distribution.

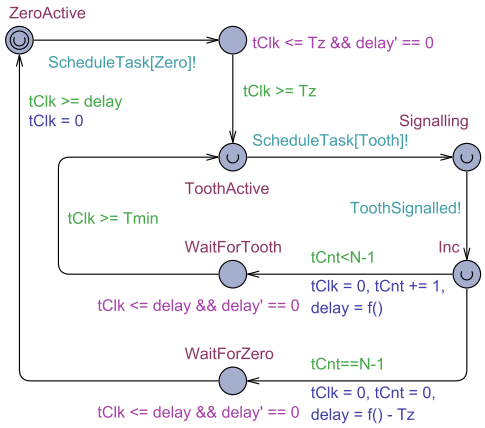


Figure 4.2: RotatingShaft template modified for UPPAAL-SMC

The raw data provided implied that the speed variations would be of fractions of microseconds in terms of delay between teeth. UPPAAL-SMC uses fine-grained discretization at least when clocks are set to non-integer values, or when guards involve general floating-point expressions. A naive implementation of jittery rotation speed would involve the former, and results in simulations taking roughly 2000 times longer at default settings for discretization step (a value of 0.01).
Furthermore, UPPAAL-SMC returned errors such as “Location RotatingShaft.WaitForZero has unbounded delay but no positive rate.”.

In the first attempt to apply the 'clock delay' pattern, we merely replaced `Tmin` and `Tmax` with `delay`. This left transitions with '`tClk <= delay - Tz`' and '`tClk >= delay - Tz`'. Our conjecture is that fined-grained discretization can cause runs to skip the moment when `tClk` equals `delay - Tz`, and somehow miss the transition to the next state, presumably due to floating point inaccuracy. The error disappeared when `delay` was initialized as "`f() - Tz`" on the next-to-last tooth. In this model, the function `f()` for RotatingShaft template is based off the minimum and maximum values which are 550 or higher, therefore '`f() - tZ`' is always strictly positive. As a note, the 'clock delay' pattern shows random errors such as the one listed above if `f()` returns 0.

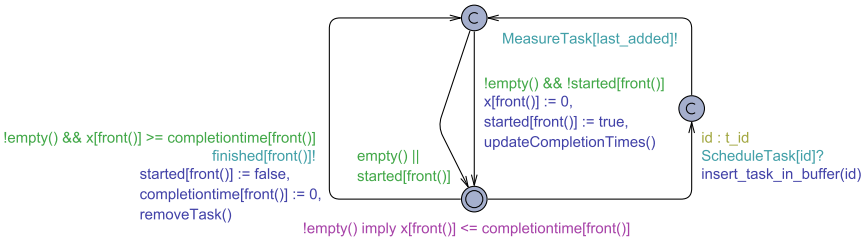


Figure 4.3: Resource template modified for UPPAAL-SMC

Arbitrary number of cylinders has been implemented. This required little adjustment in the templates, as it involved modifying code related to the resource buffer and lookup of task data. Duplicate tasks are allowed by means of increasing the task ID if the base ID for that task is already present in the buffer. The maximum number of duplicate tasks is a function of the number of cylinders and is bounded. Because the bound is a theoretical maximum and is very unlikely to be hit, there may be a large number of Measure templates which are present in the system but lay unused for simulations counted in thousands of runs. UPPAAL-SMC has a feature to dynamically spawn templates, but it has not been investigated whether this would improve the efficiency of the system by leaving out Measure automata which are never triggered.
An additional synchronisation transition was added to the Resource template for triggering the correct Measure template as the Cylinder template is unaware of running Valve tasks.

Development of the model halted at this point. The simplified SMC model does not take into account how the software would calculate the release time of Valve tasks depending on last known axis speed and intended task release time. Tasks lengths are also still constants, whereas a more realistic version of the model should include a probability distribution of individual task lengths.

Figure 4.4 shows a chart of the four Valve tasks as the axis turns. The

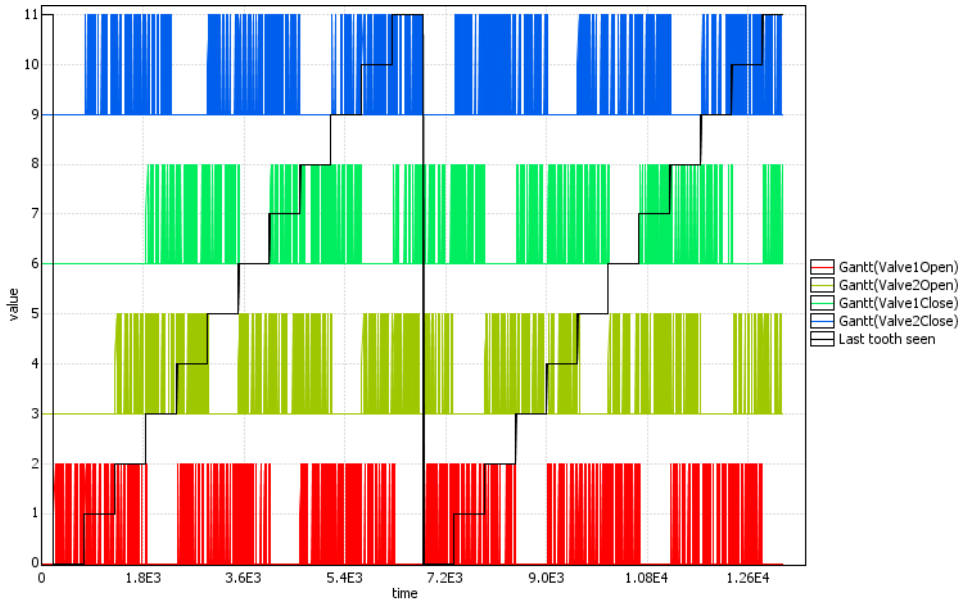


Figure 4.4: Gantt-like chart for the four Valve tasks for three cylinders

chart is of 100 runs to show where a given task may be scheduled. It also shows that even with the alteration to the starting value of `quadrant`, Valve1Close and Valve2Open still do not properly schedule in the first full rotation of the shaft.

One trick noted in the UPPAAL-SMC tutorial is the usage of clocks with zero rate for use as variable for queries. Mathematical expressions with clocks work as expected in UPPAAL-SMC, and converting integers to clocks is mostly straightforward. The ternary operator does not appear to have been updated and will not work with clocks or doubles, nor does the shorthand notation '`var += expression`' for example.
One downside of using the `Pr [variable <= upper_bound] (expression)` query to generate a probability distribution of a variable is that the probability is 1 (if the upper bound is a true upper bound of the variable). For such queries, UPPAAL-SMC with default settings ($\alpha = \epsilon = 0.05$) only needs 36 runs to determine the probability, leading to reduced accuracy in the resulting graph. This can be fixed by modifying α and ϵ in the Settings menu. We were not able to determine the exact algorithm or formula UPPAAL-SMC uses to calculate the number of runs (The formula listed in **??** does not match the numbers), therefore it was easier to just try. Table 4.1 shows the number of runs based on various values for α and ϵ.

α / ϵ	0.05	0.01	0.005	0.001	5E-4	1E-4	5E-5	1E-5
0.05	36	183	368	1843	3688	18443	-	-
0.01	51	263	528	2647	5296	26489	-	264914
0.005	57	297	597	2993	5989	-	-	-
0.001	73	377	757	3797	-	-	-	-
5E-4	79	411	826	4143	-	-	-	-
1E-4	94	491	986	4947	9899	49513	-	-

Table 4.1: The number of runs for a `Pr` query with probability 1

4.1 Results

The graphs found in this section were generated using a Measure template in figure 4.5 and queries such as '`Pr [Measure(Valve2Close).max <= 1000] (<> global == Tmax*N*2)`'. The upper bound for the clock variable does not matter as long as UPPAAL-SMC does not reach an internal maximum of 100,000 steps per run, at which point the query will terminate with an error. The expression '`<> global == Tmax*N*2`' results in UPPAAL-SMC saving the value of the clock after exactly two rotations. Systems with with an odd number of cylinders do not schedule all Valve tasks correctly during the first rotation, therefore the end of the second rotation was chosen as bound. `global` is a clock that is never reset, effectively the same as the implicit clock in SMC queries. Variable `Tmax` was set to 550 for SMC queries, removing all variance in rotation speed.

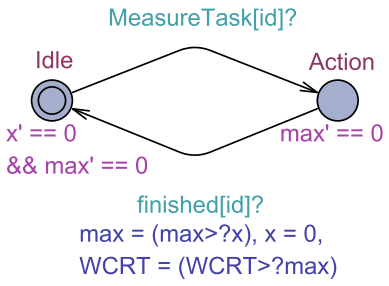
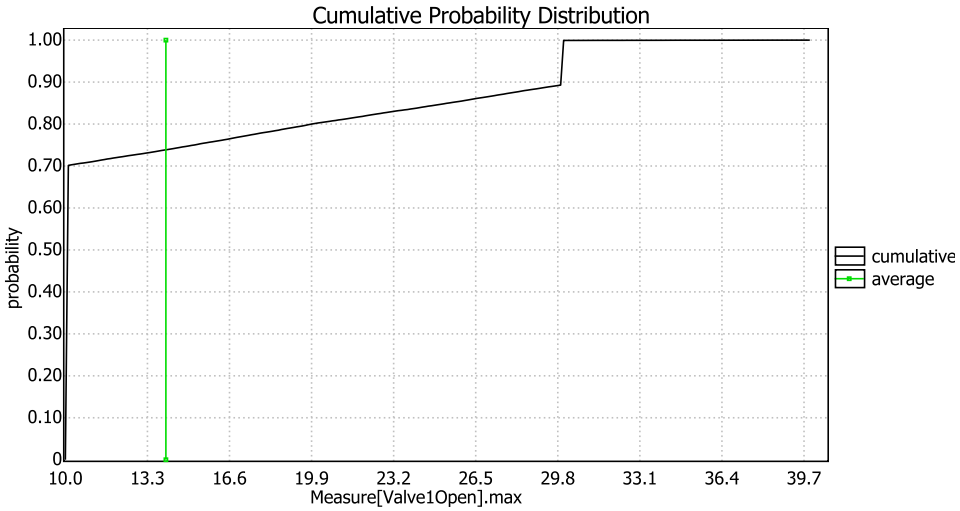


Figure 4.5: The Measure template for SMC queries. WCRT is a clock with its rate set to zero elsewhere. The Resource template emits both `MeasureTask[id]` and `finished[id]`, respectively when a task is scheduled and when a task finishes. The syntax '`max>?x`' returns the maximum of variables `max` and `x`, as the use of a ternary operator for clocks is not allowed. This pattern allows us to place variables `Measure(i).x`, `Measure(i).max`, and `WCRT` on the x-axis in graphs of `Pr` queries.

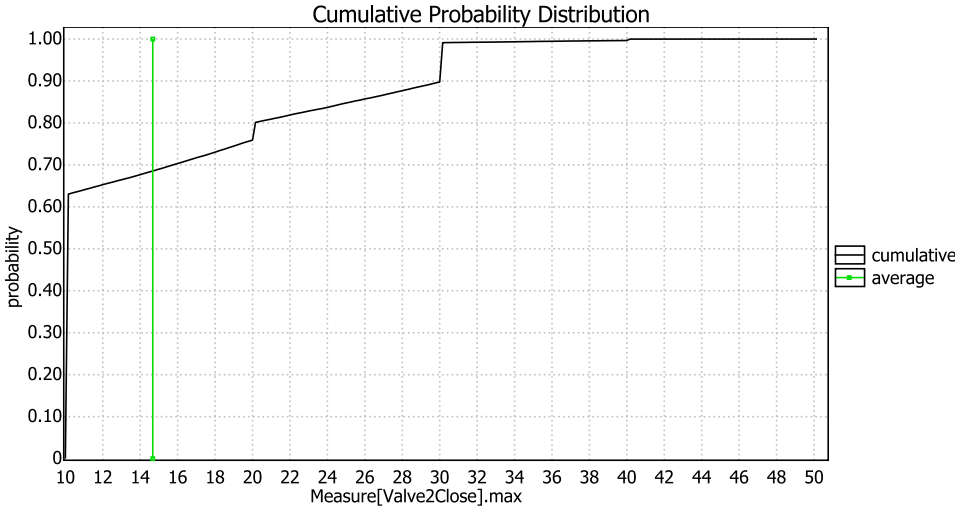
Figure 4.6 shows the cumulative probability distribution for Valve1Open and Valve2Close response times when using a uniform distribution for the release of said tasks. While this distribution is wholly inaccurate for real life scenarios it does show how the distribution develops. Calculation time for both queries is in the order of 250 seconds with a memory usage of 9MB. This gives a calculation time of 4.1 milliseconds per run, which is far faster than the 160 milliseconds the physical system would take to do two full rotations. With three cylinders and a uniform distribution, it is possible to find runs very close to the WCRT for each task.

Figure 4.7 shows that with a slightly more realistic probability distribution, the response times are consistently lower for Valve1Open and Valve2Open, while Valve1Close and Valve2Close are always scheduled at or near a Tooth task, leading to a near consistent response time of 30 milliseconds.

Then it was time to test the computing ability of UPPAAL-SMC. Figure 4.8 shows the WCRT per run of a system with 16 cylinders and 16 teeth. Computation time per run was 28 milliseconds, which again is two full rotations. The query was "`Pr [WCRT <= 200] (<> global==Tmax*N*2)`". The theoretical WCRT for 16 cylinders is 220 as per section 2.4. In 61025 runs, UPPAAL-SMC was unable to find runs with a WCRT over 80. Even with significantly more runs, the graph for the cumulative probability distribution would not change significantly, except for a possible longer tail of extremely rare events with higher WCRT. If one can already determine the chance for deadline misses to be this small in a model where the rotating shaft effectively rotates at nine times the original speed (the time per rotation is the number of teeth multiplied by a constant time value), it is not even required to investigate the more realistic model which takes more time to calculate through. In figure 4.8, the cumulative probability graphs are shown for a system with 16 cylinders and 128 teeth. Computation time per run was 79 milliseconds, still twice as fast as the physical system. With the significantly increased available time in a quadrant, the uniform distribution gives Valve tasks ample time to execute without interfering with other tasks. Running probability queries on rare events would take significantly more runs that each take more time to compute. To conclude, as with simplifying models to make them calculable with UPPAAL, thus is it more efficient to simplify and shorten intervals of UPPAAL-SMC models to increase the probability of finding rare events and proving with more certainty that the WCRT of a system is below a certain value, while at the same time decreasing calculation times.



Parameters: α=1e-005, ε=0.0001, bucket width=0.12069, bucket count=248
Runs: 61025 in total, 61025 (100%) displayed, 0 (0%) remaining
Span of displayed sample: [10, 39.932264839299]
Mean of displayed sample: 14.0422474940892 ± 0.128456087640781 (99.999% CI)



Parameters: α=1e-005, ε=0.0001, bucket width=0.16129, bucket count=249
Runs: 61025 in total, 61025 (100%) displayed, 0 (0%) remaining
Span of displayed sample: [10, 50]
Mean of displayed sample: 14.669944207911 ± 0.131325957784156 (99.999% CI)

Figure 4.6: Cumulative Probability Distributions for the highest response times per run of Valve1Open and Valve2Close for 3 cylinders and 12 teeth with a uniform distribution for all Valve tasks

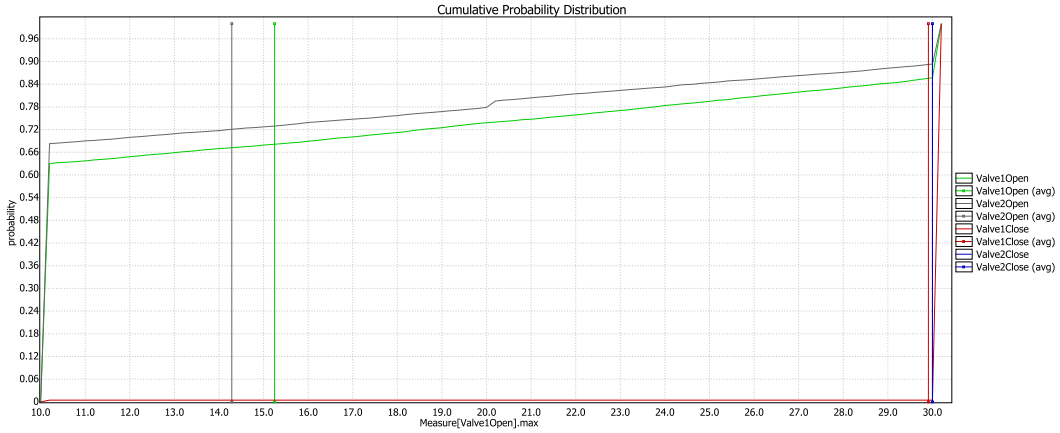
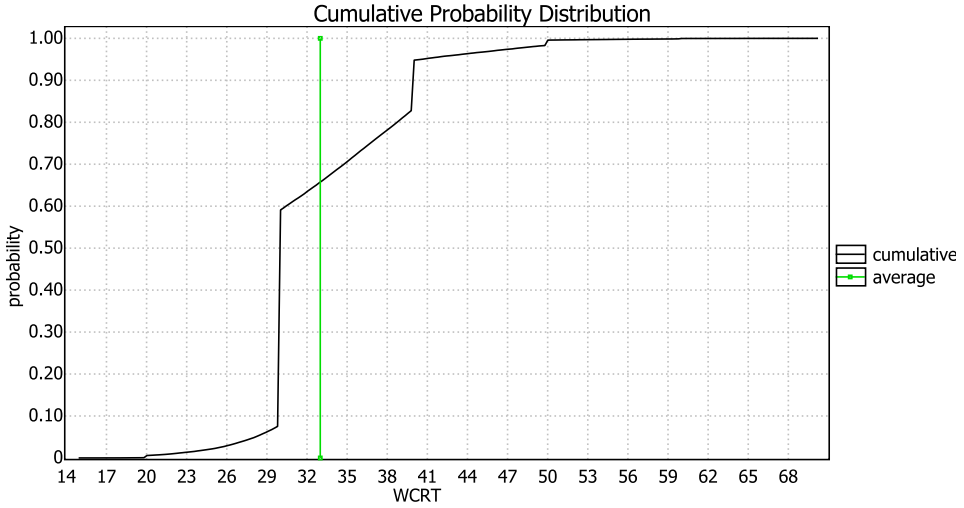
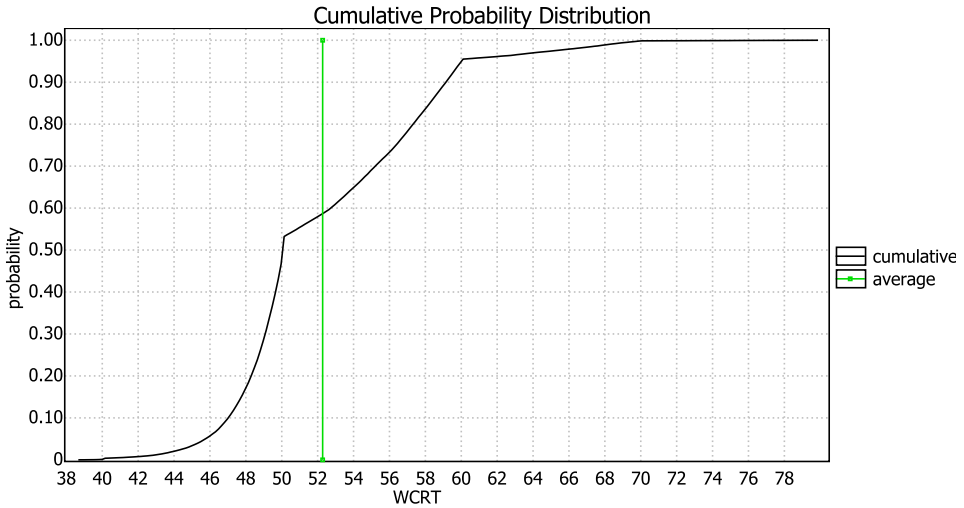


Figure 4.7: Cumulative Probability Distributions for the highest response times per run per Valve task using the minimum and maximum raw data values per cylinder as minima and maxima for uniform distributions, with $a = 0.0001$ and $e = 0.0005$, leading to 9899 runs of 3 cylinders with 12 teeth.



Parameters: α=1e-005, ε=0.0001, bucket width=0.2221, bucket count=249
Runs: 61025 in total, 61025 (100%) displayed, 0 (0%) remaining
Span of displayed sample: [14.9183116666973, 70]
Mean of displayed sample: 32.9911390491575 ± 0.0958992748249708 (99.999% CI)



Parameters: α=1e-005, ε=0.0001, bucket width=0.16616, bucket count=248
Runs: 61025 in total, 61025 (100%) displayed, 0 (0%) remaining
Span of displayed sample: [38.6678269364247, 79.8760165286562]
Mean of displayed sample: 52.2777676026483 ± 0.0939280383798349 (99.999% CI)

Figure 4.8: Cumulative Probability Distributions for WCRT per run for 16 cylinders and 16 teeth. For the first image a uniform distribution for all Valve tasks was used, while for the second the limited uniform range originally of 3 cylinders was used. Cylinder i in the 16 cylinder system used the distribution of cylinder (i mod(3)) for its Valve tasks.

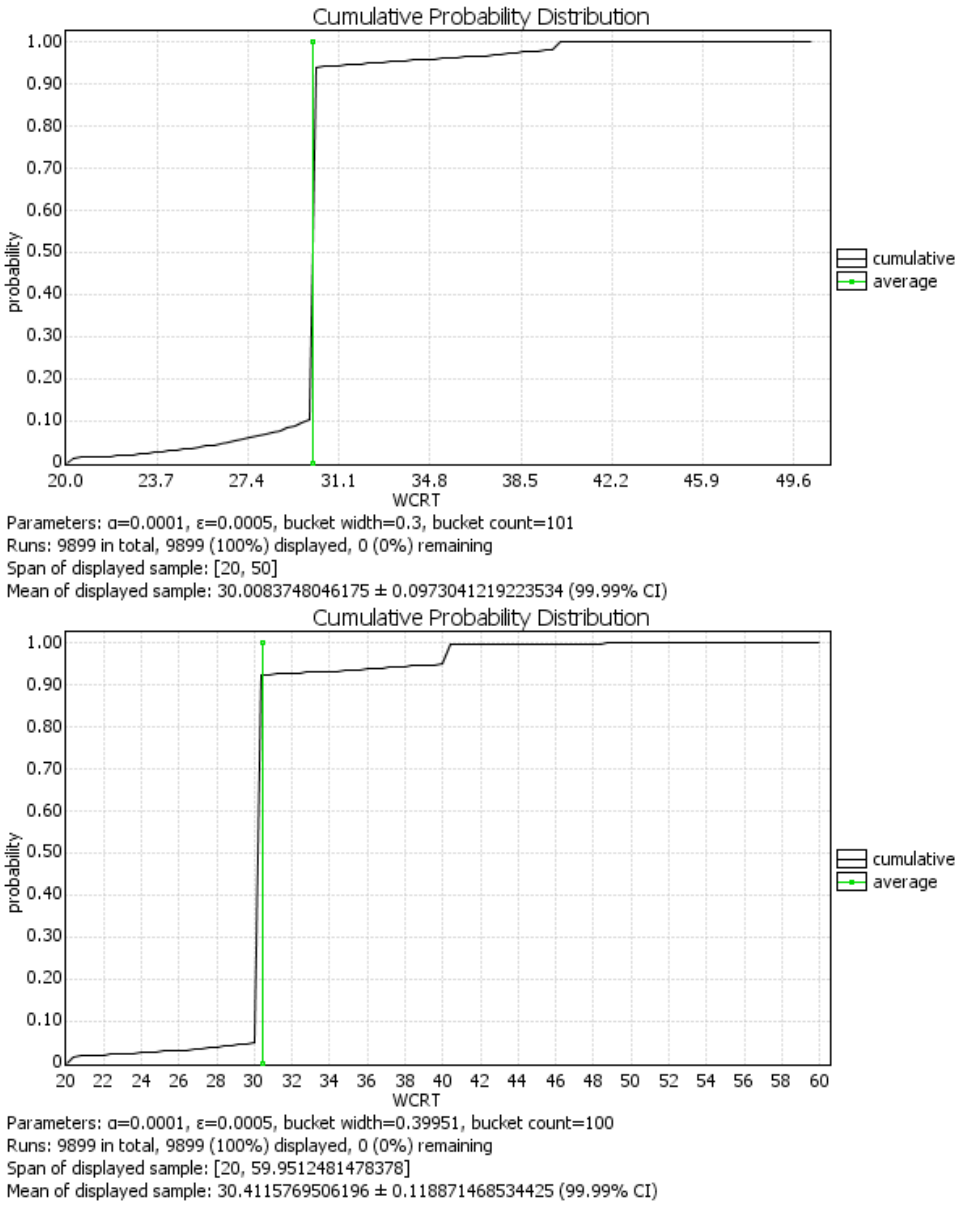


Figure 4.9: Cumulative Probability Distributions for WCRT per run for 16 cylinders and 128 teeth. The distributions for the Valve tasks use the same rules as for figure 4.8.

Chapter 5

Discussion

Working with UPPAAL-SMC was very much a mixed experience. Initially, we used UPPAAL 4.1.18 to develop the models, and attempts to use the simulator while using UPPAAL-SMC features would cause the server to crash. This was fixed in 4.1.19. UPPAAL certainly is a powerful tool to create models of systems in the real world and is user-friendly. Development on UPPAAL does not appear to be as active, with an official release 4.0.13 on 27th of September 2010, and the current development snapshot of 4.1.19 on 1st of July 2014. We ran into several (known) bugs in UPPAAL and UPPAAL-SMC. Parsing of user-defined functions can still outright crash the server if control structures reach end of file, or give non-sensical errors when defining a new variable after an if statement. The simulator tab is a very useful feature when debugging UPPAAL models, but it does not function if UPPAAL-SMC features are used. There is some support for UPPAAL to ignore clocks designated as hybrid which cannot affect the model logic, which then allows usage of standard UPPAAL queries and the simulator tab, but for models that rely on SMC for its logic this will not work.

The ability to visualize variables and other expressions is unquestionably useful for developing models and showing results. But documentation on UPPAAL-SMC is insufficient, there is one paper which serves as tutorial but is not complete [5]. The ability to select runs from a simulate query based on an expression is a fully functional feature only documented in the Herschel case study [11]. The bug tracker contains a bug report in which some undocumented built-in functions are listed [1].

Contact with Boström suggested that stiff differential equations are required to correctly calculate delays in their software. UPPAAL would be unsuitable to calculate such equations by definition of stiffness. It would be a useful future feature for UPPAAL to be able to use external libraries for calculations. It was inconvenient to have to set variables such as cylinder count explicitly in the model, where for a future feature such (constant) variables could be moved to the queries themselves for ease of use.

Chapter 6

Conclusion

This thesis extended on a Finnish case study of a Digital Hydraulic Power Management System. Given a TIMES model we created an UPPAAL model to verify the original results and then extended the model for use with UPPAAL-SMC. Among the successes were verifying deadlock freedom for systems with three to six cylinders and calculating Worst Case Response Times for systems with three to five cylinders. Beyond five cylinders, the UPPAAL model does not scale. UPPAAL-SMC was able to calculate through much larger systems, but due to the simplified probability distributions it was not viable to find runs with response times close to the theoretical worst case response times, even for systems with a high number of cylinders. That example also showed that it is sufficient to calculate through a sped-up model to guarantee that the real system has a very limited number of deadline misses over a long period of time.
UPPAAL-SMC has a lot of potential but, in its current form, is not mature. Since real-time control systems are pervasive in the area of embedded systems, we see many potential applications for a mature version of UPPAAL-SMC.

6.1 Acknowledgements

We would like to thank Pontus Boström for providing us with the TIMES models and some raw data, they have been invaluable in developing models for this thesis.

Bibliography

[1] Bug 588 - uppaal smc features: operations and limitations of double/-clock type. `http://bugsy.grid.aau.dk/bugzilla/show_bug.cgi?id=588`. Accessed: 2015-04-07.

[2] Tobias Amnell, Elena Fersman, Leonid Mokrushin, Paul Pettersson, and Wang Yi. Times b—a tool for modelling and implementation of embedded systems. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 460–464. Springer, 2002.

[3] Gerd Behrmann, Alexandre David, and Kim G Larsen. A tutorial on uppaal. In *Formal methods for the design of real-time systems*, pages 200–236. Springer, 2004.

[4] Pontus Boström, Petr Alexeev, Mikko Heikkilä, Mikko Huova, Marina Waldén, and Matti Linjama. Analysis of real-time properties of a digital hydraulic power management system. In *Formal Methods for Industrial Critical Systems*, pages 33–47. Springer, 2014.

[5] Peter Bulychev, Alexandre David, Kim Gulstrand Larsen, Marius Mikučionis, Danny Bøgsted Poulsen, Axel Legay, and Zheng Wang. Uppaal-smc: Statistical model checking for priced timed automata. *arXiv preprint arXiv:1207.1272*, 2012.

[6] Giorgio C Buttazzo. *Hard real-time computing systems: predictable scheduling algorithms and applications*, volume 24. Springer Science & Business Media, 2011.

[7] Alexandre David, Jacob Illum, Kim G Larsen, and Arne Skou. Model-based framework for schedulability analysis using uppaal 4.1. *Model-based design for embedded systems*, pages 93–119, 2009.

[8] Thomas Hérault, Richard Lassaigne, Frédéric Magniette, and Sylvain Peyronnet. Approximate probabilistic model checking. In *Verification, Model Checking, and Abstract Interpretation*, pages 73–84. Springer, 2004.

[9] Benedikt Huber and Martin Schoeberl. Comparison of implicit path enumeration and model checking based wcet analysis. In *Proceedings of the 9th International Workshop on Worst-Case Execution Time (WCET) Analysis*, pages 23–34, 2009.

[10] Axel Legay, Benoît Delahaye, and Saddek Bensalem. Statistical model checking: An overview. In *Runtime Verification*, pages 122–135. Springer, 2010.

[11] Marius Mikučionis, Kim Guldstrand Larsen, Jacob Illum Rasmussen, Brian Nielsen, Arne Skou, Steen Ulrik Palm, Jan Storbank Pedersen, and Poul Hougaard. Schedulability analysis using uppaal: Herschel-planck case study. In *Leveraging Applications of Formal Methods, Verification, and Validation*, pages 175–190. Springer, 2010.

[12] Alice Miller, Alastair Donaldson, and Muffy Calder. Symmetry in temporal logic model checking. *ACM Computing Surveys (CSUR)*, 38(3):8, 2006.

Appendix A

Appendix

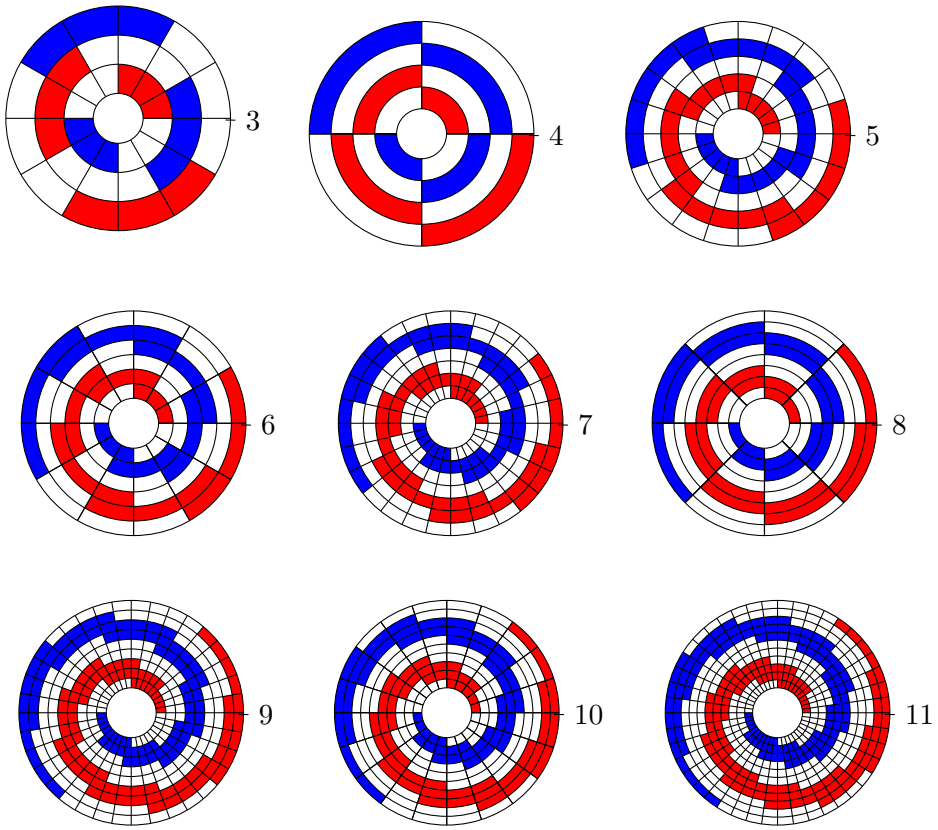


Figure A.1: Graphics detailing the intervals for Valve1Open and Valve2Open for systems with three to eleven cylinders