Introducing Formal Methods to First-Year Students in Three Intensive Weeks

Luca Aceto[1,2(✉)] and Anna Ingólfsdóttir[1]

[1] ICE-TCS, Department of Computer Science, Reykjavik University, Reykjavik, Iceland
{luca,annai}@ru.is
[2] Gran Sasso Science Institute, L'Aquila, Italy

Abstract. This paper presents a crash course whose goal is to introduce modelling and verification using model-checking technology to mostly first- and second-year bachelor students at the Department of Computer Science at Reykjavik University. The course is student driven, project based and fosters independent learning in the student body. During the course, students tackle a number of non-trivial modelling and verification tasks using the model checker Uppaal, while also practising 'soft skills' such as their communication skills, as well as their ability to work independently and as members of a team.

Keywords: Formal methods education · Modelling and verification · Timed automata · Uppaal

1 Introduction

This article describes the Real-Time Models course (henceforth abbreviated to REMO), its context and its structure. REMO is a three-week, intensive, introductory course on 'applied formal methods' we designed and have taught, individually or together, at Reykjavik University every year since the spring semester 2013.

The main goal of the REMO course is to give bachelor students at the Department of Computer Science at Reykjavik University a hands-on introduction to modelling and verification in a student-driven, project-based setting. During the course, students tackle a number of non-trivial modelling and verification tasks using the model checker Uppaal, an integrated tool environment for modelling, validation and verification of real-time systems modelled as networks of timed automata, extended with data types. As part and parcel of the course, students also hone their presentation and writing skills, as well as their ability to work

This work has been partly funded by the projects "Open Problems in the Equational Logic of Processes (OPEL)" (grant no. 196050) and "MoVeMnt: Mode(l)s of Verification and Monitorability" (grant no. 217987) of the Icelandic Research Fund.

J. F. Ferreira et al. (Eds.): FMTea 2021, LNCS 13122, pp. 1–17, 2021.
https://doi.org/10.1007/978-3-030-91550-6_1

independently and as members of a team. A number of students must follow the course in the first year of their studies, while others take it in their second or third year as an elective. This means that the course cannot assume any previous knowledge on the part of the students apart from programming and discrete mathematics. For many of the students following the course, including those in our Software Engineering BSc programme, the REMO course will provide the only opportunity to become acquainted with formal methods during their studies. In our, admittedly biased, opinion, this state of affairs is undesirable because we believe that every student graduating with a bachelor degree in a Computer-Science-related subject should have some working knowledge of 'applied formal methods'[1]. Indeed, our graduates will be the next generation of designers and developers of computing systems that will permeate the daily operations of our future society even more than they do today. This population of devices is already embedded in the fabric of our homes, shops, vehicles, farms and some even in our bodies. They help us command, control, communicate, do business, travel and entertain ourselves.

In light of the increasing complexity of computing systems, and of the fact that they control important, when not altogether safety critical, operations, we think that our students should realise that it is important to adopt high standards of quality in their development and validation, and that a key scientific challenge in computer science is to design and develop computing systems that do what they were designed to do, and do so reliably.

Our goal in the REMO course is to expose early-career bachelor students to the use of model-based approaches in the design and validation of computing systems. As discussed in Sects. 2 and 3 of this article, we do so in an intensive three-week course, during which the students use the model checker Uppaal and its underlying modelling formalism to model and analyse algorithms, games, scheduling problems and other fun scenarios with relevance to Computer Science. During the course, we focus on the use of Uppaal and introduce only the bare minimum of the underlying theoretical foundations the students really need in order to make a principled use of the tool in addressing the challenges we pose them. Moreover, we limit ourselves to presenting the features of the modelling formalism and of the tool that are relevant for the modelling and verification tasks tackled by the students. Our hope is that, after having followed this course, students will realise that the use of formal methods can have impact on the practice of the development of computing systems in a world that increasingly depends on the quality of software-controlled devices. Moreover, in our experience, students who take the course then go on to follow our master-level Modelling and Verification course or are well-equipped to take similar courses at other institutions. In addition, they serve as ambassadors for model checking technology, and formal methods in general, by enticing other students to

[1] We note, in passing, that in Iceland many of our students find well-paid jobs even before graduating with a bachelor degree and do not pursue master-level studies. To our mind, this phenomenon increases the importance of exposing them to formal methods during their bachelor studies.

follow the REMO course and by informing their co-workers of the usefulness of modelling and verification.

The rest of the paper describes the REMO course in more detail by presenting the context for course (Sect. 2), its goals and underlying pedagogical philosophy (Sect. 3), and its structure (Sect. 4). We also introduce the two 'pandemic editions' of the course and how we adapted a course designed for supervised, intensive work in class to an online setting (Sect. 5). We conclude the paper with a brief evaluation of the course based on the opinions we have received from the roughly 300 students who have followed the course since 2013, and with a discussion of some possible future steps that might increase its impact (Sect. 6).

2 Context for the Course

In each semester, teaching at Reykjavik University is divided into two distinct periods, each followed immediately by exams. The first part of the semester lasts for 12 weeks, during which students typically follow four six-ECTS courses concurrently[2]. The second part of the semester spans three weeks, which are devoted to one six-ECTS course that is taught in 'full-immersion mode'. During those three weeks, students are expected to engage in activities related to the single course they are following every working day for about eight hours per day.

Even though we have occasionally taught courses involving a substantial theoretical component during the three-week period ourselves, courses held at that time mostly have a project-based and practical component, including a focus on group work. To our mind, and based on our experience in teaching it since the spring of 2013, the REMO course fits the three-week period very well.

The course was originally designed for students in the three-year bachelor programme in Discrete Mathematics and Computer Science (DIMACS) at Reykjavik University, where it is a compulsory second-semester course. However, it is also available as an elective for students in the bachelor programmes in Computer Science and in Software Engineering. To put the students' knowledge in context, we remark that all students who enrol in the course have followed two courses in programming (Programming, Data Structures), one Computer Architecture course as well as one or two courses in Discrete Mathematics (which briefly introduce propositional logic, elementary graph theory, finite automata, grammars and regular expressions amongst many other topics). In addition, students in the DIMACS programme have taken two Calculus courses and a Linear Algebra course. Apart from testing their programs in the programming courses,

[2] The European Credit Transfer and Accumulation System (ECTS) is used within the European Higher Education Area to make studies and courses transparent and to allow students to transfer study credits between institutions, possibly located in different countries, in a seamless fashion. Depending on the country, one ECTS credit point corresponds to an average between 25 and 30 actual study hours. A bachelor-level degree course is equivalent to 180 ECTS. See https://ec.europa.eu/education/resources-and-tools/european-credit-transfer-and-accumulation-system-ects_en for more information.

none of the students taking the REMO course has any familiarity with the theory and software tools underlying modelling and verification of computing systems, the field of program correctness, and topics such as concurrency, model checking, temporal logics and real-time systems. Therefore, when 'teaching' the course, we can only rely on the students' programming experience and on their willingness to engage actively with novel and, to many of them, alien material.

3 Goals and Overall Philosophy of the Course

The main knowledge-related aims of the REMO course are

- to introduce students early in their bachelor studies to the basic ideas underlying modelling and verification of computing systems,
- to help them to develop an appreciation of the importance of those activities in the development of computing systems, and
- to make them aware of the fact that there is powerful and eminently usable tool support they can employ in their modelling and verification tasks.

Moreover, as part of the course, students develop an appreciation of the key role that models play in Computer Science, of quality criteria good models should possess, and of how models and model checkers can be used to synthesise control programs, plans and schedules satisfying a number of correctness and optimality criteria at the press of a button. In our experience, this last point is important, since students learn that, at times, it is best to describe computational tasks in a 'declarative' fashion and let our computational engines develop correct and 'optimal' algorithms for solving them on our behalf.

Since the course runs over three weeks and we want the students to be in a position to apply modelling and verification techniques already on the second day of the course, there is really room for presenting only *one* modelling formalism and *one* model checker based on it. Moreover, the course focuses solely on the application of model checking to a variety of problems and we eschew any mention of the underlying mathematical theory and algorithmics, apart from hinting at why the computational problems solved by the model checker are computationally hard. Our underlying philosophy in this course is that less is more; rather than drowning students in theoretical developments and tool features that they do not need in their modelling and verification challenges, we focus on introducing the bare necessities exactly when the students need them in a timely fashion. We trust that, having followed our introductory course, at a later stage in their studies, some of the students will be enticed to enrol in the master-level Modelling and Verification course we offer[3]—see [2] for a description of that course, which is based on the textbook [1].

[3] According to the data we have available, to date 26% of the student who followed the REMO course in the period 2013–2019 then went on to take the Modelling and Verification course. However, several of our students pursued master-level studies abroad. We expect that many of them took advanced courses on formal methods at foreign universities, but have no hard data to support this expectation.

The REMO course is centred on the seminal model of *(networks of) timed automata*, a graphical formalism for the description of real-time computing systems due to Rajeev Alur and David Dill [3]. During the course, students use the model to describe a variety of scenarios with relevance to Computer Science, and to analyse the behaviour of the systems they have modelled using the automatic verification tool Uppaal [5,6]. Uppaal is an integrated tool environment for the description, validation and verification of real-time systems modelled as networks of communicating timed automata, extended with data types.

In our course, we use the model checker Uppaal for a number of reasons. First, in our experience, students learn to use the basic features of the tool in a few hours and, after reading Frits Vaandrager's excellent introduction to the tool [17] and playing with the models accompanying that article, are ready to make and analyse their first models already by the start of the second day of the course. This opinion of ours is confirmed by Roelof Hamberg and Frits Vaandrager, amongst others, who have used the Uppaal model checker in an introductory course on operating systems for first-year Computer Science students at the Radboud University Nijmegen [13]. We think that the graphical nature of Uppaal models and the ease of use of the tool are crucial in an intensive course for students at the early stage of their bachelor studies, as these characteristics allow them to experience the usefulness of formal methods without having to understand modelling formalisms and tools with a steep learning curve. (For what it is worth, this opinion of ours is confirmed by the reports we have received from our students since the first edition of the course ran in 2013.) A second reason for using Uppaal in the course is that it is the model checking tool with which we have most familiarity, which ensures that our teaching assistants and we can provide timely answers to questions from the students. Moreover, having worked at Aalborg University with the people responsible for developing and maintaining the tool for many years, we can ask for their assistance regarding technical issues that may arise during the students' work. To our mind, prompt feedback on a variety of issues is crucial in a course that proceeds at a fast pace and lasts only three weeks.

Apart from the aforementioned goals related to the teaching of selected topics in formal methods, the REMO course also has the following pedagogical aims. During the course, students are made responsible for their own learning, and develop independence and peer-learning skills. One of the decisions we made in designing the course is that, during it, we do very little conventional lecturing. Moreover, the little lecturing we do is confined to the first morning of the course and to short sessions, during which we introduce increasingly sophisticated features in Uppaal when they may be of help for the students while they work in groups on a variety of modelling tasks.

The course is project based and student driven. One of our pedagogical tenets is that students should be prime movers in their own learning and that they can, and indeed do, challenge themselves when given the power to shape their own learning tasks. In order to entice them to do so, our course emphasises the playful aspects of modelling and verification activities, introducing important

Computer Science topics in fun and recreational settings. (See Sect. 4 for more details on the course structure.) Each student group reads the suggested material independently and solves the given modelling challenges at its own speed. Our role is to act as facilitators and to give students (hopefully helpful) hints when they have problems in their modelling work or face the state-explosion problem while verifying their models. A second important soft skill the students hone in the course is an ability to work both independently and as members of a group of four–five students. Last, but by no means least, students develop their technical-communication skills since they have to present their work both orally, as part of two conference sessions and at a final oral exam, and in writing, in the form of two project reports.

4 Structure of the Course

As mentioned above, the REMO course runs over a period of three weeks in the spring semester at Reykjavik University. (Interested readers may found some information on the 2021 edition of the course at http://www.icetcs.ru.is/remo-course/.) Each course day lasts roughly from 8:30 till 17:00. The mornings are mostly devoted to supervised independent work by the student groups. Students work independently in the afternoons, but we are available to answer their questions and to assist them in resolving issues they might encounter.

4.1 Week One: Warming up

The first week of the course serves as a warm-up period in which the students become acquainted with the context for the course, the Uppaal tool and the model of networks of timed automata it supports, and start working on a variety of modelling challenges. Nearly all the actual lecturing we do in the course is concentrated on the first morning of the first day of the course. During that course session, we introduce students to the context for the REMO course, presenting the correctness problem for computing systems as one of the key scientific challenges in Computer Science from its early days to today, and highlight various approaches to modelling and verification of computing systems, pointing to their applications in industry and to the development of trustworthy real-life applications with which students are familiar[4]. We also briefly introduce the model of networks of timed automata and the basic aspects of the query language supported by Uppaal, initially ignoring timing-related features. We typically do so in two steps. First, we discuss a version of the 'small university' system described in [1]. The goal of that very simple example is to highlight how a system can be described using automata running in parallel and communicating via synchronous handshakes. We present a deadlock-free version of that system and a variation that can exhibit a deadlock, analysing their behaviour by hand with

[4] The slides we used for this introduction in the 2021 edition of the course are available at http://www.icetcs.ru.is/remo-course/remo-intro.pdf.

the students' help. Next, we discuss the actual Uppaal model of the classic Gossiping problem (see, for instance, [14]) presented by Vaandrager in [17]. The Gossiping model serves a number of purposes at this very early stage in the course. In particular, it allows us to present some of the data types supported by Uppaal and some of the key features students will employ in their models (such as the concepts of nondeterministic selection of values for variables, and the use of guards and updates on transitions), as well as the use of the Uppaal verifier and its diagnostic-trace option to find an optimal sequence of phone calls the agents can use to share all their secrets.

The model of the Gossiping problem also highlights early on in the course one of the messages that we want students to take home, namely that the model focuses on describing *what* each agent can do at any given time given its current 'state' rather than on *how* the agents can optimally achieve their goal. Once a faithful model has been built and the goal the system should achieve has been expressed as a suitable reachability query in the Uppaal specification language, finding the optimal scheduling of phone calls is best left to the computational engine of the tool. In our experience, even for students in the age of machine learning, moving from an algorithmic and procedural way of thinking to a declarative one feels like a Copernican revolution for many students, and is best stressed right away and repeatedly in the course. Moreover, the analysis of the Gossiping model using Uppaal gives students their first introduction to the state-explosion problem. We show them how the processing time used by the tool increases substantially with the number of agents in the system. We think that this fact of Computer Science life is also worth highlighting early on in the course, since students are used to getting fast response from their devices to just about any computational task they have faced so far. As the course develops, they will see that the choices they make in their modelling tasks can crucially affect the time it takes for their verification efforts to complete, and that they might have to show some patience in waiting for an answer to their queries.

At the end of the morning session on the first day of the course, students form groups, which typically consist of four or five students, and we encourage them to spend the afternoon reading Vaandrager's Uppaal tutorial [17], to watch a video describing the Uppaal tool produced by one of our PhD students and to familiarise themselves with Uppaal by examining the models accompanying Vaandrager's introductory article.

By the start of the second day of the course, most student groups are typically ready to start working on modelling and verification challenges. We suggest that they begin by analysing the *How much can you lose?*[5] and *How much can we reach?*[6] puzzles and a number of classic mutual-exclusion algorithms. These warm-up challenges show students how they can turn pseudo-code descriptions of algorithms into Uppaal models, and introduce them to some of the conceptual challenges and classic pitfalls in concurrent programming. In order to make it easier for them to build models efficiently, we devote some time to showing

[5] See Exercise 23 at http://people.cs.aau.dk/~kgl/ESV04/exercises/index.html.
[6] See Exercise 24 at http://people.cs.aau.dk/~kgl/ESV04/exercises/index.html.

students how to express programming constructs such as loops and conditionals in Uppaal and how, using state transitions in Uppaal models, they can describe easily which operations are atomic and which are not, which is a key issue in concurrent programming. Moreover, using the Uppaal simulator and verifier, students can explore the effect that different atomicity assumptions have on the behaviour of apparently simple concurrent programs, leading to results that, at least at first sight, appear counter-intuitive.

A key (and fun) modelling challenge we pose the students early on during the first week of the course is to model the one- and two-dimensional solitaire games described in [4, Chap. 6][7] using Uppaal, and to employ the Uppaal verifier to check that the games can indeed be solved and to find solutions for them involving the least number of moves. This modelling exercise sets the stage for the two group projects on which the students work in the last two weeks of the course, and further reinforces the usefulness of declarative models.

As the first week of the course evolves, we introduce students to some of the timing-related features in Uppaal, always striving to focus on providing the least amount of information students need to use them properly in their models. In particular, we present the use of clocks in Uppaal models to express time-dependent system behaviour (with focus on guards and invariants), the peculiarities of variables of data type `clock`, the way in which time elapses in a network of timed automata and advanced features such as urgent channels as well as committed and urgent locations. Students employ these features right away in synthesising the control program for a coffee machine in the problem described at

http://people.cs.aau.dk/~kgl/ESV04/exercises/index.html#coffee.

The work done by the students during the first week of the course does *not* contribute to their final grade. We do so to allow them to explore the material independently and at their own speed, giving them enough time to become familiar with Uppaal, its underlying model and query language and to hone their modelling skills. We think that this decision of ours contributes to creating a positive atmosphere in the course in which students feel that they can learn by making mistakes, without affecting their final grades.

We devote the start of the course session on the last day of the first week of the course to a 'conference-like session', which we chair. During that session, each group of students delivers a short talk presenting their solution to one of the problems they tackled during the week to everyone involved in the course. Each presentation is followed by questions and comments from the audience, both on the quality of the modelling work and of the presentation. We stress to the students that giving good presentations based on their work is a skill they will need in their future careers, but one whose importance is not widely appreciated and that is, unfortunately, not practised sufficiently in many degree courses in Computer Science.

[7] See http://www.ru.is/faculty/luca/MV2011/solitaire.pdf for a scan of the relevant pages.

We close the first week of the course by presenting the first project for the course, so that students who want to start working on it can do so right away.

4.2 Week Two: First Project

The last two weeks in the course are mainly devoted to two group projects, which together account for 70% of the final grade for the course and form the basis for the final oral exam. In keeping with the 'recreational' atmosphere of the course, both projects entice students to explore challenging topics in formal methods in a 'serious-game setting'[8]. The projects we set our students change regularly, but the ones we describe in this section and in the subsequent one are two of our favourite ones and we have used them for the last three editions of the REMO course.

The project we have recently used for week two of the course is inspired by Vaandrager and Verbeek's article on designing vacuum-cleaning trajectories [18]. Working on it, apart from honing their modelling and verification skills, and learning about the connection between winning strategies in games and control software for a simple robotic-inspired application, students realise that automated support is needed to avoid (or at least reduce the number of) errors that human programmers make in developing even relatively simple systems.

Briefly, in their first project, students work with a vacuum-cleaning-robot problem from the book [19]. On pages 67–69 of that book[9], Wooldridge describes an example of a small robotic agent that will vacuum clean a room. In our version of the example, the room is a 3-by-3 grid and at any point the robot can move forward one step or turn clockwise 90°. The problem is to find a deterministic, memoryless strategy for the robot in which

1. its next action only depends on its current square and orientation (one of north, west, south, east), and
2. all squares are visited infinitely often.

Wooldridge gives a partial specification of such a strategy using a number of rules. Ignoring the actions of the robot having to do with sucking dirt and focussing only on the actions related to movement, the rules given by Wooldridge are:

- If $In(0,0)$ and $Facing(north)$ then $Do(forward)$.
- If $In(0,1)$ and $Facing(north)$ then $Do(forward)$.
- If $In(0,2)$ and $Facing(north)$ then $Do(turn)$.
- If $In(0,2)$ and $Facing(east)$ then $Do(forward)$.

[8] Other courses in formal methods employ recreational problems to engage students to good effect. By way of example, we limit ourselves to mentioning that Rozier has used magic-square, chess and Rubik's cube puzzles in an applied formal methods course offered to undergraduate and graduate students in Aerospace Engineering, Computer Science and Computer Engineering at Iowa State University [15].

[9] The relevant pages are available at http://icetcs.ru.is/fm-at-work/IntroductiontoMultiAgentSystemsWooldridgepp67-69.pdf.

According to Wooldridge, 'similar rules can easily be generated that will get the agent to $(2, 2)$, and once at $(2, 2)$ back to $(0, 0)$.'

In their project work, we ask the students to model the above scenario using Uppaal and to check whether Wooldridge's aforementioned claim is correct. Having realised that there is actually *no* deterministic, memoryless strategy for the vacuum cleaner that extends Wooldridge's four rules, the students then find out that one has to remove all but the first rule in order to find a suitable strategy for the robotic agent that meets the stated criteria. Next, we ask the students to consider a number of timing-based scenarios, asking them to find the fastest strategy for the robot to vacuum clean the grid. Having examined a number of specific settings of the time f it takes the robot to move forward and the time r that the robot needs to rotate, and after having mapped all the possible combinations of forward moves and rotations in a strategy, the students quickly realise that a shortest strategy is also a fastest one unless $2r < f$ and that, in that case, a strategy with 16 rotations and 10 forward moves is faster than a shortest one, which involves 12 turns and 12 forward moves.

In order to entice our students to develop scalable models and to reflect on the qualities of their models as a whole, we also ask them to consider the vacuum-cleaning scenario in a 4-by-4 grid and to assess their models vis-a-vis the seven criteria listed by Vaandrager in [17, Sect. 1.10].

Each student group delivers a project report describing their work on the vacuum-cleaning project, together with all their Uppaal models and query files. Again, we devote the start of the course session on the last day of the second week of the course to a 'conference-like session', during which each group of students delivers a short talk presenting their work on the project.

4.3 Week Three: Second Project and Final Exam

The last week of the course is devoted to the second group project and ends with a final, oral exam contributing 30% of the final grade.

The second project we have used most often over the years is based on the solitaire game Rush Hour, which is today produced by Thinkfun. Students are expected to model the game using Uppaal and to use the tool to solve the puzzle for a variety of starting configurations. (See http://www.icetcs.ru.is/remo-course/project2.html for the latest version of the project description.) Apart from being fun, Rush Hour is an example of a challenging solitaire game, which lends itself to a number of variations that can be used to exercise the students' modelling abilities.

First of all, Rush Hour is hard, for humans and machines alike! Indeed, its generalised version is PSPACE-complete [10] and, as recently shown in [7], this hardness result holds true even with only 1×1 cars and fixed blocks. Moreover, the hardest solvable configurations for the game's six-by-six board found by Collette, Raskin and Servais in [8] are fiendishly difficult for humans, involving as many as 93 moves to solve optimally[10]. Second, the game can be modelled in

[10] See http://www.icetcs.ru.is/remo-course/RushHourHardestConfigurations1.pdf for a list of the six hardest game configurations.

a number of natural ways and we encourage our students to explore at least two modelling approaches, comparing the ease with which they can be analysed using the Uppaal verifier. We think that this experience is particularly instructive for students, since it helps them realise that apparently innocuous modelling choices can have a huge effect on processing time during their verification and makes them realise the exponential growth in the number of possible interleavings in the executions of concurrent systems in a playful setting. To make state-explosion come to the fore even more, we ask students to model the Rush Hour game in which a step in the game allows one to move a vehicle more than one position at a time.

In order to make students evaluate the extensibility and reusability of their models, we ask them to model the version of Rush Hour with walls and use the Uppaal verifier to solve the hardest such puzzles given at https://www.michaelfogleman.com/rush/.

Finally, we use the game to introduce students to issues related to mutual exclusion in concurrent programming, some of which they will meet in their future courses on Operating Systems and Concurrent Programming. Concretely, we ask them to assume that a two-handed player can move two vehicles (one position) simultaneously. The player is left-handed and takes 3 s to move a vehicle one position with the left hand, but 5 s to move a vehicle one position with the right hand. The keenest students are expected to model this scenario using Uppaal and to find the fastest way of solving the puzzle from some of the starting configurations we provide. This task is optional and open ended; we stress to the students that we are mainly interested in seeing how they approach the modelling task, and in the type of possible pitfalls they identify. Having said so, most student groups over the years have attempted to solve this problem and we even had one group of students that considered a scenario in which the player is an octopus!

The last week of the course ends with a final, oral exam. Even though the course is very much project based and students do report to us when some group members are not contributing to the project work, we still need to check whether some students are free riding. To this end, we examine one student group at the time as follows. The exam session starts with a presentation in which the group being examined presents its work on the second project. We then ask questions to the group based on their work during the three weeks. Initially, questions are addressed to the group as a whole and we give everyone who wants to answer the chance to do so. If we realise that some students do not attempt to answer any of our questions, we direct some questions specifically to them to gauge how much they know about the work their group has done during the course. Each student receives an individual grade for the final exam, whereas work on the projects is graded at the group level.

5 The Two Pandemic Editions

The REMO course was largely held in a 'Renaissance-workshop style'[11] in the period 2013–2019. All student groups worked independently in the university building in a suitably-sized lecture room and in the surrounding areas, as needed. As mentioned above, our role is largely a supporting one. We are there to assist students in their independent learning and work. In those editions of the course, we were available to answer questions from each group during their supervised working sessions in the mornings, and regularly visited each group at that time to check how their work progressed and whether they had any specific issues that were stifling their progress. Whenever problems of general interest arose, we would call all groups for a brief plenary presentation addressing those problems and how they might be tackled using Uppaal, as well as providing pointers to the literature. Student course evaluations over the years indicate that students like this workshop-like setting, and feel empowered by the trust we show in them and the level of independence we expect in their work. To wit, we limit ourselves to mentioning a comment we received from one student, who later asked for a letter of references from one of us:

> I enjoyed taking your course and loved how the barrier between teacher and student was broken during the course. (I have been and will continue to recommend the course to any student at Reykjavik University that I know).

During the pandemic, we had to hold the 2020 and 2021 editions of the course fully online, which were followed by 19 and 24 students, respectively. However, despite the online setting, we strove to maintain the workshop-style structure of the course and to uphold its key pedagogical tenets mentioned in Sect. 2. To this end, we ran the course trying to mimic the in-person sessions in an online setting as faithfully as possible. The course was held on Jitsi and we were available to answer questions from the student groups at the same Jitsi link throughout the course from 8:30 till 17:00, taking shifts with one of our PhD students. We also had a Discord channel for the course and most student groups used Discord to coordinate their project work.

We started the day each morning with a scrum meeting, during which each group would report on the progress they made and on what they planned to do during the day[12]. We alternated between public scrum meetings, in which all groups were virtually present at the same time, and group-specific scrum

[11] By this, we mean that our course sessions tried to replicate the creative atmosphere of the busy workshops in which many of the great Renaissance works of art were produced. Those workshops were run by experienced artists, who trained young ones in their crafts and also learnt from the new ideas produced by their trainees.

[12] We also held impromptu scrum sessions at the start of each day in all the in-person editions of the course. However, due to the lack of face-to-face contact with the students and in order to help students structure their remote working day, we felt that we needed to be much more systematic in doing so during the two pandemic editions.

meetings. All conference sessions and the final exams were held online, following the same structure we adopted for the in-presence editions of the course.

Based on the feedback we received from the students who took the pandemic editions of the course, this set-up worked rather well and the students felt that they were making progress independently, while receiving a good level of support from us. They were pleased with the willingness we showed to answer their questions and with our short response time. The following excerpt from a particularly eloquent course evaluation we received from a second-year student in Software Engineering is representative of the students' opinions:

> 'The course was conducted entirely online, using various platforms to interact with the professors and other students. It was interesting inviting people you just met into your room, virtually, and presenting your work. It takes some time getting used to, but if this past year has taught us anything, it's that working from home is very much possible. Inevitably, some people had minor technical issues but overall, I believe that this aspect of the course, as well as all other aspects, was a success and think that we all learned something new in these three weeks. Even the professor took on a new communication tool he had never used before which was popular among the students. It's all about patience and willingness to adapt to a new environment'.

Based on that circumstantial evidence, it seems that we did manage to meet our goals to hold an engaging, student-driven course that makes them responsible for their own learning. However, the price we had to pay to do so was to devote more time and energy to the course than we typically do in an in-person setting. Even though it may seem counter-intuitive, empowering students in an online setting required more involvement from us during the afternoon course sessions than it did in an in-person setting, where students work in the university building with little or no assistance from us. Based on discussions with our colleagues both at our institution and elsewhere, as well as on the literature on teaching during a pandemic we have read over the last year (see, for instance, [9,11,12]), it is fair to say that we are not alone in feeling that way. To wit, even though such figures have to be taken with a grain of salt, it is estimated that teaching online can roughly triple the preparation time for a one-hour lecture [12]. Moreover, according to a survey mentioned in [9], university lecturers overwhelmingly thought that the pandemic has made their job more difficult than it was before, with 87% of the respondents either strongly agreeing or agreeing. Some of the possible reasons for the increased workload on teachers during the pandemic are mentioned in [11, Sect. 4.3.2]; these include the need to record lectures in advance and edit them thereafter to clarify and/or correct some of the material, the interaction with learning management systems that sometimes needed to be adjusted to support teaching taking place fully online, the use of more time for grading assignments since students ended up working alone on tasks that could be carried out in groups, and answering the same questions repeatedly and on a variety of tools used to communicate with the students taking a course.

6 Evaluation and Conclusions

The REMO course has run at Reykjavik University in the three-week part of the spring semester every year since 2013. It has been followed by roughly 300 students overall and has consistently received high grades from the students in their course evaluations. To wit, the average evaluation of the students' satisfaction with the course lecturers since 2013 is 4.9 out of 5. The average score of the course in the period 2013–2021 in the other evaluation criteria adopted by Reykjavik University is as follows:

- Course satisfaction: 4.79 out of 5.
- Assessment of the outcome of the course: 4.61 out of 5. (This metric refers, for example, to factors such as overall understanding of course content, increased interest in the academic field and study materials, and personal learning success).
- Course organisation: 4.84 out of 5. (This metric refers, for example, to factors such as the teaching plan, learning objectives, course assessment, planning of classes, and organisation of assignments).
- Evaluation of how the course assignments helped in increasing student understanding of the study material: 4.79 out of 5.

(Of course, pleasing as they may be, the results of these student course evaluations should be considered merely as a measure of student satisfaction with the course. As indicated in, for instance, the meta-analysis of faculty's 'teaching effectiveness' reported in [16], that metric is not related to student learning).

Most importantly, we believe that the course has made the students taking it aware of the existence of mature and usable modelling and verification technology developed by the research community working on formal methods over many years. Those students now have a basic understanding of what formal methods can (and cannot) do, know that they are routinely employed within the high-technology companies whose products they use daily, and have repeatedly told us that they will consider using model checkers in their future studies and careers as appropriate. Moreover, they serve as ambassadors for the course, and hopefully also for formal methods, within the rest of the student population and in their future workplaces. By way of example, we limit ourselves to citing two comments we received from students in their course evaluations:

> 'Very interesting course, introducing techniques and tools that I think it is likely that I will use in the future. I chose the course blindly, having no idea of what formal methods and model checking are. I was positively surprised by what I learnt. I will recommend this course to my colleagues. (A second-year student in Computer Science)'

> 'At the start of the course, I had absolutely no knowledge on the subject.... The main takeaway was that using human ingenuity only takes you so far when optimizing complex systems and it can be difficult to prove that you have the most efficient design without using the proper

> tools and methods we learned about in the course.... The work I did in this course is some of my proudest work during my studies. (A second-year student in Software Engineering)'

We think that 'spreading the formal-methods gospel' is important at a department like ours, since far too many of our students graduate without taking a single course that exposes them to formal modelling and verification. One of the roles of the REMO course is to try and change this state of affairs, but it is fair to say that we still have work to do, despite our efforts since 2013. For instance, the number of students taking the course is still fairly small (ranging from 18 in 2013 to 36 in 2016) and increasing it will be non-trivial in the light of some compulsory three-week courses that students in Computer Science and Software Engineering take in the spring semester. So far, first-year students for which REMO is compulsory account for 46% of all the students who have taken the course, 19% have been second-year students and 29% have been third-year or exchange students. Since the Discrete Mathematics and Computer Science degree course is a niche study programme, it is unlikely that the percentage of first-year students taking the course will increase noticeably. Therefore, we will target third-year students, even though some of them might be finalising their final bachelor projects in the spring semester.

We might also consider turning REMO into a typical twelve-week course, while maintaining its philosophy. We believe that this is doable by identifying a suitable number of modelling and verification problems that student groups can solve in two-to-four-hour, supervised study sessions. However, doing so would dilute the students' study experience somewhat, the course would lose its high-intensity flavour and we would have to compete for the students' brain cycles with the other courses running in parallel with that version of the REMO course.

Moreover, since the course has run since 2013, we should carry out a data-driven analysis of how well it achieves its learning outcomes and high-level goals. For instance, it would be interesting to collect and analyse data to try and understand whether students taking our course find that the introduction to formal methods they received has had an impact on their practice. Furthermore, we should collect student feedback on whether our 'warm-up week' does play a role in creating a nurturing and positive learning environment at the start of the course.

In conclusion, based on our experience, one can introduce formal methods to early-career bachelor students in high-intensity-training mode in as little as three weeks. Our REMO course provides a possible blueprint for doing so, while empowering students to learn and work independently, and challenging them to step out of their intellectual comfort zone and to take charge of their own development. We hope that our experience can be useful to others.

Acknowledgements. We are grateful to the anonymous reviewers and the PC chairs, who offered some detailed and insightful comments and suggestions that led to several improvements in our original manuscript. We thank Elli Anastasiadi for her excellent work as student mentor during the last three editions of the REMO course. In par-

ticular, Elli's assistance has been invaluable during the two pandemic installments in the spring of 2020 and 2021. We are most grateful to Kim G. Larsen for the Uppaal-related course material he has generously shared with us over the years and to Marius Mikucionis for the sterling support he has offered our students and us on Uppaal-related matters. Catia Trubiani provided useful feedback on a draft of this paper. Our colleagues Nidia Guadalupe López Flores and María Óskarsdóttir pointed out some interesting papers on teaching during the pandemic. Last, but not least, we thank Hildur Daví sdóttir for eloquently sharing her opinions on the 2021 edition of the REMO course with us. Any remaining infelicity is solely our responsibility.

References

1. Aceto, L., Ingólfsdóttir, A., Larsen, K.G., Srba, J.: Reactive Systems: Modelling, Specification and Verification. Cambridge University Press, Cambridge (2007)
2. Aceto, L., Ingolfsdottir, A., Larsen, K.G., Srba, J.: Teaching concurrency: theory in practice. In: Gibbons, J., Oliveira, J.N. (eds.) TFM 2009. LNCS, vol. 5846, pp. 158–175. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04912-5_11
3. Alur, R., Dill, D.L.: A theory of timed automata. Theor. Comput. Sci. **126**(2), 183–235 (1994). https://doi.org/10.1016/0304-3975(94)90010-8
4. Arnold, A., Bégay, D., Crubillé, P.: Construction and Analysis of Transition Systems with MEC. AMAST Series in Computing, vol. 3. World Scientific (1994). https://doi.org/10.1142/2505
5. Behrmann, G., David, A., Larsen, K.G.: A tutorial on Uppaal. In: Bernardo, M., Corradini, F. (eds.) SFM-RT 2004. LNCS, vol. 3185, pp. 200–236. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30080-9_7
6. Behrmann, G., David, A., Larsen, K.G., Pettersson, P., Yi, W.: Developing UPPAAL over 15 years. Softw. Pract. Exp. **41**(2), 133–142 (2011). https://doi.org/10.1002/spe.1006
7. Brunner, J., et al.: 1 × 1 rush hour with fixed blocks is PSPACE-complete. In: Farach-Colton, M., Prencipe, G., Uehara, R. (eds.) 10th International Conference on Fun with Algorithms, FUN 2021, 30 May–1 June 2021, Favignana Island, Sicily, Italy. LIPIcs, vol. 157, pp. 7:1–7:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2021). https://doi.org/10.4230/LIPIcs.FUN.2021.7
8. Collette, S., Raskin, J.-F., Servais, F.: On the symbolic computation of the hardest configurations of the RUSH HOUR game. In: van den Herik, H.J., Ciancarini, P., Donkers, H.H.L.M.J. (eds.) CG 2006. LNCS, vol. 4630, pp. 220–233. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-75538-8_20
9. Flaherty, C.: Faculty pandemic stress is now chronic. Inside Higher Ed, November 2020. https://www.insidehighered.com/news/2020/11/19/faculty-pandemic-stress-now-chronic
10. Flake, G.W., Baum, E.B.: Rush hour is PSPACE-complete, or "why you should generously tip parking lot attendants". Theor. Comput. Sci. **270**(1–2), 895–911 (2002). https://doi.org/10.1016/S0304-3975(01)00173-6
11. Flores, N.G.L., Islind, A.S., Óskarsdóttir, M.: Effects of the COVID-19 pandemic on learning and teaching: a case study from higher education. CoRR abs/2105.01432 (2021). https://arxiv.org/abs/2105.01432
12. Gewin, V.: Pandemic burnout is rampant in academia. Nature **591**, 489–491 (2021). https://doi.org/10.1038/d41586-021-00663-2

13. Hamberg, R., Vaandrager, F.W.: Using model checkers in an introductory course on operating systems. ACM SIGOPS Oper. Syst. Rev. **42**(6), 101–111 (2008). https://doi.org/10.1145/1453775.1453793
14. Hurkens, C.: Spreading gossip efficiently. Nieuw Arch. voor Wiskunde **5/1**(2), 208–210 (2000). http://www.nieuwarchief.nl/serie5/pdf/naw5-2000-01-2-208.pdf
15. Rozier, K.Y.: On teaching applied formal methods in aerospace engineering. In: Dongol, B., Petre, L., Smith, G. (eds.) FMTea 2019. LNCS, vol. 11758, pp. 111–131. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-32441-4_8
16. Uttl, B., White, C.A., Gonzalez, D.W.: Meta-analysis of faculty's teaching effectiveness: student evaluation of teaching ratings and student learning are not related. Stud. Educ. Eval. **54**, 22–42 (2017). https://doi.org/10.1016/j.stueduc.2016.08.007
17. Vaandrager, F.W.: A First Introduction to Uppaal. https://www.mbsd.cs.ru.nl/publications/papers/fvaan/handbookuppaal/. to appear in Quasimodo Handbook, J. Tretmans editor
18. Vaandrager, F.W., Verbeek, F.: Recreational formal methods: designing vacuum cleaning trajectories. Bull. EATCS **113** (2014). http://eatcs.org/beatcs/index.php/beatcs/article/view/269
19. Wooldridge, M.J.: An Introduction to MultiAgent Systems, 2nd edn. Wiley, Hoboken (2009)