Introducing Formal Verification with LEGO®

David Lawrence, Dimitri Racordon, Maximilien Colange, Steve Hostettler, Alban Linard, Edmundo López Bóbeda, Alexis Marechal, Matteo Risoldi, Nicolas Sedlmajer, Didier Buchs

Centre Universitaire d'Informatique
Université de Genève
7 route de Drize, 1227 Carouge, Suisse

Abstract. "In the end, you are a mathematician, not a computer scientist" or "Have we not already discovered everything in computer science?". Which theoretical computer scientist has not heard a similar sentence when trying to explain their research to a layperson? Promotion of theoretical computer science, and formal methods in particular, is mainly hindered by the high level of abstraction commonly used in the field, and probably furthermore complicated by a lack of education of the people thereupon.

We present in this paper an educational experiment intended to explain the importance and aim of formal methods, along with the challenges they present. It uses LEGO® robots as an anchor to the real world, introducing fun into the presentation. Primarily targeting high-school students, it forms a solid and adaptable basis to reach various audiences.

1 Introduction

Despite the formidable growth of computers in our daily lives (some talk about a "fourth industrial revolution"), masses remain poorly educated about computers and computer science. The commercial success of smartphones and tablets today, of connected objects tomorrow, proves the greed of the masses for new applications of computers. Most vendors focus on the ease of use as marketing arguments, so that the same masses tend to forget the high level of technology concentrated in their phones, and the long way we have gone since the Enigma machine. Many people do not understand why research in computer science is still undertaken, presumably because they have no idea of its possible content – the current applications being already beyond their wildest dreams.

It is therefore computer scientists' responsibility to promote computer science, and educate people about computers. It is not only a matter of a better social recognition for "geeks", computer science education is of primary importance in forthcoming issues that computer are posing to our democratic societies: surveillance, ethics, but also security and liability. Are computers reliable enough to make them responsible for nuclear power plants management? For autonomous driving? For stock markets monitoring? Such questions, that begin to emerge in the public debate, obviously ask about formal verification.

We propose in this paper our modest contribution to this topic, in the form of a small experiment, intended to be both easy to understand, easy to carry out and yet complete enough to stimulate interest of both adults and young adults. With it, we introduce the main concepts that formal methods deal with, including motivation for their study, and limits to their usage. This experiment is based on the use of a small robot, whose task is to get an object from a point A to a point B. We have developed a simple framework for basic programming of such a robot, in order to hide programming technicalities from the audience, allowing them to focus on the verification challenges. The code can then be simulated in the framework, and finally uploaded onto the real robot, to observe the successful execution of the program.

In this paper, we give a description of our kit and of its application in several contexts, including a recent event that took place at the University of Geneva, named *(R)amène ta science !*[1]. We then investigate several extensions of our framework to include more complex aspects of formal verification while keeping a level of difficulty affordable for high and post-secondary school students. The remaining of this paper is organized as follows: Section 2 exposes the main concepts related to our kit; Section 3 gives an extensive description of our kit; Section 4 summarizes our experience using it and the feedbacks we got from students and teachers; based on these, Section 5 investigates the desired goals for such an experiment, and defines an extension to our existing kit; finally Section 6 concludes with a discussion about the approach.

2 Introduction to the key concepts

We based our approach on *key concepts* to help the subject gradually build its definition of formal verification. We may understand these key concepts as individual subdivisions of the reasoning behind formal verification. Indeed, each concept is based on the previous one, asking questions that go beyond the range of this latter. That way, starting from the system to be verified itself, the subject should eventually acquire a good understanding of each concept and its limitation. Based on that idea, we defined six key concepts, as listed below. Figure 2.1 puts a visual representation on this concepts ordering, summarizing the questions they should answer.

System This concept encapsulates the understanding of software programming, that is the means by which a programmer can *build* a system. The very essence of this concept is rather intuitive; the basic idea of programming can easily be explained using better-known definitions, such as a list of instruction steps, or cooking recipes.

Validation This concept encapsulates the notion of ensuring that something works and asks how one can carry out this task. Explaining what risks caused by programming flaws may arise, and in particular what consequences they may have is usually a good way to introduce this concept.

[1] `http://ramene-ta-science.unige.ch`

Testing This concept is very close to the previous one, introducing the notions related to tests execution. It emphasizes the purposes of a test, covers questions related to the design of test scenario and even more importantly, brings in the limitations of such approach.

Simulation This concept is central to our method since it is the first one to break with the empirical approach, introducing a theoretical one. It proposes the use of a system simulation that will no longer be validated but rather *verified.*

Modeling This concept extends the notion of simulation. It encapsulates the understanding of abstraction and mathematical representation of a real system.

Verification This concept finally introduces the formal verification as its own discipline, using previously acquired knowledge to understand its differences with empirical approach, and also the limitations induced by the accuracy of the model being used.

Depending on the targeted audience, some of these concepts may be partially intuitive or even fully acquired. For instance, a software developer is likely to be well accustomed with *System* and *Validation*, while a software engineer would be more familiar with *Testing* and *Modeling.* Nevertheless, independently of the level of understanding of these concepts, the path of reasoning remains identical and allows one to gradually understand the purpose of formal verification and notions related to it.

3 Description of our kit

History. This project was elaborated after a common reflection of our Software Modeling and Verification Group (SMV) group, at the University of Geneva, of how to present verification of software to non computer engineering experts. The choice of having a Domain Specific Language (DSL), a simulation for performing the verification and finally an execution on the actual device were central to our idea. The initial project started in 2009 for the 450th birthday of the University. For this event, a video has been recorded and is available on `dailymotion.com` on a short or long version[2]. Initially the aim was to be able to perform the exercise in a whole class equipped with PCs. Later on, we were asked to do this experiment in several occasions for student visits. In 2011 the University asked us to repeat it for the *Nuit de la science* that was held in 2012. We decided, with the support of the University, to perform a redesign of the application in order to simplify its use, since then featuring an unique environment hiding the underlying IDE, a user-friendly editor, an improved simulation, and a movable board to easily transport the kit. We also introduce an interface with Scratch[3] in the application. This version was successful during *Nuit de la Science* and was experienced by people from various age and education. For some it was

[2] `https://smv.unige.ch/events/portes-ouvertes-2009`

[3] `http://scratch.mit.edu`

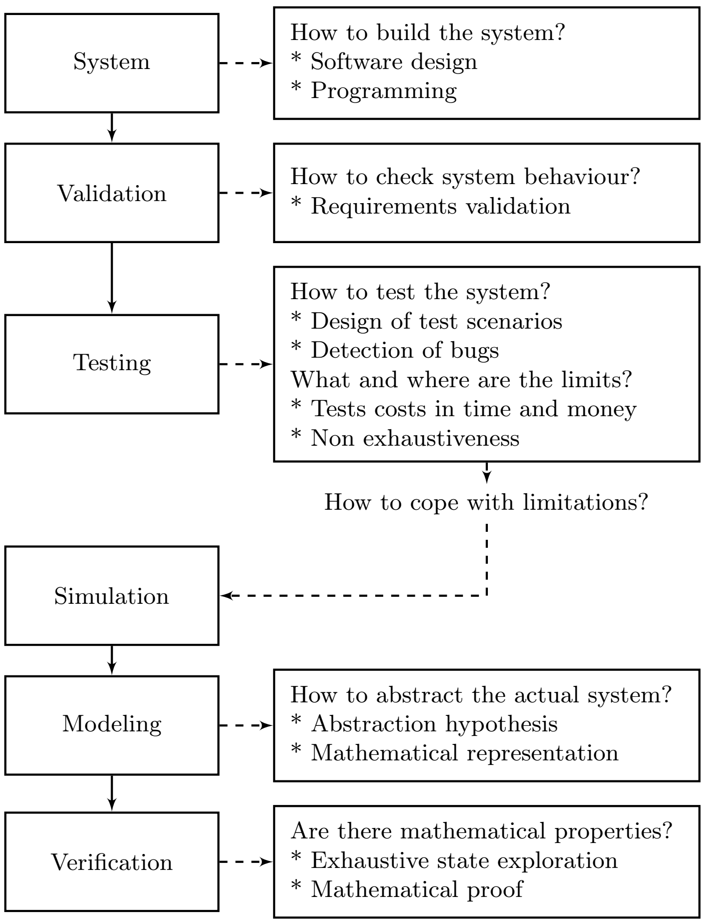

System
How to build the system?
* Software design
* Programming
Validation
How to check system behaviour?
* Requirements validation
Testing
How to test the system?
* Design of test scenarios
* Detection of bugs
What and where are the limits?
* Tests costs in time and money
* Non exhaustiveness
How to cope with limitations?
Simulation
Modeling
How to abstract the actual system?
* Abstraction hypothesis
* Mathematical representation
Verification
Are there mathematical properties?
* Exhaustive state exploration
* Mathematical proof

Fig. 2.1: reasoning path leading to our six key concepts

the opportunity to write a program for the first time. We reused the same kit for some new events: Ethnopoly 2012, High School visits in 2013 and finally *(R)amène ta science !* in 2014.

Goals. The objective of the kit is to write and validate a program that controls a movable robot, shown in Figure 3.1, which must carry an object from a given position to another area of the environment using its built-in clamp. To increase the interest in the exercise, the environment contains a large cruciform obstacle in its center that the robot should not touch under any circumstances. At any step of the programming, the student can simulate their current program to visualize its behavior, as pictured in Figure 3.2. During the code interpretation, the simulator intentionally injects small direction errors to simulate real world constraints. These errors physically represent small delays between the two direction motors, respectively left and right. The simulation ends when the objective is accomplished, when the program failed to achieve the goal or when there is no more code left to execute. In other words, one might say that the program is considered valid if and only if the robot can carry the object from its initial location to its final one without touching any obstacle and without provoking any errors due to a bad utilization of the clamp's instructions. In fact, such cases may arise if the student orders the opening of an already opened clamp, or inversely. To summarize, the verification of the robot program must fulfill three properties:

1. the object in the initial position should eventually reach its destination area;
2. the robot should never touch the obstacle;
3. the robot should never be broken by a bad usage of its features.

A complete description of the experiment workflow is distributed to the audience by the means of a flyer, as shown in Figure 3.3.

Software Application. Obviously, the primary component of the kit corresponds to the software application, which provides the student with a dedicated framework where s/he can carry out the experiment. The application consists of a modest integrated development environment that handles the code writing, compiling and execution. Additionally, it features a mean to visualize the execution of the code using a 2D schematic view of the robot and its environment. Furthermore, one of the most important features of our kit is that it proposes a DSL that greatly simplifies the programming step of the robot. The latter is composed of six basic instructions, namely:

1. `turn left`;
2. `turn right`;
3. `move forward <x> centimeters`;
4. `move backward <x> centimeters`;
5. `open clamp`;
6. `close clamp`

In fact, in order to focus on the verification aspect and not on the issues related to robotic programming, it is primordial that the programming remains as simple as possible. Therefore, the usage of such an instructions set allows the student to apprehend this step of the experiment with a reasoning very close to their natural language.

Finally, when the student wants to upload their code on the robot, the application generates the complete code of the program written with the DSL, compiles it and sends it to the robot through an USB interface.

LEGO® Robot. To observe the execution of the student's program in *the real world*, we built a LEGO® robot using the same specification described by the simulator. The robot is controlled by a NXT EV3 microcontroller[4] on which is installed LeJos[5], an alternative operating system designed for the LEGO® Mindstorms, supporting the execution of Java programs.

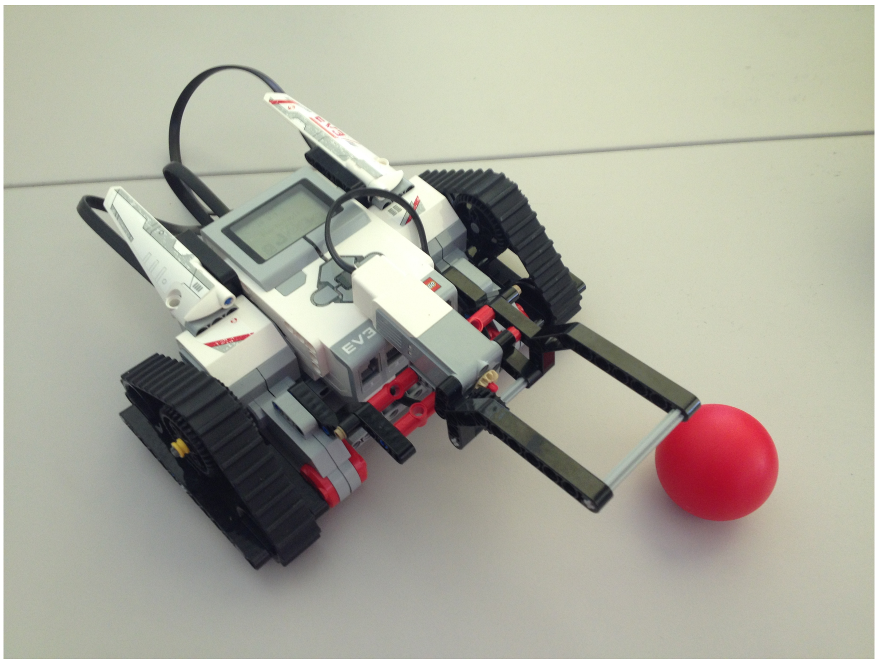


Fig. 3.1: Picture of the LEGO® robot

4 Our experience

First Edition in 2009. As stated in section 3, we tried our kit for the first time in 2009, for the 450th anniversary of the University of Geneva.

Students were globally very interested in the experiment and could understand its objective with ease. At that time, the demo was split in two phases. Students were first given a sheet of paper with the grid representing the world environment, on which they were asked to draw the path the robot would follow before programming it in the second phase. Surprisingly enough, students encountered few if no difficulties to translate their drawings into an actual program. It also helps breaking the cliché of the computer scientist always stuck behind a screen.

[4] `http://www.lego.com/en-us/mindstorms/products/ev3/31313-mindstorms-ev3`
[5] `http://www.lejos.org`

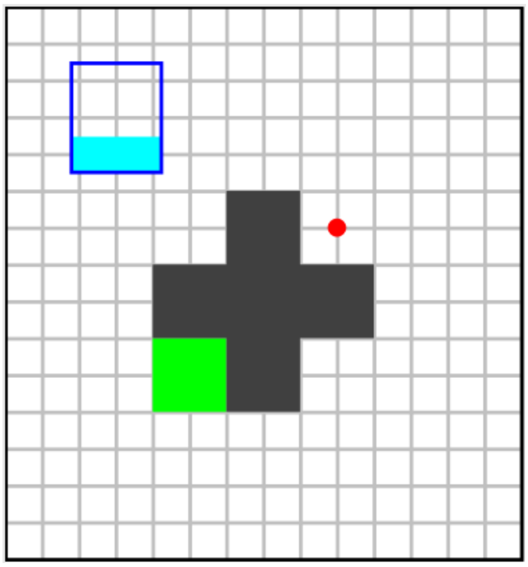
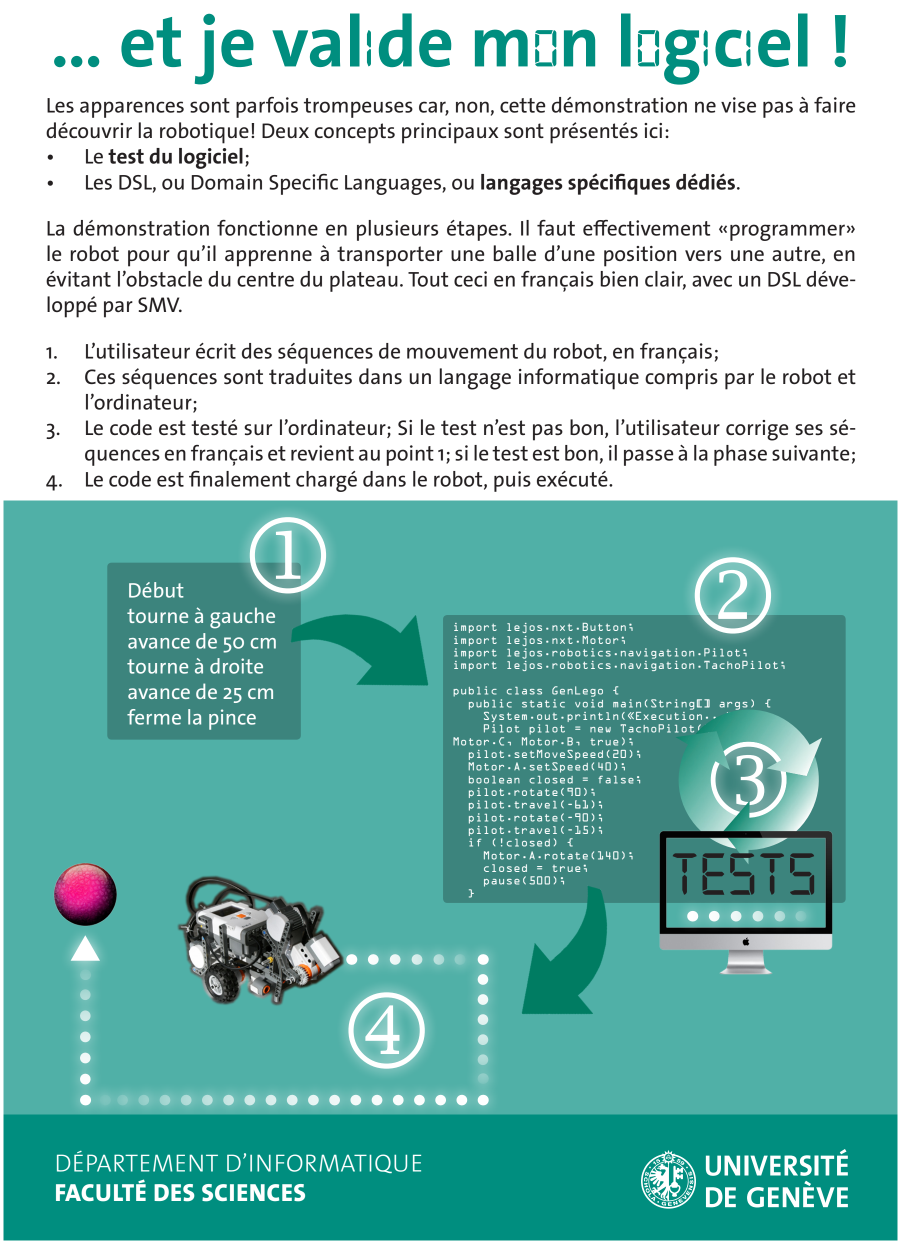


Fig. 3.2: Print screen of the simulator

(R)amène ta science ! in 2014. The event *(R)amène ta science !* took place on May 2014, as an initiative of the Faculty of Science of the University of Geneva. Its objective was to promote science studies to high school students by the means of an original concept where students acted as *ambassador* of a particular science and taught their own classmates. Several kits were made available to students, including ours, among with a small lecture on how to use them. Then, students who had learned how to use the kits passed on their knowledge and animated an interactive demonstration where their classmates could use the experiments. The originality of the concept lay in the fact that students had little assistance from the teachers, thus relying on their own words and understanding to carry out the animation.

From the students' perspective, the experiment had been perceived globally very well. Each of them could complete the objectives and even refine their own solution after having noticed their program was not optimal. After being introduced to the kit, ambassadors were quite enthusiastic about the idea of showing what they had learned to their classmates. They even turned the experiment into some sort of friendly competition to who was the fastest to provide the best program. At this point we may stress that all students could assimilate the notions required to understand the kit in very little time. Because they could use the simulator as a *try and fail* tool to build their program, programming itself had not been an issue. Nevertheless, we observed that some of them were confused about the representation on the simulator and were at first not fully capable of matching it to the real world. This led to some misuse of the direction instructions, the language being composed of *right* and *left* movement



... et je valide mon logiciel !

Les apparences sont parfois trompeuses car, non, cette démonstration ne vise pas à faire découvrir la robotique! Deux concepts principaux sont présentés ici:

- Le **test du logiciel**;
- Les DSL, ou Domain Specific Languages, ou **langages spécifiques dédiés**.

La démonstration fonctionne en plusieurs étapes. Il faut effectivement «programmer» le robot pour qu'il apprenne à transporter une balle d'une position vers une autre, en évitant l'obstacle du centre du plateau. Tout ceci en français bien clair, avec un DSL développé par SMV.

1. L'utilisateur écrit des séquences de mouvement du robot, en français;
2. Ces séquences sont traduites dans un langage informatique compris par le robot et l'ordinateur;
3. Le code est testé sur l'ordinateur; Si le test n'est pas bon, l'utilisateur corrige ses séquences en français et revient au point 1; si le test est bon, il passe à la phase suivante;
4. Le code est finalement chargé dans le robot, puis exécuté.

```
Début
tourne à gauche
avance de 50 cm
tourne à droite
avance de 25 cm
ferme la pince
```

```
import lejos.nxt.Button;
import lejos.nxt.Motor;
import lejos.robotics.navigation.Pilot;
import lejos.robotics.navigation.TachoPilot;

public class GenLego {
  public static void main(String[] args) {
    System.out.println(«Execution..
    Pilot pilot = new TachoPilot(
Motor.C, Motor.B, true);
  pilot.setMoveSpeed(20);
  Motor.A.setSpeed(40);
  boolean closed = false;
  pilot.rotate(90);
  pilot.travel(-61);
  pilot.rotate(-90);
  pilot.travel(-15);
  if (!closed) {
    Motor.A.rotate(140);
    closed = true;
    pause(500);
  }
```

DÉPARTEMENT D'INFORMATIQUE
FACULTÉ DES SCIENCES

UNIVERSITÉ DE GENÈVE

Fig. 3.3: Flyer distributed to the audience

commands. Interestingly, we could not observe the same alertness on the experiment objectives with the teachers. Indeed, none of these teachers were computer scientists and since they were merely observers, they could not catch the notions as well as their students.

In all the different events where we used our kit, we noticed that one drawback of our current approach is that it does not focus enough on the verification aspect. Indeed, because we use a robot, people tend to first think they are being introduced either to programming or robotics. It is only once the issue of verification and validation have been made clear that the audience was able to fully capture the essence of our message. We believe that this problem might well be solved by further extensions of our kit.

5 Extending the experiment

As described above, our current demonstration kit does not match all the objectives described in Section 2. We consider in this section extensions of the current kit, following two axis:

- an extension to match all the goals described in Section 2, more specifically the formal verification part;
- adaptations to different audiences.

These axis are orthogonal and therefore considered separately.

5.1 Extending the kit to describe formal verification

Verifying Goal Completion. As we have seen, our current demo is centered around the motion of the robot: its abstract modeling as perfect straight lines and right angles, its imperfection to capture the real behaviors such as drifting . . . This modeling allows a simulation, introducing random imperfections, so as to validate a program as *being able* to achieve the goal (moving the object to the target zone). If now we want to *verify* such a program, we should check that the goal is always achieved, whatever the drifting and imperfection in the robot motion. Once again, the notion of drifting and motion imperfection should be defined formally and integrated to the formal model. It seems natural that the possible paths are embedded within a cone around the perfect path, and the possible final positions for the object within a zone. If this zone is included in the target zone, then the goal is always achieved. In addition, we also need to make sure that the cone never intersects what would be the geometrical representation of the obstacle. Based on this principle, a method to formally verify the objective completion would be to first compute the bounds of this cone. The easiest way to do so is to compute them step by step, summing the error computed at the previous steps. For each step, the result of this computation is an ellipse representing the robot possible positions after the completion of the step where the two dimensions are *1*) the error $\Delta = \delta \cdot d$ on the total traveled distance d; *2*) the error $\lambda = \tan\alpha \cdot d$ on the robot direction, α being the maximum angle of the direction error. Parameters δ and α may be adjusted depending on the assumed precision of the robot within the simulation model.

Figure 5.1 illustrates two steps of computation. The point s is the initial position of the robot, t the targeted position after the first step and u the position after second step. As we can see, t is surrounded with an ellipse representing the possible actual position once the distance d_1 traveled. This same ellipse is then surrounded by a new one, representing the additional error induced after travelling distance d_2.

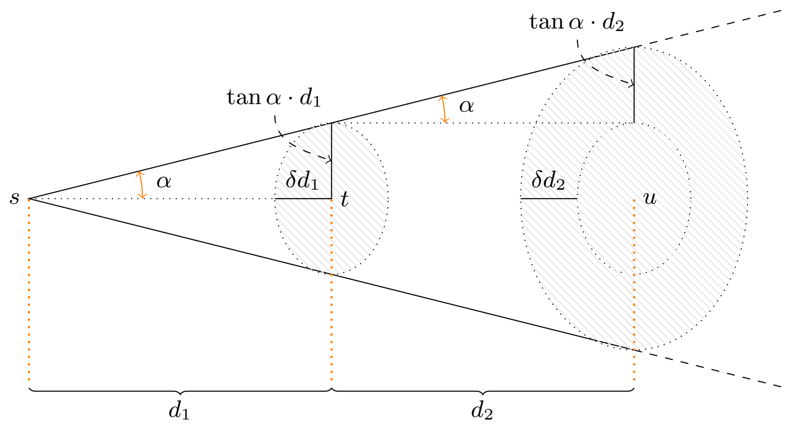


Fig. 5.1: Illustration of the method to verify the objective completion

At this point, we remark that the motion errors are cumulative: the more motion steps, the larger the final position zone, and the less likely to have a guarantee for the program. This is not a desirable property for a demo, as we would like the audience to be able to produce correct (i.e. that pass the verification step) programs. A solution is to widen the target zone, but then the goal would appear too easy.

Furthermore, it consists in verifying a physical property that has little to do with the program written by the audience. It renders the active participation (writing a program for the robot) somewhat artificial. We want a stronger link between the programs written by the participants and the verification step; the audience should be more motivated, therefore more involved, thus strengthening the impact of the demo.

Finally, this kind of verification involves mostly geometric calculus and can be solved analytically. We fear that this example would not make clear for the audience the distinction between mathematics and computer science. We believe that techniques such as finite model-checking would better differentiate computer science from other sciences in the audience's eyes.

Verifying the Cooperative Algorithm. For all these reasons, we propose to extend the scenario with a cooperative task. The task of moving an object from an initial position to a target zone remains the same, but now there are two (or more) robots. Their first action is to decide which one will undertake this task, through a leader election protocol. The goal can be also slightly changed: moving the object from an initial position to an intermediate one, and then to a final one, the two parts of the path must be done by different robots. The election thus decides which one undertakes the first part, and which one the second part. Another simple cooperative scenario involves an obstacle on the target path: one robot is responsible for removing this obstacle (e.g. push a button to open a door), the other one is responsible for moving the object to the target zone.

The key addition here is the leader election protocol, as the system to be modeled and verified. Such a protocol is a perfect teaching example, as it allows to exhibit non-trivial undesirable behaviors (such as deadlocks or non-termination).

Enhanced Robots. We first have to add a communication capability to our robots. The Mindstorms pack features Bluetooth communication that can be used for communication between robots.

Enhanced DSL. To keep the demo accessible to a large audience, we need to add to our DSL a few instructions regarding the infrared sensor. To keep things simple for the audience, we think that defining listeners is a good approach. The program to be designed is thus a set of listener, associating a sequence of actions to a message. One of these listeners should contain the code for robot motion, to be executed by the leader once it has been elected. Again for the sake of simplicity, we think we should restrict the type of messages. Three solutions come to mind:

– allow only a set of pre-defined messages;
– allow to send an integer in a message;
– allow to send a string in a message.

The first solution seem over-protective, and it raises the problem of naming the pre-defined messages: generic naming (such as `msg1`, `msg2` ...) is likely to be an obstacle to the audience. Although mapping generic names to arbitrary purposes may seem natural to a computer scientist, it is not at all the case for high-schoolers. For the same reason, the integer solution is to be ruled out. The third (strings) solution allows flexibility and readability: users choose their messages, but they have to ensure that all sent messages are taken account of in listeners. It is also a method prone to typos. Although we favor this solution, we keep in mind that this is an important point to be tested.

We also have to choose how messages are cast. Bluetooth allows both unicast and broadcast. As for the DSL, both modes should be available: a typical first step in a leader election protocol consists of all working agents broadcasting their IDs, while unicast is necessary to break the initial symmetry. IDs (such as letters A, B ...) will be assigned to robots, the DSL offering to send a message to a robot identified by its ID.

To sum up, the structure of the code is changed, and a program now consists of a set of listeners. Among them, there should be a special one to be executed by the leader only once elected, and an initial one to be executed at the beginning (typically the broadcast of IDs), although this initial step can also be hardcoded. Each listener (except the special ones above) are parameterized by a message to react to. This listener structure spares both loops and 'receive' statements. Two new statements are added, one to unicast a message to a specific robot, one to broadcast a message. We will also need a symbolic name for the ID of the robot, as well as operations over the IDs, or at least comparisons, which also naturally lead to a conditional statement ('if'). Finally, our DSL is enhanced with two sending statements, a conditional statement and comparison between IDs.

All of these are sufficient to implement simple election algorithms such as Chang and Roberts [1] or HS algorithm [2] where agents are assumed to be in a ring topology. This can be simulated by an increment operation on IDs.

Enhanced Demo. In order to illustrate the possible behaviors, we plan to have a few predefined runs of the same leader election protocol: one that succeeds, and two failing for different reasons (such as deadlocking and non-terminating). The first one shows that the protocol works, and the second one that it is flawed. The explanation of the flaw can be a good introduction to concurrency problems, to which any audience should be aware of our networked world. The depth of the discussion about the deadlock should of course be calibrated according to the audience and the available time. Once these two runs have been shown, one or two formal methods are presented to the audience: their principles, and how to apply them in our use case. The framework we use should embed at least one verification tool, along with a translator from the DSL to an input format for this tool. A patched, correct version the leader election protocol is then presented, and tested with the verification tool. In a final step, the audience is given the opportunity to design their own programs to achieve the task. These programs are then validated through simulation and verified (or not) through verification, before being uploaded onto the robots for the show.

Implementation. The idea of a renewed demo is quite recent, and its implementation has not been undertaken yet. This explains some of the question marks above, especially about the programming tools left to the audience for the leader election protocol design. We still have to find a good balance, to keep the goal challenging, yet without loosing the audience with too complex constructs or concepts. This step will require some testing time. We nevertheless have hired an intern to implement the new version of the demo, whose work should start during Summer 2014.

5.2 Adapting the demo to various audiences

Considering the event *(R)amène ta science !*, our demo was primarily focused on high-school students. We are also likely to take part in similar local events, rather focused on middle-school students, and on an occasional basis. Another promotional event, *la Nuit de la Science*, targets a family audience. We thus seek to adapt our demo to adults.

First of all, the use of the LEGO® robot is deliberate, as an anchor to the real world – it is much simpler to talk about a robot than about abstract concepts such as algorithms or abstract machines. This robot seems to be an important means to catch the attention of children and teenagers, and even older when we experimented the demo with some colleagues. Even for adults, the anchor to the physical world is important, even for people with a strong, but loosely related to computer science, background.

Furthermore, the concepts that we identified as educational goals were carefully chosen for their relative simplicity. Thus, we consider that the adaptation to the audience mostly resides in the *degree of details* achieved in the precision. The key to an appropriate adaptation is to avoid introducing notions not directly related to goals that would go beyond the comprehension of the audience. When middle-schoolers would only be introduced to the distinction between validation and verification, to modeling as an unfaithful representation of the reality, senior engineers would be explained e.g. the biases introduced during modeling, both conscious and unconscious, and their possible repercussions on the limits of the subsequent verification, or the demo could focus on the integration of the verification process within a product development cycle. Admittedly, we do not have such elaborate scenario, as such opportunity has not occurred to us yet.

As for the middle- and high-schoolers, our new kit is yet to be tested, but we believe to have achieved a solid demo for these cases, and a strong basis for further adaptations.

6 Conclusion

Benefits of our approach As desired, our kit proved to be easy to understand and easy to carry on with the students who had been using it. Adapting our level of assistance, we could use it in several classes and events, targeting primary to high school students, and we believe we might use it to teach young adults too. The biggest advantage of our approach surely lays in the fact that the objective of the experiment is simple to understand but yet introduces issues related to the verification and validation. First we demonstrate by the example the importance of validation by using the robot as a substitute of a valuable asset. Second we give the student tools to understand notions such as model and simulation, thus helping him to take a glimpse at much more complicated concepts such as system properties and verification of these latter.

We may also stress that unlike what we first thought, our experiment was still challenging for quite a lot of people. This observation led us to reconsider the way we present our science to non-computer scientist since we realize that most of the time we fail to capture simple elements, thinking they are too trivial to deserve an explanation.

Difficulties and Future Work. There are two difficulties in the demo we propose here. The first one has been identified during previous events: although the robots are intended to be an anchor to the physical world, it blurs our target message, as the demo is often perceived as oriented to robotic programming, rather than verification. We thus have to work our speech to constantly emphasize on the verification purpose.

We have proposed an extension to our kit, that remains unimplemented. An intern will start working on this in the following weeks, and will hopefully grant us soon with a new working demo. But there remain uncertainties regarding the difficulty of the proposed task. We have considered some variations of our demo, that will have to be tested on live audiences in order to find the appropriate balance. Nevertheless, we hope that our efforts will contribute to a better understanding between formal methods and the public.

References

1. Chang, E., Roberts, R.: An improved algorithm for decentralized extrema-finding in circular configurations of processes. Commun. ACM 22(5), 281–283 (May 1979), http://doi.acm.org/10.1145/359104.359108
2. Lynch, N.A.: Distributed algorithms. Morgan Kaufmann (1996)