TEACHING ABSTRACTION IN MATHEMATICS AND COMPUTER SCIENCE
A Computer-supported Approach with Alloy

M. Simonot, M. Homps and P. Bonnot
LIASD, Université Paris 8- IUT Montreuil, 140 Rue de la Nouvelle France, 93100 Montreuil, France

Keywords: Abstraction, Alloy, Lightweight Formal Method, Modelling, Teaching Mathematics for Computing, Computer Supported Education.

Abstract: Abstraction skill is difficult but essential when learning computer science and mathematics. In this paper, we present an experience from the Computer Science Department of IUT de Montreuil in which mathematics and computing teaching work together in order to provide abstract thinking skills to students. Our approach is to connect mathematics and software engineering through a computer-supported approach which uses - in mathematics and software engineering undergraduate courses- Alloy, a lightweight formal method from MIT. Alloy serves as a continuum from micro-modelling activities -used to enforce an abstract understanding of basic mathematic concepts- to real software modelling practice. It enables us to elaborate a kind of exploratory learning where students are actively engaged.

1 INTRODUCTION

Abstraction is a vital skill when learning Computer Science as well as Mathematics. Keith Devlin (Devlin, 2003) explains clearly the importance of abstraction for Computer Science : “ Once you realize that computing is all about constructing, manipulating, and reasoning about abstractions, it becomes clear that an important prerequisite for writing (good) computer programs is the ability to handle abstractions in a precise manner“. But learning abstraction is hard and many students tend to stay at a low level of abstraction, focusing immediately on the code rather than the design.

The need for abstraction gives to mathematics course an important part of computing curriculum. But many students have a negative attitude.

Observing this difficulties raise the following question: how mathematics and computing can work together in order to provide abstract thinking skills to students?

In this paper we address this question. As we are concerned with Computer Science curriculum, our first choice was to concentrate on software modelling and on discrete mathematics, logic and sets. Indeed, software modelling is a good way to practice abstraction and discrete mathematics is its mathematical background. In order to connect as soon as possible mathematical concepts with Computer Science, our second choice was to introduce mathematical notions with some kind of micro-modelling activities where students use these notions to express things.

But in order to appropriate the notions our students must take an active part. To address this problem our third choice was to adopt a computer-supported approach. with the help of Alloy (Jackson, 2006) (Alloy, 2007), a lightweight formal method dedicated to rapid software modelling created by Daniel Jackson’s, group at MIT. This tool allows us to elaborate a pedagogy based on the Actions-Process-Object model (Fenton and Dubinsky, 1996) which is considered today as the central model of concept formation.

Our proposal can be summarized as follows : our approach is to connect mathematics and software engineering through the use- in mathematics and software engineering courses- of Alloy. Alloy serves as a continuum from micro-modelling activities -used to enforce an abstract understanding of basic mathematic concepts- to real software modelling practice.

This leads us to reorganize our curriculum as follows : the first mathematics course of our curriculum is devoted to logic, sets functions and relations, induction .... Every time a notion is introduced, the students have an Alloy laboratory session whose role is the appropriation of this new notion. Laboratory ex-

Simonot M., Homps M. and Bonnot P..
TEACHING ABSTRACTION IN MATHEMATICS AND COMPUTER SCIENCE - A Computer-supported Approach with Alloy.
DOI: 10.5220/0003898302390245
In *Proceedings of the 4th International Conference on Computer Supported Education* (CSEDU-2012), pages 239-245
ISBN: 978-989-8565-07-5
Copyright © 2012 SCITEPRESS (Science and Technology Publications, Lda.)
239

ercises are elaborated following the APO model. At the end of the course, as students familiarize themselves with the notions and the tool, some laboratory sessions are devoted to introduce them to a model driven approach of software engineering and to give them a mathematical representation of key concepts of computing such as inheritance, composition, abstract classes . . .

Alloy is then from time to time used in Computer science courses such as programming language, database relational model, Uml modelling, when introducing computing concepts or formalisms and in order to give them a mathematical meaning. In this way, the work made with mathematics and Alloy doesn't substitute for usual computing topics. It improves them by giving them a common language.

The cross-disciplinary characteristic of Alloy labs requires a mixed teaching team. Our team is composed of 2 mathematics and one computing teacher. Each lab session is elaborated in common and taught in parallel.

2 RELATED WORKS

Computer-supported approach has been extensively used in mathematic education in order to enforce students motivation and to develop an active attitude. A common approach is to use web technology as in (Gregory, 2004). Other works (Siller and Greefrath, 2009) share with us a focus on mathematical modelling but most of them use software tools as Computer Algebra systems (CAS), Dynamic Geometry Software (DGS) or Spredsheet programs (SP). Our approach is very closed from (Fenton and Dubinsky, 1996). We adopt their APO model of concept formation together with a computer-supported approach The main differences are (1) that we use a lightweight formal method while they use a programming language, (2) that our approach doesn't restrict to mathematics but is cross-disciplinary. We use the same tool to enforce an abstract understanding of basic mathematic concepts as well as to practice real software modelling.

The idea to introduce Object-Oriented concepts with the preliminary use of a modeling language is also present in (Hadar and Hadar, 2007). In this paper, authors insist on two properties of UML : abstraction barrier and visual representation. Those properties are also properties of Alloy and have motivate our choice. The use of UML for Object-oriented concepts introduction is also present in (Bennedsen and Caspersen, 2004). UML and Alloy serve the same goal : let the students have the "big picture" of programs, but the use of Alloy lets also the students have a mathematical understanding of O.O concepts.

The belief that the main intellectual challenge of software engineering is abstraction and design is commonly shared and leads a lot of institutions to introduce formal methods as soon as possible in undergraduate curriculum. (Sotiriadou and Kefalas, 2000) describes a formal method course in the second year of the undergraduate curriculum which focuses on software modelling and uses the Z formal specification language. In Rochester Institute of Technology (RIT) (Lutz, 2006), (Naveda et al., 2009) the software engineering program has been redesigned in order to expose students to both formal and informal modelling throughout the entire curriculum. Discrete mathematics take place in the first year and formal methods for specification and design such as Z, VDM, and more recently Alloy are introduced as soon as possible.

In (Boyatt and Sinclair, 2008) a software design undergraduate module which uses Alloy is described. Like us, they highlight two characteristics of Alloy: experiment and visualization which help shape the mathematical understanding of a model and allow a kind of exploratory learning. The fact that the use of Alloy is compartmentalized in one module is mentioned in the paper as a limit of their approach. To this effect, We can say that our work extends this work because of our use of Alloy as a continuum from mathematic concepts learning to software design modelling.

Few works address our problem of how to decompartmentalize mathematics and computing learning. We can cite (Sekerinski, 2009) where mathematics is used as the unifying force of all core elements of software design. But, in this approach the focus is placed on specifications, algorithms and correctness proofs with the use of Weakest Preconditions Calculus while we focus on mathematical modelling. We can also cite (Cohoon and Knight, 2006) which describes efforts made at Virginia University to connect software engineering and discrete mathematics. Their solution differs from ours: they systematically motivate introduction of discrete mathematic notions by preliminary software engineering problem studies.

3 ALLOY

Like all formal method, Alloy is made of a language and a tool which supports the language. An Alloy text (a model) is made of sets and relations definitions and of formulas expressing constraints over them. But Alloy is a lightweight formal method i.e. a method which has a formal basis but is limited in its scope.

The limitation of Alloy is in the analyzer supporting the language: it uses SAT solving techniques to check the model. The analyzer therefore doesn't provide general results as theorem proving does, but it gives a quick way of exploring instances of the model and generating counterexamples.

Alloy provides two commands: `run` and `check`. With the `run` command, Alloy tries to find instances which satisfy definitions and constraints of the model. If it succeeds, satisfiability of the model is proved, but if it fails, we don't have guarantee of unsatisfiability. The `check` command is used to check if a given formula is a consequence of the model. In this case, The analyzer tries to find a counterexample. A success proves that the assertion is wrong, a failure doesn't prove anything.

Let us illustrate Alloy with a simple example, inspired from the famous scene of the Sergio Leone's film: *the good, the bad and the ugly*:
-"You see, in this world, there are two kinds of people, my friend : those with loaded guns and those who dig. You dig."

We need a `Human` set partitioned by 2 disjoint subsets `GunMan` and `Digger` and also a `Hole` set and a `Treasure` set. We define 3 relations in order to express the facts that gunmen threats diggers, that diggers dig holes and that holes contains treasures. The Alloy model is given in listing 1.

Listing 1: The gunman model.

```
sig Hole{contain : lone Treasure}
sig Treasure{}
abstract sig Human{}
sig Digger extends Human{ dig: set Hole}
sig Gunman extends Human{threat :set Digger}
```

Figures 1(a) shows an instance obtained with the `Run` command. Our model, doesn't enforce the `contain` relation to be injective. Executing `check{injective[contain,Treasure]}` will produce counterexamples as shown in figure 1(b).

4 LEARNING MATHEMATIC NOTIONS WITH THE HELP OF ALLOY

4.1 Visualization and Active Learning

Alloy doesn't prove anything but provides a quick way of exploring instances of models, generating counterexamples and visualizing the results. In other

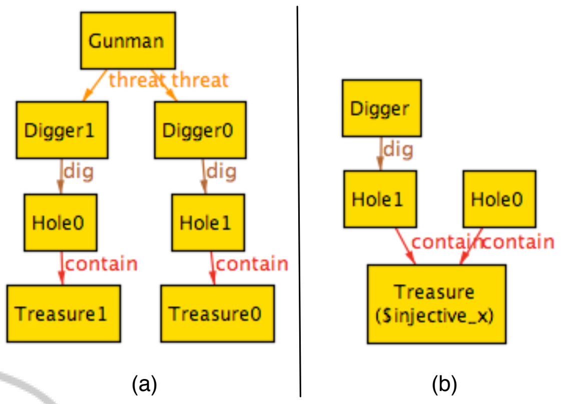

Figure 1: An instance (a) and a counterexample (b).

words, the tool shows a representation of what the student has written. The teacher can then adopt an attitude where he never gives the answer to an exercise but helps the student to find how the tool will give it. This kind of iterative interaction with the tools imposes active learning, let the student be responsible for what he says and allows him to interiorize the notions.

4.2 A Constructivist Approach

In this section, we will explain how Alloy allows a pedagogy based on the Action-Process-Object model (Fenton and Dubinsky, 1996) which is considered today as the central model of concept formation.

According to this model, three successive mental constructions are needed in order to develop an understanding of a mathematical concept. At the first stage, the learner performs **actions** on object, according to a given algorithm. When the learner gains the ability to refer to these actions using symbols or is able to imagine the action without needing to perform it explicitly, we say that the actions were transformed into a **process**. The last stage is transforming the process into an **object**: the learner can refer to the entity concept much like he refers to a physical entity.

Let us consider again our GunMan example in order to illustrate the elaboration of exercises corresponding to the three stages of the APO model dedicated to the learning of relation concept.

According to APO model, at the beginning of the learning process, students need to work on concrete examples and to follow step by step instructions. For this purpose, it is convenient to work with a fix instance of the model. Let us then add the Alloy text of listing 2 to the model-listing 1 in which sets and relations are extensionally defined:

Listing 2: A fix instance of gunman model.

```
one sig chest,nugget extends Treasure{}
one sig theBad,theUgly extends Digger{}
one sig theGood,other extends Gunman{}
one sig h1,h2,h3  extends Hole{}
fact{
contain =  h1->chest + h2->nugget  &&
threat  =  theGood->theUgly + theGood->theBad +
           other->theBad &&
dig     =  theBad->h1 + theUgly->h3 + theBad->h3
}
```

We can then ask the students to perform step by step the compose definition and to interact with Alloy for self-assessment as in the following exercise:

Exercise 1. *Add the following 2 sentences to the GunMan model and complete the* `mystery` *fact by giving the tuples corresponding to the compose of* `dig` *and* `contain`*:*
```
fact mystery{ m1= //to be completed}
check { m1= dig.contain}
```

After the definition has been processed, we can move to a little more abstract level as in the following exercise.

Exercise 2. *Add the following 2 sentences to the GunMan model and give a meaningful name to* `m2`*. You can have help by exploring instances with Alloy:*
```
fact { m1= dig.contain}
fact { m2= threat.m1}
```

Understanding `m2` supposes to take `m1` as a whole, as an object. To be able to give a meaningful name to `m2` (for instance `possess`), requires then a mental representation of `m2`, i.e. of the compose concept itself. Alloy visualization of instances and work in autonomy contributes strongly to this process.

The next step is to ask students to define their own relations.

Exercise 3. *Define the* `threatWith` *relation which maps gunmen to gunmen who threat at least a same person. Is your relation reflexive (to be test with Alloy)? How to proceed if we don't want it to be reflexive?*

At this stage, students must be able to combine relations in order to define a new one. Contrary to the other steps, Alloy will be helpful only after they have formulated a definition, as a test tool. Moreover they are engaged in an iterative (micro-)modelization process: using mathematic notions to express concrete phenomenons.

Previous exercises work on concrete relations. It is now time to free from them. Three kinds of activities are provided: Analyze of given models, model design and mathematical problems. The first and the second will be detailed in the next section. Mathematical problems are the way to connect the work done in Alloy laboratory sessions with traditional paper and pencil mathematic sessions. Here is an example:

Exercise 4. *Is a symmetric and transitive relation necessary reflexive? Use Alloy to check your guess and then prove it on a paper.*

We evaluate student's ability to build an interaction with Alloy in which the student role is restricted to the formulation and the investigation of the problem. Alloy counterexamples should lead to understand that only a *total* symmetric and transitive relation is necessarily reflexive. It can be first checked with Alloy and then proved on paper.

5 A MODELLING-FIRST APPROACH OF COMPUTING

As soon as students have acquired the basic mathematical notions, they are able to model simple software systems with Alloy. To begin software modelling in the math course is interesting because in math course students don't ask for writing code. Students don't balk at doing such exercises because they see the connection with computing. Furthermore, modelization is an abstraction process and modelization exercises are a perfect way to practice abstraction. From the computer science point of view, it allows to expose our students to software modelling from the beginning of our curriculum and to give a mathematical intuition of basic concepts of computing.

5.1 Introducing Object Oriented Notions

Let us take the example of a static description of a multimedia library. Such a library has several media. Each media has a unique reference, a unique title and a unique borrower. Different media must have different reference. This exercise which is a standard UML exercise can be modelled with Alloy with the basic mathematical notions previously used as shown in listing 3.

As it can be noticed, Alloy language has been elaborated in order to be mathematically simple and

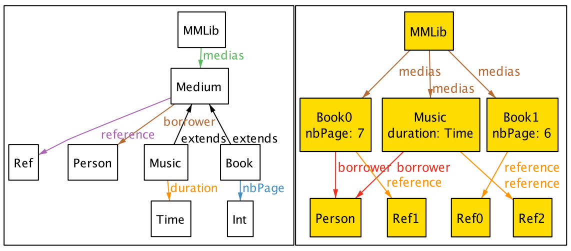

Figure 2: Multi media library with meta model and instance.

also to be closed to object programming and modelling language. We define sets and relations over sets, but it looks like class and instance variables. In fact, it gives a mathematical semantic of object language.

Listing 3: Multi media library model.

```
sig MMLib{
 medias : set Medium
}
abstract sig Medium{
 reference  : one Ref,
 borrower: lone Person
}
fact{injective[reference ,Ref] }
sig Person, Ref, Time{}

sig Book extends Medium{
 nbPage: one Int
}
sig Music extends Medium{
 duration : one Time
}
```

When students will be introduced to object programming, they will have this kind of mathematical intuition to engage with the subject.

The same work can be done with inheritance as shown in listing 3. The Alloy keyword `extends` has been first explained with notions like inclusion, partition and disjoint subsets. When students will be introduced to inheritance, they will be able to look at inheritance as inclusion. This is the key for a good use of inheritance in object conception, for understanding that subclasses inherit instance variables and methods from super-classes, and to understand Liskov Substitution Principle.

There is also a strong connection between UML class diagram and meta-models generated by Alloy, as illustrated by figure 2. Our use of Alloy let students have a unified view of programming and software design. Lastly, the generated instances of software models such as the second box of figure 2 help to consider an object program as a graph, i.e. as a set of related objects which cooperate. Introduction of notions as responsibility delegation will be easier.

5.2 Introducing Theoretical Aspects of Computing

As Alloy is a formal method, it obviously allows to introduce theoretical notions such as specifications, Pre/Post conditions, system invariant, algorithm refinement of specifications, finite state machines and so on. Each of these topics relies on the possibility to model the dynamic of a system. In this case, Alloy visualisation provides is a kind of symbolic execution of the system which allows to test specifications. One of the key point is that these notions can be introduced in a simple way and without the need of proofs. Students are in this way exposed to a declarative approach of computing.

6 ASSESSMENT

In this section we will analyze to what extent we have achieved our goals.

Is Alloy Really Simple to Use? One of our hypothesis was the fact that Alloy, from our point of view, seemed to be very simple to use. This point is confirmed. We didn't provide any lessons devoted to the use of the tool nor Alloy tutorial lab sessions. The use of Alloy was explained degree by degree along lab sessions devoted to mathematical exercises. The most difficult issue for students is to engage interaction with the tool. We relate this fact with the question of active learning and explain it in the next subsection.

Does Computer Supported Education with Alloy Help Active Learning? We have insisted all over this paper that simplicity and visualization tool offered by Alloy is a key point for active learning. The teacher never gives the response to an exercise but helps the student to find the way Alloy will give it. It is in our opinion a key point to let the students be active. But active learning -even with a well adapted tool as Alloy- had to be imposed. Students resist to engage in this kind of self exploratory learning and teachers have to struggle in order to impose it. Furthermore, our hypothesis that students will enjoy doing maths with computer turns out to be false. More precisely, Alloy labs students assessment follows a cycle. They enjoy in the beginning, they don't like when they realize that it requires to be active and it is better at the end when they get accustomed. We think that the reasons are: (1) to be active needs efforts, (2) doing maths with Alloy is forgetting calculus and focusing on concept meaning. It is precisely what our students rarely do before, and what is needed after. In conclusion, despite the difficulties inherent to active learning, our experience convince us to the relevance of our approach.

Does Our Computer-supported Approach Help Mathematic Education? One of our goals was to enforce a deep understanding of basic mathematical concepts. During Alloy labs, students manipulate these basic notions as explained in section 4. This kind of session is well achieved by a large part of students. Moreover, encountered difficulties relate most of the time to a real mathematic difficulty and very rarely to an Alloy specificity : quantifier alternations, implication .... In exams, Alloy exercises are generally well achieved. From this point of view, Alloy labs allow us to insist on difficult points and on the necessity to master basic concepts. The point that needs to be improved is the connection with the usual paper and pencil mathematic activity. Even if Alloy labs are inside the mathematic course, even if we elaborate exercises which show the connection and even if teachers point out connections during lessons, students tend to compartmentalize usual mathematics and Alloy activity. We think that the reason is that seeing connections needs to master the field and it has convinced us to work on many more decompartmentalizations.

Does Our Approach Help Computing Education? The question of the connection between mathematics and computing was asked to the students and surprisingly, a lot of them answered positively. It convinces us to the relevance of : (1) beginning with logic sets and discrete mathematics, (2) having a team composed with both mathematic and computing teachers, (3) using mathematic to model software systems. No doubt, placing mathematical modeling in mathematic course is efficient. It is very surprising to see students writing mathematical model docilely in math courses while they balk at writing UML model in usual software engineering courses Our approach doesn't completely remedy the student dislike to abstract design of software. Students prefer writing code to modelling but, due to our experience, they have better abstract skills for modelling and they know, because they practice models from the beginning of their studies that it cannot be ignored.

7 CONCLUSIONS

We have described in this paper a computer-supported experience from the Computer Science department of IUT de Montreuil-Paris 8, whose goal is to connect mathematics and software engineering through the use-in mathematics as well as in software engineering course- of the lightweight formal method Alloy. Alloy serve as a continuum from micro-modelling activities -used to enforce an abstract understanding of basic mathematic concepts- to real software modelling practice. We explained our original use of this tool for teaching mathematic concepts following the APO model and for introducing O.O. concepts.

In conclusion, we think that our approach benefits to Mathematics as well as to Computer Science: from the Mathematics point of view, the break with methods used before in math teaching gives a boost and develops problem solving skills. From the computer scientist point of view, it allows to expose our students to software modelling from the beginning of our curriculum , as it is recommended by many current approaches, in order to move our students from procedural thinking to abstract one.

REFERENCES

Alloy (2007). The alloy analyzer. http://alloy.mit.edu.

Bennedsen, J. and Caspersen, M. E. (2004). Programming in context: a model-first approach to cs1. In *SIGCSE'04*, pages 477–481.

Boyatt, R. and Sinclair, J. (2008). Experiences of teaching a lightweight formal method. In *Formal Methods in Computer Science Education 2008.*

Cohoon, J. and Knight, J. (2006). Connecting discrete mathematics and software engineering. In *36th ASEE/IEEE Frontiers in Education Conference*, pages 13–18.

Devlin, K. (2003). Introduction. *Commun. ACM*, 46:36–39.

Fenton, W. E. and Dubinsky, E. (1996). *Introduction to discrete mathematics with ISETL.* Springer.

Gregory, P. (2004). Motivating computing students to learn mathematics. *MSOR Connections*, 4(3).

Hadar, I. and Hadar, E. (2007). An iterative methodology for teaching object oriented concepts. *Informatics in Education*, 6(1):67–80.

Jackson, D. (2006). *Software Abstractions: Logic, Language and Analysis.* The MIT Press.

Lutz, M. (2006). Experiences with alloy in undergraduate formal method. In *Proceedings of 2006 American Society of Engineering Education Conference,, Chicago, IL. June 2006.*

Naveda, J. F., Lutz, M. J., Vallino, J. R., Reichlmayr, T., and Ludi, S. (2009). The road we've traveled: 12 years of undergraduate software engineering at the rochester institute of technology. In *ITNG*, pages 690–695.

Sekerinski, E. (2009). Teaching the unifying mathematics of software design. In *WCCCE '09: Proceedings of* *the 14th Western Canadian Conference on Computing Education*, pages 109–115, New York, NY, USA. ACM.

Siller, H. S. and Greefrath, G. (2009). Mathematical modelling in class regarding to technology. In *CERME 6*.

Sotiriadou, A. and Kefalas, P. (2000). Teaching formal methods in computer science undergraduates. In *International Conference on Applied and Theoretical Mathematics*.