Logic training through algorithmic problem solving

João F. Ferreira [1], Alexandra Mendes [1], Alcino Cunha [2], Carlos Baquero [2], Paulo Silva [2], L. S. Barbosa [2], and J. N. Oliveira [2]

[1] School of Computer Science, University of Nottingham, Nottingham, England
joao@joaoff.com, afm@cs.nott.ac.uk
[2] CCTC & Dep. Informatics, Minho University, Braga, Portugal
{mac,cbm,paufil,lsb,jno}@di.uminho.pt

Abstract. Although much of mathematics is algorithmic in nature, the skills needed to formulate and solve algorithmic problems do not form an integral part of mathematics education. In particular, logic, which is central to algorithm development, is rarely taught explicitly at pre-university level, under the justification that it is implicit in mathematics and therefore does not need to be taught as an independent topic. This paper argues in the opposite direction, describing a one-week workshop done at the University of Minho, in Portugal, whose goal was to introduce to high-school students calculational principles and techniques of algorithmic problem solving supported by calculational logic. The workshop resorted to recreational problems to convey the principles and to software tools, the *Alloy Analyzer* and *Netlogo*, to animate models.

Keywords: equational logic, calculational method, problem solving, algorithm derivation

1 Introduction and overview

It is consensual that Logic plays an essential role in rigorous software development. But the converse is also true, even if less well-known: literacy in logic, i.e., the ability to make productive use of the logic methods and tools, can be improved through algorithmic problem solving. Actually, two decades of research on *correct-by-construction* program design have created a new discipline of algorithmic problem solving and shed light on the underlying mathematical structures, modelling, and reasoning principles. In particular, it emphasises goal-directed, calculational construction of algorithms as opposed to more traditional guess-and-verify methodologies. Starting with the pioneering work of Dijkstra and Gries [8, 13], and in particular, through the development of the so-called *algebra of programming* [5, 4], a *calculational style* [3, 11, 7] emerged, emphasising the use of systematic mathematical calculation in the design of algorithms. The realisation that such a style is equally applicable to logical arguments [8, 13] and that it can greatly improve on traditional verbose proofs in natural language has led to a systematisation that can, in return, also improve exposition in

the more classical branches of mathematics. In particular, lengthy and verbose proofs (full of natural language explanations for "obvious" steps) are replaced by easy-to-follow calculations presented in a standard layout which replaces classical implication-first logic by variable-free algebraic reasoning [11, 12].

Such a systematisation of a calculational style of reasoning, proceeding in a essentially syntactic way, greatly improves on the way proofs are presented. In particular it may help to overcome the typical justification for omitting proofs in school mathematics: that they are difficult to follow for all but exceptional students.

However, and although much of mathematics is algorithmic in nature, the skills needed to formulate and solve algorithmic problems do not form an integral part of mathematics education. Also, the teaching of computer-related topics at pre-university level focuses on enabling students to be effective users of information technology, rather than equip them with the skills to develop new applications or to solve new problems.

In such a context, this paper reports on a concrete case-study on exploiting and combining the dynamics of algorithmic problem solving and calculational reasoning to introduce logic in high-school as a live and productive tool. A tool to boost the abilities students need to overcome the challenges they will encounter through life. This experiment shows that logic skills can be trained through simple problems that emphasise formalisation and calculation. For example, logic puzzles where the goal is to solve simultaneous equations on Booleans, can be introduced by analogy with simultaneous equations on numbers. High-school students already learn how to solve simultaneous equations on numbers; going from the reals to the simpler Boolean domain, where each variable is either true or false, seems a natural step to follow. Furthermore, illustrating how logic can be used to model and solve algorithmic problems, improves the students' abilities to solve problems in general. Related research, leading to similar conclusions, is reported in [6, 1, 2, 10].

2 An educational experiment

In July 2010, we organised a one-week workshop for Portuguese high-school students (aged between 14 and 17) on algorithmic problem solving. The goal was to show, through active involvement in tackling concrete problems, how the principles and techniques developed by computing scientists can be used to model and solve complex problems. There were 13 students enrolled in the workshop, all above-average students with a high interest on mathematics.

The workshop provided the opportunity to assess how pre-university students react to the calculational method and proof format. Two tools were used to increase interactivity and to show how machines can assist in problem-solving: *Alloy Analyzer* [15], to prototype models, and *NetLogo* [16], a multi-agent programmable modelling environment.

A summary of the plan of activities is shown in table 1. The week was divided into three main parts, detailed in the sequel. The first two days were used to

introduce computer modelling, supported by Alloy. The two following days were devoted to basic concepts in algorithmic problem solving: concision in naming the elements of a problem, symmetry, calculational logic, and invariants. These concepts were introduced using a *pen-and-paper* approach to reinforce the idea that computers are not needed to explore the topics that underlie computing. The final day was devoted to modelling complex systems in *NetLogo*.

Day	Activities
1	Introduction to Alloy. Modelling of a simple logic problem.
2	Modelling "The chameleons of Camelot" in Alloy.
3	Importance of concision and symmetry in algorithmic problem solving. River-crossing problems.
4	Introduction to calculational logic (through a logic puzzle). Introduction to invariants. Definitive solution to the problem "The chameleons of Camelot".
5	Introduction to *NetLogo*. Modelling of the problem "The chameleons of Camelot" in *NetLogo*.

Table 1. Plan of activities (the duration of each session was 3 hours).

Modelling problems in Alloy Alloy is a formal specification language based on relational logic (first-order logic enriched with relational product, composition, meet, converse and other relational operators). Alloy is getting increasingly popular in the software engineering community since it embodies a *lightweight* approach to formal methods: its minimalist syntax and straightforward semantics (centered in the unifying concept of relation) make it particularly easy to learn and well-suited for automatic verification. Bounded verification of Alloy assertions can be performed by the *Alloy Analyzer* tool: the model is translated to propositional logic and fed to an off-the-shelf SAT solver; when found, counter-examples are graphically displayed for better comprehension.

Alloy proved quite effective in this workshop. Set-theory is part of high-school curriculum, and the knowledge of the students was enough to understand the usage of relations as a specification formalism and even to develop small models after a one morning introductory course. As an example, in day 1, the students used Alloy to determine when they can be their own grandfathers (more precisely, when someone is her own step-grandfather). More important than the tackled problems, was the realisation that logic could be made "alive" with the help of computational tools: after modelling, *Alloy Analyzer* was used to explore properties of the problem in an interactive way.

Pen-and-paper approach The pen-and-paper approach was central to the workshop dynamics. Our starting point was the following logic puzzle:

> In an abridged version of Shakespeare's *Merchant of Venice*, Portia had two caskets: gold and silver. Inside one of these caskets, Portia had put

her portrait, and on each was an inscription. Portia explained to her suitor that each inscription could be either true or false but, on the basis of the inscriptions, he was to choose the casket containing the portrait. If he succeeded, he could marry her.

The inscriptions were:

Silver: *The portrait is not in this casket.*
Gold: *Exactly one of these inscriptions is true.*

Which casket contained the portrait? What can we deduce about the inscriptions?

One way of solving the problem is to introduce the variables pg for "the portrait is in the gold casket", ps for "the portrait is in the silver casket", ig for "the inscription in the gold casket is true", and is for "the inscription in the silver casket is true". Then, we are given:

$$(ig \equiv (ig \equiv \neg is)) \wedge (is \equiv \neg ps) \wedge (pg \equiv \neg ps) \quad .$$

The calculation that determines the values of the variables is a straightforward exercise in calculational logic:

$$\begin{array}{ll} & (ig \equiv (ig \equiv \neg is)) \wedge (is \equiv \neg ps) \wedge (pg \equiv \neg ps) \\ = & \quad \{ \quad \text{associativity} \quad \} \\ & ((ig \equiv ig) \equiv \neg is) \wedge (is \equiv \neg ps) \wedge (pg \equiv \neg ps) \\ = & \quad \{ \quad \text{reflexivity and negation} \quad \} \\ & (\mathsf{false} \equiv is) \wedge (is \equiv \neg ps) \wedge (pg \equiv \neg ps) \\ = & \quad \{ \quad \text{substitution of equals for equals} \quad \} \\ & (\mathsf{false} \equiv is) \wedge (\mathsf{false} \equiv \neg ps) \wedge (pg \equiv \neg ps) \\ = & \quad \{ \quad \text{negation and substitution of equals for equals} \quad \} \\ & (\mathsf{false} \equiv is) \wedge (\mathsf{true} \equiv ps) \wedge (pg \equiv \mathsf{false}) \quad . \end{array}$$

After being introduced to the rules used in the calculation above, the students came up with the same solution very easily.

Modelling complex systems in NetLogo The use of *Netlogo* allowed the introduction of a few examples of how very simple models can give rise to complex interactions. The covered examples included models for forest fires, community segregation, and soil erosion. Following these examples, and inspecting the code, the students could assess classical cases of complex systems and emergent behaviour.

In particular, the forest fire model shows how tree density plays an important role in the percentage of forest burned by a fire event. By playing with the model, students could perceive critical values in the density that when reached lead to a

phase transition where most of the forest is burned. The language is so accessible that, even at first contact, some students could add to the model the influence of wind to the fire propagation.

Unifying the three parts As we can see in table 1, the problem "The chameleons of Camelot" was discussed in each of the three parts. In fact, the problem was used to unify them. Using *Alloy Analyzer*, the students were able to find examples of arguments for which the problem can be solved. However, since Alloy models are verified in bounded domains, a definitive answer could not be obtained for all arguments, namely those for which there is no solution. We then modelled the problem using pen and paper and we were able to get a definitive answer. Finally, we modelled the problem in *NetLogo*. The graphical interface of the tool enriched the experience and allowed the students to interact with the problem. In the next section, we describe the problem and provide more details about how the three different approaches complement each other.

3 An algorithmic problem: the chameleons of Camelot

One of the problems extensively used in the workshop was a generalisation of "The chameleons of Camelot", as stated in [14, p. 140] (a more recent and accessible reference is [17]). The problem was used to help students to recognise, model, and solve algorithmic problems. In particular, it has a good potential to introduce non-determinism, problem decomposition, invariants, and program termination.

Problem statement On the island of Camelot there are three different types of chameleons: grey chameleons, brown chameleons, and crimson chameleons. Whenever two chameleons of different colours meet, they both change colour to the third colour. For which number of grey, brown, and crimson chameleons is it possible to arrange a succession of meetings that results in all the chameleons displaying the same colour?

For example, if the number of the three different types of chameleons is 4, 7, and 19 (irrespective of the colour), we can arrange a succession of meetings that results in a monochromatic state:

$$(4\,,7\,,19)\to(6\,,6\,,18)\to(5\,,5\,,20)\to\cdots\to(0\,,0\,,30) \quad .$$

On the other hand, if the number of chameleons is 1, 2, and 3, it is impossible to make them all display the same colour.

Modelling the problem in Alloy The first step towards the solution was to model the problem in Alloy. After modelling the rules governing the evolution of the chameleon colony, *Alloy Analyzer* was used to explore different combinations of colours and detect which could evolve to a monochromatic state (automatically detecting the succession of meetings leading to that state). This provided useful insight into finding the logical invariant constraining the state, and motivated the students for the following *pen-and-paper* approach.

Pen-and-paper solution Although *Alloy Analyzer* could be used to determine if given colonies of chameleons could reach a monochromatic state, it could not give a definitive answer when there was no solution. This limitation motivated a new approach. Using g, b, and c to denote, respectively, the number of grey, brown, and crimson chameleons, the first step was to model the algorithm that underlies the problem and to decompose it into two simpler parts:

$$
\begin{array}{l}
\mathsf{do} \quad g \neq b \wedge g \neq c \wedge b \neq c \;\rightarrow \\
\qquad \mathsf{if} \quad 0 < g \wedge 0 < b \;\rightarrow\; g\,,b\,,c \;:=\; g{-}1\,,b{-}1\,,c{+}2 \\
\qquad \Box \quad 0 < g \wedge 0 < c \;\rightarrow\; g\,,b\,,c \;:=\; g{-}1\,,b{+}2\,,c{-}1 \\
\qquad \Box \quad 0 < b \wedge 0 < c \;\rightarrow\; g\,,b\,,c \;:=\; g{+}2\,,b{-}1\,,c{-}1 \\
\qquad \mathsf{fi} \\
\mathsf{od} \\
\{ \quad g = b \;\vee\; g = c \;\vee\; b = c \quad \}\;;
\end{array}
$$

Two classes of chameleons are now equally numbered. Arrange a meeting between all the chameleons of these two classes.

$$\{ \quad (g = 0 \wedge b = 0) \vee (g = 0 \wedge c = 0) \vee (b = 0 \wedge c = 0) \quad \} \quad .$$

The algorithm consists of a loop (enclosed between the keywords do and od) that terminates when two classes of chameleons are equally numbered. Once we reach such a state, the problem is easy to solve.

The loop executes while at least one of the three guards (the conditions at the left of the arrow $\rightarrow$) is satisfied. If more than one guard is satisfied, the block operator ($\Box$) ensures that only one of the three assignments is chosen non-deterministically. The first assignment, for example, corresponds to a meeting between a grey chameleon and a brown chameleon: provided that there are chameleons of both these colours, the number of grey chameleons (g) and brown chameleons (b) both decrease by 1, whilst the number of crimson chameleons (c) increases by 2.

The next step of the solution was to find an invariant of the three assignments. Based on the postcondition of the loop, we calculated the invariant

$$g \cong b \pmod 3 \;\vee\; g \cong c \pmod 3 \;\vee\; b \cong c \pmod 3 \quad .$$

This pointed to the conclusion that if the initial numbers of chameleons do not satisfy the invariant, that is, if no two initial numbers are congruent modulo 3, it is impossible to organise a succession of meetings that results in all the chameleons displaying the same colour. At this point, the students understood why *Alloy Analyzer* could not find any succession of meetings when two initial numbers were not congruent modulo 3.

Finally, we discussed how to remove the non-determinism from the algorithm shown above so that we can guarantee termination.

The *pen-and-paper* approach to this problem is fully described in [9] (including the solution, notes on how to present the problem, and exercises).

Modelling the problem in NetLogo Modelling this problem in *NetLogo* allowed a closer simulation of how the encounters could occur in a spacial setting. The model places the chameleon population in a virtual torus and allows them to roam at constant speed. When a pair of chameleons reaches a given proximity threshold they interact and possibly change colour. As the simulation proceeds, a graph shows the population sizes for each colour along time. The interface allows either a randomised allocation of colours to a given total population size or an individual assignment of each initial colour population.

The experiment helped to determine how fast (when possible) one can observe convergence to a single colour state, in this idealised movement model. It also illustrated, empirically, that although some initial colour distributions can allow a set of encounters that leads to single colour convergence, this is statistically highly unlikely for all but the smallest sized populations.

4 Conclusions

At least from the point of view of the 13 students present, who were asked to fill an anonymous assessment survey at the end, the workshop was a success. The proposed questions sought open answers, where students could provide their opinion not being limited to a few standard cases. Although this does not allow for a statistical treatment of the results, it provides us with a more personal feedback and interesting insights about the outcome of the workshop.

Most students wrote that their expectations were exceeded. Some remarked the workshop has further stimulated their interest in mathematics, while other stated to have learned something about how software is developed and the activity of the professionals working in this field. Asked whether anything in that activity came as a surprise, most of them pointed out that the connection between programming computers and solving logical problems was by and large unexpected. They were also surprised by the accessible and interesting contents, while highlighting the overall quality of the sessions. All the students considered the pace of this workshop appropriate, although some recognised that it was faster than in high-school.

They were expecting to have more contact with programming languages and computers, but in the end, they seem to have understood that clear and structured reasoning is the key to solve problems and write good software, and that computers can be useless if the programs they put in motion are not carefully designed. They also remarked they have enjoyed approaching general problems and understood the role of abstraction and genericity in programming.

Certainly, no general conclusions can be extracted from a limited and single experiment. In general, however, the challenge placed to students was surprisingly well received and the feedback was quite positive. For example, the students *calculated* the solution of a logic puzzle very easily, which suggests that calculational logic can indeed be introduced at high-school level. Furthermore, most of them enjoyed the recreational flavour of the problems, and, at the end of the week, they were able to apply techniques like invariants by themselves. They

also liked the interactivity provided by the software tools that we have used, and, most important, they enjoyed being challenged.

Acknowledgements On-going collaboration with Roland Backhouse is deeply acknowledged. This research was supported by the MATHIS project under contract PTDC/EIA/73252/2006. The first two authors were further supported by FCT grants SFRH/BD/24269/2005 and SFRH/BD/29553/2006, respectively.

References

1. Back, R.J., Mannila, L., Peltomaki, M., Sibelius, P.: Structured derivations: A logic based approach to teaching mathematics. In: FORMED 2008: Formal Methods in Computer Science Education, Budapest (2008)
2. Back, R.J., von Wright, J.: Mathematics with a little bit of logic: Structured derivations in high-school mathematics (2006)
3. Backhouse, R.C.: Mathematics and programming. A revolution in the art of effective reasoning. Inaugural Lecture, University of Nottingham (2001)
4. Backhouse, R.C., Hoogendijk, P.F.: Elements of a relational theory of datatypes. In: Möller, B., Partsch, H., Schuman, S. (eds.) Formal Program Development. pp. 7–42. Springer Lect. Notes Comp. Sci. (755) (1993)
5. Bird, R., Moor, O.: The Algebra of Programming. Series in Computer Science, Prentice-Hall International (1997)
6. Boute, R.: Using Domain-Independent Problems for Introducing Formal Methods, LNCS, vol. 4085, chap. 22, pp. 316–331. Springer-Verlag (2006)
7. Dijkstra, E.W.: On the economy of doing mathematics. note EWD1130 (1992)
8. Dijkstra, E.W., Scholten, C.S.: Predicate Calculus and Program Semantics. Springer Verlag, NY (1990)
9. Ferreira, J.F.: Principles and Applications of Algorithmic Problem Solving. Ph.D. thesis, School of Computer Science, University of Nottingham (2010)
10. Ferreira, J.F., Mendes, A.: Student's feedback on teaching mathematics through the calculational method. In: 39th ASEE/IEEE Frontiers in Education Conference. IEEE (2009)
11. van Gasteren, A.J.M.: On the Shape of Mathematical Arguments. Springer Lect. Notes Comp. Sci. (445) (1990)
12. Gries, D., Feijen, W.H.J., van Gasteren, A.J.M., Misra, J.: Beauty is our Business. Springer Verlag (1990)
13. Gries, D., Schneider, F.: A Logical Approach to Discrete Mathematics. Springer Verlag, NY (1993)
14. Honsberger, R.: In Polya's Footsteps: Miscellaneous Problems and Essays (Dolciani Mathematical Expositions). The Mathematical Association of America (1997)
15. Jackson, D.: Software Abstractions: Logic, Language, and Analysis. The MIT Press (2006)
16. Tisue, S., Wilensky, U.: Netlogo: A simple environment for modeling complexity. In: in International Conference on Complex Systems. pp. 16–21 (2004)
17. Winkler, P.: Puzzled: Understanding relationships among numbers. Commun. ACM 52(5), 112 (2009)