Issues in the Design of an Environment to Support the Learning of Mathematical Generalisation*

Darren Pearce[1], Manolis Mavrikis[2], Eirini Geraniou[2], and Sergio Gutiérrez[1]

London Knowledge Lab[1,2]

[1] Birkbeck College {darrenp,sergut}@dcs.bbk.ac.uk
[2] Institute of Education {m.mavrikis,e.geraniou}@ioe.ac.uk

Abstract. Expressing generality, recognising and analysing patterns and articulating structure is a complex task and one that is invariably problematic for students. Nonetheless, very few systems exist that support learners explicitly in the process of mathematical generalisation. We have addressed this by developing a novel environment that supports users in their reasoning and problem-solving of generalisation tasks. We have followed a stakeholder-centred design process, integrating feedback and reflections from twenty-four children, five teachers and a variety of other stakeholders. This paper focuses on several inter-related design issues that have been informed by this iterative process and demonstrates how the system can be used for a typical generalisation task to foster an appreciation of generality and indeed algebra.

1 Introduction

In the traditional mathematical curriculum, algebra is a means of expressing generality. However, generalisation is so implicit in algebra that experts no longer notice the strategies they have integrated into their thinking [1]. This causes problems for students who perceive algebra as an *endpoint* rather than a tool for problem solving [2].

Several learning environments have been developed and integrated in classroom contexts over the last few years that attempt to help students in algebra and problem solving. However, the vast majority of these environments (e.g. [3–5]) are aimed at students who already have at least a basic understanding of algebra and attempt to develop students' understanding of various representations such as tables and graphs. These learning environments therefore do not deal explicitly with the generalisation difficulties that students face before they are comfortable with algebra. A different approach could focus on helping students derive generalisations from patterns. For example, in Mathsticks [6] students use a subset of LOGO commands to work on patterns and regularities constructed out of matchsticks. This allows them to explore how the variables within the task relate to each other. Despite some successes, difficulties remain, and these tend to coalesce around the need for significant pedagogic support from the teacher to provide a bridge to algebraic symbolism and generalisation.

* The authors would like to acknowledge the rest of the members of the MiGen team and financial support from the TLRP (e-Learning Phase-II, RES-139-25-0381).

This paper presents a mockup mathematical microworld — ShapeBuilder — that attempts to address these issues by supporting users in their reasoning and solving of generalisation problems. As the user constructs their model of the problem, they implicitly use the power of algebra and, as such, student experiences of the system serve to provide a smooth transition to the teaching of algebra and an intuitive justification as to why algebra is such a useful and powerful tool.

Throughout the development of ShapeBuilder, we have followed a stakeholder-centred design process,[3] interleaving software development phases with small-scale pilot studies with groups of children of our target age (11–14 years old). We have also integrated feedback from various other stakeholders such as teachers and teacher educators. This co-design with teachers is critical since former studies have shown that the use of educational tools in the education of mathematics must be carefully integrated within the classroom context [6]. In addition, studies about the adoption of educational software highlight that teachers would like the opportunity to be more involved in the entire design process of computer-based environments for their students [7, 8].

The remainder of the paper is structured as follows. Section 2 discusses the theoretical background of mathematical generalisation and the difficulties of developing this kind of thinking in young learners. Section 3 then briefly describes the key functionality of ShapeBuilder. This is followed by detailed discussions of various inter-related design issues in Section 4. Section 5 presents a concrete task we have used extensively for exploring generalisation, discussing its typical classroom deployment and how ShapeBuilder can be useful in its exploration. Section 6 draws together the various issues described in the paper and discusses future work.

2 Theoretical Background

The difficulty that algebraic thinking poses to children has been thoroughly studied in the field of mathematics education [9, 10]. One of the significant issues is that generalisation problems are frequently presented to students in confusing ways and this is compounded by strict constraints on the teaching approaches used [11]. These difficulties have to be investigated in the context of the curriculum, the nature of the tasks posed and the tools available for their solution [2]. The general tension in schools is towards pattern spotting. As mentioned by many authors [2, 12], most instructions emphasise the numeric aspect of patterning. These unfortunately lead to the variables becoming obscured and limit students' ability to conceptualise relationships between variables, justify the rules and use them in a meaningful way [11]. In addition, teachers tend to teach "the abstracted techniques isolated from all context" or alternatively "the technique as a set of rules to be followed in specific contexts" [13] to help their students find the rule. This could result in students' own powers atrophying due to lack of use [14].

Another difficulty secondary school students face is their inexperience with the use of letters. They struggle to grasp the idea of letters representing any value (e.g. [15]) and lack some of the mathematical vocabulary needed to express generality at this age. Even though it is a reasonable idea to introduce algebra early, there is still the issue

[3] This term is intended to encompass user-centred design and learner-centred design since our design process has integrated feedback from stakeholders other than users/learners.

of how to introduce it so that students can make the transition from simple arithmetic to algebra smoothly. Other researchers [16] describe that students' written responses lacked precision which supports the view of primary school students' inexperience with the mathematical language. Even if students succeed in expressing generality, they do so in natural language. The right design of tasks though has the potential to encourage students to write expressions in a general form rather than give a description in words. This articulation process needs to be addressed so that students can learn to express their thinking using algebraic notation. Deployed appropriately, ICT can help students understand different representations — the symbolic, the iconic and the numeric — and reinforce connections between them once they realise the relationships and the equivalence of different representations.

Students are required to learn techniques to pass exams and use examples as ways to learn different techniques, whereas working on different examples should help them realise "how the calculations are done, with an eye to seeing if they generalise" [14]. The idea of 'seeing the general through the particular' is a powerful way to introduce students to generalisation [14]. It is important, though, to introduce different approaches to students and allow them to explore. In this way, students are more likely to "strengthen their own powers, and at the same time, because of the pleasure experienced in exploiting their own powers, actually find mathematics enjoyable, creative and involving" [14]. This can be further enhanced by having students construct their own mathematical models [2]. This modelling approach seeks not only to foster seeing the general in the particular by construction and exploration, but also a sense of ownership of the abstraction process, which as claimed above is of great value to the students.

Learning is an active process and learners construct mental models and theories of the world as they understand it. This latter idea is characterised as constructivism. Papert ([17]), though, introduced another notion inspired by constructivism, which is characterised as constructionism. This supports "the idea that learners build knowledge structures particularly well in situations where they are engaged in constructing public entities" [12, p.61]. Such a constructionist pedagogical approach allows students to explore and construct their own models and can be usefully adopted when building systems. In contrast to much of the content of modern school mathematics textbooks, tasks within such systems can aim at generating understanding rather than inducing repetitive behaviour.

3 ShapeBuilder

ShapeBuilder[4] is an environment which aims to encourage structured algebraic reasoning of 11–14-year-old students. It allows the learner to create constants, variables and arbitrarily-complex compound expressions involving basic algebraic operations. These expressions can then be used to define various different shapes. Once a shape is defined, the user can move it, attach it to other shapes and alter its defining expressions as desired. The actual numerical values of expressions can be found by dragging them into the evaluator.

[4] Please see the project web-site (`http://www.migen.org/`) to download this and other project-related software.

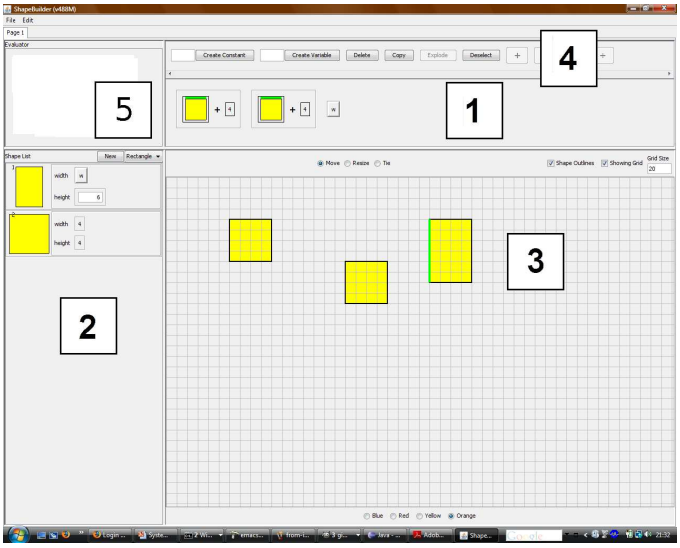


Fig. 1: Screenshot of ShapeBuilder, showing the main areas: Expression Palette (1), Shape List (2), main interaction area (3), Expression Toolbar (4), Evaluator (5).

A critical feature of the software allows users to *define expressions using shapes*. Specifically, by double-clicking on an edge of a shape, the user is able to obtain an *icon-variable* [18] which, at all times, evaluates to the current value of that dimension of the shape. These icon-variables are *bona fide* expressions and, as such, can be combined with other expressions in the usual way.

Figure 1 shows the layout of the ShapeBuilder. The following section explains the design issues behind its crucial functionalities, and their evolution through our stakeholder-centred iterative design process.

4 Design issues

Due to the nature of the tasks that the system is able to address, and the developmental level of the students, there is a pragmatic requirement to design the user interface in an optimal way: not only should it be intuitive for all students to use but it should also support their cognitive processes and reduce their memory load in relation to how the software works. As explained in Section 1, one of the aims of our project is to develop tools that provide assistance to learners and advice to teachers based on analyses of individual students and the activities of the group overall. This requires striking a balance between the need to design parts of the system with intelligent support in mind, and the need to take into account pedagogical and HCI considerations.

Several pilot studies have been conducted so far, involving 26 sessions with children in our age range (11–14 years old). They are helping to raise implications for the design of the system's intelligent support. However, due to the constructionist principles be-

hind our approach, the expressive power of the system and the freedom for students to interact with it in an exploratory manner are not compromised by the need for intelligent pedagogic support.

4.1 Direct Manipulation

In initial iterations of the software, creating a new rectangle required the user to select two expressions and click on the 'Make Rectangle' button (Figure 2a).[5] This creates a new shape on the canvas at a default location and adds its details as an entry in the Shape List (Figure 2b).

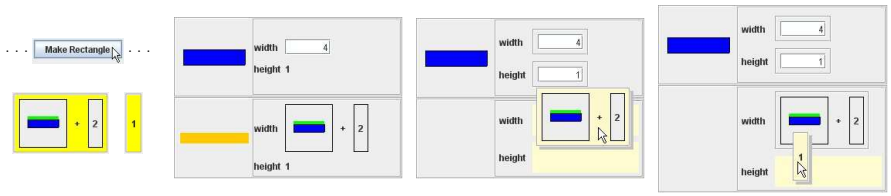


Fig. 2: Using direct manipulation. **(a)** The user selects two expressions and clicks on 'Make Rectangle'; **(b)** This creates the appropriate entry in the ShapeList. **(c)** The user drags an expression for the width; **(d)** and for the height.

As part of our reflections through contact with users and other stakeholders, it became clear that this interface was far from intuitive. Primarily, this was due to the fact that the user's action of clicking on the 'Make Rectangle' button was dislocated from the system responses of creating an entry in the Shape List and a shape in the interaction area. We addressed this issue through allowing the user to drag expressions to slots within the Shape List as shown in Figures 2c-d. Once all appropriate expressions are specified, the shape is created at a default location.

This refinement relates to the notion of 'distance' in direct manipulation interfaces [19] as it reduces 'the effort required to bridge the gulf between the user's goals and the way they must be specified to the system' [20]: if the user wants an expression to be used as the width for a shape, they simply drag it to the appropriate location; there is no need to understand that they must first select two expressions and then click on a specific button to create a shape.

Other aspects of the interface also relate to issues of directness. For example, recent iterations introduced the facility for cloning shapes through dragging their thumbnail from the Shape List to the interaction area. This is even required for creation of the initial shape rather than it being created at a default location. In future work, the creation of values and icon-variables will both adopt direct manipulation interface metaphors.

[5] The expression selected first was used as the width and the second as the height.

In general, such interface design considerations are important for a constructionist environment since it has the potential to increase the clarity of the construction and make the process more intuitive.

4.2 Variables, Constants and Values

Initially the software allowed the user to create variables whose value could change and constants whose values were fixed. These were displayed differently on the screen. However, during user trials it was often the case that constraining the user to choose between these two types *at creation time* led to unnecessary and confusing interaction. Consider for example the situation where a user decides they want the side of an already-created shape to be a variable rather than a constant. With this constraint in place, they would have to create a new variable and put it in place of the original constant. This situation becomes more complex if the constant that they want to change is buried deep within another expression. Not only is this sequence of steps an issue in terms of usability but, as part of a system that will feature intelligent support, detecting that such an episode of interaction solely achieves the transformation of a constant into a variable introduces an unwanted element of complexity.

In view of this, we experimented with another user interface which allowed the user to convert variables to constants and vice versa at any time. This was achieved through the use of a 'Toggle Lock' tool. Using this new tool on a variable would 'convert' it to a constant and, similarly, using this tool on a constant would 'convert' it to a variable.

Through various discussions with teachers and teacher educators, it later became clear that the use of the terms 'variable' and 'constant' were problematic in themselves since, in terms of the National Curriculum, a variable is an entity with a name such as x which varies in some way. Since the representation of variables within ShapeBuilder was purely in terms of its current value and not in terms of a name, this could be a potential source of confusion for users of the system. As a result of these considerations, the most recent iteration of the software development disengages entirely from this false dichotomy of constant and variable and provides a tool that creates a new *value*. The user can switch between locked values ('constants') and unlocked values ('variables') using the Toggle Lock tool as before. An important aspect of this final configuration is that values are always created *locked*. In this way, it is clear to the user and the system (for the purposes of intelligent support) whether they want the values to be able to change or not.

4.3 Building with 'n'

When expressing generality, one of the main difficulties is the use of variables for expressing universal concepts (e.g. n representing the number of people in *any* room). This basic foundation of algebra is a difficult skill for children to acquire. Previous research has shown that children start viewing variables as static objects with no general meaning and then pass through a series of steps. These steps include viewing variables as concrete numbers then generalised numbers and, finally, as general entities [9].

When designing our system, one of our main concerns was to give the learner the facility to 'build with n'. We were conscious that using letters to denote variables

was likely to prove difficult for learners so we addressed this problem by using 'icon-variables' [18]. Icon-variables are iconic representations of an attribute of an object (such as the height of a rectangle). They are defined from objects and can be used to construct expressions that, in turn, define other objects.

Icon-variables are a pictorial representation of a concept and provide a way to identify a general concept that is easier for young learners to comprehend. They can be used in exactly the same way as other expressions: copied, deleted, used in operations (e.g. addition), used to define other objects, etc. In this way, they are an intermediate step that scaffold the use of variables.

This design feature holds significant potential to lead to generalised thinking. For example, a constant and an icon-variable can be added to express relationships such as "the width of shape B is the width of shape A plus 2", or "shape A is twice as high as shape B". These expressions can be built on screen and used as a resource for reasoning by the user.

4.4 Evaluating Expressions

As explained in Section 2, a typical problem with the deployment of generalisation tasks is the premature engagement with specific numbers. Avoiding this problem was therefore an important requirement of the software.

Initially, this was achieved through providing tooltips on expressions. When the user hovered with the mouse cursor over an expression, a tooltip would appear showing the result of evaluating the expression. This is a standard interface metaphor wherein waiting over various screen areas for a moment leads to more information. It was precisely this status of 'additional information' that satisfied the design requirement of number as secondary.

Although during the student trials, users had minimal difficulty in using this software feature, we still concluded that its use was nonetheless problematic. This was due to the fact that the use of the tooltip was *ambiguous*; when a user displays a tooltip, it is not clear whether it is deliberate or whether they are attending to it appropriately. For the user, this means that the tooltip is potentially distracting if it is displayed when unwanted. For the system — in which intelligent support is an important design consideration — unambiguous interaction is essential as recognised by a large body of work since the early 1990s [21]. The use of a tooltip for evaluating expressions therefore can increase the noise in the system.

In view of these considerations, we included a specific screen area within the interface — the 'Evaluator'. It is *only* through this component that a user can evaluate expressions explicitly.[6] To demonstrate the behaviour of this component, consider the case where the user has created a rectangle (of, say, width 4 and height 3) and wants to evaluate expressions for both its area and its perimeter. They place these in the Evaluator and it displays their initial values as shown in Figure 3a. If the width of the rectangle is now changed to 5, the value of both these expressions then changes as a result since

[6] Currently, all expressions can be evaluated by the user manually in that they could carry out the operations on the expressions themselves. The incentive to use the Evaluator therefore is to save calculation time and ensure calculation accuracy.

they both depend on the width of the rectangle. Given this dependency, the Evaluator obscures the evaluation component with question marks (Figure 3b). In order to see the new values of these expressions, the user must now click on these question marks (in any order they choose). In this way, an expression within the evaluator requires *explicit* action from the user to view values as they change, which not only directs their attention and therefore reduces their cognitive load but also provides information to the system increasing the amount and quality of the input (also referred to as 'bandwidth' [22]) for intelligent components of the system.

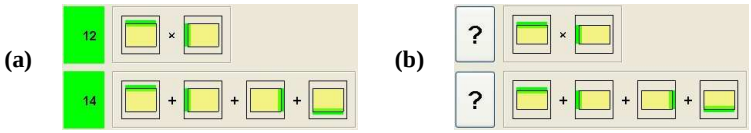


Fig. 3: The behaviour of the Evaluator. **(a)** Two expressions first evaluated; **(b)** Both expressions have changed.

5 Pond Tiling: A Concrete Task for Exploring Generalisation

As a first exploration of using ShapeBuilder, we focused on one particular generalisation problem: pond tiling, which is typical in the British algebra curriculum for children in our target age group. The simplest version of this task can be described as follows: "given a rectangular pond with an actual integer length and breadth, how many 1×1 tiles are needed to surround it?". We have focused on the pond-tiling activity both because of its appropriateness for presentation on a computer and the fact that it naturally lends itself to a variety of different representations. The task encourages students to find expressions for the number of tiles needed based on the length and the breadth of the pond. Four different ways students could visualise the pond and its surrounding tiles are shown in Figure 4. Given these different arrangements, students can be asked to have interesting discussions about the equivalence of the expressions derived from each viewpoint. There is then an incentive to develop some of the basic rules of algebra intuitively such as commutativity and associativity.

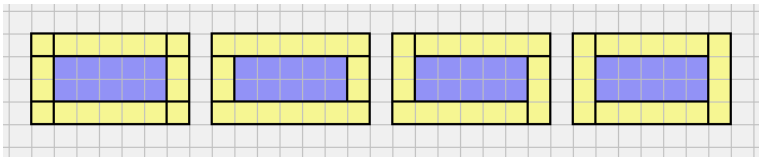


Fig. 4: Tilings corresponding to the general algebraic equations: $2l+2b+4$, $2l+2(b+2)$, $2(l+1)+2(b+1)$, and $2(l+2)+2b$ (l is the length of the pond; b the breadth).

The following sections illustrate how this problem is usually approached in the classroom and how the introduction of ShapeBuilder affords a different approach, in line with the constructionist principles discussed in Section 2.

5.1 Pond Tiling in the Classroom

In order to tackle pond tiling and similar generalisation problems in the classroom context, students are often advised to try different values for the length (l) of the pond and arrange them in a table such as the one shown in Table 1. The numbers in the table are then used to determine the relationship between l and the number of tiles. This of course enables students to generate a relationship using pattern spotting based on the numbers in the table. However, this is far from ideal since this relationship is independent of the structure of the initial problem.

Length	Number of tiles	Difference
1	8	
2	10	+2
3	12	+2
...	...	+2
50	?	+2

Table 1. A table of differences, typically used to find general rules.

Another disadvantage of this approach is that it is limited to problems involving one variable. In cases where more than one variable is involved, the approach becomes cumbersome. A further — and more crucial — issue is that this representation is usually given to students without any justification. As a result, students learn to become perfect executors of 'tricks' without being prompted to think of *why*, but rather only *how* similar problems were solved by the teacher. Lastly, since there is no sense of ownership in the abstraction process, it becomes less meaningful.

5.2 Pond Tiling in ShapeBuilder

As presented in Section 4, students can use ShapeBuilder to construct their own models, play with these creations and put them to the test. Users can either build specific constructions using explicit values or use icon-variables (as discussed in Section 4.3) so that their constructions are general. Figure 5 illustrates the construction of a general tiling using icon-variables.

During the pilot studies, one typical yet erroneous approach that students take is to construct their solution for a *specific* pond. After the solution has been constructed, the given values of the problem (e.g. width and height of the pond) can be changed 'surprising' the student who has to figure out what was wrong with the construction. In the dynamic geometry literature, this operation is referred to as 'messing up' [23]. An example of this is shown in Figure 6a where the width of the rectangle is defined by the

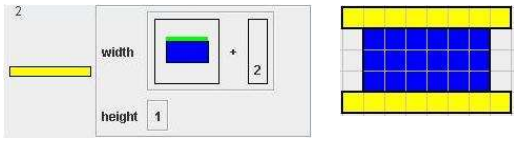
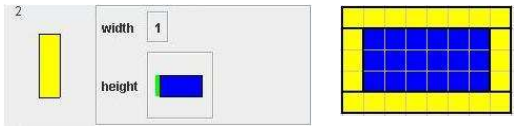
Step 1 The user creates two independent variables, clicks on 'New Shape' and then drags these expressions to define the width and height of their pond (rectangle).	
Step 2 The user creates an iconic expression for the pond width and add 2 to it. They then use this to define the width of the horizontal tile. They copy it (by dragging) to create two instances.	
Step 3 They then create an iconic expression for the height of the pond and use this to define the height of the vertical tile. Once again, they copy this through dragging.	

Fig. 5: Using ShapeBuilder for the pond tiling task. Each step shows the latest entry in the shape list and the current state of the canvas.

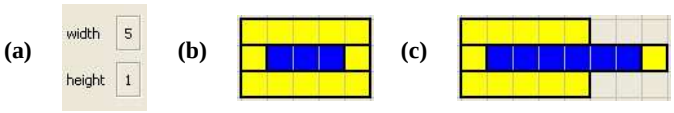


Fig. 6: 'Messing up'. **(a)** The definition of the long rectangle; **(b)** the tiling for pond width 3; **(c)** and 6.

constant 5. This looks correct for a pond of width 3 (Figure 6b). However, if the width of the pond is changed by the teacher (or indeed another student or, in the future, the system itself), the construction is no longer valid (Figure 6c).

The challenge for students, therefore, is to construct a solution that is impervious to 'messing up' in this way. This 'incentive to generalise' [1] provides students with the opportunity to realise that there is an advantage in using icon-variables and promotes thinking in terms of abstract characteristics of the task rather than specific numbers, thus leading to a type of mathematical generalisation.

The next step for students is to name their expressions. They typically produce phrases such as 'width of the swimming pool' or 'number of tiles'. This initiates an articulation process. Students could be further prompted to use words instead of phrases and, later on, letters instead of words. As discussed in the background, this process should help them make a smooth transition to the use of letters in the traditional algebraic symbolism. Depending on the task, there are opportunities for students to see the point in the use of letters since, for instance, naming their objects or expressions will enable them to refer to them in a laconic way when they collaborate with fellow students. Furthermore, in line with the constructionist approach, the process of constructing ob-

jects or expressions and then naming them as they wish engages students and deviates in a positive way from the traditional teaching method of algebra.

6 Conclusions and Future Work

The need to recognise, express and justify generality is at the core of mathematical thinking and scientific enquiry. However, it has been consistently shown to be complex and problematic for students. Several systems have been developed to help students in algebra but assisting the development of mathematical generalisation still presents important challenges. Although it may seem that it is the students' responsibility to actively construct mathematics for themselves, it is unreasonable to expect they will do it on their own. This is particularly true in the case of mathematics because it is not observable: unlike the physical world, mathematics only exists within people's minds. This paper presents a system that could act as a mediator between the learner and mathematics and therefore assist students visually in their mathematical knowledge and development of generalisation.

The tool follows a constructionist approach, allowing students to create shapes and expressions and see the relationships between them. The paper has presented the design of the system, describing and discussing several design issues: the need for direct manipulation; the importance of how values are presented in the system; the need for building with 'n' using icon-variables; and the issue of evaluating expressions.

Our aim is to foster the idea of seeing the general through the particular by construction and exploration, but also to promote a sense of ownership of the abstraction process. However, we recognise that this also depends on the activities undertaken and the educational settings in general. The paper presented how the tool has been applied to a well-known generalisation problem and discussed the added value compared to the traditional classroom-based approach. Future work will formally evaluate its effectiveness. In addition, the iterative design and development of the system and the suggested activities will continue to co-evolve. It is important to find the right balance between the degree of structure and degree of flexibility built into the learning process. While we recognise the importance of giving clear guidance and support to students to achieve the learning objectives, we also need to give them freedom to explore, experiment, enjoy, interact and arrive at their own generalisations.

Next steps in our research include exploring collaboration among learners within a classroom context and integrating intelligent support within the system to assist students and teachers in the teaching and learning of mathematical generalisation.

References

1. Mason, J., Graham, A., Johnston-Wilder, S.: Developing Thinking in Algebra. Paul Chapman Publishing (2005)
2. Noss, R., Healy, L., Hoyles, C.: The construction of mathematical meanings: Connecting the visual with the symbolic. Educational Studies in Mathematics **33**(2) (1997) 203–233
3. Tall, D., Thomas, M.: Encouraging versatile thinking in algebra using the computer. Educational Studies in Mathematics **22**(2) (April 1991) 125–147

4. Roschelle, J., Kaput, J.: SimCalc MathWorlds: Composable components for calculus learning. Communications of the ACM **39** (1996) 97–99
5. Kieran, C., Yerushalmy, M.: Research on the role of technological environments in algebra learning and teaching. In Stacey, K., H. Shick, H., Kendal, M., eds.: The Future of the Teaching and Learning of Algebra. The 12th ICMI Study. Volume 8 of New ICMI Study Series. Kluwer Academic Publishers (2004) 99–152
6. Hoyles, C., Healy, L.: Visual and symbolic reasoning in mathematics: Making connections with computers. Mathematical Thinking and Learning **1**(1) (1999) 59–84
7. Underwood, J., Cavendish, S., Dowling, S., Fogelman, K., Lawson, T.: Are integrated learning systems effective learning support tools? Computers and Education **26** (1996) 33–40
8. Pelgrum, W.: Obstacles to the integration of ICT in education: results from a world-wide educational assessment. Computers and Education **37** (2001) 163–178
9. Küchemann, D., Hoyles, C.: Investigating factors that influence students' mathematical reasoning. In: PME XXV. Volume 3. (2001) 257–264
10. Healy, L., Hoyles, C.: A study of proof conceptions in algebra. Journal for Research in Mathematics Education **31**(4) (2000) 396–428
11. Moss, J., Beatty, R.: Knowledge building in mathematics: Supporting collaborative learning in pattern problems. International Journal of Computer-Supported Collaborative Learning **1** (2006) 441–465
12. Noss, R., Hoyles, C.: Windows on mathematical meanings: Learning cultures and computers. Dordrecht: Kluwer (1996)
13. Sutherland, R., Mason, J.: Key aspects of teaching algebra in schools. QCA, London (2005)
14. Mason, J.: Generalisation and algebra: Exploiting children's powers. In Haggarty, L., ed.: Aspects of Teaching Secondary Mathematics: Perspectives on Practice. Routledge Falmer and the Open University (2002) 105–120
15. Duke, R., Graham, A.: Inside the letter. Mathematics Teaching Incorporating Micromath **200** (2007) 42–45
16. Warren, E., Cooper, T.: Generalising the pattern rule for visual growth patterns: Actions that support 8 year olds' thinking. Educational Studies in Mathematics **67** (2008) 171–185
17. Papert, S.: Mindstorms: Children, Computers, and Powerful Ideas. Perseus Books, U.S. (1993)
18. Gutiérrez, S., Mavrikis, M., Pearce, D.: A learning environment for promoting structured algebraic thinking in children. In: Int. Conf. in Advanced Learning Technologies (ICALT '08), IEEE (2008)
19. Schneiderman, I.E.: Direct manipulation: A step beyond programming languages. IEEE Computer **16** (1983) 57–69
20. Hutchins, E.L., Hollan, J.D., Norman, D.A.: Direct manipulation interfaces. Human-Computer Interaction **1** (1985) 311–338
21. Orey, M., Nelson, W.: Development principles for intelligent tutoring systems: Integrating cognitive theory into the development of computer-based instruction. Educational Technology Research and Development **41**(1) (March 1993) 59–72
22. VanLehn, K.: Student modeling. In Polson, M., Richardson, J., eds.: Foundations of Intelligent Tutoring Systems. Hillsdale, NJ: Erlbaum (1988) 55–78
23. Healy, L., Heltz, R., Hoyles, C., Noss, R.: Messing up. Micromath **10** (1994) 14–17