Designing to see and share structure in number sequences

Journal Item

Designing To See And Share Structure In Number Sequences

By Yishay Mor, Richard Noss, Celia Hoyles, Ken Kahn and Gordon Simpson
London Knowledge Lab, Institute of Education, University of London, UK
yishaym@gmail.com

the International Journal for Technology in Mathematics Education, 13 (2) : 65-78, 2006.

Received: 7th October 2006 Revised: 8 March 2006

This paper reports on a design experiment in the domain of number sequences conducted in the course of the WebLabs *project. We iteratively designed and tested a set of activities and tools in which 10-14 year old students used the* ToonTalk *programming environment to construct models of sequences and series, and then shared their models and their observations about them utilising a web-based collaboration system. We report on the evolution of a design pattern (programming method) called 'Streams' which enables students to engage in the process of summing and 'hold the series in their hand', and consequently make sophisticated arguments regarding the mathematical structures of the sequences without requiring the use of algebra. While the focus of this paper is mainly on the design of activities, and in particular their epistemological foundations, some illustrative examples of one group of students' work indicate the potential of the activities and tools for expressing and reflecting on deep mathematical ideas.*

1 INTRODUCTION

This paper reports on a set of activities designed for students to construct number sequences and sum them, and consequently to be encouraged to reason and argue about the structure of number sequences, both face to face and at a distance. It is an initial component of a more general effort in which they engaged in a range of mathematical activities exploring issues such as cardinality and convergence.

Pattern recognition and generalisation are fundamental to mathematical thinking and a fruitful pathway into algebraic thinking. In the words of John Dossey (1998) "*From whence does algebra grow? It grows from the study of growth itself. One of the first places students see growth is when they look at patterns and patterns of numbers*" (p 20). Kieran (1997) reviews several examples of how activities originating in observation of patterns in numeric or graphical sequences can create opportunities for introducing algebraic thinking. Sasman *et al* (1999) note that such an approach is implicit in the design of many national curricula, yet, as noted by Zazkis & Liljedahl (2002), most of the research focuses on either fundamental counting sequences or on advanced mathematical concepts. At one end of the spectrum we find studies such as Steffe (1988; 1994) and Olive (2001) which illuminate the construction of the basic number sequence at an early age. At the other we find Davis & Vinner (1986), Tall & Schwarzenberger (1978), Cornu (1991) and, more recently, Oehrtman (2003), Kidron *et al* (2001), Sriskanda (2003), and Floris (2004), all discussing learning of limits of sequences and functions, typically in the context of advanced high-school and college students. In between these extremes, the literature is dominated by the potentials and issues emerging from observing number patterns.

Mason (1996) notes that school algebra is traditionally centred on numbers, and on functions of numbers. Observing and reasoning about patterns in number sequences is an opportunity for learners to experience the process of mathematical generalisation. Yet at the same time, a number of researchers, including Radford (2000) and Noss *et al* (1997), point to the difficulties students encounter in shifting from pattern spotting to structural understanding. Students often tend to base their conclusions on superficial or incidental patterns they observe in the sequence, rather than on arguments referring to its structure. Although the use of structural reasoning increases modestly with age, Küchemann & Hoyles (2005) note that empirical reasoning remains widespread. The study reported here attempts to address the gap between fundamental and advanced concepts, by designing learning experiences which allow students to construct bridges from their primary intuitions to mathematical concepts of number sequences and series.

In the case of number sequences, some of the aforementioned researchers have suggested that one of the obstacles to developing an appreciation of structure is students' tendency towards a recursive view, that is, identifying the relationship between consecutive terms rather than its general rule of the sequence (e.g. describing the sequence described by the function $f: x \rightarrow 2x + 1$ as "add 2").

Several attempts have been made to explain these difficulties. Cottrill *et al* (1996) use APOS theory, while others propose co-variation, correspondence, or a property-oriented view (Confrey & Smith, 1994; Salvit, 1997). Regardless of the interpretative framework, two observations are universal: first, that number-pattern spotting is a predominant solution strategy, and second that the recursive form is a predominant description strategy. Indeed, pattern spotting lacks the definitiveness of a formal argument, and the recursive form does not generalise easily to functions of the real numbers ($f{:}R \rightarrow R$). Yet the association between *recursive* and *lack of structure* suggests that in some cases, researchers might be confusing structure with representation. Consider the sequence: *1, 4, 7, 10…*

It can be represented in closed form, as: $a_n = 1 + 3 * n$

Or recursively as: $a_0 = 1\,; a_n = a_{n-1} + 3$

Both are functions. One is a function of the natural numbers, the other a function of the previous term. Yet whereas the former conflicts with base intuitions, the latter stems from them. The disassociation between the perceived structure of the sequence and its taught representation means that the student needs to tackle two seemingly unrelated challenges: the mathematical object and the algebraic representation. The result is, as noted in Noss *et al* (1997, p 205):

> … algebraic formulation is often disconnected from the activity which precedes it, a meaningless extra that neither illuminates the problem nor provides a means for validating its solution. Algebra is viewed as an endpoint, a problem solution in itself rather than a tool for problem solving.

From a design point of view, the challenge is to construct learning environments that are contiguous with existing knowledge, rather than seeking to replace it. Weigand (1991) has posited that *iteration sequences* offer rich mathematical experiences that should be exploited in activity design. We too start from the supposition that we need to construct activities and tools that allow students to start from intuitive forms, formalise them, and develop alternative ways to explore problem situations.

2 AN ALTERNATIVE REPRESENTATION FOR SEQUENCES

This study adopts, in common with our approach over many years, the harnessing of a programming language as a medium of mathematical expression that builds on students' intuitive ideas of a particular mathematical domain (see Noss & Hoyles, 1996, for a historical overview of this approach, and its rationale). The idea of developing alternative representational forms has a concrete consequence in the domain of number sequences. Traditionally, the predominant representation of a sequence in computer programming was as a *list*: an ordered set of items. This abstract definition needs to be implemented with respect to the particulars of the chosen language. Common educational implementations attempt to capture the essence of the formal definition of a sequence, as a function $f{:}N \rightarrow R$. These representations are static – at any given point in time, their content is fixed. Additionally while lists are not limited in length and can be extended on the fly, any actual list at any given time is, of course, finite. This could be a source of epistemic conflict. While we talk about infinite lists, the objects we manipulate are inherently finite, and the algorithms used are geared towards finiteness. Furthermore, by emphasising the $f{:}N \rightarrow R$ formalisation of sequences it risks a conflict with students' recursive intuition $\left(f : a_n \rightarrow a_{n+1}\right)$.

Sacristán (1997) proposes an alternative approach that uses recursive programs as a representation of infinite sequences. She focused on establishing intuitions by visualising the sequences, an approach which proved successful. However, she stresses the need to supplement this approach with alternative representations in terms of numeric values. This was achieved by allowing students directly to manipulate the code that instantiated the visual representation. Seeing the visualisation and the sequence unfold together gradually allowed the students to consider the sequence as a process and object and helped them to identify local structure. Such a dual view, argue Sfard (1991), Tall and Gray (1993), and others, is fundamental to mathematical thinking. Sacristán's design called for task specific programs, which balanced functional richness with code simplicity, so that students could observe the visualisation and then tweak the code that produces it. Unfortunately, simplicity is often achieved at the price of generality. A function created to display one sequence cannot be used to plot another. Arguably, this can be overcome by a minor code change. Yet we may want students to work with code components as building blocks without the need to recode them.

We propose an alternative approach that generates the terms dynamically, as these are needed. This is precisely the idea behind the *Stream* structure. A stream is a dynamic representation of a sequence. In object oriented languages (such as JAVA or C++) it is implemented by an object with a read ( ) method (function) which retrieves the next term every time it is invoked. The idea of 'streams' is not new. Abelson & Sussman (1996, section 3.5) noted, for example:

> If time is measured in discrete steps, then we can model a time function as a (possibly infinite) sequence. … we will … model change in terms of sequences that represent the time histories of the systems being modeled. To accomplish this, we introduce new data structures called streams. From an abstract point of view, a stream is simply a sequence. However, we will find that the straightforward implementation of streams as lists … doesn't fully reveal the power of stream processing.

Shapiro (1988) explains why 'streams' are most useful in concurrent systems, those in which many processes are executed in parallel. Streams provide a structured mechanism of dividing work between processes using an assembly line metaphor: every process sends out a stream of outputs, which are passed as a stream of inputs to the next. While process II is busy, say, with the 5th term, process I is already generating the 6th, and process III can work on the 4th. Streams are used as a fundamental mechanism in UNIX for communicating between applications and operating system processes (SUN, 2005) and the primary input – output framework in Java (Eckel, 2002). In many scenarios, streams have computational advantages over lists. A detailed comparison is, however, beyond the scope of this paper.

In educational contexts, the potential of streams would most likely be realised in concurrent languages, and one such – *ToonTalk* (Kahn, 1996) – was utilised in the present study. *ToonTalk* is a language and a programming environment designed to be accessible by children from a wide range of ages, without compromising computational and expressive power. It does so by embedding complex programming constructs in a video-game setting as shown in Figure 1. In *ToonTalk*, every programming structure is concretised as an animated cartoon object: robots (labelled 2 in Figure 1) stand for programs, boxes (labelled 3) for data structures, birds (5) for message sending, nests (6) for message receiving, scales for comparisons, trucks for process spawning, and bombs for process termination. The toolbox (11) contains the data types and operators, while the notebook (12) provides a standard library of stored procedures.

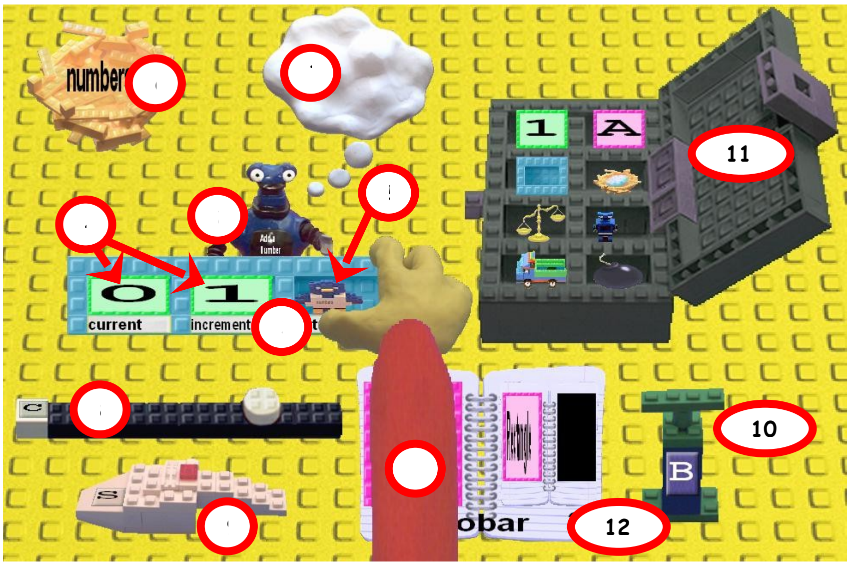


Figure 1 The ToonTalk Environment

The user directly manipulates objects using a virtual 'hand' (labelled 1 in Figure 1), or with tools such as the magic wand for copying (labelled 8), vacuum cleaner (9) for cutting, pasting and erasing or bicycle pump (10) for changing object size. Programs are created by training a robot – directly leading it through the steps of a task it is required to perform. The robot remembers what it is trained to do, but only for the specific set of values with which it was 'trained'. These are stored in the robot's thought bubble (7). The robot's memory can then be generalised by 'vacuuming'; that is, by erasing the values and leaving an empty slot for 'any value'. Thus the concept of variables is introduced implicitly through the programming metaphors. Needless to say, this mode of programming is very different from that used in traditional text-based languages, and induces different patterns and styles of problem solving.

The implementation of 'streams' in *ToonTalk* is straightforward (Mor *et al*, 2004). A robot generating a 'stream' uses a box with the internal variables it needs and a bird for carrying the terms out. A robot processing that stream uses a box in which it holds the nest of the bird from the first robot. Robots can be chained in this manner to construct complex "assembly lines" from simple self-contained processes. Furthermore, *ToonTalk* is a concurrent language, which means that several programs (robots) can run concurrently. This allows us to generate a sequence and add up its terms at the same time, while keeping the two processes clearly distinguishable.

As we noted above, children's intuition of sequences tends to be predominantly recursive. The stream design pattern allows us to capture this intuition and formulate it as code. A typical implementation would use an internal variable to store the value of the previous term, and use that to generate the next. For example, consider the sequence $a_n = 2^n$. In order to generate it, a robot needs a box with two holes: one for the current value, and one for the output bird. The robot iteratively multiplies the current value by two and passes a copy of the result to the bird, as illustrated in Figure 2.

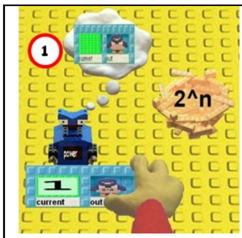
 Give the robot its input box	 The robot multiplies 'current' by 2	 And passes a copy to the bird
 The bird will carry it to its nest	 Sequence terms are stacked on the nest	 While the robot repeats its actions indefinitely.

Figure 2: Generating the powers of 2 sequence

We have mentioned the importance of maintaining a cohesive process-product view of sequences. In traditional list programming, the process (generating the sequence) is decoupled from the product (an enumeration of the first *n* terms). Similarly, the relationship between successive terms of the sum series is lost, as each one is computed independently from the original list. A stream is an object which ties together the process and the product. This is particularly true in a language such as *ToonTalk*, which allows the user to observe the execution of code, by watching the animated components of the program play out their programmed behaviour. Decomposing a complex structure into a chain of simple ones encourages students to toggle between the process view (in this case, "summing up") and the product / object view (in this example, the sequence of sums).

A final rationale for the stream approach concerns the nature of infinity and its (unsurprising) difficulty as a concept for learners. Several researchers (Tirosh, 1991; Li & Tall, 1993; Falk and Lavy, 1989; Dubinsky *et al*, 2005) have commented on the tension between potential and actual infinity. Any manifestation of infinity in a computational medium is inevitably potential, since the computer's memory and processing power is finite. The stream pattern is as close as possible to this intuitive concept of infinity; it will continue providing terms indefinitely until it is interrupted. It can also possibly provide a bridge towards the conception of actual infinity: since it is not possible to count the length of a stream (as is possible with lists), the stream object itself represents all terms of the sequence – *ad infinitum*. Again, the power of streams is not in the representation of any specific infinite process – but in the possibility of combining and manipulating infinite processes.

2.1 The Experimental Context

The work reported here formed part of the *WebLabs* Project (www.weblabs.eu.com, European Union, Grant # IST-2001-32200), which aimed to explore new ways of constructing and expressing mathematical and scientific knowledge in communities of young learners. Our approach brought together two traditions: *constructionist learning* as described by Harel & Papert (1991) and collaborative *knowledge-building* in the spirit of Scardamalia & Bereiter (1994). We use *ToonTalk* as our primary platform for construction, building open 'toolsets' for students to construct models, and supplementing these with other appropriate tools as necessary (for example, *Excel*). As a parallel and intimately related development, we have designed and built a web-based collaboration system called *WebReports* for sharing and discussing these constructions. This system allows students to seamlessly embed their models in free form text documents they publish them on the web. Thus the central tenets of the approach are that students simultaneously *build* and *share* models of their emerging mathematical knowledge. The details of the system have been described elsewhere, by, for example, Mor *et al* (2005), and Simpson *et al* (2006).

Alongside the technical development, a main focus of *WebLabs* was on designing and testing a set of activity sequences to support learning. Our activities followed a common pattern. We identify our learning aims and then began each activity sequence by discussing an intriguing mathematical theme. We encourage students to propose conjectures or derive concrete questions to explore, which are then formulated by us into modelling/programming tasks. Students complete these tasks individually or in pairs and publish their individual models (*ToonTalk* programs) along with their observations about them, in personal *webreports* (we use *WebReports* to refer to the system and *webreports* to refer to actual documents within it). Students are encouraged to comment on each other's models, which are then used as input to an instructor-led group discussion. The product of this discussion is a *group webreport* which represents the shared understandings of the group, a process that encourages students to reflect on their work, to acknowledge the need to construct rigorous arguments for their claims, and to negotiate socio-mathematical and socio-technical norms within the (international) community, using the terminology of Yackel & Cobb (1995).

Ideally, at this point in the students' work the *webreport* would be reviewed by another group, perhaps in another country, and an inter-group discussion would ensue, using the *WebReports* 'comment' mechanism (see Figure 3 for an illustration of the *WebLabs* common activity framework). In fact, we rather seldom succeeded in orchestrating such a discussion, largely due to pragmatic limitations but also because of the difficulty in establishing a distributed community of practice. More usually, a class was split to produce group *webreports* and each group then elected a representative to present their *webreport* to the whole class using the electronic whiteboard for whole class discussion.

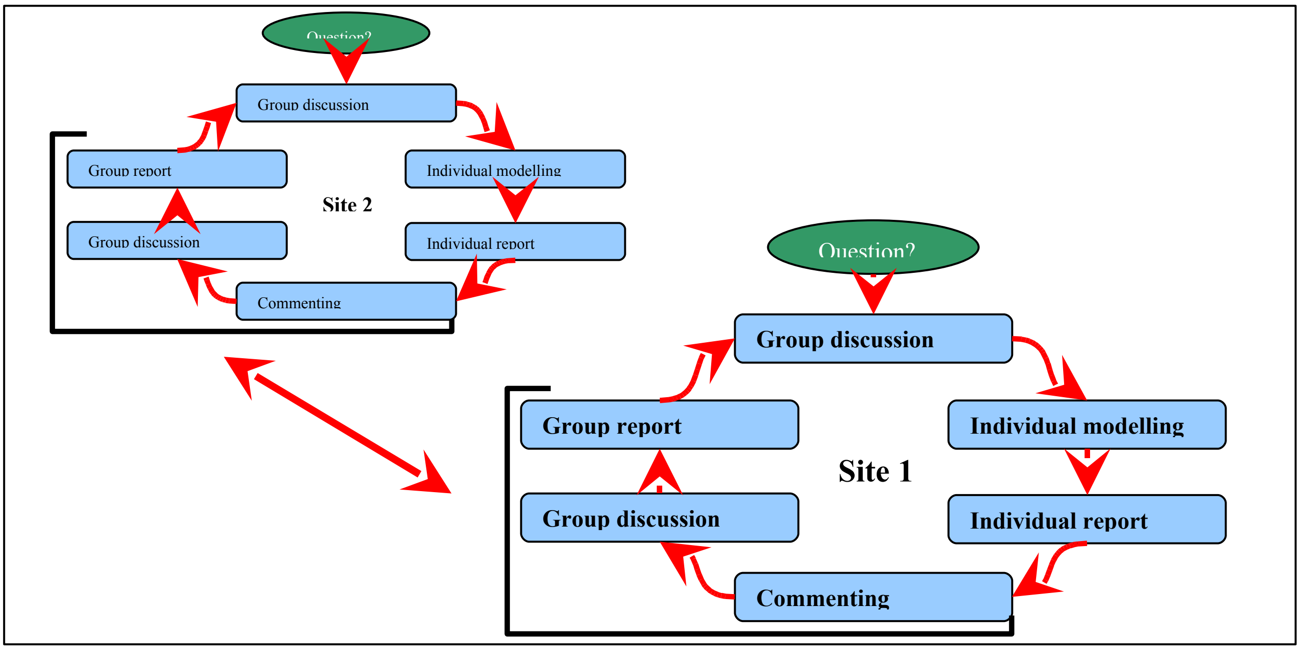


Figure 3 WebLabs common activity framework

The above pattern of activities emerged following iterative design and evaluation in this domain and others, for example 1D collisions, as described in Simpson *et al* (2005)

3 DESIGN OF TOOLS AND ACTIVITIES FOR SEQUENCES AND SERIES

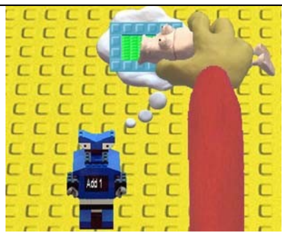
3.1 The preliminary programming task

The first activity we used was the "Add-a-number challenge". Its motivating question was posed more as a *ToonTalk* puzzle than a mathematical one. We asked the students "*how would you train a robot* [the *ToonTalk* equivalent of 'program a procedure'] *to count 1, 2, 3, 4, and so on*"? As expected, students would generally propose a construction similar to the one in Figure 4, and we would follow their instructions on the interactive whiteboard. Even this preliminary activity confronts students with one of the most fundamental concepts of algebra: the idea of variables and generalisation. This concept is prompted by a unique affordance of *ToonTalk*.

Most programming languages distinguish clearly between constants and variables. Code is written for the general case ("*any n*") and tested for specific cases (or written for a singular setting). *ToonTalk* employs *programming by example*. This means that robots are trained for specific values, which can then be generalized by 'erasing' the specific value from the robot's memory. In the case of this task, generalisation is required immediately - after the robot runs once, the value of current is no longer 0, and needs to be generalised if we want the robot to continue counting past 1.

However, this solution has a serious shortcoming - the robot does not keep a record of the numbers it generates. Since all computations are done 'in place', the only term of the sequence we can access is the last. This problem provides a motivation for introducing *birds* - *ToonTalk*'s message-passing mechanism. Whenever a *bird* is given an object, it will carry it to its *nest*. If we provide our robot with a bird, and train it to hand the sequence term over to it, we will have them stacked on the nest as the robot runs.

Train the robot to take a number 1 from the toolbox and drop it on the input, to increment it.



Generalise the program by erasing the value of the input from the robot's memory.

Give the robot its input box. The robot will continually repeat the actions it has been taught.

Figure 4 Training a robot to countHowever, this solution has a serious shortcoming - the robot does not keep a record of the numbers it generates. Since all computations are done 'in place', the only term of the sequence we can access is the last. This problem provides a motivation for introducing birds - ToonTalk's message-passing mechanism. Whenever a bird is given an object, it will carry it to its nest. If we provide our robot with a bird, and train it to hand the sequence term over to it, we will have them stacked on the nest as the robot runs.

3.2 Training Add-a-number: from the natural numbers to arithmetic progression

The first programming task that we set for students was to train a robot to generate the natural numbers, and send them to a nest. To scaffold students' work, we provide an *active worksheet*: a *webreport* template that includes the instructions for the task and questions related to it. Students create a *webreport* of their own by clicking a button on this page, and use the prompts on it to scaffold their report. The unique feature which makes the worksheet 'active' is that the *ToonTalk* tools required for the task are embedded in it, and at the click of the mouse students can load them into their programming environment. In this particular case, the worksheet contains a *task-in-a-box* (see Figure 5) - a *ToonTalk* box containing task instructions, an untrained robot, an input box, and output nest.

The task-in-a-box serves several purposes at once. First, it helps students overcome the shift in medium from a (mainly textual) web page to the animated programming environment. More important, it supports their work by providing the input box to be used in training. Last, it implicitly sets a standard for packaging and sharing *ToonTalk* models, one that can be used to establish a shared culture for discussion and exploration within and across school sites.

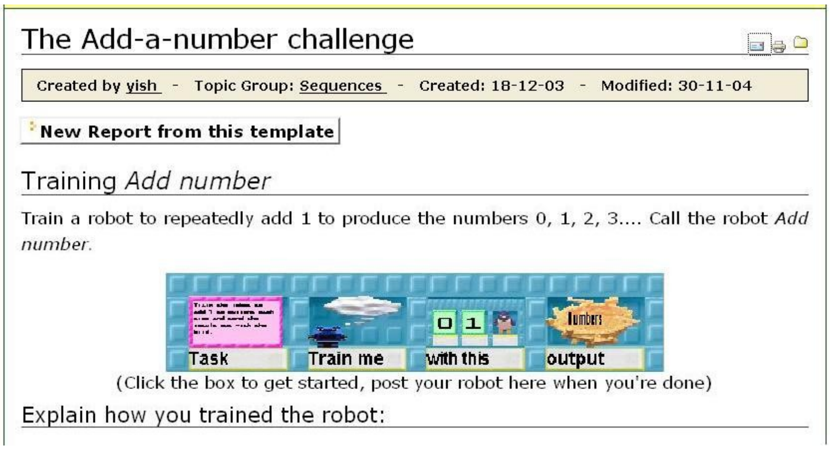


Figure 5 Add a number active worksheet and task-in-a-box

Note, as illustrated in Figure 5, that the input box contains two holes with numbers: an increment and the current value. A third hole contains the output bird. The robot needs to be trained to perform two actions: hand a copy of current to the output bird, and then drop a copy of the increment over current (thus adding them). This is the first occurrence of the *Stream pattern*: the numbers are sent out to the nest, one after the other, *'ad infinitum'*.

From a mathematical point of view, the *streams* method generates the natural numbers by repeated application of the *successor* function. At this stage, this kind of observation was *not* shared with the students. Instead, the students were engaged in constructing this procedure, manipulating it and using it as a building block in larger constructions, thus establishing their concept of the natural numbers as an object, the product of this process. The structure of the input box we provide the students provokes them to take generalisation one step further. The preliminary task already illustrated the need to generalize the current variable. However, by generalising the increment as well, students can use the program to generate any arithmetic progression! The next part of the worksheet asks students to predict which sequences can be generated by their robot and which cannot. These questions aim to promote students' mathematical conjecturing and argumentation, and specifically raise their awareness to the relationship between the *procedural* and the *structural* facets of sequences. After reflecting on these examples, students are asked to provide one more sequence that their robot can generate and one it cannot. The latter question is probably the hardest - in order to say that the robot *cannot* produce a sequence one has to argue about the nature of the *class* of sequences it *can* produce.

3.3 Training Add-up: constructing the partial sums series as an operation on a sequence

Once students have posted their Add-a-number robot and answered the questions, they are introduced to the next task: train a robot to add up the terms of a sequence. We refer to this as the Add-up robot. Mathematically, this robot embodies the concept of a sequence of partial sums, and implements it as a function on the domain of sequences: for any given sequence, it will produce the sequence of its partial sums. In concrete terms, we give the nest of the first sequence to the Add-up robot, which sums the numbers coming in to that nest, and sends the results out to its output nest Figure 6 illustrates how Add-a-number and Add-up robots are working in conjunction. The arrows, step numbers and dark bubbles (added for clarity) show the order of operations as an element passes down the stream. Students chain the Add-a-number robot to the Add-up robot by placing the former's output nest in the latter's box. Add-a-number generates an arithmetic sequence by repeatedly adding a copy of the 'add' to the current value (1) and copying the result to the out bird (2). The out bird carries this result to the nest in Add-up's input box (3). Add-up then adds this value to the total (4) and copies the result to its own out bird (5), which takes it to its nest (6) .

Figure 6 "Chaining" Add-a-number to Add-up

Directing students to this pattern addresses two aims. First, our initial experiments have shown – as suggested by the literature – that students tend to become confused between source sequences and the corresponding sequences of partial sums. This confusion causes difficulties in reasoning about limits, and sequence behaviour in general. Second, by using one sequence as an input to a process which generates another one, we address both process and object perspectives and encourage students to construct connections between them.

Students are asked to construct the Add-up robot and post it on their *webreport*. They then chain it with the Add-a-number robot, and experiment with summing different sequences. Next, they are asked to answer some questions regarding the chain of robots. Observing the patterns in and between these examples can lead to conjectures regarding the rules governing the co-variance of the source sequence and the corresponding sequence of partial sums. The Add-a-number and Add-up phase of activities is concluded by a group discussion, driven by the goal of composing a consensus *webreport* based on the individual *webreports* for Add-a-number and Add-up. First, the Add-a-number robot is constructed by the group. Students instruct the teacher how to train the robot, and where others disagree, discuss their solutions until a consensus is reached. After the robot is trained and posted, the students continue to discuss the answers to the questions in the worksheet, and when necessary the teacher displays students' individual *webreports* to refresh their memory. This process is iterated for the Add-up robot.

4 A FEW ILLUSTRATIVE EXAMPLES OF ACTIVITIES (AND LEARNING)

While the main focus of this paper is on the design of the activities and their close-knit relationship with the knowledge we were attempting to build, we provide here a very general overview and a few illustrative examples of what one group of students did and – we hope – learned. The results reported here are from an experiment conducted in London in autumn 2004. This experiment involved a group of 10 boys, aged 13-14, for 6 hourly sessions and a full day workshop. The activities were also undertaken by students in Bulgaria, Cyprus and Portugal, although we do not report these here. Our main sources of data are the models and texts that students published, and *in-activity-probes*: short interviews – typically up to five minutes – conducted while a student was engaged in an activity and referring to it. The use of this tool aims at capturing the process of knowledge construction and allows students to express their situated abstractions in the context that they are formed.

Most students found the activities engaging. They completed the tasks successfully, and then refined their answers through collaboration. They identified the natural numbers as a case of arithmetic progression, and then expanded that class to a more general one. They used formalisations derived from their ToonTalk experience to make sophisticated mathematical arguments.

4.1 Using Robot structure as a stepping stone to mathematical structure

Establishing elaborate norms of mathematical discourse is a lengthy process, the foundations of which can be seen in this activity. At the initial stage of the activity, we saw a shift from *modelling a particular robot to thinking about a class of sequences*. This shift was prompted by questions about which sequences their robots could and could not generate. Interestingly, some students interpreted this as which sequences could you generate with a robot *similar* to yours (see the example provided in Figure 8). This indicates that they did not see the robot as an isolated item, but as a representative of a mathematical class.

Students' initial classifications matched our expectations from the literature. For example, they exclude negative numbers or fractions from the arithmetic progression class (defined by the Add-a-number robot). These classifications were refined, and later elaborated, in the course of the activity. We saw two forces driving this refinement: students testing of their conjectures by manipulating the tools they created, and challenging each other's claims through webreports commenting and face to face discussion. The dynamics of learning through collaboration and communication are discussed further below.

BOB (we use students' internet pseudonyms as anonymised identifiers) published a characteristic report on Add-a-number. Following the worksheet structure, he first included the robot he had constructed and answered the questions regarding the specific sequences. He then reflected on the on the task, generalising the class of sequences that can be generated by such a robot:

> How would you explain to a friend what kind of sequences your robot can generate, and how it can be used to generate those sequences?
>
> *It can generate quite a lot of sequences, e.g. x 3 by copying the number 2 times, then dropping, one by 1 the numbers onto the original. Or, the nine times table, by copying the number 8 times etc. However, it cannot produce non whole numbers (I think)*
>
> Describe one sequence that cannot be generated by it, and explain why.
>
> *Dividing e.g. by 4.*

Note that BOB does not consider changing the input to the robot, but sees a class of robots which can be trained 'similarly' to the one he trained. This class of robots corresponds to the class of arithmetic progression sequences.

Martin, on the other hand, sees that he does not need to retrain the robot. In an 'in-activity-probe' taken right after he trained his first robot, we asked him if he could use it to generate other sequences and how.

> *M: Just go into it, I have to change it but I won't have to erase it, I could add 1 for that – 1 there* [points to 'add this' hole], *I could just make it add 2, I could just change its function to have to add 2.*

In other words, use the same process but change the input. Another in-activity-probe at the same stage shows an interesting confusion. As before, we probed Albert immediately after he had trained his Add-a-number robot, and asked him how to use it to generate other sequences. He suggested changing the increment from 1 to 3. We asked him what sequence would result:

> *Albert: Going up time 3 .. er.. times 3 sequence.*
>
> *Y: Times 3 sequence.*
>
> *A: Or add 3 sequence.*

Albert is confusing the closed form (a_n = 3*n) with the recursive form ($a_n = a_{n-1} + 3$). The term *times 3* relates to the *times 3 table* with which students are familiar from school. On the other hand, the term *add 3* relates to students intuition, whish also maps directly to the *ToonTalk* representation.

4.2 Distinguishing process, parameters and product

Students came to recognise that observed patterns can be decomposed into generic processes and the varying parameters they are initiated with. This realisation is a further step towards developing a structural view. This distinction is made salient by the particular design pattern we promoted for the Add-a-number robot. In the naïve implementation of the robot, both the process (incrementing by a constant) and its parameters (the value of the increment) are defined by the robot's training. By asking the students to train the robot with a box that contains the increment of 1, we enable them to later generalise that increment to any number. The process defines the class of sequences (arithmetic progression) while the value defines its domain (natural, integer or rational numbers).

Indeed, students acknowledged this distinction, and characterised the classes of sequences associated with the robot. They were able to explain how modifying the parameters affects the generated sequence, and what's more, how it determines its mathematical properties.

The streams pattern was very conducive to this affect. By seeing the output sequence not just as a pattern of numbers, but as the result of a composite process, students were able to decompose this process to its original elements and achieve a sophisticated analysis of the sequence structure. Albert suggested providing his robot with the initial value 1 and increment 4, to generate the sequence {*1, 4, 7...*}. We asked if this sequence would ever contain a number divisible by 3:

> *A: Yes, No, one above.*
>
> *Y: How can you get the same robot to generate a sequence which will have numbers divisible by 3?*
>
> *A: Yes, you change their current to 0 when you start off.*

Albert sees which properties of the sequence are determined by the process and which by the initial conditions, and how these interact.

Constructing the Add-up robot inspired further sophistication of students' arguments. Again, when asked if a particular sequence (*2, 6, 16, 20, 30 …*) can be generated by the chain they remarked:

> No. Because you can't retrain the robot. You would need to add these boxes (4 and 10) or retrain the robot to alternate between adding 4 and 10 (change the value by typing).

This is less of a programming claim than a mathematical one. Add-a-number can generate any arithmetic sequence. Chain it with Add-up, and you get a corresponding sum series. The sequence at hand is the result of alternately adding 4 and 10 – and is neither of the first form nor the second. $a_n = a_{n-1} + 4 : n = 2m$, $a_n = a_{n-1} + 10 : n = 2m+1$.

But when Allen presented their response in a group discussion, a second observation emerged:

> We believe if you change the 16 to a 12 it would be fine. If you um… started with um… with the 'in' as 2 'cause each it'll go up by 2: [*pointing at the spaces between the sequence terms*] 2, 4, 6, 8, 10, 12 and so on. So you get the answer, uh, and that would be a way without actually having two birds, which is impossible.

The sequence was not a sum series of an arithmetic progression, but if you changed the third term to 12 it would be. Not only did he note the structure of the sequence, he also saw how a new structure could be constructed from it.

4.3 Developing programming and mathematical norms

The task-in-a-box method proved to be highly effective. Students picked up the standards we set by imitation, without us having to detail explicitly the conventions for programming, packaging and publishing ToonTalk models. Figure 7 shows Luminardi's Add-a-number robot, as he had published it in his report. Using the scheme we initiated in the active worksheet but appropriating to his needs, he replaced box labels to describe how his tool should be used. Establishing such conventions as norms was crucial to facilitate communication in later phases. It ensured that models shared by one student would be readable by another. Furthermore, they were important even from an individual perspective, as they allowed students to easily revisit work they had done, reuse tools they had constructed or reflect on the evolution of their ideas.

Figure 7 Luminardi's add a number robot

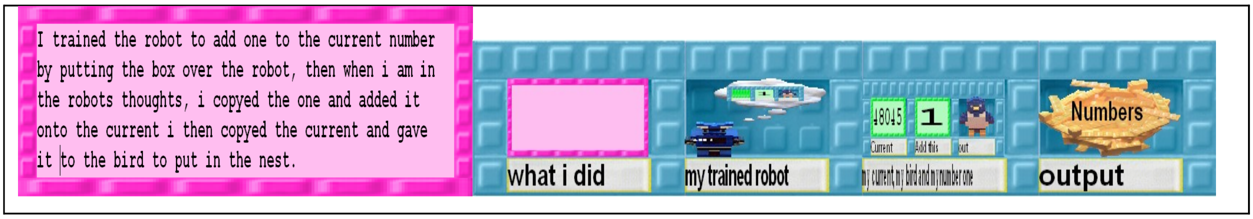


Figure 8 Superpat313's add a number and its description

While these norms emerged at the level of programming style, they evolved into a standard of mathematical discourse. Students quickly got into the habit of attaching written descriptions to their models, labelled 'description' or 'read this'. Superpat313 published his robot in a similar way (see Figure 8). He followed the same convention (from the description text in the leftmost hole to the nest in the rightmost). In contrast to Luminardi's product, the labels on Superpat313's report suggest that he was more focused on presentations ('what I did', 'my trained robot') than on usage.

As expected, most of the descriptions by students were procedural. Nevertheless, they constitute a first step towards students' reflective articulation of their work. For example, students used the box we provided as a package for the task to package their completed models. By doing so, they adopted the programming conventions we set without us needing to impose them explicitly. Students appropriated the packaging scheme to their needs – changing the box labels and adding box cells (holes in *ToonTalk* terminology) as needed. This packaging convention goes beyond aesthetics: it standardizes the use of 'streams', and prompts students to attach a reflective text to their model.

4.4 Refining individual knowledge through communication and collaboration

Our original vision saw the *WebReport* system mainly as a vehicle for highly structured communication between remote groups. We envisioned one group publishing a well thought out report on its findings, another responding to it, until a joint consensus report emerges from the discussion. We now realise that this view was over-ambitious. We have managed, in several cases, to facilitate inter-group discussion around a report, but this never went as far as publishing a joint report. We believe that the reasons behind this are predominantly pragmatic, rather than fundamental. The need to coordinate activity calendars, the length of time required to compose group reports and the availability of technology are the prime factors we found for the lack of success in this area. We believe that all this would change in an environment where such inter-group discussion is embedded in curricular activity.

In a way, these pragmatic shortcomings worked to our advantage. We split the London group into two sub-groups, let each one publish a group report and comment on the others report, and then had representatives of each group present the report and the responses to the class. This procedure led to an animated discussion in which the students reflected deeply on the activity. Writing the group report and group comments provided a sense of joint enterprise, which was highly engaging. The realisation that they are publishing to a wider audience entailed a strong commitment of students to their text. Having the text, and no less important – the ToonTalk models – available for discussion proved to be valuable cognitive aids for discussion. Students could draw on these artefacts to stimulate their memory or support their arguments.

The need to negotiate a common agreement provoked students to refine their classification of sequences. The dichotomy of 'sequences that can / cannot be generated by Add-a-number' gave way to a hierarchy of 'sequences that can / cannot be generated under such and such conditions'.

At the same time we, as designers, learnt an important lesson: these forms of collaboration are as effective in a traditional, face-to-face, setting as they are for a remote one. We conjecture that in fact, in order for a medium to successfully promote remote collaboration it should first be grounded in existing classroom practices.

After publishing their individual *webreports* on Add-a-number, the London students had a quick group discussion about which sequences it can generate. They had no problem generalising to a wide range of arithmetic progression sequences, including negative progression:

> Q: Now if I wanted to generate the sequence that's written down here, that's 2, 3, 4, 5?
>
> L: Change the current to 1.
>
> :
>
> Y: The next thing is, what about this sequence? Minus 1, minus 2, minus 3, minus 4? Yeah, Mark.
>
> M: Change the current to 0 and change the add this to -1.
>
> G: So what is it that stays the same and what changes to create this sequence compared to 1, 2, 3, 4, ...?
>
> Axel: Um, its just, it ... what it does stays the same but just the numbers it uses are changed.

Many students retrained a robot for each sequence when working individually. We were surprised to note that the same students saw that you could obtain the required sequences by changing the input conditions, without changing the robot - that is, they saw the process. We asked them to explain this phenomenon. They offered two factors. Albert put it down to reflection:

> Well, when you're like, um, working on it yourself and you're just doing it, you don't really think about all the different ways you just like try and do it in the one way you think you would ... you know how to do it. If you actually like, discuss it you kind of try a different way.

While Alan suggested confidence:

> um, maybe when you're changing it in the box, you may do something wrong and it will muck up so you think it doesn't work and you go on to do training robots.

The students then split into two groups. Each group published a consensus *webreport*.

Whereas most students were very cryptic in their personal reports, often skipping some of the questions, the group reports are far more elaborate.

Yishay's group's responses regarding the Add a number sequences are given in Table 1:

Sequence	Explanation
2, 3, 4, 5…	Yes. Start the current on 1.
-1, -2, -3, -4…	Yes. Start the "in" as -1, current as 0.
-7, -6, -5, -4..	Yes. Start the current as -8.
2, 4, 6, 8…	Yes. Have the in as 2 (current as 0).
5, -1, -7…	Yes. In is -6, current is 11.
0, 3, 6, 9,... 1.1, 2.1, 3.1, 4.1... -4, -6.9, -9.8, -12.7...	current -3, in is 3 in is 1, current 0.1 current -1.1, in is -2.9

Table 1. Yishay's group's responses regarding the Add a number sequences

Gordon's group's responses regarding the Add a number sequences are given in Table 2.

Sequence	Explain
2, 3, 4, 5…	Change the *Current* to 1 (The *Add this* value being 1)
-1, -2, -3, -4…	Change the *Add this* value to -1 (the *Current* being 0)
-7, -6, -5, -4..	Change the *Current* value to -8 (the *Add this* value being 1)
2, 4, 6, 8…	Change the *Current* value to 0 (the *Add this* value being 2)
5, -1, -7…	Change the *Current* value to 11 (the *Add this* value being -6)
Write down a sequence of your own, which can be generated by your robot.	8, 16, 32, 64...
Write down a sequence of your own, which **cannot** be generated by your robot.	*Triangle* numbers. 1, 3, 6, 10, 15... (The *Add this* keeps changing) You also can't to *Perfect* numbers.

Table 2. Gordon's group's responses regarding the Add a number sequences

After they completed their reports, each group commented on the reports from the other group. Having both groups on the same site gave us the opportunity to allow them to present their comments verbally. Several comments dealt with establishing norms. For example, Gordon's group posted a comment titled 'can you explain' which included the following issues:

> We think you should use the boxes labels instead of "in" and "n". Or you can say what in and n mean.
>
> In your answers in the first table, you haven't completed your all your statements - you haven't said what value the other field ("in") should hold.
>
> You haven't defined "r".

When presenting it, Albert argued why the other group should have used precise variable names:

> Ok, what we thought… our team … on looking at your, um, report … saw that you'd used 'in' … um… 'in' and 'n' to express your um… expressions or what. So we think that you should, um, have used the box labels instead of 'in' and 'n' 'cause otherwise we can't tell what 'in' and 'n' mean.

Yishay's team classified the sequences that can be generated by the Add-a-number robot as "*Any sequence that adds the same number each step to the current*" and noted:

> you cannot do:
>
> - square numbers
> - anything where you times or divide
> - in can't go up in prime numbers
> - any sequence with two stages
> - triangular numbers
>
> It can only go up (or down) in the same number each time.

To which the other group commented:

> You assumed that you cannot multiply or divide, but this can be done. You can also do the square numbers, by using ^2. We disagree with your statement "any sequence with two stages" because you could use advanced formulas (*2; +1).

This second group of students had discovered that *ToonTalk* allows them to replace the additive variable with any unary function. Instead of adding the value of this variable to the current term, it would apply the function it. In doing so they had re-formalised arithmetic sequences as a special case of iterative sequences. Later in the discussion, Alan (from Yishay's group) responded to these comments, acknowledging what they had learnt from them:

> Um, again, before we knew that you could use the semi-colon, before we knew you could do the semi-colon, we didn't think you could do multiplication or division.

Not only had Alan's group obtained a richer concept of sequence structure through the discussion, they acknowledge the evolution of their knowledge. This acknowledgment is important in the individual meta-cognitive sense, as it promotes critical reflection. It is also important in the development of a community of practice, in which changing one's stance is legitimised.

4.5 Using ToonTalk language to talk about sequences

ToonTalk terminology became part of the students' repertoire, allowing them to develop a formalism for rigorously describing and sharing information about sequences. During discussions students could refer to their models as taken-as-shared, and use them to fill the gaps in their mathematical terminology for describing sequences. In several groups, students consistently referred to the natural numbers as "*the add-one sequence*". Thus, where a mathematician would say "$a_n = B*n + C$, *where* $B = 3$ *and* $C = 1$" our students would say "an Add-a-number robot with current equals 1 and add-this equals 3". Not only are these descriptions equivalent, they are both as precise. Once the formalism is established, the natural numbers become an instance of the class of arithmetic sequences. Furthermore, using a recursive formalism meant that the arithmetic sequences became in turn a special case of a much larger class, as the examples in the previous section demonstrate.

Using this language, students can construct new structures from existing ones by applying conceptual metaphor (Lakoff & Núñez, 1998). They could describe the powers of 2 in terms of "replacing the add-this in Add-a-number with a times two".

Students became very proficient at using the tools, both in terms of completing the programming tasks, and in terms of using the tools they created when appropriate. During a group discussion, Alan demonstrated how to train the Add-up robot. When he finished training, he tested it on the floor.

> Alan: It's run out of numbers.
>
> Y: So if you wanted not to run out of numbers?
>
> A: You'd create... you can create an add a number robot.
>
> Y: Do you have the one you've created on the web?
>
> A: yeah…
>
> Y: so go fetch it.

Alan downloaded the Add-a-number robot he built in a previous session and chained it to his newly built Add-up robot. He explained:

> *So you take it out* [takes out input box] *and you want to take this robot out* [takes add a number robot out of the box] *and you give this numbers box to the in* [takes Add a number's nest and puts it in add up's hole] *and then you start this robot up* [points to Add a number]

We then asked Paul to give a commentary on Alan's robots while they ran:

> *The 'add up' robot is taking the numbers from the nest which says numbers I think, and the numbers in the numbers nest are coming from the other sequence which the other robot is doing so he's taking these numbers and he's adding them on to the total creating a different sequence out of the other sequence.*

> *Celia: What is this different sequence that it's created? This last sequence what is it, can you describe it?*
>
> *P: It's,(pause) it adds, it's going up I think, (laugh) it's going up 1 and adding that number on each time to the total.*

While the first part of his description is procedural, his answer to Celia's question shows a deeper understanding. He describes a process which isn't seen on screen: the result of the application of Add-up to the output of Add-a-number.

Students made sophisticated structural arguments using ToonTalk terminology. While discussing the sequence *2, 6, 16, 20, 30 ...* they argued whether it could be produced by the Add-a-number and Add-up robots. One group claimed you "needed to retrain the robot to first add 4 then add 10". However, Simon remarked:

> *I think ... that sequence would work if it was possible to have two birds [unclear] one nest, because you're using two alternating sequences and you're putting them all into one, like, end result basically*
>
> *Richard: and if you could have two birds, what would these two alternating sequences be?*
>
> *Simon: it would be… one would be 'add 4' and the other one would be 'add 10'*

Simon saw the sequence as a composite of two other sequences, a structure he did not have the tools to describe algebraically – and might not for several years – but could argue about in a coherent manner using a language situated in his programming experiences.

5 CONCLUSIONS

Our main conclusion is that, under rather carefully controlled circumstances and with a great deal of design effort, the modelling approach (in which students construct and share programs that express the organisation of rich phenomena) can assist in developing students' understandings of structure and consistency in mathematical situations. For some of these phenomena, it is likely to take several years before our students will encounter the armoury of algebraic tools that would enable them to conduct a detailed study. Given this scenario, the tools we designed did provide an interim solution to this difficulty that at least led to the students engaging with non-trivial ideas in mathematics. The examples we include demonstrate students' engagement with ideas of variables, partial sums, equivalence and rate of change.

We saw how the 'streams' design pattern allowed students to mould their intuitions into a situated formalism with which they could explore quite complex ideas, and argue convincingly and with commitment for their hypotheses. Nevertheless, it would be premature to argue that we can explicitly illustrate any relationship between our students' activities and their subsequent ability to handle algebraic expression: this putative linkage is, more generally, an area that warrants subsequent research.

By constructing robots that generate number sequences and then publishing their theories about the class of sequences which could be generated by their robots, students were led to reflect on structures rather than merely patterns. Their initial conjectures were, not unexpectedly, based on simple patterns. At first, students did not make implicit mathematical statements: their discourse remained strictly within the bounds of what was straightforward in the *ToonTalk* domain, based on simple procedures – add 2, 'times by 3' etc. Yet given time, the statements students made regarding the sequences that they constructed illustrate how they came to transcend the purely procedural view, and associated the process of generating the sequence with its mathematical structure. This can be seen even in the early examples of identifying the type of sequences that can be generated by robots *similar* to the one they constructed. Using the 'streams' paradigm, the sum series was modelled by passing the output of one process as an argument to another. This approach encourages a view of sequences that is both a process that unfolds and an object which can be manipulated by another process. As a result, some students began clearly to express the relationship between a sequence and its series, an issue many learners find – again unsurprisingly – confusing.

Collaboration and discussion played a central role in the construction of individual and group knowledge. The need to publish their thoughts in writing, and in a public medium, provoked students to reflect on their experiences and intuitions. The process of writing a joint report required that they find a shared mathematical language, and revisit their arguments. Reading others' reports critically, encouraged attention to detail. Yet all these results were contingent on two major facets: that the students had *something engaging to talk about*, and that they had a *reason to talk about it*. In our case, the former consisted of their models and conjectures, and the latter was built into the activity structure.

In conclusion, we note that our evidence confirms the claim, well observed in the literature, that students' intuitions of number sequences are intuitively recursive (that is, between successive terms of the co-domain, rather than as a relation between corresponding elements of the domain and the co-domain). The 'streams' approach seemed to offer a viable bridge between these two ways of expressing a sequence as a function. Of course, this is not to suggest that the standard functional form should be rejected; rather to propose a more approachable route into sequences and functions that is contiguous with students' intuitions. Once students have established their skills, our expectation would be to address functions of the natural numbers in a 'streamlike' way, for example, by chaining an Add-a-number robot with an Apply-function robot, which applies a function to every incoming input and outputs a stream of new values. This idea was employed in our design of activities on cardinality of infinite sets, as discussed by Kahn *et al* (2005).

As for the group of students referred to in this paper, the power of the 'stream' approach was indeed revealed several months later, when they returned to study sequences but at a much higher level. We challenged them, for example, to construct sequences that 'get closer to zero but never go below it'. Although they had not been working in this domain for very long, the students demonstrated impressive proficiency in 'stream' programming – using it to construct complex sequences and initiate surprising explorations.

REFERENCES

Abelson, H. and Sussman, G. (1996). *Structure and Interpretation of Computer Programs.* Cambridge, MA, MIT. 2nd edition.

Confrey, J., and Smith, E. (1994). Exponential functions, rate of change, and the multiplicative unit. *Educational Studies in Mathematics*, **26**, 135-164.

Cornu, B. (1991). Limits. In ed. D. Tall. *Advanced Mathematical Thinking*. pp 153-166. Dordrecht, Kluwer.

Cottrill, J., Dubinsky, E., Nichols, D., Schwingendorf, K., Thomas, K. and Vidakovic, D. (1996). Understanding the limit concept: beginning with a coordinated process. *Journal of Mathematical Behavior*, **15**, 167-192.

Davis, R. and Vinner, S. (1986). The notion of limit: Some seemingly unavoidable misconception stages. *Journal of Mathematical Behavior*, **5**, 281-303.

Dossey, J. (1998). Making algebra dynamic and motivating: A national challenge. *In The Nature and Role of Algebra in the K-14 Curriculum: Proceedings of a National Symposium*. pp 17-22. Washington, DC, National Academies Press.

Dubinsky, E., Weller, K., McDonald, M., & Brown, A. (2005). Some historical issues and paradoxes regarding the concept of infinity: An APOS based analysis: Part 1. *Educational Studies in Mathematics*, **58 No 3**, 335-359.

Eckel, B. (2002). *Thinking in Java.* Upper Saddle River, NJ, Prentice Hall.

Falk, R. and Lavy, S. (1989). How big is an infinite set? Exploration of children's ideas. *In Proceedings of the Thirteenth Annual Conference of the International Group for the Psychology of Mathematics Education*. Vol. 1 pp 252-259.

Floris, R. (2004). Some didactical variables for the study of numerical sequences using a mathematical pocket computer. *In Procceedings of ICME 10*. Copenhagen, Denmark. July 4-11, 2004.

Harel, I. and Papert, S. (1991). *Constructionism*. Norwood, NJ, Ablex.

Kahn, K. (1996). ToonTalk - An Animated Programming Environment for Children, *Journal of Visual Languages and Computing*, **7 No.2**, 197-217.

Kahn, K., Sendova, E., Sacristán, A. and Noss, R. (2005). M*aking Infinity Concrete by Programming Never-ending Processes*. Paper presented during the Symposium on the *WebLabs* project at the 7th International Conference on Technology in Mathematics Teaching, Bristol, UK, July 2005.

Kidron I., Zehavi, N. and Openhaim, E. (2001). Teaching the limit concept in a CAS environment: students dynamic perceptions and reasoning. In *Proceedings of the 25th International Conference for the Psychology of Mathematics Education*. Vol 3 pp 241-248. Utrecht

Kieran, C. (1997). Mathematical concepts at the secondary school level: the learning of algebra and functions. In eds. P. Bryant and T. Nunes. *Learning and Teaching Mathematics: An International Perspective*. pp 133-158. Hove, Psychology Press.

Küchemann, D. and Hoyles, C. (2005). Pupils' awareness of structure on two number / algebra questions. In *Proceedings of the Fourth Conference of the European Society for Research in Mathematics Education (CERME 4)*.

Lakoff, G. and Núñez, R. (1998). Conceptual metaphor in mathematics. In ed. J. Koenig. *Discourse and Cognition: Bridging the Gap*. Stanford, CA, CSLI Publications.

Li L. and Tall, D. (1993). Constructing different concept images of sequences and limits by programming. *In Proceedings of the 17th International Conference for the Psychology of Mathematics Education*. Vol 2, pp 41-48.

Mason, J. (1996). Expressing generality and roots of algebra. In eds. N. Bednarz, C. Kieran and L. Lee. *Approaches to Algebra: Perspectives for Research and Teaching*. pp 65-86. Dortrecht, Kluwer.

Mor, Y., Hoyles, C., Kahn, K., Noss, R. and Simpson, G. (2004). Thinking in progress. *Micromath*, **20 No 2**, 17-23.

Mor, Y., Tholander, J. and Holmberg, J. (2005). Designing for cross-cultural web-based knowledge building. In *Proceedings of CSCL '05: The Tenth International Conference on Computer Support for Collaborative Learning, Taipei, Taiwan.*

Noss, R. and Hoyles, C. (1996). *Windows on Mathematical Meanings: Learning Cultures and Computers*. Dordrecht, Kluwer.

Noss, R., Healy, L. and Hoyles, C. (1997). The construction of mathematical meanings: connecting the visual with the symbolic. *Educational Studies in Mathematics*, **33 No 2**, 203-233.

Oehrtman, M. (2003). Strong and weak metaphors for limits. In *Proceedings of the 27th Annual Meeting of the International Group for the Psychology of Mathematics Education (PME)*. Honolulu, Hawaii, vol 3, 397-404.

Olive, J. (2001). Children's number sequences: an explanation of Steffe's constructs and an extrapolation to rational numbers of arithmetic. *The Mathematics Educator*, **11 No 1**, 4-9.

Papert, S. (1991). Situating constructionism. In eds. I. Harel and S. Papert. *Constructionism*. Norwood, NJ: Ablex.

Radford, L. (2000). Signs and meanings in students' emergent algebraic thinking: a semiotic analysis. *Educational Studies in Mathematics*, **42 No 3**, 237-268

Sacristán, A. I. (1997). *Windows on the Infinite: Creating Meanings in a Logo-based Microworld.* Unpublished PhD Thesis, University of London, Institute of Education.

Salvit, D. (1977). An alternate route to the reification of function. *Educational Studies in Mathematics*, **33**, 259-281.

Sasman, M., Olivier, A. and Linchevski, L. (1999). Developing and stimulating generalisation thinking processes and skills. In *Fifth Annual Congress of the Association for Mathematics Education of South Africa*. Vol 1, pp 177-182. Port Elizabeth, SA, Port Elizabeth Technikon.

Scardamalia, M. and Bereiter, C. (1994). Computer support for knowledge-building communities. *Journal of the Learning Sciences*, **3 No 3**, 265-283.

Sfard, A. (1991). On the dual nature of mathematical conceptions: reflections on processes and objects as different sides of the same coin. *Educational Studies in Mathematics*, **22**, 1-36

Shapiro, E. (1988). Systolic programming: a paradigm of parallel processing. In ed. E. Shapiro. *Concurrent Prolog: Collected Papers*. pp 207-242. Cambridge, MA, MIT Press.

Simpson, G., Hoyles, C. and Noss, R. (2005). Designing a programming-based approach for modelling scientific phenomena. *Journal of Computer Assisted Learning*, **21 No 2**, 143-158.

Simpson, G., Hoyles, C. and Noss, R. (2006). Exploring the mathematics of motion through construction and collaboration. *Journal of Computer Assisted Learning*, **22 No 2**, 114-136.

Sriskanda, N. (2003). Using *Excel* spreadsheet to understand the limiting value of a sequence. In *Electronic Proceedings of the Sixteenth Annual International Conference on Technology in Collegiate Mathematics*. Chicago.

Steffe, L. (1988). Children's construction of number sequences and multiplying schemes. In eds. J. Heibert and M. Behr. *Number Concepts and Operations in the Middle Grades*. pp 119-140. Hillsdale, NJ, Lawrence Erlbaum Associates.

Steffe, L. (1994). Children's multiplying schemes. In eds. G. Harel and J. Confrey. *The Development of Multiplicative Reasoning in the Learning of Mathematics*. pp 3-40. Albany, NY, State University of New York Press.

SUN (2005). *STREAMS Programming Guide*. Santa Clara, CA: Sun Microsystems.

Tall, D. and Schwarzenberger, R. (1978). Conflicts in the learning of real numbers and limits. *Mathematics Teaching*, **82**, 44-49.

Tall, D. and Gray, E. (1993). Success and failure in mathematics: the flexible meaning of symbols as process and concept. *Mathematics Teaching*, **142**, 6-10

Tirosh, D. (1991). The role of students' intuitions of infinity in teaching the Cantorial theory. In ed. D. Tall. *Advanced mathematical thinking*. pp 199-214. Dordrecht, Kluwer.

Weigand, H. (1991). Iteration sequences and their representations. *Educational Studies in Mathematics*, **22 No 1**, 411-437.

Yackel, E. and Cobb, P. (1995). Classroom socio-mathematical norms and intellectual autonomy. In *Proceedings of the 19th International Conference for the Psychology of Mathematics Education*. Vol 3 pp 264-271.

Zazkis, R. and Liljedahl, P. (2002). Generalization of patterns: the tension between algebraic thinking and algebraic notation. *Educational Studies in Mathematics*, **49 No 3**, 379-402.

BIOGRAPHICAL NOTES

Yishay Mor holds an MSc in computer science from the Hebrew University, Jerusalem. Before being a researcher with the Weblabs project he was a software engineer with Cisco systems. He is now a researcher with the Learning patterns project.

Richard Noss is Professor of Mathematics Education, and co-director of the *London Knowledge Lab*. He has a Masters degree in pure mathematics, and a PhD in mathematical education. Professor Noss was, until recently, the Pro-director (ICT) of the Institute of Education.

Celia Hoyles is Professor of Mathematics Education at the Institute of Education. She is the first recipient of the Hans Freudenthal Medal, and was awarded an OBE in 2004. She is currently the Government's Chief Adviser for Mathematics.

Gordon Simpson holds a BE in Electrical and Electronic Engineering and MSc in Psychology from the University of Canterbury, New Zealand.

Ken Kahn holds a PhD in computer science and artificial intelligence from MIT. He is the designer and developer of *ToonTalk*. Prior to this, he researched programming languages at Uppsala University and Xerox PARC.