An excerpt from a draft of ***"Exploring Cardinality by Constructing Infinite Processes"*** *by Ken Kahn, Evgenia Sendova, and Richard Noss*

The role of concurrency and communication in exploring infinity

ToonTalk's support for concurrency and communication enables many activities involving multiple interacting infinite processes. ToonTalk robots are implicitly iterative – a robot executes an infinite loop unless the programmer explicitly programmed for it to terminate. All programming languages are capable of expressing an infinite computation such as generating all the natural numbers. Many programming languages also implement the concept of threads or processes so that a computation can consist of more than one infinite computation. An important and unique feature of ToonTalk is that it supports communication and coordination between these infinite processes.

Consider the challenge of constructing a program to compute the natural numbers. In an ordinary programming language, this can be accomplished by a program such as:

```
set n to 0 then
do forever set n to n + 1
```

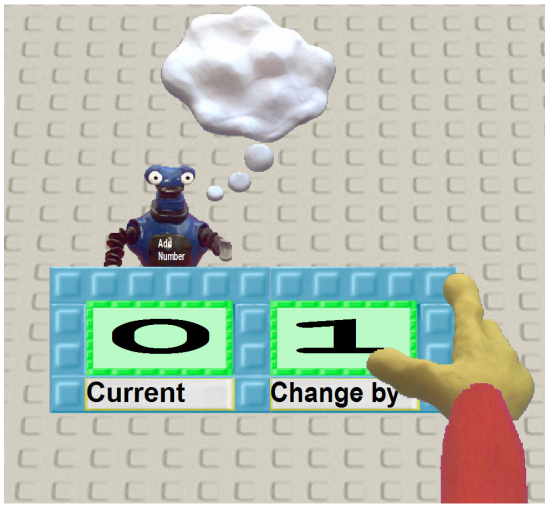
A program to compute the natural numbers

This computes the natural numbers but the numbers are not visible to the programmer. To see the numbers a print command can be added:

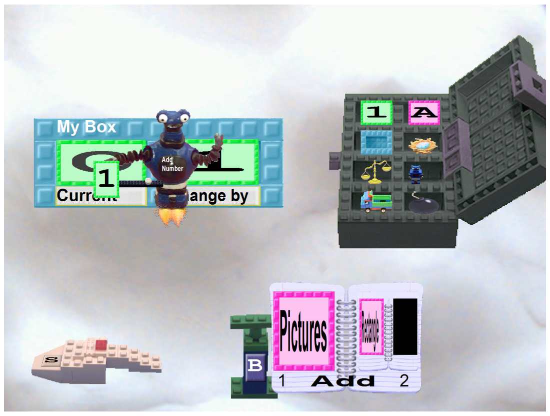
```
set n to 0 then
do forever set n to n + 1 then print n
```

A program to compute and display the natural numbers

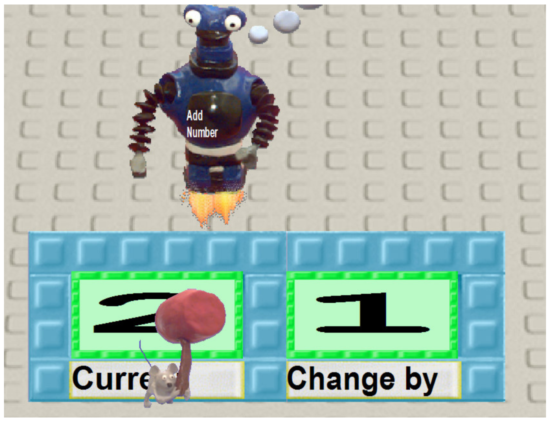
In ToonTalk all computations occur inside of a virtual animated world. If a robot is trained to drop 1 on the number in its box then when given a box with a 0 in it, an animation will be displayed of the robot picking up a 1, dropping it on the number in the box, the numbers being smashed together to produce 1 (the sum of 0 and 1). The robot is then seen picking up another 1, dropping it on top of the 1 in the box, and the number in the box becoming 2. And so on. There is no need for something like a print command.



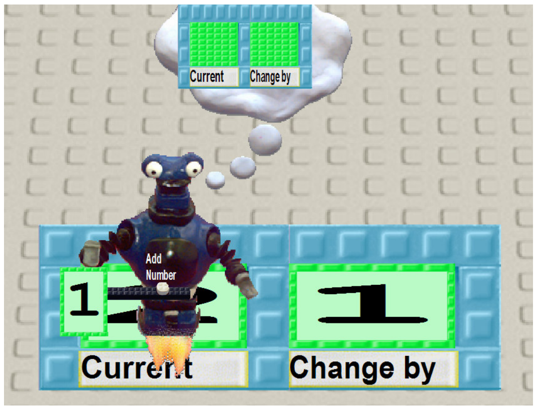
Screen shot 1 – Initiating the training of a robot to repeatedly add



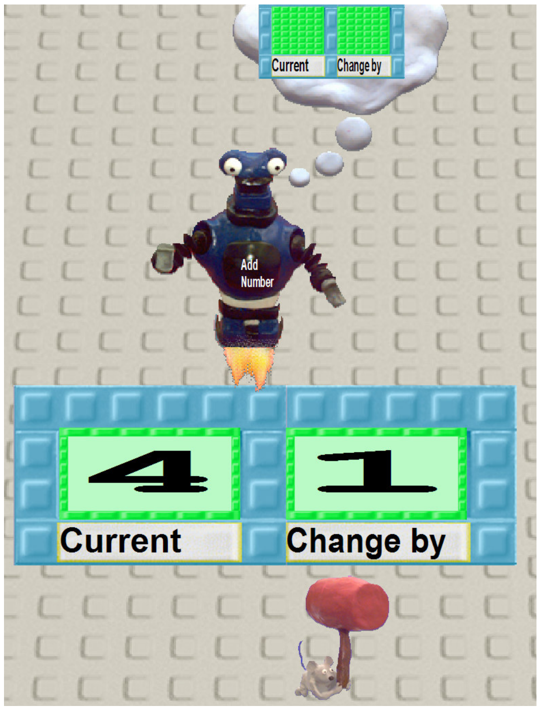
Screen shot 2 – Training the robot to add copies of the contents of the second hole to the contents of the first hole



Screen shot 3 – The robot in the middle of its second iteration



Screen shot 4 – The robot in the middle of its third iteration



Screen shot 5 – The robot in the middle of its fourth iteration

Suppose now one wants to compute both the natural numbers and the even natural numbers. The conventional way to do this is to edit the program to do both tasks within the main loop:

```
set n to 0 then
set even to 0 then
do forever
    set n to n + 1 then print n
    set even to even + 2 then print even
```

A program to compute the natural numbers and the even numbers

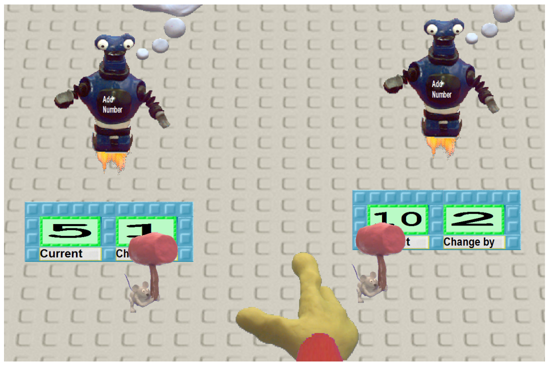
This approach quickly becomes unwieldy as independent tasks are implemented by intertwined actions within a single loop. It is difficult to analyse the process of generating the natural numbers or the process of generating the even numbers because the two processes are interleaved. There is neither computational modularity nor conceptual modularity in this approach.

If there are two or more concurrent processes, it is more general and modular to implement them so that each process runs independently of the others:

```
spawn a process to
    set n to 0 then
    do forever
        set n to n + 1 then print n in window1
then
spawn a process to
    set even to 0 then
    do forever
        set even to even + 2 then print even in window2
```

A concurrent program to compute the natural numbers and the even numbers

This version uses separate windows for the two sequences so the outputs of the processes are not interleaved arbitrarily. In ToonTalk one could simply copy the *Add Number* robot and give one copy a box containing 0 and 1 to compute the natural numbers and the other robot is given a box containing 0 and 2.



Screen shot 6 – Two copies of the robot running simultaneously with different state

The problem with the conventional program, even if it uses multiple processes, is that the processes are not computing something that can be used by other programs. We want the process that computes the natural numbers to produce an object that reifies the sequence of natural numbers. We want to be able to do operations upon sequences such as doubling each element. A programming language with lists or dynamic arrays could be used to express a program that constructs the sequence of natural numbers:

```
set natural_numbers to nothing then
spawn a process to
    set n to 0 then
    do forever
```

```
                set n to n + 1 then insert n at the end of natural_numbers
```

A concurrent program to compute a list of the natural numbers

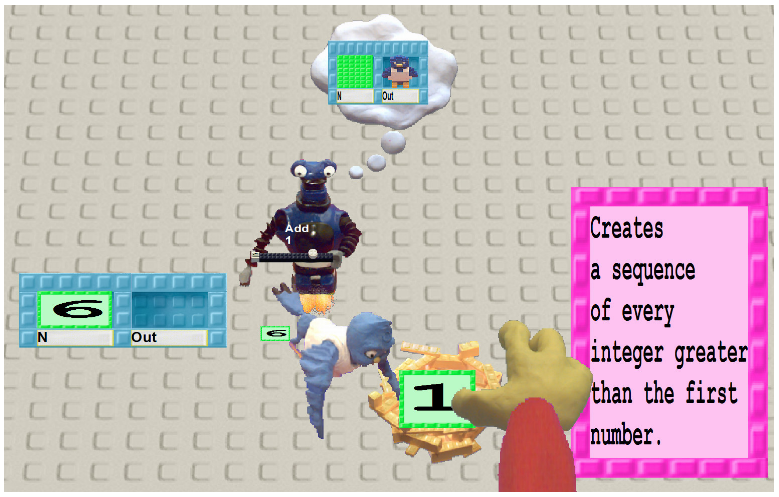
Another process can then produce the even numbers by doubling each natural number:

```
set even_natural_numbers to nothing then
set remaining_natural_numbers to natural_numbers then
spawn a process to
        do forever
                wait until the rest of the remaining_natural_numbers
                        is not empty then
                insert 2 * the first element of remaining_natural_numbers
                        at the end of even_natural_numbers then
                set remaining_natural_numbers
                        to the rest of remaining_natural_numbers
```

A computing a list of the even numbers from a list of natural numbers

This is getting very complicated. And most languages don't have a way to express the fact that the loop in this process needs to wait for the other process to produce numbers. And there are critical subtleties regarding the meaning of "insert … at the end". Is this process working on a data structure or a variable that the other process is modifying? We consider this a dead-end approach for all but advanced computer science students.

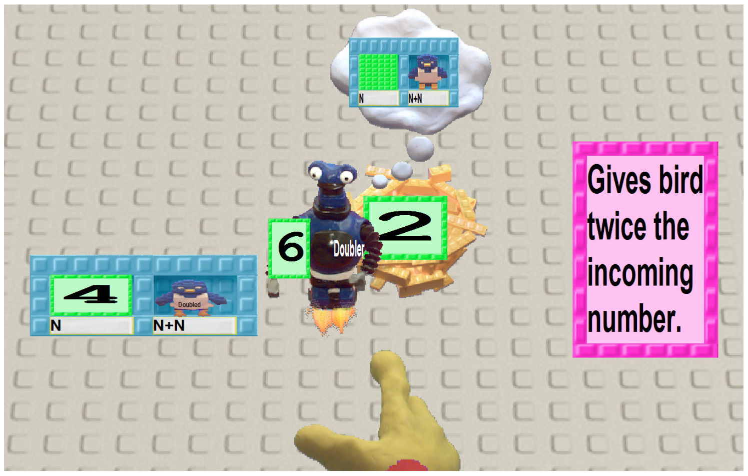
What is needed are child-friendly ways of expressing both the communication of data between processes and the coordination of the processes. ToonTalk is unique in providing this. Communication is accomplished within ToonTalk by birds that behave like carrier pigeons. If the programmer's persona (direct manipulation) or a robot (program execution) drops something on a bird, she'll take it to her nest. If she's already put something else on the nest then she'll put the new item on the bottom of the stack of earlier items. She then returns and can be given more items to put on her nest.


N
Out
Add 1
6
1
Creates a sequence of every integer greater than the first number.

Screen shot 7: The Add 1 robot producing the natural numbers

This is how one-to-one communication is accomplished. Robots, and therefore processes, can communicate this way if one robot gives things to a bird and the other robot removes things from the nest. A one-to-many communication pattern emerges if nests are copied, a many-to-one pattern emerges if birds are copied, and many-to-many if both are copied.

This bird/nest mechanism is also how robots coordinate their activities. If a robot is expecting a number in a hole in the box it is working on and instead finds an empty nest there the robot will wait until a bird arrives and puts something on the nest before proceeding. This implicitly performs the function of the "wait" command in the previous example.



Screen shot 8: The Doubler robot doubling the natural numbers

A nest repeatedly used in this way represents a stream of values. Streams are an advanced computer science concept (Sussman and Abelson 1996) that ToonTalk has made concrete and metaphorical. There is a close correspondence between the computer science concept of infinite streams and the mathematical concept of the infinite sequences. The computer scientist, however, treats an infinite stream as an object that one applies operations to. For example, an operation may double each element of the stream. Or it may compute the partial sums thereby producing a series from a sequence. Some operations such as one that splits a stream into multiple streams or one that combines streams into one expand the ways mathematical sequences are typically treated.

The concurrency and communication mechanisms within ToonTalk affords users with the opportunity of experiencing these processes concretely as separate teams of robots working simultaneously that are repeatedly giving numbers to birds. It also provides a way to experience each infinite sequence concretely as a nest where birds are continually delivering numbers. A user can hold in their hand a nest that, if they waited forever, would have all the natural numbers on it. And they routinely perform operations upon these infinite objects.