Università di Genova
DIPARTIMENTO INFORMATICA, BIOINGEGNERIA, ROBOTICA E INGEGNERIA DEI SISTEMI

Computer Science Workshop
PhD program in Computer Science and Systems Engineering

Enhanced Regular Corecursion for Data Streams

Pietro Barbieri

Introduction

As we venture deeper into the Internet of Things (IoT) era, stream processing is becoming increasingly important. With our recent work, we propose a stream calculus to lay the foundations of a tool for real time analysis of potentially infinite flowing data series.

Main Objectives

- Develop a calculus to define and manipulate infinite streams
- Provide a procedure to check whether a stream is well-defined
- Achieve a good compromise between expressive power and decidability

State of the Art

Two complementary approaches to manipulate streams:

Lazy Evaluation

Streams defined by arbitrary functions and inspected as much as needed.

Pros

- Widely known and well-established solution for stream processing
- Supports both regular (cyclic) and non-regular streams

Cons

- Operations that need to inspect the whole stream cannot be computed
- Allows the definition of incorrect streams due to high expressive power

Examples

```
one_two = 1:2:one_two    //1:2:1:2:1:...
from n = n:from(n+1)  //naturals from n
head (from 0)    ⟶ 0
allPositive one_two ⟶ ⊥ //non-termination
bad_stream = 0:tail bad_stream
```

Regular Corecursion

Streams are finitely represented by sets of equations. The executions avoids non-termination by keeping track of already processed calls.

Pros

- The entire stream can be inspected because it is finitely represented by a set of equations

Cons

- Fails to model non-regular streams

Examples

```
allPositive(one_two())   ⟶ true
head(from(0)) ⟶ ⊥ //non-termination
```

Our Solution

- Enhances regular corecursion
  - Not only constructors are allowed in equations defining streams
  - Supports regular and (some) non-regular streams
- Provides a procedure to check whether a stream definition is correct
  - `bad_stream()` ⟶ runtime error

Examples

Simple cyclic streams

```
repeat(n) = n:repeat(n)  //n:n:n:n:...
one_two() = 1:2:one_two()  //1:2:1:2...
```

Note: `[+]` `[*]` `[/]` are pointwise operations on streams, `^` computes the tail.

Non-regular streams

```
nat() = 0:(nat()[+]repeat(1))
fact() = 1:((nat()[+]repeat(1))[*]fact())
fib() = 0:1:(fib()[+]fib()^)
```

`nat()` ⟶ | 0 | 1 | 2 | 3 | ... |

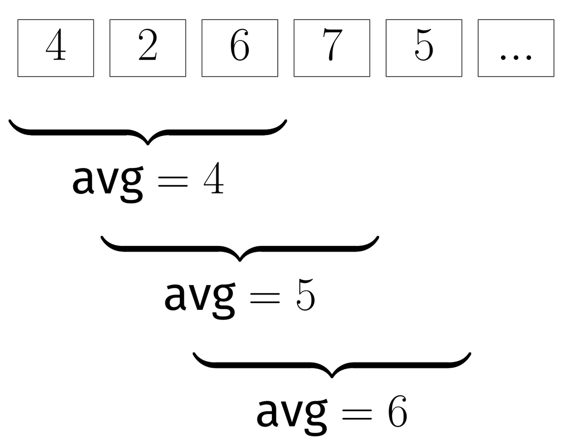
`fact()` ⟶ | 1 | 2 | 6 | 24 | ... |

`fib()` ⟶ | 0 | 1 | 1 | 2 | ... |

Functions for stream processing

```
aggr(n,s) = if n<=0 then repeat(0)
            else s[+]aggr(n-1,s^)
avg(n,s) = aggr(n,s)[/]repeat(n)
```

Below you find an example of execution of avg over a window of size 3



Forthcoming Research

- Make function definitions more *flexible*
  - The user is allowed to specify the behaviour in presence of a cycle
- Introduce a static type system to prevent runtime errors

Reference paper:

Davide Ancona, Pietro Barbieri, Elena Zucca. Enhanced Regular Corecursion for Data Streams. ICTCS21

CONTACTS

Davide Ancona
Davide.Ancona@unige.it

Elena Zucca
Elena.Zucca@unige.it

Pietro Barbieri
pietro.barbieri@edu.unige.it

CSW