Interpretation of Stream Programs

Romain Péchoux

Université de Nancy 2, LORIA

24 october 2010
Outline

Goal

There is a need of general studies in order to develop new formal methods and tools for the analysis of stream programs.

Several studies [Sijtsma 89, Klop et al. 89,97,07] have focused on the notion of productivity, i.e. the ability to produce a normal form of the n-th output element.

However formal methods on the complexity analysis are lacking.

The methodology

This talk presents a combination of works with :

- Marco Gaboardi (Universita di Bologna)
- Mathieu Hoyrup (INRIA), Emmanuel Hainry (Nancy 1) and Hugo Férée (ENS Lyon)

using quasi-interpretaion-based techniques that are useful to prove quantitative properties of stream programs.
Setting and Motivations

Motivations

▶ Program resource control and resource usage certification.
▶ Properties for programs working on infinite data structures, notably streams.

Settings

▶ Lazy Functional Programming Languages, like Haskell, enable a treatment of stream programs in a finitary way.
▶ Use of static analysis interpretation methods inspired by:
  ▶ Quasi-interpretation [Bonfante et al. 01] and
  ▶ Sup-interpretation [Marion et al. 06]
▶ The presented criteria permit to build upper bounds on the number and size of computed stream elements.
Streams

Streams can be used to formalize network communication flows, audio and video signals flows, etc.

For simplicity, I will consider streams as infinite sequences of natural numbers:

\[ 0 : 1 : 2 : 3 : 4 : 5 : 6 : \cdots : 65512 : 65513 \cdots \]

\[ 1 : 1 : 2 : 3 : 5 : 8 : 13 : \cdots : 46368 : 75025 : \cdots \]

\[ 0 : 1 : 1 : 0 : 1 : 0 : 0 : 1 : 1 : 0 : 0 : 1 : 0 : 1 : 1 : 0 \cdots \]

In the sequel we will consider programs built over numerals and addition and extended by particular function definitions.
It is convenient to distinguish between two different kinds of programs dealing with stream:

- **Stream constructor** programs

- **Stream function** programs

Clearly the former are a particular case of the latter.
Streams in lazy functional languages

Lazy functional programming languages like Haskell allow us to define streams by means of (typed) expressions:

\[
\begin{align*}
\text{repeat} & : \text{Nat} \to \text{[Nat]} \\
\text{repeat } x &= x : (\text{repeat } x) \\
\text{fibo} & : \text{[Nat]} \\
\text{fibo} &= 1 : 1 : (\text{sadd fibo (tail fibo)})
\end{align*}
\]

We restrict the considered programs to a first order fragment of Haskell.

We use a lazy evaluation semantics without sharing:

\[
\begin{align*}
\text{fibo} \downarrow 1 : 1 : (\text{sadd fibo (tail fibo)}) \\
\text{repeat } 3 \downarrow 3 : (\text{repeat } 3)
\end{align*}
\]
Preliminaries on Programs

- We will extensively use the standard function definition:

\[
\begin{align*}
\text{take} & :: \text{Nat} \to [\alpha] \to [\alpha] \\
\text{take } 0\ s & = \text{nil} \\
\text{take } (x + 1)\ \text{nil} & = \text{nil} \\
\text{take } (x + 1)\ (y : ys) & = y : (\text{take } x\ ys)
\end{align*}
\]

\[
\begin{align*}
!! & :: [\alpha] \to \text{Nat} \to \alpha \\
(x : xs)!!0 & = x \\
(x : xs)!!(y + 1) & = xs !! y
\end{align*}
\]

- "Approximation lemma" : provides finitary methods

\[
s = r \iff \forall n \ ((\text{take } n\ s) = (\text{take } n\ r))
\]

- We will write \(e_n\) and \(e \upharpoonright n\) as a short for \(e!!n\) and \(\text{take } n\ e\).
We also need a program forcing the evaluation of programs:

\[
\text{eval} :: A \to A
\]

\[
\text{eval} \ (c \ e_1 \ \cdots \ e_n) = \hat{C} \ (\text{eval} \ e_1) \ \cdots \ (\text{eval} \ e_m)
\]

where \(\hat{C}\) is a program representing the strict version of the primitive constructor \(c\).

In particular when it is applied to stream expressions it diverges:

\[
\begin{align*}
\text{eval fibo} & \uparrow \quad \text{eval (repeat 3)} \uparrow \\
\text{eval fibo} & \downarrow^\infty 1 : 1 : 2 : 3 : \cdots 75025 \cdots \\
\text{eval (repeat 3)} & \downarrow^\infty 3 : 3 : 3 : \cdots 3 \cdots \\
\text{eval (take 5 fibo)} & \downarrow 1 : 1 : 2 : 3 : 5
\end{align*}
\]

For this reason in what follows we write \(e \downarrow^v v\) for \(\text{eval} \ e \downarrow^v v\).
A program $P$ admits an interpretation if there is an assignment $\langle \cdot \rangle :$

- **total**: $\forall t, \langle t \rangle : (\mathbb{R}^+)^n \rightarrow \mathbb{R}^+$

- **monotonic**: $X_i \geq Y_i \Rightarrow \langle t \rangle (\ldots, X_i, \ldots) \geq \langle t \rangle (\ldots, Y_i, \ldots)$,

such that for each definition in $P$ of the shape $f \ p_1 \cdots p_n = e$:

$$\langle f \ p_1 \cdots p_n \rangle \geq \langle e \rangle$$

Moreover, a program $P$ admits an additive interpretation if $\langle \cdot \rangle$ is such that for every symbol $c$ of arity $n$:

- $\langle c \rangle = 0$ if $n = 0$

- $\langle c \rangle (X_1, \cdots, X_n) = \sum_{i=1}^{n} X_i + \alpha_c$, with $\alpha_c \geq 1$ otherwise.
Consider again the example:

\[
\text{repeat : } \text{Nat} \rightarrow [\text{Nat}]
\]
\[
\text{repeat } x = x : (\text{repeat } x)
\]

the assignment \((-\cdot\)) defined as

\[
(x) = X \quad (\cdot : )(X, Y) = \max(X, Y) \quad (\text{repeat})(X) = X
\]

It is total and monotonic and we check that

\[
(\text{repeat } x) \geq (x : (\text{repeat } x))
\]

We have:

\[
(\text{repeat } x) = (\text{repeat})(X) = X
\]
\[
= \max(X, X) = (\cdot : )(X, X)
\]
\[
= (\cdot : )(X, (\text{repeat} X))
\]
\[
= (x : (\text{repeat } x))
\]
Properties of Interpretations

- **Interpretation respects evaluation**
  Given a program \( P \) admitting the interpretation \( (\_\_\_\_) \), for every closed expression \( e \) we have
  - if \( e \downarrow v \) then \( |e| \geq |v| \)
  - if \( e \downarrow v \) then \( |e| \geq |v| \)

- **Interpretation does not respect infinitary evaluation**
  If \( \text{fibo} \downarrow \infty \) \( u = 1 : 1 : 2 : 3 : 5 : 8 : \cdots \), we do not expect:
  \[
  |\text{fibo}| \geq |u|
  \]

- **Interpretation vs. expression size**
  Given a program \( P \) admitting the interpretation \( (\_\_\_\_) \), there is a function \( G : \mathbb{R}^+ \to \mathbb{R}^+ \) such that for each expression \( e \):
  \[
  |e| \leq G(|e|)
  \]
First property - Global Upper Bound

- Each element is bounded by a function in the maximal size of the input elements.
- A program generated using:

```plaintext
repeat :: Nat \rightarrow [Nat]
repeat x = x : (repeat x)
zip :: [\alpha] \rightarrow [\alpha] \rightarrow [\alpha]
zip (x : xs) ys = x : (zip ys xs)
square :: [Nat] \rightarrow [Nat]
square (x : xs) = (mul x x) : (square xs)
```

is bounded by some \( k \).

- For example:

```
square (zip (repeat 5) (square (zip (repeat 7) (repeat 4))))
```

is bounded by \( k = 2401 = 7^4 \).
Global Upper Bound - Intuition

\[ |a_i| \leq \max_{r=0}^{\infty} |a_r| = k \]

\[ s = \begin{array}{ccc}
\cdots & a_i & \cdots & a_1 \\
\end{array} \]

\[ |b_j| \leq F(|e|, k) \]

\[ b_1 \cdots b_j \cdots = f(e, s) \]
Global Upper Bound - Intuition

Clearly the input stream may have no maximal element:

\[
|a_i| \leq \max_{r=0}^{\infty} |a_r| = k
\]

\[
s = \overbrace{\cdots a_i \cdots a_1 \cdots}^{f_e} = f(e, s)
\]

\[
|b_j| \leq F(|e|, k)
\]

\[
b_1 \cdots b_j \cdots
\]
Global Upper Bound - Intuition

Stream constructors are just a particular case:

\[ |a_i| \leq \max_{r=0}^{\infty} |a_r| = k \]

\[ s = \cdots a_i \cdots a_1 \]

\[ |b_j| \leq F(|e|, k) \]

\[ b_1 \cdots b_j \cdots = f(e, s) \]
Global Upper Bound - Formally

A function $f : [\sigma'] \rightarrow \tau \rightarrow [\sigma]$ of $P$ has a global upper bound if there is $F : \mathbb{R}^+ \rightarrow \mathbb{R}^+$ such that, for every stream expression $s \in P$ and for every expression $e \in P$ if

$$(f \ s \ e)_n \downarrow \nu \ \nu$$

then

$$F(\max(|s|, |e|)) \geq |\nu|$$

A program $P$ has a global upper bound if every function symbol in it enjoys this property.
GUB Criterion

- A program $P$ satisfies the GUB criterion if it admits an additive interpretation $\langle - \rangle$ such that:

$$\langle : \rangle (X, Y) = \max(X, Y)$$

- The interpretations of GUB programs well behave with respect to the selection function $!!$. In particular if $e_n \Downarrow_v v$ then

$$\langle e \rangle \geq \langle v \rangle$$

- Theorem
  If a program is GUB then each stream function $f$ has a global upper bound $F$. 
GUB Examples

- The program including \texttt{repeat}, \texttt{zip} and \texttt{square} is GUB, it admits the interpretation:

\[
\begin{align*}
\langle \text{repeat} \rangle(X) &= X + 1 \\
\langle \text{zip} \rangle(X, Y) &= \max(X, Y) \\
\langle \text{square} \rangle(X) &= X^2 \\
\langle :\rangle(X, Y) &= \max(X, Y)
\end{align*}
\]

- Define \texttt{fun} \( x = \text{square} \ (\text{zip} \ (\text{repeat} \ 9) \ x) \).

Taking \( F(X) = \langle \text{square} \rangle(\langle \text{zip} \rangle(\langle \text{repeat} \ 9 \rangle, G(X)) \)

\[
\text{If } (\text{fun} \ e) \Downarrow \nu \ m \\
\text{then } F(|e|) = (\max(10, G(|e|)))^2 \geq |m|
\]
Second property - Local Upper Bound

- Each element is **bounded** by a function in the maximal size of the input elements and of its position in the output stream.
- Consider the following stream program:

  
  
  \[
  \begin{align*}
  \text{nats} & : \text{Nat} \rightarrow \text{[Nat]} \\
  \text{nats} \; x &= x : (\text{nats} \; (x + 1)) \\
  \text{sad} & : \text{[Nat]} \rightarrow \text{[Nat]} \rightarrow \text{[Nat]} \\
  \text{sad} \; (x : xs) \; (y : ys) &= (\text{add} \; x \; y) : (\text{sad} \; xs \; ys)
  \end{align*}
  \]

- the size of each element of a stream $s$ built only using $\text{nats}$ and $\text{sad}$ as

  \[
  \text{sad} \; \text{nats} \; 3 \; (\text{sad} \; \text{nats} \; 5 \; \text{nats} \; 4)
  \]

  is **not globally bounded** but is bounded by the function $(3 \times 5) \times (n + 1)$ where $n$ is its position in the stream.
We consider also dependencies on the local index $n$ as follows:

\[
|a_i| \leq \max_{r=0}^{\infty} |a_r| = k
\]

\[
s = \begin{array}{c}
\ldots
a_i \\
\ldots
a_1
\end{array}
\]

\[
|b_n| \leq F(e, k, n)
\]

\[
f_e(b_1 \ldots b_n \ldots) = f(e, s)
\]
A function $f : [\sigma'] \rightarrow \tau \rightarrow [\sigma]$ of $P$ has a **local upper bound** if there is $F : \mathbb{R}^+ \rightarrow \mathbb{R}^+$ such that, for every stream expression $s \in P$ and for every expression $e \in P$ if

$$(f \ s \ e)_n \Downarrow_v \mathbf{v}$$

then

$$F(\max(|s|, |e|, |n|)) \geq |\mathbf{v}|$$

A program $P$ has a **local upper bound** if every stream function symbol enjoys this property.

$|a_i| \leq \max_{r=0}^{\infty} |a_r| = k$

$s = \cdots a_i \cdots a_1$

$f_e$

$|b_n| \leq F(e, k, n)$

$b_1 \cdots b_n \cdots = f(e, s)$
Local Upper Bound - Example

The following program computing the Fibonacci sequence:

\[
1 : 1 : 2 : 3 : 5 : 8 : 13 : \cdots : 46368 : 75025 : \cdots
\]

has clearly a local upper bound. The size of each element in position \(n\) is bounded by \(2^n\):

\[
\text{fib} :: [\text{Nat}] \\
\text{fib} = 0 : (1 : (\text{sad} \text{fib} (\text{tail} \text{fib}))) \\
\text{sad} :: [\text{Nat}] \rightarrow [\text{Nat}] \rightarrow [\text{Nat}] \\
\text{sad} (x : xs) (y : ys) = (\text{add} x y) : (\text{sad} xs ys)
\]

\[
\text{tail} :: [\alpha] \\
\text{tail} x : xs = xs
\]
A program $P$ admits a **parametrized interpretation** if there is a total **parametrized assignment** $(|-)_L$ over $\mathbb{R}^+$, such that:

- it is extended to expressions by

\[
\begin{align*}
(|\text{hd}:\text{tl}|)_L &= (\_)_L((|\text{hd}|)_L, (|\text{tl}|)_{L-1}) & \text{Case:} \\
(|t\ e_1\cdots e_n|)_L &= (|t|)_L(|e_1|)_L, \ldots, (|e_n|)_L) & t \neq: \\
\end{align*}
\]

- for each definition $f \xrightarrow{p} e : (|f\ p|)_L \geq (|e|)_L$

**Proposition**

- i) If $e \xrightarrow{v} v$ then $(|e|)_r \geq (|v|)_r$
- ii) If $e \xrightarrow{v} v$ then $(|e|)_r \geq (|v|)_r$
- iii) $\exists G, \forall e, (|e|)_r \leq G(|e|, r)$
A program $P$ satisfies the LUB criterion if it admits an additive and monotonic (also in $L$) parametrized additive interpretation $(\cdot)_L$ such that:

$$(\cdot)_L(X, Y) = \max(X, Y)$$

LUB programs has suitable properties wrt $!!$:

If $e_n \downarrow v$

then $(e)_n \geq (v)_0$

Theorem

If a program is LUB then each stream function $f$ has a local upper bound $F$. 
LUB Examples

- The program for the Fibonacci sequence consisting of tail, sad, and fib is LUB. It admits the additive interpretation:

  \[
  \langle 0 \rangle_L = 0 \quad \langle +1 \rangle_L(X) = X + L + 1 \quad \langle : \rangle_L(X, Y) = \max(X, Y) \\
  \langle \text{sad} \rangle_L(X, Y) = \langle \text{add} \rangle_L(X, Y) = X + Y \quad \langle \text{tail} \rangle_L(X) = X \quad \langle \text{fib} \rangle_L = 2^L
  \]

- The program including nats and sad is LUB, it admits the interp.:

  \[
  \langle \text{nats} \rangle_L(X) = X + L \quad \langle \text{sad} \rangle(X, Y) = X + Y \\
  \langle : \rangle(X, Y) = \max(X, Y)
  \]

Consider the definition \(\text{dubnat } x = \text{sad } (\text{nats } 5) (\text{nats } x)\). Take \(F(X) = \langle \text{sad} \rangle_X(\langle \text{nats } 5 \rangle_X, \langle \text{nats} \rangle_X(X))\) and \(k = \max(|e|, |n|)\), if

\[
(\text{dubnat } e)_n \downarrow m
\]

then \(F(k) = \langle \text{sad} \rangle_k(\langle \text{nats } 5 \rangle_k, \langle \text{nats} \rangle_k(k)) = 3k + 5 \geq m\)
The third and fourth properties - Length and size-based I/O Upper Bounds

- The number of output elements is bounded by a function in the number (resp. size) of read input elements.

- Consider the following stream program:

```
zip :: [α] → [α] → [α × α]
zip (x:xs) ys = x:(zip ys xs)
double :: [Nat] → [Nat]
double (x:xs) = x : (x : (double xs ))
```

- the number of output written elements of a stream function \( f \) defined by:

\[
f \ x = \text{double}(\text{zip } (\text{double } x) (\text{double } x))
\]

is bounded by a constant \( k(=8 \text{ in this case}) \).
Length-based I/O Upper Bound - Intuition

We consider dependencies between the number of input readings and the number of output writings:

\[ s = \cdots a_i \cdots a_1 = f(e, s) \]

\[ j \leq F(e, i) \]

\[ b_1 \cdots b_j \cdots = f(e, s) \]
Length-based I/O Upper Bound - Formally

\[ s = \cdots a_i \cdots a_1 \quad f_e \quad j \leq F(e, i) \quad b_1 \cdots b_j \cdots = f(e, s) \]

A function \( f : \lbrack \sigma' \rbrack \rightarrow \tau \rightarrow \lbrack \sigma \rbrack \) of \( P \) has a length based I/O upper bound if there is \( F : \mathbb{R}^+ \rightarrow \mathbb{R}^+ \) such that, for every stream expression \( s \in P \) and for every expression \( e \in P \) if

\[ \text{length}(f \ s \rvert_n \ e) \downarrow_v m \]

then

\[ F(\max(|s|, |e|, |n|)) \geq |m| \]

A program \( P \) has a length based I/O upper bound if every stream function in it enjoys this property.
LBUB Criterion

- A program is **LBUB** if it admits an additive interpretation \( (\cdot-L) \) such that

\[
(|:|)(X, Y) = Y + 1 \\
(|+1|)(X) = X + 1
\]

- **Theorem**
  - If a program is **LBUB** then each stream function in it has a length based I/O upper bound.
LBUB Examples

The program including zip and double is LBUB, it admits the interpretation:

\[(\text{zip})(X, Y) = X + Y \quad (\text{double})(X) = 2X \quad (\text{post})(X, Y) = Y + 1\]

Consider the definition

\[
\text{fun } x = \text{double } (\text{zip } (\text{double } x) (\text{double } x))
\]

Take \(F(X) = (\text{double})((\text{zip})((\text{double})(X), (\text{double})(X))),\) and \(k = \max(|n|, |s|),\) if

\[
\text{length}((\text{funone } s |_{n}) \downarrow m)
\]

then

\[
F(k) = (\text{double})((\text{zip})((\text{double})(k), (\text{double})(X))) = 8 \times k \geq m
\]
Oracle Turing Machine

input tape

query tape

output tape

answer tape

Alternative model: OTM’ if $F(w)$ is substituted to $F(|w|)$ on the oracle answer tapes
Oracle Turing Machine

- Costs: Usual step: 1, Oracle step $F(\vert w \vert)$

- An OTM $(\mathbb{N} \to \mathbb{N}) \to \mathbb{N} \to \mathbb{N}$ has time complexity $T$ if it halts before $T(\vert F \vert, \vert x \vert)$ steps on oracle $F$ and input $x$, where

  $$|F|(n) = \max_{k \leq n} |F(k)|$$

- Second order poly $P := c \mid X \mid P + P \mid P \times P \mid Y(P)$

- A function is **OTM-Poly** if it is computed by an OTM of time complexity $P$, second order polynomial.

- A function is **BFF** if it is computed by an OTM' of time complexity $P$, second order polynomial.
Second order polynomial interpretations

We slightly modify interpretations by:

- The interpretation of a function $f : [\text{Nat}] \to \text{Nat} \to [\text{Nat}]$ is a function $(\langle f \rangle) : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}^2 \to \mathbb{N}$.
- The interpretation of a function $f : [\text{Nat}] \to \text{Nat} \to \text{Nat}$ is a function $(\langle f \rangle) : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N} \to \mathbb{N}$.
- $\langle y \rangle(Z) = Y(Z)$
- $\langle z \rangle(X, Y, Z + 1) = 1 + X + Y(Z)$ and $\langle z \rangle(X, Y, 0) = 1 + X$. 

Second order poly $P := c \mid X \mid P + P \mid P \times P \mid Y(P)$
( polyExp $PE := c \mid X \mid PE + PE \mid PE \times PE \mid Y(2^{PE})$ )

**Definition** A program is WFPoly (WFpolyExp) if it admits a second order polynomial (polyExp) interpretation such that for each definition $fp = e$, $(\langle fp \rangle) > (\langle e \rangle)$. 
**Definition** A program is WFPoly (WFPolyExp) if it admits a second order polynomial (polyExp) interpretation such that for each definition \( fp = e \), \(|fp| > |e|\).

**Theorem**
- A WFPoly (and WFpolyExp) program is productive
- The set of functions computed by WFPoly programs is exactly OTM-Poly
- The set of functions computed by WFPolyExp programs is exactly BFF
Future works

Further developments:

- Extend the work on OTM
- Study how to combine the different criteria in order to capture a wider set of examples
- Adapt the techniques presented here to study space properties relevant for synchronous stream processing languages, e.g. buffering, memory leak, reachability.