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 ouput 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-intrepretation-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 :

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

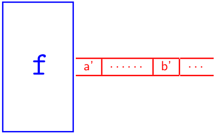
$$1 : 1 : 2 : 3 : 5 : 8 : 13 : \cdots : 46368 : 75025 : \cdots$$

$$0 : 1 : 1 : 0 : 1 : 0 : 0 : 1 : 1 : 0 : 0 : 1 : 0 : 1 : 1 : 0 \cdots$$

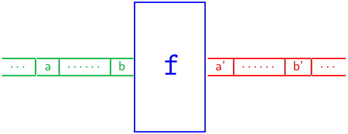
- In the sequel we will consider programs built over numerals and addition and extended by particular function definitions.

Representation of Stream programs

- 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 :

$$\begin{aligned}
\mathtt{repeat} :\ & \mathtt{Nat} \to [\mathtt{Nat}] \\
\mathtt{repeat}\ \mathtt{x} =\ & \mathtt{x} : (\mathtt{repeat}\ \mathtt{x}) \\
\mathtt{fibo} :\ & [\mathtt{Nat}] \\
\mathtt{fibo}\ =\ & 1 : 1 : (\mathtt{sadd}\ \mathtt{fibo}\ (\mathtt{tail}\ \mathtt{fibo}))
\end{aligned}$$

- We restrict the considered programs to a first order fragment of Haskell.
- We use a lazy evaluation semantics $\Downarrow$ without sharing :

$$\begin{aligned}
\mathtt{fibo} \Downarrow\ & 1 : 1 : (\mathtt{sadd}\ \mathtt{fibo}\ (\mathtt{tail}\ \mathtt{fibo})) \\
\mathtt{repeat}\ 3 \Downarrow\ & 3 : (\mathtt{repeat}\ 3)
\end{aligned}$$

Preliminaries on Programs

- We will extensively use the standard function definition :

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

$$\begin{aligned}
&!! :: [\alpha] \to \texttt{Nat} \to \alpha \\
&(\texttt{x} : \texttt{xs})!!0 = \texttt{x} \\
&(\texttt{x} : \texttt{xs})!!(\texttt{y}+1) = \texttt{xs}\ !!\ \texttt{y}
\end{aligned}$$

- "Approximation lemma" : provides finitary methods

$$\texttt{s} = \texttt{r} \iff \forall \texttt{n}\ ((\texttt{take n s}) = (\texttt{take n r}))$$

- We will write $\texttt{e}_{\underline{\texttt{n}}}$ and $\texttt{e} \restriction_{\underline{\texttt{n}}}$ as a short for $\texttt{e}\ !!\ \underline{\texttt{n}}$ and $\texttt{take}\ \underline{\texttt{n}}\ \texttt{e}$.

eval

- We also need a program forcing the evaluation of programs :

$$\begin{aligned}&\texttt{eval} :: \texttt{A} \to \texttt{A}\\ &\texttt{eval} \quad (\mathbf{c}\ \texttt{e}_1\ \cdots\ \texttt{e}_n) \ = \ \hat{\texttt{C}}\ (\texttt{eval}\ \texttt{e}_1)\ \cdots\ (\texttt{eval}\ \texttt{e}_m)\end{aligned}$$

 where $\hat{\texttt{C}}$ is a program representing the strict version of the primitive constructor **c**.

- in particular when it is applied to stream epressions it diverges :

$$\texttt{eval fibo} \Uparrow \qquad \texttt{eval (repeat 3)} \Uparrow$$
$$\texttt{eval fibo} \Downarrow^{\infty} 1:1:2:3:\cdots 75025 \cdots$$
$$\texttt{eval (repeat 3)} \Downarrow^{\infty} 3:3:3\cdots 3\cdots$$
$$\texttt{eval (take 5 fibo)} \Downarrow 1:1:2:3:5$$

- for this reason in what follows we write $\texttt{e} \Downarrow_v \texttt{v}$ for $\texttt{eval e} \Downarrow \texttt{v}$.

Interpretation

A program P admits an interpretation if there is an assignment $(\!|-|\!)$:

- total : $\forall \texttt{t},\ (\!|\texttt{t}|\!) : (\mathbb{R}^+)^n \to \mathbb{R}^+$
- monotonic : $X_i \geq Y_i \Rightarrow (\!|\texttt{t}|\!)(\ldots, X_i, \ldots) \geq (\!|\texttt{t}|\!)(\ldots, Y_i, \ldots)$,

such that for each definition in P of the shape $\texttt{f}\ \texttt{p}_1 \cdots \texttt{p}_n = \texttt{e}$:

$$(\!|\texttt{f}\ \texttt{p}_1 \cdots \texttt{p}_n|\!) \geq (\!|\texttt{e}|\!)$$

Moreover, a program P admits an additive interpretation if $(\!|-|\!)$ is such that for every symbol **c** of arity n :

- $(\!|\mathbf{c}|\!) = 0 \quad$ if $n = 0$
- $(\!|\mathbf{c}|\!)(X_1, \cdots, X_n) = \sum_{i=1}^{n} X_i + \alpha_{\mathbf{c}}$, with $\alpha_{\mathbf{c}} \geq 1$ otherwise

An Example of Interpretation

Consider again the example :

$$\begin{aligned} \texttt{repeat} &: \ \texttt{Nat} \to [\texttt{Nat}] \\ \texttt{repeat x} &= \ \texttt{x} : (\texttt{repeat x}) \end{aligned}$$

the assignment $(\!|-|\!)$ defined as

$$(\!|\texttt{x}|\!) = X \quad (\!|:|\!)(X, Y) = \max(X, Y) \quad (\!|\texttt{repeat}|\!)(X) = X$$

It is total and monotonic and we check that

$$(\!|\texttt{repeat x}|\!) \geq (\!|\texttt{x} : (\texttt{repeat x})|\!)$$

We have :

$$\begin{aligned} (\!|\texttt{repeat x}|\!) &= (\!|\texttt{repeat}|\!)\ (X) = X \\ &= \max(X, X) \ = \ (\!|:|\!)\ (X, X) \\ &= (\!|:|\!)\ (X, ((\!|\texttt{repeat}|\!) X) \\ &= (\!|\texttt{x} : (\texttt{repeat x})|\!) \end{aligned}$$

Properties of Interpretations

- Interpretation respects evaluation
  Given a program P admitting the interpretation $(\!|-|\!)$, for every closed expression e we have
  - if $\mathsf{e} \Downarrow \mathtt{v}$ then $(\!|\mathsf{e}|\!) \geq (\!|\mathtt{v}|\!)$
  - if $\mathsf{e} \Downarrow_v \mathtt{v}$ then $(\!|\mathsf{e}|\!) \geq (\!|\mathtt{v}|\!)$
- Interpretation does not respect infinitary evaluation
  If $\mathtt{fibo} \Downarrow^\infty \mathtt{u} = 1 : 1 : 2 : 3 : 5 : 8 : \cdots$, we do not expect :

$$(\!|\mathtt{fibo}|\!) \geq (\!|\mathtt{u}|\!)$$

- 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 :

$$(\!|\mathsf{e}|\!) \leq G(|\mathsf{e}|)$$

First property - Global Upper Bound

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

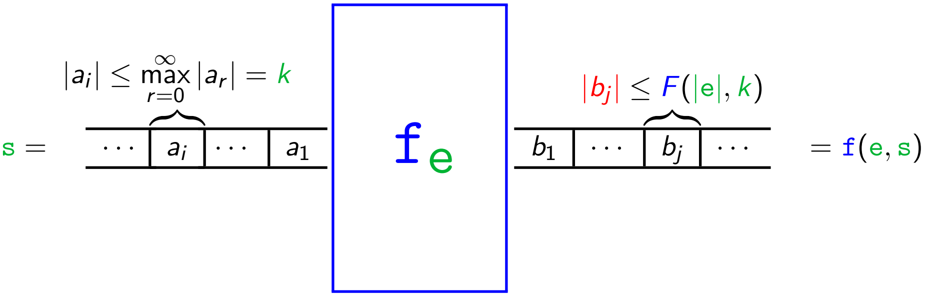
$$\begin{aligned}
&\mathtt{repeat} :: \mathtt{Nat} \to [\mathtt{Nat}] \\
&\mathtt{repeat\ x} = \mathtt{x} : (\mathtt{repeat\ x}) \\
&\mathtt{zip} :: [\alpha] \to [\alpha] \to [\alpha] \\
&\mathtt{zip}\ (\mathtt{x} : \mathtt{xs})\ \mathtt{ys} = \mathtt{x} : (\mathtt{zip\ ys\ xs}) \\
&\mathtt{square} :: [\mathtt{Nat}] \to [\mathtt{Nat}] \\
&\mathtt{square}\ (\mathtt{x} : \mathtt{xs}) = (\mathtt{mul\ x\ x}) : (\mathtt{square\ xs})
\end{aligned}$$

  is bounded by some k.
- For example :

$$\mathtt{square}\ (\mathtt{zip}\ (\mathtt{repeat}\ 5)\ (\mathtt{square}\ (\mathtt{zip}\ (\mathtt{repeat}\ 7)\ (\mathtt{repeat}\ 4))))$$

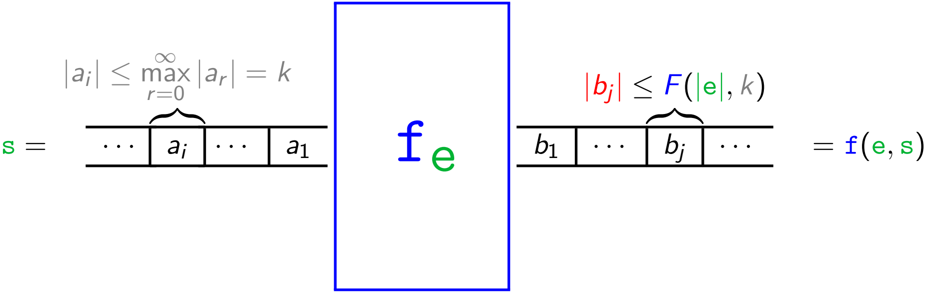
  is bounded by $k = 2401 = 7^4$.

Global Upper Bound - Intuition



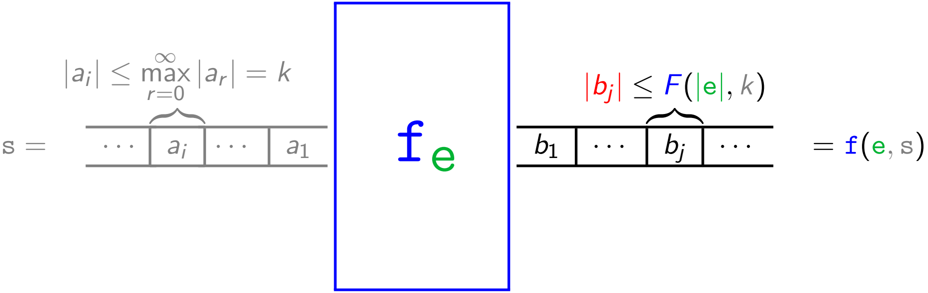
Global Upper Bound - Intuition

Clearly the input stream may have no maximal element :

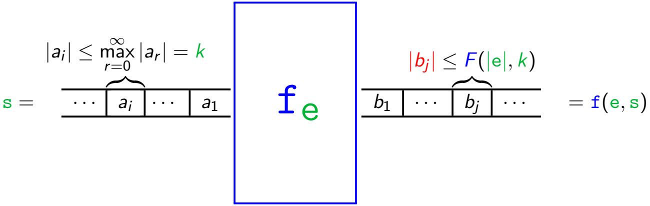


Global Upper Bound - Intuition

Stream constructors are just a particular case :



Global Upper Bound - Formally



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

$$(\mathtt{f}\ \mathtt{s}\ \mathtt{e})_{\underline{\mathtt{n}}} \Downarrow_v \underline{\mathtt{v}}$$

  then

$$F(\max(|\mathtt{s}|, |\mathtt{e}|)) \geq |\underline{\mathtt{v}}|$$

- A 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 $(\!|-|\!)$ such that :

$$(\!|:|\!)(X, Y) = \max(X, Y)$$

- The interpretations of GUB programs well behave with respect to the selection function !!. In particular if

$$\mathtt{e}_{\underline{\mathtt{n}}} \Downarrow_v \mathtt{v}$$

then

$$(\!|\mathtt{e}|\!) \geq (\!|\mathtt{v}|\!)$$

- **Theorem**
If a program is GUB then each stream function $\mathtt{f}$ has a global upper bound F.

GUB Examples

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

$$(\!|\mathtt{repeat}|\!)(X) = X + 1 \quad (\!|\mathtt{zip}|\!)(X, Y) = \max(X, Y)$$

$$(\!|\mathtt{square}|\!)(X) = X^2 \quad (\!|:|\!)(X, Y) = \max(X, Y)$$

- Define $\mathtt{fun}\ \mathtt{x} = \mathtt{square}\ (\mathtt{zip}\ (\mathtt{repeat}\ 9)\ \mathtt{x})$.
  Taking $F(X) = (\!|\mathtt{square}|\!)((\!|\mathtt{zip}|\!)((\!|\mathtt{repeat}\ 9|\!), G(X))$

$$\text{If } (\mathtt{fun}\ \mathtt{e}) \Downarrow_v \underline{\mathtt{m}}$$

$$\text{then } F(|\mathtt{e}|) = (\max(10, G(|\mathtt{e}|)))^2 \geq |\underline{\mathtt{m}}|$$

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 :

$$\begin{aligned}
\mathtt{nats} &:: \mathtt{Nat} \to [\mathtt{Nat}] \\
\mathtt{nats}\ \mathtt{x} &= \mathtt{x} : (\mathtt{nats}\ (\mathtt{x}+1)) \\
\mathtt{sad} &:: [\mathtt{Nat}] \to [\mathtt{Nat}] \to [\mathtt{Nat}] \\
\mathtt{sad}\ (\mathtt{x} : \mathtt{xs})\ (\mathtt{y} : \mathtt{ys}) &= (\mathtt{add}\ \mathtt{x}\ \mathtt{y}) : (\mathtt{sad}\ \mathtt{xs}\ \mathtt{ys})
\end{aligned}$$

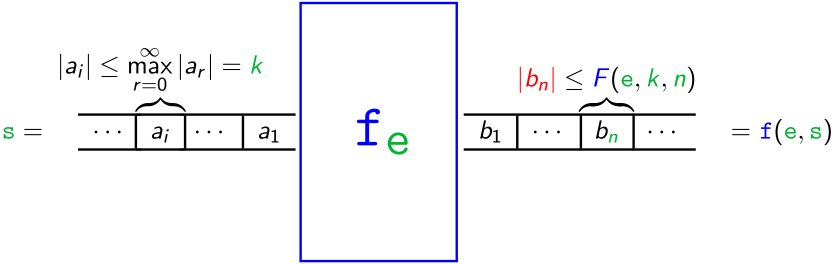
- the size of each element of a stream $\mathtt{s}$ built only using $\mathtt{nats}$ and $\mathtt{sad}$ as

$$\mathtt{sad}\ \mathtt{nats}\ 3\ (\mathtt{sad}\ \mathtt{nats}\ 5\ \mathtt{nats}\ 4)$$

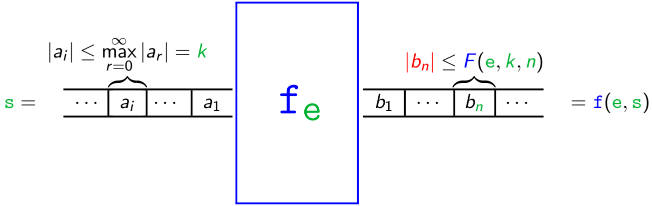
is not globally bounded but is bounded by the function $(3 \times 5) \times (n+1)$ where n is its position in the stream.

Local Upper Bound - Intuition

We consider also dependencies on the local index n as follows :



Local Upper Bound - Formally



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

  $$(\mathtt{f\ s\ e})_{\underline{\mathtt{n}}} \Downarrow_v \underline{\mathtt{v}}$$

  then

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

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

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 :

$$\begin{aligned} \texttt{fib} &:: [\texttt{Nat}] \\ \texttt{fib} &= \underline{\texttt{0}} : (\underline{\texttt{1}} : (\texttt{sad fib (tail fib)})) \end{aligned} \qquad \begin{aligned} \texttt{tail} &:: [\alpha] \\ \texttt{tail x : xs} &= \texttt{xs} \end{aligned}$$

$$\begin{aligned} \texttt{sad} &:: [\texttt{Nat}] \to [\texttt{Nat}] \to [\texttt{Nat}] \\ \texttt{sad (x : xs) (y : ys)} &= (\texttt{add x y}) : (\texttt{sad xs ys}) \end{aligned}$$

Parametrized Interpretations

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

$$\begin{array}{rcll} (\!|\mathtt{hd} : \mathtt{tl}|\!)_{\mathtt{L}} & = & (\!|:|\!)_L((\!|\mathtt{hd}|\!)_{\mathtt{L}}, (\!|\mathtt{tl}|\!)_{\mathtt{L-1}}) & \text{Case } : \\ (\!|\mathtt{t}\ \mathtt{e}_1 \cdots \mathtt{e}_n|\!)_{\mathtt{L}} & = & (\!|\mathtt{t}|\!)_{\mathtt{L}}((\!|\mathtt{e}_1|\!)_{\mathtt{L}}, \ldots, (\!|\mathtt{e}_n|\!)_{\mathtt{L}}) & \mathtt{t} \neq : \end{array}$$

- for each definition $\mathtt{f}\ \overrightarrow{\mathtt{p}} = \mathtt{e}$: $(\!|\mathtt{f}\ \overrightarrow{\mathtt{p}}|\!)_{\mathtt{L}} \geq (\!|\mathtt{e}|\!)_{\mathtt{L}}$

Proposition

i) If $\mathtt{e} \Downarrow \mathtt{v}$ then $(\!|\mathtt{e}|\!)_r \geq (\!|\mathtt{v}|\!)_r$

ii) If $\mathtt{e} \Downarrow_v \mathtt{v}$ then $(\!|\mathtt{e}|\!)_r \geq (\!|\mathtt{v}|\!)_r$

iii) $\exists G,\ \forall \mathtt{e},\ (\!|\mathtt{e}|\!)_r \leq G(|\mathtt{e}|, r)$

LUB Criterion

- A program P satisfies the LUB criterion if it admits an additive and monotonic (also in L) parametrized additive interpretation $(\!|-|\!)_{\mathtt{L}}$ such that :

$$(\!|:|\!)_{\mathtt{L}}(X, Y) = \max(X, Y)$$

- LUB programs has suitable properties wrt !! :

$$\text{If } \mathrm{e}_{\underline{\mathtt{n}}} \Downarrow_{v} \mathtt{v}$$

$$\text{then } (\!|\mathrm{e}|\!)_{n} \geq (\!|\mathtt{v}|\!)_{0}$$

- **Theorem**
  If a program is LUB then each stream function $\mathtt{f}$ 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 :

$$(\!|0|\!)_L = 0 \quad (\!|+1|\!)_L(X) = X + L + 1 \quad (\!|:|\!)_L(X, Y) = \max(X, Y)$$

$$(\!|\mathtt{sad}|\!)_L(X, Y) = (\!|\mathtt{add}|\!)_L(X, Y) = X + Y \quad (\!|\mathtt{tail}|\!)_L(X) = X \quad (\!|\mathtt{fib}|\!)_L = 2^L$$

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

$$(\!|\mathtt{nats}|\!)_L(X) = X + L \quad (\!|\mathtt{sad}|\!)(X, Y) = X + Y$$

$$(\!|:|\!)(X, Y) = \max(X, Y)$$

Consider the definition $\mathtt{dubnat}\ \mathtt{x} = \mathtt{sad}\ (\mathtt{nats}\ 5)\ (\mathtt{nats}\ \mathtt{x})$. Take $F(X) = (\!|\mathtt{sad}|\!)_X((\!|\mathtt{nats}\ 5|\!)_X, (\!|\mathtt{nats}|\!)_X(X))$ and $k = \max(|\mathtt{e}|, |\underline{\mathtt{n}}|)$, if

$$(\mathtt{dubnat}\ \mathtt{e})_{\underline{\mathtt{n}}} \Downarrow \underline{\mathtt{m}}$$

then $F(k) = (\!|\mathtt{sad}|\!)_k((\!|\mathtt{nats}\ 5|\!)_k, (\!|\mathtt{nats}|\!)_k(k)) = 3k + 5 \geq \underline{\mathtt{m}}$

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 :

$$\begin{aligned}
\texttt{zip} &:: [\alpha] \to [\alpha] \to [\alpha \times \alpha] \\
\texttt{zip}\ (\texttt{x} : \texttt{xs})\ \texttt{ys} &= \texttt{x} : (\texttt{zip ys xs}) \\
\texttt{double} &:: [\texttt{Nat}] \to [\texttt{Nat}] \\
\texttt{double}\ (\texttt{x} : \texttt{xs}) &= \texttt{x} : (\texttt{x} : (\texttt{double xs}))
\end{aligned}$$

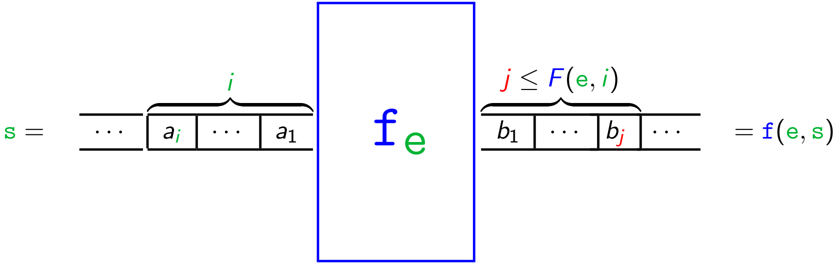
- the number of output written elements of a stream function `f` defined by :

$$\texttt{f x} = \texttt{double}(\texttt{zip}\ (\texttt{double x})\ (\texttt{double x}))$$

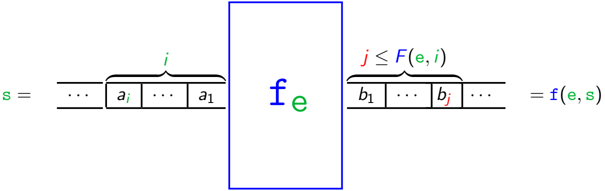
is bounded by a constant k(=8 in this case).

Length-based I/O Upper Bound - Intuition

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



Length-based I/O Upper Bound - Formally



- A function $\mathtt{f} :: [\sigma'] \to \tau \to [\sigma]$ of P has a length based I/O upper bound if there is $F : \mathbb{R}^+ \to \mathbb{R}^+$ such that, for every stream expression $\mathtt{s} \in P$ and for every expression $\mathtt{e} \in P$ if

$$\mathtt{length}(\mathtt{f}\ \mathtt{s}\restriction_{\underline{\mathtt{n}}}\ \mathtt{e}) \Downarrow_v \underline{\mathtt{m}}$$

  then

$$F(\max(|\mathtt{s}|, |\mathtt{e}|, |\underline{\mathtt{n}}|)) \geq |\underline{\mathtt{m}}|$$

- 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 $(\!|-|\!)$ 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 :

$$(\!|\texttt{zip}|\!)(X,Y) = X+Y \quad (\!|\texttt{double}|\!)(X) = 2X \qquad (\!|:|\!)(X,Y) = Y+1$$

Consider the definition

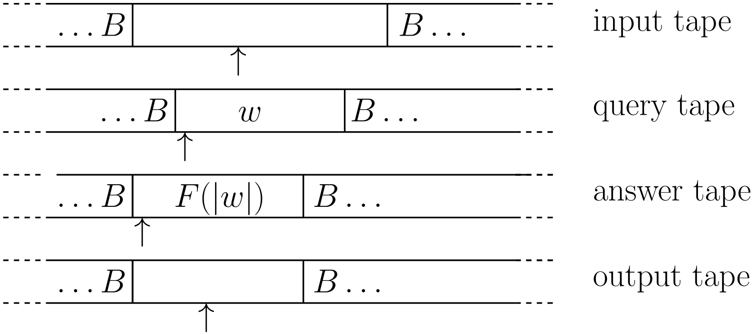
$$\texttt{fun x} = \texttt{double (zip (double x) (double x))}$$

Take $F(X) = (\!|\texttt{double}|\!)((\!|\texttt{zip}|\!)((\!|\texttt{double}|\!)(X), (\!|\texttt{double}|\!)(X)))$, and $k = \max(|\underline{\mathtt{n}}|, |\mathtt{s}|)$, if

$$\texttt{length}(\texttt{funone s}\restriction_{\underline{\mathtt{n}}}) \Downarrow \underline{\mathtt{m}}$$

then
$F(k) = (\!|\texttt{double}|\!)((\!|\texttt{zip}|\!)((\!|\texttt{double}|\!)(k), (\!|\texttt{double}|\!)(X))) = 8 \times k \geq \underline{\mathtt{m}}$

Oracle Turing Machine



Alternative model : OTM' if $F(w)$ is subsituted to $F(|w|)$ on the oracle answer tapes

Oracle Turing Machine

- Costs : Usual step : 1, Oracle step $F(|w|)$
- An OTM $(\mathbb{N} \to \mathbb{N}) \to \mathbb{N} \to \mathbb{N}$ has time complexity T if it halts before $T(|F|, |x|)$ 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 $\mathtt{f} : [Nat] \to Nat \to [Nat]$ is a function $(\!|\mathtt{f}|\!) : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}^2 \to \mathbb{N}$.
- The interpretation of a function $\mathtt{f} : [Nat] \to Nat \to Nat$ is a function $(\!|\mathtt{f}|\!) : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N} \to \mathbb{N}$.
- $(\!|y|\!)(Z) = Y(Z)$
- $(\!|:|\!)(X, Y, Z+1) = 1 + X + Y(Z)$ and $(\!|:|\!)(X, Y, 0) = 1 + X$.

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 $\mathtt{fp} = \mathtt{e}$, $(\!|\mathtt{fp}|\!) > (\!|\mathtt{e}|\!)$.

Characterizations

Definition A program is WFPoly (WFPolyExp) if it admits a second order polynomial (polyExp) interpretation such that for each definition $\mathtt{fp} = \mathtt{e}$, $(\!|\mathtt{fp}|\!) > (\!|\mathtt{e}|\!)$.

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.