Data-Oblivious Stream Productivity

Jörg Endrullis    Clemens Grabmayer    Dimitri Hendriks

Vrije Universiteit – Universiteit Utrecht – Vrije Universiteit

NWO Projects *Infinity* and *ProvCyc*

PAM, CWI

May 7, 2008
Outline

1. Introduction
   Productivity. Previous Approaches.
   First Paper. New Results.

2. Stream Specifications
   Motivating Examples
   Definitions and Classes

3. Data-Oblivious Analysis
   Intuition
   Definition and Properties

4. The Production Calculus

5. Translation into Production Terms

6. Deciding Data-Oblivious Productivity

7. Conclusion
   Summary. Further Plans.
   Tools
Productivity

- ‘Productivity’ first used by Dijkstra (1980).
- Slogan: *For programming with infinite structures, productivity is what termination is for programming with finite structures.*
- Productivity captures the notion of *unlimited progress*, of ‘working’ programs, producing defined values indefinitely.

Related questions:
- When do we accept an infinite object defined in terms of itself?
- When does a finite set of equations *constructively* define a *unique* infinite object?
Productivity of Stream Specifications

- $A^\omega := \{ \sigma \mid \sigma : \mathbb{N} \to A \}$ the set $A^\omega$ of streams over $A$
- ‘:’ is the stream constructor symbol: $a : \sigma$ denotes the result of prepending $a \in A$ to $\sigma \in A^\omega$
- A recursive stream specification
  \[ M = \ldots M \ldots \]
  is productive if the process of continually evaluating $M$ results in an infinite constructor normal form:
  \[ M \xrightarrow{} a_0 : a_1 : a_2 : \ldots \]

- Productivity is undecidable in general (in fact $\Pi^0_2$-complete).
- But for restricted formats computable sufficient conditions or decidability can be obtained.
Examples

Example

\[
\begin{align*}
\text{read}(x : \sigma) &= x : \text{read}(\sigma) \\
\text{fast_read}(x : y : \sigma) &= x : y : \text{fast_read}(\sigma) \\
\text{fives} &= 5 : \text{read}(\text{fives}) \\
\text{fives'} &= 5 : \text{fast_read}(\text{fives'}) \\
\text{zip}_2(x : \sigma, y : \tau) &= x : y : \text{zip}_2(\sigma, \tau) \\
\text{zip}_1(x : \sigma, \tau) &= x : \text{zip}_1(\tau, \sigma) \\
\text{sevens} &= 7 : \text{zip}_2(\text{sevens}, \text{tail}(\text{sevens})) \\
\text{sevens'} &= 7 : \text{zip}_1(\text{sevens'}, \text{tail}(\text{sevens'}))
\end{align*}
\]
Productivity Recognition: Previous Approaches

- **Wadge (1981):** ‘cyclic sum test’ (*limited, computable criterion*).
- **Sijtsma (1989):** mathematical theory of productivity based on ‘production moduli’ (*mathem., not directly computable criteria*).
- **Coquand (1994):** ‘guardedness’ as a syntactic criterion for productivity (*automatable, but restrictive criterion*).
- **Telford and Turner (1997):** extend the notion of guardedness by a method in the flavour of Wadge.
- **Hughes, Pareto, and Sabry (1996):** introduce a type system for proving productivity (*automatable criterion*).
- **Buchholz (2004):** type system for proving productivity, two forms:
  - using unrestricted production moduli (*general, not automatable*);
  - a decidable subsystem with limited moduli (*automable criterion, handles all examples of Telford & Turner*).
Our First Paper: Productivity Decision

Productivity of Stream Definitions (Endrullis, Grabmayer, Hendriks, Isihara, Klop), FCT’07:

- A decision algorithm for productivity on the large and natural class of pure stream spec’s.

- ‘Large class’: The stream functions allowed in pure stream spec’s contain all automatic sequences (Allouche, Shallit).

- Idea behind the decision algorithm:
  - The process of evaluation of a pure stream spec can be modelled by dataflow of pebbles in a finite pebbleflow net.
  - The production of a pebbleflow net associated with a pure stream spec (amount of pebbles the net can produce at its output port) can be calculated by reducing nets to trivial nets.
Main New Results: Productivity Recognition/Decision

- All previous approaches use a ‘quantitative’ analysis that abstracts away from concrete values of stream elements. We formalise this by data-oblivious rewriting.
- We introduce the notion of data-oblivious productivity.
- We identify two syntactical classes of stream spec’s: flat and (general) pure spec’s.
- For flat stream spec’s we obtain a decision method for data-oblivious productivity, yielding a computable, data-obliviously optimal criterion for productivity.
- For pure stream spec’s we obtain a decision method for productivity.
Map of Stream Specifications

\[ \text{data-obliviously recognizable} \]

\[ \text{P} = \text{productive} \]
\[ \text{F} = \text{flat} \]

Our contribution:
\[ \square = \text{automated recognition} \]
\[ \blacksquare = \text{decision} \]
Stream Specification

Example

\[
\begin{align*}
T &\rightarrow 0 : \text{zip}(\text{inv}(T), \text{tail}(T)) \quad \text{stream layer} \\
\text{tail}(x : \sigma) &\rightarrow \sigma \\
\text{zip}(x : \sigma, \tau) &\rightarrow x : \text{zip}(\tau, \sigma) \quad \text{function layer} \\
\text{inv}(x : \sigma) &\rightarrow i(x) : \text{inv}(\sigma) \\
i(0) &\rightarrow 1 \quad i(1) \rightarrow 0 \quad \text{data layer}
\end{align*}
\]

This is a productive stream definition obtaining the Thue-Morse sequence:

\[
T \rightarrow 0 : 1 : 1 : 0 : 1 : 0 : 0 : 1 : 1 : 0 : 1 : 0 : 1 : 1 : 0 : \ldots
\]
Stream Specification

Example

\[
\begin{align*}
J &= 0 : 1 : \text{even}(J) \\
\text{even}(x : \sigma) &\rightarrow x : \text{odd}(\sigma) \\
\text{odd}(x : \sigma) &\rightarrow \text{even}(\sigma)
\end{align*}
\]

This stream definition is not productive: \( J \rightarrow 0 : 1 : 0 : 0 : \text{even}^\omega \)
Motivating Examples

\[ J \rightarrow 0 : 1 : 0 : 0 : \text{even}^\omega \]

\[ J \rightarrow 0 : 1 : \text{even}(J) \]
\[ \text{even}(J) \rightarrow \text{even}(0 : 1 : \text{even}(J)) \]
\[ \rightarrow 0 : \text{odd}(1 : \text{even}(J)) \]
\[ \rightarrow 0 : \text{even}(\text{even}(J)) \]

\[ \text{even}^2(J) \equiv \text{even}(\text{even}(J)) \rightarrow \text{even}(0 : \text{even}(\text{even}(J)))) \]
\[ \rightarrow 0 : \text{odd}(\text{even}^2(J)) \]

\[ \text{odd}(\text{even}^2(J)) \rightarrow \text{odd}(0 : \text{odd}(\text{even}^2(J)))) \]
\[ \rightarrow \text{even}(\text{odd}(\text{even}^2(J))) \]

\[ \text{odd}(\text{even}^2(J)) \rightarrow \text{even}(\text{odd}(\text{even}^2(J))) \]
\[ \rightarrow \text{even}^2(\text{odd}(\text{even}^2(J))) \]
\[ \rightarrow \ldots \rightarrow \text{even}^n(\text{odd}(\text{even}^2(J))) \rightarrow \ldots \]
\[ \rightarrow \text{even}^\omega \]

Hence: \[ J \rightarrow 0 : 1 : 0 : 0 : \text{even}^\omega. \]
Stream TRS

A stream TRS is a

- finite \( \{S, D\}\)-sorted, orthogonal, constructor TRS \(\langle \Sigma, R \rangle\), with
- signature partition \(\Sigma = \Sigma_S \cup \Sigma_D\) into stream symbols and data symbols, and

For the definition of stream spec’s we also assume:

- stream signature partition \(\Sigma_S = \{\cdot\} \cup \Sigma_{str} \cup \Sigma_{fun}\), where
  - ‘\(\cdot\)’ the stream constructor symbol,
  - \(\Sigma_{str}\) a set of stream constant symbols having only data arguments;
  - \(\Sigma_{fun}\) a set of stream function symbols with usually at least one stream argument.
Stream Specification

Definition

Let \( T = \langle \Sigma, R \rangle \) a stream TRS with part’s \( \Sigma = \Sigma_{str} \uplus \Sigma_{fun} \uplus \{ : \} \uplus \Sigma_D \) and \( R = R_{str} \uplus R_{fun} \uplus R_D \). \( T \) is a **stream specification** if:

- \( R_{str} \) stream layer
- \( R_{fun} \) function layer
- \( R_D \) data layer

\[
\begin{align*}
R_{str} & \quad \text{stream layer} \\
R_{fun} & \quad \text{function layer} \\
R_D & \quad \text{data layer}
\end{align*}
\]

\( \{ : \} \)

\( \Sigma_{str} \) a set of constant symbols, contains \( M_0 \), the root of \( T \).

\( R_{str} \) is the set of defining rules \( \rho_M : M \rightarrow s \) for every \( M \in \Sigma_{str} \).

- The **data-layer** \( T_d = \langle \Sigma_D, R_D \rangle \) is a terminating \( D \)-sorted TRS.
- The underlying **stream function specification**
  \( T_{fun} = \langle \Sigma_{fun} \uplus \{ : \} \uplus \Sigma_D, R_{fun} \uplus R_D \rangle \) is a TRS.
- \( T \) is exhaustive for the defined symbols in \( \Sigma \).
Stream Specification (Layered setup)

Remark

- every layer may use symbols from a lower layer, not vice versa
- isolated data-layer:
  - data-layer symbols are stream independent, excluding stream-dependent functions like:
    \[ \text{head}(x : \sigma) \rightarrow x \]
  - by exhaustivity for \( \Sigma_D \) and strong normalization of \( T_d \), closed data terms rewrite to constructor normal forms
Stream Specification (layered setup)

Remark

- **stream layer:**
  - only sortedness restrictions are imposed on how the rules in the stream layer make use of the symbols in the other layers.

- **separate function layer:**
  - we are interested in **managable** stream functions that define **well-defined streams** or finite **stream prefixes**.
  
  The rule
  
  \[ f(\sigma) \rightarrow 0 : \text{head}(\text{tail}(f(\sigma))) : f(\sigma) \quad \text{(excluded!)} \]

  defines an operation on streams that produces an output stream with undefined odd elements.
Production of a term. Productivity of a stream spec.

Let \( \overline{\mathbb{N}} := \mathbb{N} \cup \{\infty\} \) the coinductive natural numbers.

**Definition**

Let \( \mathcal{T} = \langle \Sigma, R \rangle \) a stream definition.

- The production \( \Pi_{\mathcal{T}}(t) \) of a stream term \( t \in \text{Ter}(\Sigma) \):
  \[
  \Pi_{\mathcal{T}}(t) := \sup \{ n \in \mathbb{N} \mid t \rightarrow u_1 : \ldots : u_n : t' \} \in \overline{\mathbb{N}}.
  \]

- \( \mathcal{T} \) is called productive if \( \Pi_{\mathcal{T}}(M_0) = \infty \).

**Proposition**

A stream definition \( \mathcal{T} \) is productive if and only if

\[
M_0 \rightarrow u_1 : u_2 : u_3 : u_4 : \ldots
\]
Stream Specifications (Properties I)

A stream spec $\mathcal{T}$ is called

- **flat**: all rules in the function layer $R_{\text{fun}}$ of $\mathcal{T}$ are flat: no nested occurrences of stream function symbols on the right-hand side.

  Excludes a rule: $e(x : \sigma) \rightarrow x : e(e(\sigma))$.

- **pure**: $\mathcal{T}$ is flat, and for every symbol $f \in \Sigma_{\text{fun}}$ the defining rules of $f$ in $R_{\text{fun}}$ have the same consumption/production behaviour: they coincide (mod. renaming of variables) if all outermost data-subterms are replaced by $\bullet$.

  This excludes defining rules:
  
  $f(0 : x : \sigma) \rightarrow x : x : f(0 : \sigma)$
  
  $f(1 : x : \sigma) \rightarrow x : f(0 : \sigma)$.
Stream Specification (flat, non-pure)

Example

\[
\begin{align*}
Q & \rightarrow a : R \\
R & \rightarrow b : c : f(R) \\
\hspace{1cm} f(a : \sigma) & \rightarrow a : b : c : f(\sigma) \\
\hspace{1cm} f(b : \sigma) & \rightarrow a : c : f(\sigma) \\
\hspace{1cm} f(c : \sigma) & \rightarrow b : f(\sigma) \\
\end{align*}
\]

\[stream\ layer\]

\[function\ layer\]

\[data\ layer\]

... is \textbf{productive} and specifies the \textbf{ternary Thue-Morse sequence}:

\[Q \rightarrow a : b : c : a : c : b : a : b : c : b : a : c : \ldots\]
Stream Specification (flat, pure)

Example

\[ Q \rightarrow \text{diff}(M) \]
\[ M \rightarrow 0 : \text{zip}(\text{inv}(M), \text{tail}(M)) \]

**stream layer**

\[ \text{zip}(x : \sigma, \tau) \rightarrow x : \text{zip}(\tau, \sigma) \]
\[ \text{inv}(x : \sigma) \rightarrow i(x) : \text{inv}(\sigma) \]
\[ \text{tail}(x : \sigma) \rightarrow \sigma \]
\[ \text{diff}(x : y : \sigma) \rightarrow x(x, y) : \text{diff}(y : \sigma) \]

**function layer**

\[ i(0) \rightarrow 1 \]
\[ i(1) \rightarrow 0 \]
\[ x(0, 0) \rightarrow b \]
\[ x(0, 1) \rightarrow a \]
\[ x(1, 0) \rightarrow c \]
\[ x(1, 1) \rightarrow b \]

**data layer**

... is productive and also specifies the ternary Thue-Morse sequence.
Stream Specification (flat, non-pure)

Example (Hamming numbers)

| Definition | Expression |
|------------|------------|
| \( H \rightarrow 1 \) | merge(times(H, 2), merge(times(H, 3), times(H, 5))) |
| times(x : \( \sigma \), y) | m(x, y) : times(\( \sigma \), y) |
| merge(x : \( \sigma \), y : \( \tau \)) | aux(\( \sigma \), \( \tau \), x, y, cmp(x, y)) |
| aux(\( \sigma \), \( \tau \), x, y, lt) | x : merge(\( \sigma \), y : \( \tau \)) |
| aux(\( \sigma \), \( \tau \), x, y, eq) | x : merge(\( \sigma \), \( \tau \)) |
| aux(\( \sigma \), \( \tau \), x, y, gt) | y : merge(x : \( \sigma \), \( \tau \)) |
| cmp(0, 0) | eq |
| cmp(0, s(y)) | lt |
| cmp(s(x), 0) | gt |
| cmp(s(x), s(y)) | cmp(x, y) |
| m(0, y) | → 0 |
| a(0, y) | → y |
| a(s(x), y) | → s(a(x, y)) |
| m(s(x), y) | → a(y, m(x, y)) |
Stream Specifications (Properties II)

A stream spec $\mathcal{T}$ is called

- **friendly-nesting**: all rules in the function layer $R_{fun}$ are flat, or contained in a subset $\tilde{R} \subset R_{fun}$ of friendly-nesting rules:
  - every $\rho \in \tilde{R}$
    - consumes in each stream argument at most one element,
    - produces at least one, and
    - all defining rules of stream functions occurring on the right-hand side of $\rho$ are again in $\tilde{R}$.

**Example**

\[
\begin{align*}
  f(x : \sigma, \tau) & \rightarrow x : x : g(f(\sigma, x : \tau)) \\
g(x : \sigma) & \rightarrow x : g(x : f(\sigma, \sigma))
\end{align*}
\]
**Stream Specification (friendly-nesting)**

### Example

| Stream Function | Definition |
|-----------------|------------|
| nats → 0 : ×(ones, ones) | stream layer |
| ones → s(0) : ones | |
| ×(x : σ, y : τ) → m(x, y) : add(times(τ, x), ×(σ, y : τ)) | |
| times(x : σ, y) → m(x, y) : times(σ, y) | function layer |
| add(x : σ, y : τ) → a(x, y) : add(σ, τ) | |
| a(x, 0) → x | data layer |
| a(x, s(y)) → s(a(x, y)) | |
| m(x, 0) → 0 | |
| m(x, s(y)) → a(m(x, y), x) | |

× defines the convolution product stream operation \(\langle σ, τ \rangle \mapsto σ \times τ\):

\[
(σ \times τ)(i) = \sum_{j=0}^{i} σ(j) \cdot τ(i − j)
\]  
(for all \(i \in \mathbb{N}\))
Map of Stream Specifications

Data-obliviously recognizable

\( P = \text{productive} \)

\( F = \text{flat} \)

Our contribution:

- \( \text{automated recognition} \)
- \( \text{decision} \)

Endrullis, Grabmayer, Hendriks

PAM, CWI, May 7, 2008
Data-Oblivious Analysis

\[
\begin{align*}
T &\rightarrow f(0:1:T) & \text{stream layer} \\
(\rho_{f0}) &:\ f(0:x:\sigma) \rightarrow 0:1:f(\sigma) \\
(\rho_{f1}) &:\ f(1:x:\sigma) \rightarrow x:f(\sigma) & \text{function layer} \\
\end{align*}
\]

This specification is productive:

\[
T \rightarrow 0:1:f(T) \rightarrow 0:1:0:1:f(f(T)) \rightarrow \ldots \rightarrow 0:1:0:1: \ldots ,
\]

but, disregarding the identity of data, the rewrite sequence:

\[
T \rightarrow f(\bullet:\bullet:T) \rightarrow^{\rho_{f1}} \bullet:f(T) \rightarrow \ldots \rightarrow \bullet:f(\bullet:f(\bullet:f(\ldots))) .
\]

is possible. Hence the specification is not data-obliviously productive.
What is a data-oblivious analysis of productivity?

- ‘quantitative reasoning’
- knowledge about concrete values of data elements is ignored
- abstract from the concrete data values

Example

\[ f(0 : x : \sigma) \rightarrow x : x : f(0 : \sigma) \quad f(\bullet : \bullet : \sigma) \rightarrow \bullet : \bullet : f(\bullet : \sigma) \]
\[ f(1 : x : \sigma) \rightarrow x : f(0 : \sigma) \quad f(\bullet : \bullet : \sigma) \rightarrow \bullet : f(\bullet : \sigma) \]

Taking into account that 0 is supplied to the recursive call:

- \( n \mapsto 2n \div 3 \) is the tight lower bound on the production relation

However, a data-oblivious analysis ignores this information:

- \( n \mapsto n \div 1 \) is the data-oblivious lower bound
We formalise data-oblivious term rewriting as two-player game:

- rewrite player $R$ performs the usual term rewriting
- data-exchange player $G$ arbitrarily exchanges data elements

Example ($f(0 : x : \sigma) \rightarrow x : x : f(0 : \sigma), f(1 : x : \sigma) \rightarrow x : f(0 : \sigma)$)

Data-oblivious rewriting of the term $f(0 : 1 : 0 : \sigma)$:

```
\begin{align*}
f(0 : 1 : 0 : \sigma) \xrightarrow{G} & f(1 : 0 : 0 : \sigma) \\
0 : f(0 : 0 : \sigma) \xrightarrow{R} & 0 : f(1 : 1 : \sigma) \\
0 : 1 : f(0 : \sigma) \xrightarrow{G} & 0 : 1 : f(0 : \sigma)
\end{align*}
```

Definition (data-oblivious lower (upper) bound on the production)

... of a term $s$ is the infimum (supremum) of the production of $s$ with respect to all possible strategies for the data-exchange player $G$. 
It is sufficient to use ‘data-value-invariant‘ strategies for \( \mathcal{G} \):

- for \( \mathcal{G} \) we can abstract from the data elements in favour of \( \bullet \)

**Definition (The data abstraction \([s]\) of a term \(s\))**

... is obtained from \(s\) by replacing all data terms in \(s\) with \(\bullet\).

**Example**

\[
\begin{align*}
&\quad f(\bullet : \bullet : \bullet : \sigma) \quad \overset{\mathcal{G}}{\longrightarrow} \quad f(1 : 0 : 0 : \sigma) \\
&\quad \bullet : f(\bullet : \bullet : \sigma) \quad \overset{\mathcal{G}}{\longrightarrow} \quad 0 : f(1 : 1 : \sigma) \\
&\quad \bullet : \bullet : f(\bullet : \sigma) \quad \overset{\mathcal{G}}{\longrightarrow} \quad 0 : 1 : f(0 : \sigma)
\end{align*}
\]

**Definition (The data-guess function \(\mathcal{G}\))**

... instantiates all \(\bullet\) by closed data terms in constructor normal form.
Definition

The data-oblivious production range of a term \( s \in Ter(\Sigma)_S \) is:

\[
\overline{do}_T(s) := \{ \Pi_G([s]) \mid G \text{ a data-guess function on } T \}.
\]

The data-oblivious lower and upper bound on the production of \( s \):

\[
\underline{do}_T(s) := \inf(\overline{do}_T(s))
\]
\[
\overline{do}_T(s) := \sup(\overline{do}_T(s))
\]

Definition

An stream specification \( T \) is called

- data-obliviously productive if \( \overline{do}_T(M_0) = \infty \)
- data-obliviously non-productive if \( \overline{do}_T(M_0) < \infty \)
Data-Oblivious Productivity versus Productivity

**Proposition**

Let $\mathcal{T} = \langle \Sigma, R \rangle$ be a stream specification.

- For all stream terms $s \in \text{Ter}(\Sigma)_S$, we have
  
  $\overline{do}_T(s) \leq \Pi_T(s) \leq \overline{do}_T(s)$.

*Hence:*

- data-oblivious productivity implies productivity;
- data-oblivious non-productivity implies non-productivity.
Periodically Increasing Functions

**Definition**

Let \( f : \mathbb{N} \rightarrow \mathbb{N} \).

- \( f \) is **eventually periodic** if \( \langle f(0), f(1), f(2), \ldots \rangle \) is eventually periodic.
- \( f \) is **periodically increasing** if it is non-decreasing, and its derivative \( f' : \mathbb{N} \rightarrow \mathbb{N} \) with \( f'(n) = f(n+1) - f(n) \) is eventually periodic.

**Representation:** pairs \( \langle \alpha, \beta \rangle \in \mathcal{I} \) with \( \alpha \in \{-, +\}^* \), \( \beta \in \{-, +\}^+ \) where + stands for output, − for input.

**Production function** \( \pi_{\langle \alpha, \beta \rangle} \) for \( \langle \alpha, \beta \rangle \in \mathcal{I} \): if \( s = \alpha\beta\beta\ldots \) then

\[
\pi_{\langle \alpha, \beta \rangle}(n) := \begin{cases} 
\text{number of “+” from left until the } (n+1)\text{-th “−” in } l
\end{cases}
\]

\[\infty \ldots \text{ if there are less than } n+1 \text{ symbols in } l\]

**Abbreviation:** \( \alpha\beta \) for \( \langle \alpha, \beta \rangle \).

Example: identity function is represented by \( --+ \).
Production Terms

Definition

For $\mathcal{V}$ a set of recursion variables, the set $\mathcal{P}$ of production terms is generated by:

$$ p ::= k \mid x \mid \sigma(p) \mid \mu x . p \mid \text{min}(p, p) $$

where $x \in \mathcal{V}$, $\sigma \in \mathcal{I}$, and $k$ is a numeral for $k \in \mathbb{N}$.

The production $\prod(p) \in \mathbb{N}$ of a closed production term $p \in \mathcal{P}$ is defined by induction on the term structure, interpreting:

- $\mu$ as the least fixed point operator,
- $\sigma$ as $\pi_\sigma$,
- $k$ as $k$, and
- $\text{min}$ as $\text{min}$.

$r$-ary Gates: production term contexts $\text{min}_r(\sigma_1(\Box_1), \ldots, \sigma_r(\Box_r))$. 
Definition

The reduction relation $\rightarrow_R$ on production terms is defined as the compatible closure of:

- $\sigma_1(\sigma_2(p)) \rightarrow \sigma_1 \circ \sigma_2(p)$
- $\sigma(\min(p_1, p_2)) \rightarrow \min(\sigma(p_1), \sigma(p_2))$
- $\mu x.\min(p_1, p_2) \rightarrow \min(\mu x.p_1, \mu x.p_2)$
- $\mu x.p \rightarrow p \quad \text{if } x \notin \text{FV}(p)$
- $\mu x.\sigma(x) \rightarrow \text{fix}(\sigma)$
- $\mu x.x \rightarrow 0$
- $\sigma(k) \rightarrow \pi_{\sigma}(k)$
- $\min(k_1, k_2) \rightarrow \min(k_1, k_2)$
Reduction $\rightarrow_R$

Properties of $\rightarrow_R$:
- production preserving;
- confluent and terminating;
- normal forms are numerals.

**Theorem**

For all $p \in \mathcal{P}$:

$$\Pi(p) = k,$$

where $k$ is the uniquely determined $\rightarrow_R$-normal form of $p$. 
Translation into Production Terms

We use a translation that maps every flat stream spec $\mathcal{T}$ with root $M_0$ to a production term $[M_0]$ such that

$$do_{\mathcal{T}}(M_0) = \Pi([M_0]).$$

- **function layer translation**: Obtain a family $\{[f]\}_{f \in \Sigma_{\text{fun}}}$ of gates such that, for every $f \in \Sigma_{\text{fun}}$, the gate $\gamma_f$ represents the data-oblivious lower bound of $f$ in $\mathcal{T}$.

  (Involves solving an originally infinite ‘io-term specification’.)

- **stream layer translation**: Using the family $f \in \Sigma_{\text{fun}}$ of gates, obtain a production term $[M_0]^\mathcal{F}$ such that $\Pi([M_0]^\mathcal{F}) = do_{\mathcal{T}}(M_0)$.

  (Involves expanding the stream layer rules step by step and a finite loop-checking procedure.)
Algorithm $\textbf{DOP}$: Deciding Data-Oblivious Productivity

1. Take as input: a flat stream specification $\mathcal{T} = \langle \Sigma, R \rangle$.

2. Compute the translation of stream function symbols $f$ into gates $[f]$, yielding a family $\mathcal{F} := \{[f]\}_{f \in \Sigma_{\text{fun}}}$ of gates.

3. Construct the production term $[M_0]^\mathcal{F}$ of the root $M_0$ of $\mathcal{T}$ with respect to the family of gates $\mathcal{F}$.

4. Compute the production $k$ of $[M_0]^\mathcal{F}$ using the reduction rel. $\rightarrow_R$.

5. Give the following output:
   - If $k = \infty$: "$\mathcal{T}$ is data-obliviously productive";
   - else $k \in \mathbb{N}$: "$\mathcal{T}$ is not data-obliviously productive".
Deciding D-O Productivity. Recognising Productivity.

Theorem

Data-oblivious productivity of flat stream specifications is decidable. I.p., the algorithm $\text{DOP}$ decides data-oblivious productivity of flat stream specifications.

Since data-oblivious productivity implies productivity, we get a computable, data-obliviously optimal criterion for productivity:

Corollary

A flat stream specification $\mathcal{T}$ is productive if the algorithm $\text{DOP}$ recognizes $\mathcal{T}$ as data-obliviously productive.
Recognising and Deciding Productivity.

For pure stream spec’s productivity coincides with data-oblivious productivity. Hence DOP gives rise to a decision algorithm.

**Theorem**

*Productivity is decidable for pure stream spec’s.*

Furthermore, a variant of DOP can be used as a:

- computable criterion of productivity for friendly-nesting stream spec’s.
Map of Stream Specifications

- $P = \text{productive}
- F = \text{flat}$

Our contribution:
- $\square = \text{automated recognition}$
- $\blacksquare = \text{decision}$
Summary

- **Previous Approaches**: sufficient conditions for productivity, not automatable or only for a limited subclass

- **FCT’07-Paper**: decision algorithm for productivity of pure stream spec’s

- **New Results**:
  1. a computable, data-obliviously optimal, sufficient condition for productivity of flat stream spec’s;
  2. a decision method for productivity on pure stream spec’s with duplication/additional supply in stream arg’s (extension of FCT);
  3. an extension of 1 to stream spec’s with friendly nesting, disregarding data-oblivious optimality;
  4. a tool automating 1, 2 and 3 available at: [http://infinity.few.vu.nl/productivity](http://infinity.few.vu.nl/productivity).

Endrullis, Grabmayer, Hendriks

Data-Oblivious Stream Productivity
Present and Future Work

- A precise **complexity analysis** of our algorithms.

- **Data-aware methods** for recognising productivity.

- A **refined pebbleflow semantics** that accounts for the delay of evaluation of stream elements as made possible by lazy evaluation strategies. Think of Sijtsma’s example: $S \rightarrow 0 : \text{head}(\text{tail}^2(S)) : S$.

- A theory of **reducibility between streams**.

- Can our results be used to obtained general results clarifying under which conditions **term graph rewriting** can be viewed as a semantics for infinitary rewriting?
Our Papers and Tools.

Please visit http://infinity.few.vu.nl/productivity to find:

- Endrullis, Grabmayer, Hendriks, Isihara, Klop: *Productivity of Stream Definitions*, Proceedings of FCT 2007, LNCS 4637, pages 274–287, 2007;

- Endrullis, Grabmayer, Hendriks, Isihara, Klop: *Productivity of Stream Definitions*, journal submission;

- Endrullis, Grabmayer, Hendriks: *Data-Oblivious Stream Productivity*, extended abstract;

- access to our tools:
  - Endrullis: tool implementing the decision algorithm for data-oblivious productivity;
  - Isihara: pebbleflow visualization tool.
Thanks for your attention!