Data-Oblivious Stream Productivity

Jörg Endrullis Clemens Grabmayer Dimitri Hendriks

Vrije Universiteit Amsterdam – Universiteit Utrecht – Vrije Universiteit Amsterdam

The Netherlands

LPAR 2008, Doha, Qatar

23–27 Nov

Overview

- streams, specifying streams
- productivity
- stream specifications (formally)
- recognizing productivity: literature
- our contribution:
  - data-oblivious rewriting and d-o productivity
  - new stream specification formats
  - results: automated productivity recognition/decision
- tool demo

Specifying Streams

- a stream over A is an infinite sequence of elements from A.
- using the stream constructor symbol `":"`, we write streams as:

$$a_0 : a_1 : a_2 : \ldots .$$

Example (Thue–Morse stream)

$\mathsf{T} \to 0 : 1 : \mathsf{f}(\mathsf{tail}(\mathsf{T}))$		*stream constant*
$\mathsf{f}(x : \sigma) \to x : \mathsf{i}(x) : \mathsf{f}(\sigma)$ $\mathsf{tail}(x : \sigma) \to \sigma$		*stream functions*
$\mathsf{i}(0) \to 1$	$\mathsf{i}(1) \to 0$	*data functions*

one finds: T

Specifying Streams

- a stream over A is an infinite sequence of elements from A.
- using the stream constructor symbol `":"`, we write streams as:

$$a_0 : a_1 : a_2 : \ldots .$$

Example (Thue–Morse stream)

$\mathsf{T} \to 0 : 1 : \mathsf{f}(\mathsf{tail}(\mathsf{T}))$	*stream constant*
$\mathsf{f}(x : \sigma) \to x : \mathsf{i}(x) : \mathsf{f}(\sigma)$ $\mathsf{tail}(x : \sigma) \to \sigma$	*stream functions*
$\mathsf{i}(0) \to 1 \qquad \mathsf{i}(1) \to 0$	*data functions*

one finds: $\mathsf{T} \twoheadrightarrow 0 : 1 : \mathsf{f}(\mathsf{tail}(\underline{\mathsf{T}}))$

Specifying Streams

- a stream over A is an infinite sequence of elements from A.
- using the stream constructor symbol `":"`, we write streams as:

$$a_0 : a_1 : a_2 : \ldots .$$

Example (Thue–Morse stream)

$\mathsf{T} \to 0 : 1 : \mathsf{f}(\mathsf{tail}(\mathsf{T}))$	*stream constant*
$\mathsf{f}(x : \sigma) \to x : \mathsf{i}(x) : \mathsf{f}(\sigma)$ $\mathsf{tail}(x : \sigma) \to \sigma$	*stream functions*
$\mathsf{i}(0) \to 1 \qquad \mathsf{i}(1) \to 0$	*data functions*

one finds: $\mathsf{T} \twoheadrightarrow 0 : 1 : \mathsf{f}(\underline{\mathsf{tail}}(0 \underline{:} 1 : \mathsf{f}(\mathsf{tail}(\mathsf{T}))))$

Specifying Streams

- a stream over A is an infinite sequence of elements from A.
- using the stream constructor symbol `":"`, we write streams as:

$$\mathsf{a}_0 : \mathsf{a}_1 : \mathsf{a}_2 : \ldots .$$

Example (Thue–Morse stream)

$\mathsf{T} \to 0 : 1 : \mathsf{f}(\mathsf{tail}(\mathsf{T}))$	*stream constant*
$\mathsf{f}(x : \sigma) \to x : \mathsf{i}(x) : \mathsf{f}(\sigma)$ $\mathsf{tail}(x : \sigma) \to \sigma$	*stream functions*
$\mathsf{i}(0) \to 1 \qquad \mathsf{i}(1) \to 0$	*data functions*

one finds: $\mathsf{T} \twoheadrightarrow 0 : 1 : \underline{\mathsf{f}}(1 \underline{:} \mathsf{f}(\mathsf{tail}(\mathsf{T})))$

Specifying Streams

- a stream over A is an infinite sequence of elements from A.
- using the stream constructor symbol ":", we write streams as:

$$\mathsf{a}_0 : \mathsf{a}_1 : \mathsf{a}_2 : \ldots .$$

Example (Thue–Morse stream)

$\mathsf{T} \to 0 : 1 : \mathsf{f}(\mathsf{tail}(\mathsf{T}))$	*stream constant*
$\mathsf{f}(x : \sigma) \to x : \mathsf{i}(x) : \mathsf{f}(\sigma)$ $\mathsf{tail}(x : \sigma) \to \sigma$	*stream functions*
$\mathsf{i}(0) \to 1 \qquad \mathsf{i}(1) \to 0$	*data functions*

one finds: $\mathsf{T} \twoheadrightarrow 0 : 1 : 1 : \underline{\mathsf{i}(\underline{1})} : \mathsf{f}(\mathsf{f}(\mathsf{tail}(\mathsf{T})))$

Specifying Streams

- a stream over A is an infinite sequence of elements from A.
- using the stream constructor symbol `":"`, we write streams as:

$$a_0 : a_1 : a_2 : \ldots .$$

Example (Thue–Morse stream)

$\mathsf{T} \to 0 : 1 : \mathsf{f}(\mathsf{tail}(\mathsf{T}))$	*stream constant*
$\mathsf{f}(x : \sigma) \to x : \mathsf{i}(x) : \mathsf{f}(\sigma)$ $\mathsf{tail}(x : \sigma) \to \sigma$	*stream functions*
$\mathsf{i}(0) \to 1 \qquad \mathsf{i}(1) \to 0$	*data functions*

one finds: $\mathsf{T} \twoheadrightarrow 0 : 1 : 1 : 0 : \mathsf{f}(\mathsf{f}(\mathsf{tail}(\mathsf{T})))$

Specifying Streams

- a stream over A is an infinite sequence of elements from A.
- using the stream constructor symbol ":", we write streams as:

$$a_0 : a_1 : a_2 : \ldots .$$

Example (Thue–Morse stream)

$\mathsf{T} \to 0 : 1 : \mathsf{f}(\mathsf{tail}(\mathsf{T}))$	*stream constant*
$\mathsf{f}(x : \sigma) \to x : \mathsf{i}(x) : \mathsf{f}(\sigma)$ $\mathsf{tail}(x : \sigma) \to \sigma$	*stream functions*
$\mathsf{i}(0) \to 1 \qquad \mathsf{i}(1) \to 0$	*data functions*

one finds: $\mathsf{T} \twoheadrightarrow 0 : 1 : 1 : 0 : 1 : 0 : 0 : 1 : 1 : 0 : 0 : 1 : \mathsf{f}(\mathsf{f}(\mathsf{f}(\mathsf{tail}(\mathsf{T}))))$

Specifying Streams

- a stream over A is an infinite sequence of elements from A.
- using the stream constructor symbol ":", we write streams as:

$$a_0 : a_1 : a_2 : \ldots .$$

Example (Thue–Morse stream)

$\mathsf{T} \to 0 : 1 : \mathsf{f}(\mathsf{tail}(\mathsf{T}))$		*stream constant*
$\mathsf{f}(x : \sigma) \to x : \mathsf{i}(x) : \mathsf{f}(\sigma)$ $\mathsf{tail}(x : \sigma) \to \sigma$		*stream functions*
$\mathsf{i}(0) \to 1$	$\mathsf{i}(1) \to 0$	*data functions*

one finds: $\mathsf{T} \twoheadrightarrow 0 : 1 : 1 : 0 : 1 : 0 : 0 : 1 : 1 : 0 : 0 : 1 : 0 : 1 : 1 : 0 : \ldots$

Productivity

- captures: unlimited production of well-defined values for infinite data types
- corresponds to: termination in a well-defined result for finite data types

Definition

a stream specification is productive if lazy/fair evaluation of its root M_0 results in an infinite constructor normal form:

$$M_0 \twoheadrightarrow a_0 : a_1 : a_2 : \ldots \, .$$

- highly undecidable (basic version Π^0_2-complete, others worse)
- but for restricted formats computable sufficient conditions or decidability can be obtained

Stream Specification

Example

$$
\begin{array}{ll}
\hline
\mathsf{T} \to 0 : 1 : \mathsf{f}(\mathsf{tail}(\mathsf{T})) & \\
\mathsf{f}(0 : \sigma) \to 0 : 1 : \mathsf{f}(\sigma) & \textit{stream layer} \\
\mathsf{f}(1 : \sigma) \to 1 : 0 : \mathsf{f}(\sigma) & \\
\mathsf{tail}(x : \sigma) \to \sigma & \\
\hline
& \textit{data layer} \\
\hline
\end{array}
$$

is a productive stream definition of the Thue–Morse stream:

$$\mathsf{T} \twoheadrightarrow 0 : 1 : 1 : 0 : 1 : 0 : 0 : 1 : 1 : 0 : 0 : 1 : 0 : 1 : 1 : 0 : \ldots$$

Stream Specification

Example

$$\mathsf{J} \to 0 : 1 : \mathsf{even}(\mathsf{J})$$

$$\mathsf{even}(x : \sigma) \to x : \mathsf{odd}(\sigma) \qquad \textit{stream layer}$$

$$\mathsf{odd}(x : \sigma) \to \mathsf{even}(\sigma)$$

data layer

is not productive:

$$\mathsf{J} \twoheadrightarrow 0 : 1 : 0 : 0 : \mathsf{even}(\mathsf{even}(\ldots))$$

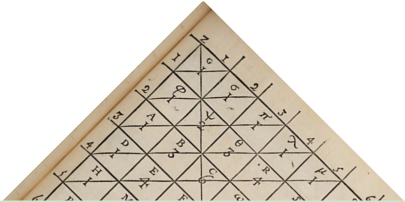
We formalize stream specifications as:

- $\{S, D\}$-sorted TRSs $\mathcal{T} = \langle \Sigma, R \rangle$
- Σ_S stream symbols and Σ_D data symbols

Definition (Stream Specification)

R_S	*stream layer*
R_D	*data layer*

1. $\mathsf{M}_0 \in \Sigma_S$ with arity 0, the root of $\mathcal{T}$.
2. $\langle \Sigma_D, R_D \rangle$ is a terminating, D-sorted TRS, the data layer of $\mathcal{T}$.
3. $\mathcal{T}$ is exhaustive



1
1 1
1 2 1
1 3 3 1
1 4 6 4 1

Example (Pascal's triangle)

$$
\begin{array}{rcll}
\hline
\mathsf{P} & \to & 0 : \mathsf{s}(0) : \mathsf{g}(\mathsf{P}) & \\
\mathsf{g}(\mathsf{s}(x) : y : \sigma) & \to & \mathsf{a}(\mathsf{s}(x), y) : \mathsf{g}(y : \sigma) & \textit{stream layer} \\
\mathsf{g}(0 : \sigma) & \to & 0 : \mathsf{s}(0) : \mathsf{g}(\sigma) & \\
\hline
\mathsf{a}(x, \mathsf{s}(y)) & \to & \mathsf{s}(\mathsf{a}(x, y)) & \\
\mathsf{a}(x, 0) & \to & x & \textit{data layer} \\
\hline
\end{array}
$$

is a productive stream specification of the Pascal's triangle:

$$\mathsf{P} \twoheadrightarrow 0 : 1 : 0 : 1 : 1 : 0 : 1 : 2 : 1 : 0 : 1 : 3 : 3 : 1 : 0 : \ldots$$

Recognizing Productivity in the Literature

(DO) Data-Oblivious Approach:

- precise quantitative analysis of the stepwise consumptions/ productions during the evaluation of a stream spec

$$\mathsf{even}(0 : 1 : \mathsf{even}(J)) \to 0 : \mathsf{odd}(1 : \mathsf{even}(J))$$

- neglecting information about which concrete data-elements are processed

$$\mathsf{even}(\bullet : \bullet : \mathsf{even}(J)) \to \bullet : \mathsf{odd}(\bullet : \mathsf{even}(J))$$

(Wadge, Sijtsma, Coquand, Gimenez, Telford/Turner, Hughes/Pareto/Sabry, Buchholz, E/G/H/Isihara/Klop).

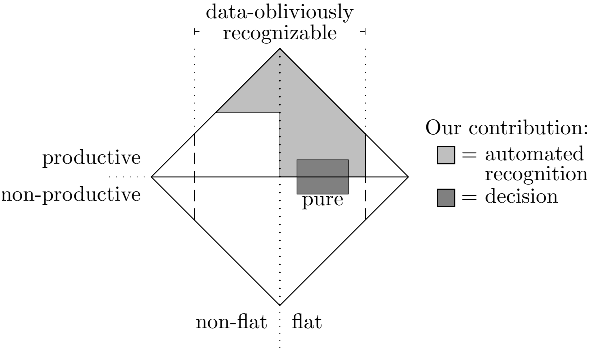
Our Contribution

- give a theoretical analysis of the approach (DO), introducing data-oblivious rewriting in order to:
  - get clear about its theoretical limitations
- exhaust the possibilities of the approach (DO) for the large class of 'flat' specifications, obtaining data-oblivious optimality
- adapt the approach (DO) for stream functions that are defined by exhaustive case distinction:

$$\mathsf{f}(0 : \sigma) \to 0 : 0 : 1 : \mathsf{f}(\sigma)$$
$$\mathsf{f}(1 : \sigma) \to 0 : 1 : \mathsf{f}(\sigma)$$

New Concepts/Definitions

- data-oblivious rewriting;
- data-oblivious productivity;
- stream specification formats: pure $\subsetneq$ flat $\subsetneq$ friendly nesting.



Data-Oblivious Rewriting

formalized by a two-player game between:

- a data-exchange player $\mathcal{G}$ can exchange data elements arbitrarily
- a rewrite player $\mathcal{R}$ can perform usual term rewriting steps

player $\mathcal{G}$ can help or handicap the rewrite player

$\Rightarrow$ for a d-o analysis we have to quantify over all strategies of $\mathcal{G}$

Definition (data-oblivious lower bound on the production of a term s)

$\underline{do}_{\mathcal{T}}(s)$:= worst case production of s (number of elements) by quantification over all strategies for $\mathcal{G}$

A stream spec $\mathcal{T}$ is data-obliviously productive if $\underline{do}_{\mathcal{T}}(\mathsf{M}_0) = \infty$.

Data-oblivious productivity implies productivity

Proposition

The data oblivious production is always $\leq$ the data aware production:

$$\underline{do}_{\mathcal{T}}(s) \leq \Pi_{\mathcal{T}}(s) .$$

Hence, data-oblivious productivity implies productivity.

Proof.

A possible strategy for the exchange player $\mathcal{G}$ is 'do nothing'. $\square$

Example

Example (Pascal's triangle)

$$\mathsf{P} \to \mathsf{0} : \mathsf{s}(\mathsf{0}) : \mathsf{g}(\mathsf{P})$$
$$\mathsf{g}(\mathsf{s}(x) : y : \sigma) \to \mathsf{a}(\mathsf{s}(x), y) : \mathsf{g}(y : \sigma)$$
$$\mathsf{g}(\mathsf{0} : \sigma) \to \mathsf{0} : \mathsf{s}(\mathsf{0}) : \mathsf{g}(\sigma)$$

data abstracted we have:

$$\mathsf{P} \to \bullet : \bullet : \mathsf{g}(\mathsf{P})$$
$$\mathsf{g}(\bullet : \bullet : \sigma) \to \bullet : \mathsf{g}(\bullet : \sigma)$$
$$\mathsf{g}(\bullet : \sigma) \to \bullet : \bullet : \mathsf{g}(\sigma)$$

The data oblivious lower bound on the production function of g is:

$$n \mapsto n \dot{-} 1$$

Which can be used to conclude productivity of P.
Hence P is data-obliviously productive.

Example, Limits of Data-Oblivious Analysis

Example

$$\mathsf{T} \to \mathsf{f}(1 : \mathsf{T}) \qquad \mathsf{f}(0 : \sigma) \to \mathsf{f}(\sigma) \qquad \mathsf{f}(1 : \sigma) \to 1 : \mathsf{f}(\sigma)$$

This specification is productive:

$$\mathsf{T} \twoheadrightarrow 1 : \mathsf{f}(\mathsf{T}) \twoheadrightarrow 1 : 1 : \mathsf{f}(\mathsf{f}(\mathsf{T})) \twoheadrightarrow \ldots \twoheadrightarrow\!\!\!\!\rightarrow 1 : 1 : 1 : 1 : \ldots \;,$$

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

$$\mathsf{T} \to \mathsf{f}(\bullet : \mathsf{T}) \to^{\rho_{\mathsf{f1}}} \mathsf{f}(\mathsf{T}) \twoheadrightarrow \ldots \twoheadrightarrow\!\!\!\!\rightarrow \mathsf{f}(\mathsf{f}(\mathsf{f}(\ldots))) \;.$$

is possible. Hence the specification is not data-obliviously productive.
(that is, productivity of this specification cannot be proven data blindly)

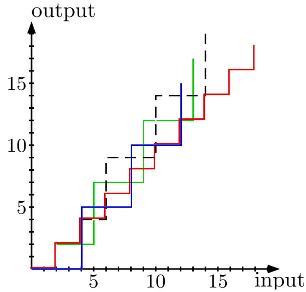
D-O Lower Bounds of Stream Functions

What distinguishes our approach from previous approaches:
we calculate with the exact data-oblivious lower bounds:

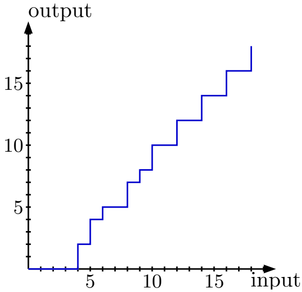
$$\mathsf{f}(\sigma) \to \mathsf{g}(\sigma, \sigma)$$

$$\mathsf{g}(0 : y : \sigma, x : \tau) \to 0 : 0 : \mathsf{g}(\sigma, \tau)$$

$$\mathsf{g}(1 : \sigma, x_1 : x_2 : x_3 : x_4 : \tau) \to 0 : 0 : 0 : 0 : 0 : \mathsf{g}(\sigma, \tau)$$



possible function-call traces for f



exact lower bound $\underline{do}_{\mathcal{T}}(\mathsf{f})$

Flat Stream Spec's

$\mathcal{T}$ is called flat: in rules for stream functions, no nested occurrences of stream function rules on their right hand sides.

Example (Ternary Thue-Morse stream)

$$
\begin{aligned}
\mathsf{Q} &\to \mathsf{a} : \mathsf{R} \\
\mathsf{R} &\to \mathsf{b} : \mathsf{c} : \mathsf{f}(\mathsf{R}) \\
\mathsf{f}(\mathsf{a} : \sigma) &\to \mathsf{a} : \mathsf{b} : \mathsf{c} : \mathsf{f}(\sigma) \\
\mathsf{f}(\mathsf{b} : \sigma) &\to \mathsf{a} : \mathsf{c} : \mathsf{f}(\sigma) \\
\mathsf{f}(\mathsf{c} : \sigma) &\to \mathsf{b} : \mathsf{f}(\sigma)
\end{aligned}
$$

stream layer

data layer

$\mathsf{Q} \twoheadrightarrow \mathsf{a} : \mathsf{b} : \mathsf{c} : \mathsf{a} : \mathsf{c} : \mathsf{b} : \mathsf{a} : \mathsf{b} : \mathsf{c} : \mathsf{b} : \mathsf{a} : \mathsf{c} : \ldots$

Theorem

For flat stream spec's we can decide data-oblivious productivity.

Pure Stream Spec's

$\mathcal{T}$ is called pure: the defining rules for a stream function all have the same consumption/production behaviour.

Example

$$\mathsf{inv}(0:\sigma) \to 1:\mathsf{inv}(\sigma) \qquad \mathsf{inv}(1:\sigma) \to 0:\mathsf{inv}(\sigma)$$

Non-example:
$$\mathsf{g}(0:x:\sigma) \to x:x:\mathsf{g}(\sigma) \qquad \mathsf{g}(1:x:\sigma) \to x:\mathsf{g}(\sigma)\,.$$

Proposition

For pure stream spec's: productivity = data-oblivious productivity.

Theorem

We can decide productivity of pure stream specifications.

Stream Specification (flat, non-pure)

Example (Pascal's triangle stream)

$$\begin{array}{rcll} \mathsf{P} & \to & 0 : \mathsf{s}(0) : \mathsf{g}(\mathsf{P}) & \\ \mathsf{g}(\mathsf{s}(x) : y : \sigma) & \to & \mathsf{a}(\mathsf{s}(x), y) : \mathsf{g}(y : \sigma) & \textit{stream layer} \\ \mathsf{g}(0 : \sigma) & \to & 0 : \mathsf{s}(0) : \mathsf{g}(\sigma) & \\ \hline \mathsf{a}(x, \mathsf{s}(y)) & \to & \mathsf{s}(\mathsf{a}(x, y)) & \textit{data layer} \\ \mathsf{a}(x, 0) & \to & x & \end{array}$$

$$\mathsf{P} \twoheadrightarrow 0 : 1 : 0 : 1 : 1 : 0 : 1 : 2 : 1 : 0 : 1 : 3 : 3 : 1 : 0 : \ldots$$

Stream Specification (pure)

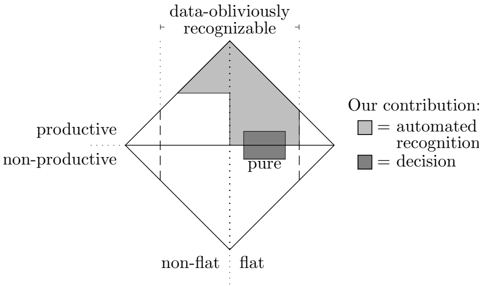
Example (Thue–Morse stream)

$$\begin{array}{rcll}
\mathsf{T} & \to & 0 : 1 : \mathsf{f}(\mathsf{tail}(\mathsf{T})) & \\
\mathsf{f}(0 : \sigma) & \to & 0 : 1 : \mathsf{f}(\sigma) & \textit{stream layer} \\
\mathsf{f}(1 : \sigma) & \to & 1 : 0 : \mathsf{f}(\sigma) & \\
\mathsf{tail}(x : \sigma) & \to & \sigma & \\
\hline
 & & & \textit{data layer}
\end{array}$$

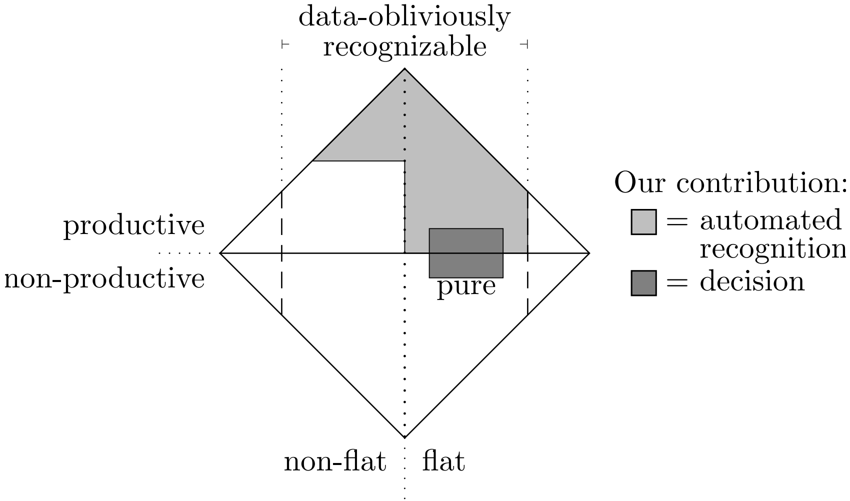
$$\mathsf{T} \twoheadrightarrow\!\!\!\!\to 0 : 1 : 1 : 0 : 1 : 0 : 0 : 1 : 1 : 0 : 0 : 1 : 0 : 1 : 1 : 0 : \ldots$$

Results

1. for flat stream spec's:
   a decision method for data-oblivious productivity, yielding a computable, data-obliviously optimal criterion for productivity
2. for pure stream spec's: a decision method for productivity
3. for friendly nesting stream spec's: a computable criterion for productivity
4. an online-tool automating (1), (2), and (3)



Map of Stream Specifications



Deciding D-O Productivity

1. Input: a flat stream specification $\mathcal{T}$.
2. Stream function translation: for the stream functions f in $\mathcal{T}$, compute their d-o lower bounds $[f] : \overline{\mathbb{N}} \to \overline{\mathbb{N}}$ (periodically increasing functions).
3. Stream constant translation: using (2), translate the root M_0 of $\mathcal{T}$ into a production term $[M_0]$.
4. Production calculation: compute the production $\Pi([M_0])$ of $[M_0]$ in a production calculus (by a confluent, terminating TRS).
5. Decision taking: if $\Pi([M_0]) = \infty$ then $\mathcal{T}$ is d-o productive, else $\mathcal{T}$ is not d-o productive.

Tool Demo. Further Aims.

Productivity Recognition Tool

- Use it at: `http://infinity.few.vu.nl/productivity`

Further Aims:

- (DA) Data-Aware Approach:
  - taking account of which concrete data-elements are processed during the evaluation of a stream spec
  - devise data-aware methods for productivity recognition