Data-Oblivious Stream Productivity

Jörg Endrullis
Department of Computer Science
Vrije Universiteit Amsterdam
De Boelelaan 1081a, NL-1081 HV Amsterdam
joerg@few.vu.nl

Clemens Grabmayer
Department of Philosophy
Universiteit Utrecht
Heidelberglaan 8, NL-3584 CS Utrecht
clemens@phil.uu.nl

Dimitri Hendriks
Department of Computer Science
Vrije Universiteit Amsterdam
De Boelelaan 1081a, NL-1081 HV Amsterdam
diem@cs.vu.nl

Abstract

We are concerned with demonstrating productivity of specifications of infinite streams of data, based on orthogonal rewrite rules. In general, this property is undecidable, but for restricted formats computable sufficient conditions can be obtained. The usual analysis, also adopted here, disregards the identity of data, thus leading to approaches that we call data-oblivious. We present a method that is provably optimal among all such data-oblivious approaches. This means that in order to improve on our algorithm one has to proceed in a data-aware fashion.[1]

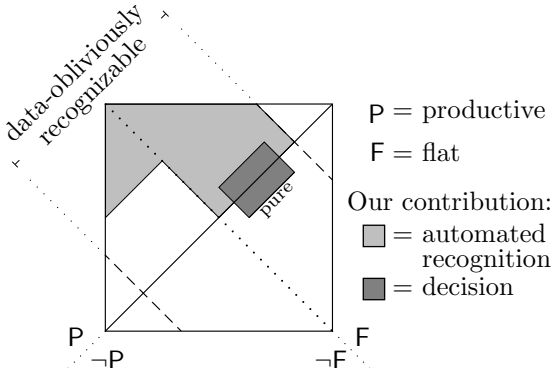

Figure 1: Map of stream specifications

1 Introduction

For programming with infinite structures, productivity is what termination is for programming with finite structures. Productivity captures the intuitive notion of unlimited progress, of 'working' programs producing defined values indefinitely. In functional languages, usage of infinite structures is common practice. For the correctness of programs dealing with such structures one must guarantee that every finite part of the infinite structure can be evaluated, that is, the specification of the infinite structure must be productive.

We investigate this notion for stream specifications, formalized as orthogonal term rewriting systems. Common to all previous approaches for recognizing productivity is a quantitative analysis that abstracts away from the concrete values of stream elements. We formalize this by a notion of 'data-oblivious' rewriting, and introduce the concept of data-oblivious productivity. Data-oblivious (non-) productivity implies (non-)productivity, but neither of the converse implications holds. Fig. 1 shows a Venn diagram of stream specifications, highlighting the subset of 'data-obliviously recognizable' specifications where (non-) productivity can be recognized by a data-oblivious analysis.

We identify two syntactical classes of stream specifications: 'flat' and 'pure' specifications, see the description below. For the first we devise a decision algorithm for data-oblivious productivity. This gives rise to a computable, data-obliviously optimal, criterion for productivity: every flat stream specification that can be established to be productive by whatever data-oblivious argument is recognized as productive by this criterion (see Fig. 1). For the subclass of pure specifications, we establish that data-oblivious productivity coincides with productivity, and thereby obtain a

[1]This research has been partially funded by the Netherlands Organisation for Scientific Research (NWO) under FOCUS/BRICKS grant number 642.000.502.

decision algorithm for productivity of this class. Additionally, we extend our criterion beyond the class of flat stream specifications, allowing for 'friendly nesting' in the specification of stream functions; here data-oblivious optimality is not preserved.

In defining the different formats of stream specifications, we distinguish between rules for stream constants, and rules for stream functions. Only the latter are subjected to syntactic restrictions. In flat stream specifications the defining rules for the stream functions do not have nesting of stream function symbols. This format allows for exhaustive pattern matching on data to define stream functions, thus possibly leading to multiple defining rules for an individual stream function symbol. Since the quantitative consumption/production behavior of a symbol f might differ in the defining rules of f, all one can settle for in a data-abstracted analysis is the use of lower bounds for recognizing productivity. If for all stream function symbols f in a flat specification $\mathcal{T}$ the defining rules for f coincide, disregarding the identity of data-elemens and modulo renaming of stream variables, then $\mathcal{T}$ is called pure.

Our decision algorithm for data-oblivious productivity of flat stream specifications determines the tight data-oblivious lower bounds on the production behavior of every stream function, and uses these bounds to calculate the data-oblivious production of stream constants. We briefly explain both aspects. Consider the stream specification $\mathsf{A} \to \mathsf{0}:\mathsf{f}(\mathsf{A})$ together with the rules $\mathsf{f}(\mathsf{0}:\sigma) \to \mathsf{1}:\mathsf{0}:\mathsf{1}:\mathsf{f}(\sigma)$, and $\mathsf{f}(\mathsf{1}:\sigma) \to \mathsf{0}:\mathsf{f}(\sigma)$, defining the stream $\mathsf{0}:\mathsf{1}:\mathsf{0}:\mathsf{1}:\ldots$ of alternating bits. For f the function $id\colon\ n \mapsto n$ is the tight data-oblivious lower bound. Further note that the function $suc\colon\ n \mapsto n+1$ is the optimal 'production modulus' of the function prepending a data element to a stream term. Thus the data-oblivious production of A can be computed as $\mu(suc \circ id) = \infty$, where $\mu(f)$ is the least fixed point of $f : \overline{\mathbb{N}} \to \overline{\mathbb{N}}$, and hence A is productive. As a comparison, only a 'data-aware' approach is able to establish productivity of a specification like $\mathsf{B} \to \mathsf{0}:\mathsf{g}(\mathsf{B})$ with $\mathsf{g}(\mathsf{0}:\sigma) \to \mathsf{1}:\mathsf{0}:\mathsf{g}(\sigma)$, and $\mathsf{g}(\mathsf{1}:\sigma) \to \mathsf{g}(\sigma)$. The data-oblivious lower bound of g is $n \mapsto 0$, which makes it impossible for any conceivable data-oblivious approach to recognize productivity of B.

Our results are summarized as follows:

(i) For the class of 'flat' stream specifications we give a computable, data-obliviously optimal, sufficient condition for productivity.

(ii) We show decidability of productivity for the class of 'pure' stream specifications, an extension of the format in [2].

(iii) Disregarding data-oblivious optimality, we extend (i) to stream specifications with 'friendly nesting'.

(iv) A tool automating (i), (ii) and (iii), retrievable from: `http://infinity.few.vu.nl/productivity`.

Related work. Previous approaches [6, 3, 7, 1] employed data-oblivious reasoning (without using this name for it) to find sufficient criteria ensuring productivity, but did not aim at optimality. The data-oblivious production behavior of a stream function f is bounded below by a 'modulus of production' $\nu_{\mathsf{f}} : \mathbb{N}^k \to \mathbb{N}$ with the property that the first $\nu_{\mathsf{f}}(n_1, \ldots, n_k)$ elements of $\mathsf{f}(t_1, \ldots, t_k)$ can be computed whenever the first n_i elements of t_i are defined. Sijtsma develops an approach allowing arbitrary production moduli $\nu : \mathbb{N}^k \to \mathbb{N}$, which, while providing an adequate mathematical description, are less amenable to automation. Telford and Turner [7] employ production moduli of the form $\nu(n) = n + a$ with $a \in \mathbb{Z}$. Hughes, Pareto and Sabry [3] use $\nu(n) = \max\{c \cdot x + d \mid x \in \mathbb{N}, n \geq a{\cdot}x{+}b\} \cup \{0\}$ with $a, b, c, d \in \mathbb{N}$. Both classes of production moduli are strictly contained in the class of 'periodically increasing' functions which we employ in our analysis. As we show, the set of data-oblivious lower bounds of flat stream functions is exactly the set of periodically increasing functions.

The present work generalizes that of [2]; see the discussion after the definition of pure specifications (Def. 6.4).

Overview. In Sec. 2 we define the notion of stream specification, and the syntactic format of flat specifications. In Sec. 3 we formalize the notion of data-oblivious rewriting. In Sec. 4 we introduce a production calculus as a means to compute the production of the data-abstracted stream specifications, based on the set of periodically increasing functions. A translation of stream specifications into production terms is defined in Sec. 5. Our main results, mentioned above, are collected in Sec. 6. We conclude and discuss some future research topics in Sec. 7.

2 Stream Specifications

We introduce the notion of stream specification. An example is given in Fig. 2. This is a productive specification

$\mathsf{Q} \to \mathsf{a}:\mathsf{R}$ $\mathsf{R} \to \mathsf{b}:\mathsf{c}:\mathsf{f}(\mathsf{R})$	*stream layer*
$\mathsf{f}(\mathsf{a}:\sigma) \to \mathsf{a}:\mathsf{b}:\mathsf{c}:\mathsf{f}(\sigma)$ $\mathsf{f}(\mathsf{b}:\sigma) \to \mathsf{a}:\mathsf{c}:\mathsf{f}(\sigma)$ $\mathsf{f}(\mathsf{c}:\sigma) \to \mathsf{b}:\mathsf{f}(\sigma)$	*function layer*
	data layer

Figure 2: Example of a flat stream specification.

that defines the ternary Thue–Morse sequence, a square-free word over $\{\mathsf{a}, \mathsf{b}, \mathsf{c}\}$; see, e.g., [5]. Indeed, evaluating this specification, we get: $\mathsf{Q} \twoheadrightarrow \mathsf{a{:}b{:}c{:}a{:}c{:}b{:}a{:}b{:}c{:}b{:}a{:}c{:}\ldots}$

Stream specifications consist of three layers: a *stream layer* (top) where stream constants are specified using stream and data function symbols that are defined by the rules of the *function layer* (middle) and the *data-layer* (bottom, which is empty in Fig. 2). Each layer may use symbols defined in a lower layer, but not vice-versa.

The three-tiered setup of stream specifications is motivated as follows. In order to abstract from the termination problem when investigating productivity, the data layer is a term rewriting system on its own, and is required to be strongly normalizing. Moreover, an isolated data layer prevents the use of stream symbols by data rules, see the discussion below Def. 2.1.

The reason to distinguish a separate function layer is of a more pragmatic nature. We are interested in the productivity of recursive stream specifications that make use of a library of 'manageable' stream functions. By this we mean a class of stream functions defined by a syntactic format with the property that their data-oblivious lower bounds are contained in a set of computable production moduli that is effectively closed under composition, minimum and taking least fixed points. As such a format we define the class of flat stream function specifications (Def. 2.2) for which data-oblivious lower bounds are precisely the 'periodically increasing' functions (see Sec. 4).

Thus only the rules in the function layer are subject to syntactic restrictions. But no conditions other than well-sortedness are imposed on how the defining rules for the stream constant symbols in the stream layer make use of the symbols in the other layers.

Stream specifications are formalized as many-sorted, orthogonal, constructor term rewriting systems [8]. We distinguish between *stream terms* and *data terms*. For the sake of simplicity we consider only one sort S for stream terms and one sort D for data terms. Without any complication, our results extend to stream specifications with multiple sorts for data and for stream terms.

Let $\mathcal{T} = \langle \Sigma, R \rangle$ be a term rewriting system (TRS). We define $\mathcal{D}(\Sigma) := \{root(l) \mid l \to r \in R\}$, the set of *defined symbols*, and $\mathcal{C}(\Sigma) := \Sigma \setminus \mathcal{D}(\Sigma)$, the set of *constructor symbols*. Then $\mathcal{T}$ is called a *constructor TRS* if for every rewrite rule $\rho \in R$, the left-hand side is of the form $f(t_1, \ldots, t_n)$ with $t_i \in Ter(\mathcal{C}(\Sigma))$; then ρ is called a *defining rule for* f. We call $\mathcal{T}$ *exhaustive* for $f \in \Sigma$ if every term $f(t_1, \ldots, t_n)$ with closed, constructor terms t_i is a redex.

A *stream TRS* is a finite $\{S, D\}$-sorted, orthogonal, constructor TRS $\langle \Sigma, R \rangle$, where the signature $\Sigma = \Sigma_S \uplus \Sigma_D$ is partitioned into the set Σ_S of *stream symbols* and the set Σ_D of *data symbols*. The signature $\Sigma_S = \{:\} \uplus \Sigma_{fun} \uplus \Sigma_{str}$ is further partitioned into ':', the *stream constructor symbol*, with arity $(D, S) \to S$, the set of *stream function symbols* Σ_{fun}, having at least one stream argument, and the set of *stream constant symbols* Σ_{str}, having only data arguments. We assume, w.l.o.g., that for all $f \in \Sigma_{fun}$ its stream arguments are in front; that is, $ar(f) \in S^{ar_s(f)}D^{ar_d(f)} \to \{S, D\}$, where $ar_s(f)$ and $ar_d(f) \in \mathbb{N}$ are called the *stream arity* and the *data arity* of f, respectively.

We come to the definition of stream specifications.

Definition 2.1. Let $\mathcal{T} = \langle \Sigma, R \rangle$ be a stream TRS with partitions $\Sigma = \Sigma_{str} \uplus \Sigma_{fun} \uplus \{:\} \uplus \Sigma_D$ and $R = R_{str} \uplus R_{fun} \uplus R_D$. $\mathcal{T}$ is a *stream specification* if it consists of 3 layers:

R_{str}	*stream layer*	
R_{fun}	*function layer*	*stream function specification*
R_D	*data layer*	

and the following conditions hold:

(i) The *data-layer* $\mathcal{T}_d = \langle \Sigma_D, R_D \rangle$ is a terminating TRS.

(ii) The underlying *stream function specification* $\mathcal{T}_{fun} = \langle \Sigma_{fun} \uplus \{:\} \uplus \Sigma_D, R_{fun} \uplus R_D \rangle$ is a TRS.

(iii) $\mathcal{T}$ is exhaustive for the defined symbols in Σ.

(iv) Σ_{str} is a set of constant symbols containing a distinguished symbol M_0, called the *root of* $\mathcal{T}$. R_{str} is the set of *defining rules* $\rho_{\mathsf{M}}\colon \mathsf{M} \to s$ for every $\mathsf{M} \in \Sigma_{str}$.

Note that exhaustivity for Σ_D together with strong normalization of $\mathcal{T}_d$ guarantees that closed data terms rewrite to constructor normal forms, a property known as sufficient completeness [4].

A restriction imposed by the hierarchical setup of stream specifications is that symbols defined in the data-layer are *stream independent*. The reason to exclude data rules that make use of stream terms is to prevent the output of possibly undefined stream elements. A stream dependent data function, like $\mathsf{head}(x : \sigma) \to x$, possibly causes 'look-ahead' as in the following well-defined example from [6]:

$$\mathsf{S} \to \mathsf{S}(3) : \mathsf{0} : \mathsf{tail}(\mathsf{S})\,,$$

where $\mathsf{S}(n) := \mathsf{head}(\mathsf{tail}^n(\mathsf{S}))$.

Definition 2.2. Let $\mathcal{T}$ be a stream specification. A stream function rule is called *flat* if its right-hand side does not contain nested occurrences of stream function symbols. $\mathcal{T}$ is called *flat* if all rules in the function layer R_{fun} are flat.

Fig. 3 shows an alternative specification of the ternary Thue–Morse sequence, this time constructed from the binary Thue–Morse sequence specified by M. Both of the specifications given in Figs. 2 and 3 are flat. The latter belongs to the subclass of pure specifications, see Sec. 6.

$\mathsf{Q} \to \mathsf{diff}(\mathsf{M})$ $\mathsf{M} \to 0 : \mathsf{zip}(\mathsf{inv}(\mathsf{M}), \mathsf{tail}(\mathsf{M}))$	*stream layer*
$\mathsf{zip}(x:\sigma,\tau) \to x : \mathsf{zip}(\tau,\sigma)$ $\mathsf{inv}(x:\sigma) \to \mathsf{i}(x) : \mathsf{inv}(\sigma)$ $\mathsf{tail}(x:\sigma) \to \sigma$ $\mathsf{diff}(x:y:\sigma) \to \mathsf{x}(x,y) : \mathsf{diff}(y:\sigma)$	*function layer*
$\mathsf{i}(0) \to 1$ $\quad \mathsf{i}(1) \to 0$ $\mathsf{x}(0,0) \to \mathsf{b}$ $\quad \mathsf{x}(0,1) \to \mathsf{a}$ $\mathsf{x}(1,0) \to \mathsf{c}$ $\quad \mathsf{x}(1,1) \to \mathsf{b}$	*data layer*

Figure 3: Example of a pure stream specification.

Definition 2.3. Let $\mathcal{T}$ be a stream specification. A subset $\tilde{R} \subseteq R_{fun}$ is called *friendly nesting* if for every rule $\rho \in \tilde{R}$ we have that (1) it consumes in each argument at most one stream element, (2) it produces at least one, and (3) the defining rules for the stream function symbols occurring on the right-hand side of ρ are again in $\tilde{R}$. A rule $\rho \in R_{fun}$ is *friendly nesting* if it is contained in a friendly nesting subset of R_{fun}. Finally, $\mathcal{T}$ is called *friendly nesting* if every rule in R_{fun} is either flat or friendly nesting.

Definition 2.4. Let $\mathcal{T}$ be a stream specification. For all stream terms $t \in Ter_\infty(\Sigma)_S$ we define by

$$\#_:(t) := \sup\{n \in \mathbb{N} \mid t = t_1 : \ldots : t_n : t'\}$$

the *number of leading stream constructors of* t.

Let $\mathcal{S} = \langle Ter_\infty(\Sigma)_S, \to\rangle$ be an abstract reduction system (ARS). The *production function* $\Pi_\mathcal{S} : Ter(\Sigma)_S \to \overline{\mathbb{N}}$ *for* $\mathcal{S}$ is defined for all $s \in Ter_\infty(\Sigma)_S$ by:

$$\Pi_\mathcal{S}(s) := \sup\{\, \#_:(t) \mid s \twoheadrightarrow_\mathcal{S} t \,\}\,.$$

$\Pi_\mathcal{T}$ is called the *production function of* $\mathcal{T}$. We say that $\mathcal{T}$ *is productive for a stream term* s if $\Pi_\mathcal{T}(s) = \infty$. $\mathcal{T}$ is called *productive* if $\mathcal{T}$ is productive for its root M_0.

3 Data-Oblivious Analysis

We formalize the notion of data-oblivious rewriting and introduce the concept of data-oblivious productivity. The idea is a quantitative reasoning where all knowledge about the concrete values of data elements during an evaluation sequence is ignored. For example, consider the following stream specification:

$$\begin{aligned} \mathsf{T} &\to \mathsf{f}(0:1:\mathsf{T}) \\ \mathsf{f}(0:x:\sigma) &\to 0:1:\mathsf{f}(\sigma) && (\rho_{\mathsf{f}0}) \\ \mathsf{f}(1:x:\sigma) &\to x:\mathsf{f}(\sigma) && (\rho_{\mathsf{f}1}) \end{aligned}$$

The specification of T is productive:

$$\mathsf{T} \to^2 0:1:\mathsf{f}(\mathsf{T}) \to^2 0:1:0:1:\mathsf{f}(\mathsf{f}(\mathsf{T})) \twoheadrightarrow \ldots\,.$$

During the whole rewrite sequence ($\rho_{\mathsf{f}1}$) is never applied. However, disregarding the identity of data makes ($\rho_{\mathsf{f}1}$) applicable and allows for the following rewrite sequence:

$$\mathsf{T} \to \mathsf{f}(\bullet:\bullet:\mathsf{T}) \to^{(\rho_{\mathsf{f}1})} \bullet:\mathsf{f}(\mathsf{T}) \twoheadrightarrow \bullet:\mathsf{f}(\bullet:\mathsf{f}(\bullet:\mathsf{f}(\ldots)))\,,$$

producing only one element. Hence the specification of T is not data-obliviously productive.

Data-oblivious term rewriting can be thought of as two-player game between a *rewrite player* $\mathcal{R}$ which performs the usual term rewriting, and an *opponent* $\mathcal{G}$ which before every rewrite step is allowed to arbitrarily exchange data elements for (sort-respecting) data terms in constructor normal form. The data-oblivious lower (upper) bound on the production of a stream term s is the infimum (supremum) of the production of s with respect to all possible strategies for the opponent $\mathcal{G}$. Fig. 4 depicts data-oblivious rewriting of the above stream specification T; by exchanging data elements, the opponent $\mathcal{G}$ enforces the application of ($\rho_{\mathsf{f}1}$).

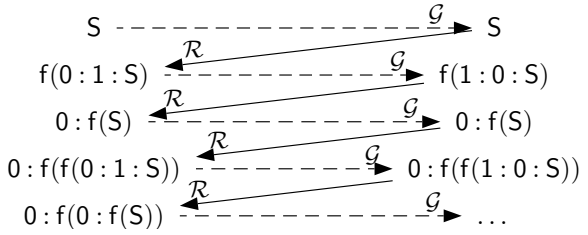

Figure 4: Data-oblivious rewriting

At first glance it might appear natural to model the opponent using a function $\mathcal{G} : Ter(\Sigma_D) \to Ter(\Sigma_D)$ from data terms to data terms. However, such a per-element deterministic exchange strategy preserves equality of data-elements. Then the following specification of M:

$$\begin{aligned} \mathsf{M} &\to \mathsf{g}(\mathsf{Z},\mathsf{Z}) & \mathsf{Z} &\to 0:\mathsf{Z} \\ \mathsf{g}(0:\sigma,0:\tau) &\to 0:\mathsf{g}(\sigma,\tau) & \mathsf{g}(0:\sigma,1:\tau) &\to \mathsf{g}(\sigma,\tau) \\ \mathsf{g}(1:\sigma,1:\tau) &\to 0:\mathsf{g}(\sigma,\tau) & \mathsf{g}(1:\sigma,0:\tau) &\to \mathsf{g}(\sigma,\tau) \end{aligned}$$

would be productive for every such $\mathcal{G}$, which is clearly not what one would expect of a data-oblivious analysis.

We model the opponent as a function from stream terms to stream terms: $\mathcal{G} : Ter(\Sigma)_S \to Ter(\Sigma)_S$. It can be shown that it is sufficient to quantify over strategies for $\mathcal{G}$ for which $\mathcal{G}(s)$ is invariant under exchange of data elements in s for all terms s. Therefore we first abstract from the data elements in favour of symbols $\bullet$ and then define the opponent $\mathcal{G}$ on the abstracted terms.

Definition 3.1. Let $\mathcal{T} = \langle \Sigma, R \rangle$ be a stream specification and let $[\![\Sigma]\!] := \{\bullet\} \uplus \Sigma_S$. Given a stream term $s \in Ter(\Sigma)_S$, its data abstraction $[\![s]\!] \in Ter([\![\Sigma]\!])_S$ is defined by:

$$[\![\sigma]\!] = \sigma\,, \qquad [\![u : s_0]\!] = \bullet : [\![s_0]\!]\,,$$
$$[\![\mathsf{f}(s_1, \ldots, s_n)]\!] = \mathsf{f}([\![s_1]\!], \ldots, [\![s_n]\!])\,.$$

Definition 3.2. Let $\mathcal{T} = \langle \Sigma, R \rangle$ be a stream specification. A function $\mathcal{G} : Ter([\![\Sigma]\!])_S \to Ter(\Sigma)_S$ is called a *data-exchange function on* $\mathcal{T}$ if for all $r \in Ter([\![\Sigma]\!])_S$ it holds: $[\![\mathcal{G}(r)]\!] = r$, and $\mathcal{G}(r)$ is in data-constructor normal form.

Definition 3.3. For $\mathcal{G}$ a *data-exchange function*, we define the ARS $\mathcal{A}_{\mathcal{T},\mathcal{G}} \subseteq Ter([\![\Sigma]\!])_S \times Ter([\![\Sigma]\!])_S$ as follows:

$$\mathcal{A}_{\mathcal{T},\mathcal{G}} := \{l \to [\![r]\!] \mid l \in Ter([\![\Sigma]\!]),\ r \in Ter(\Sigma) \text{ with } \mathcal{G}(l) \to_{\mathcal{T}} r\}\,.$$

The *data-oblivious production range* $\overline{\underline{do}}_{\mathcal{T}}(s)$ of a stream term $s \in Ter(\Sigma)_S$ is defined as follows:

$$\overline{\underline{do}}_{\mathcal{T}}(s) := \big\{\Pi_{\mathcal{B}}([\![s]\!]) \mid \mathcal{G} \text{ a data-exchange function on } \mathcal{T},\ \mathcal{B} \subseteq \mathcal{A}_{\mathcal{T},\mathcal{G}} \text{ an outermost-fair strategy}\big\}\,.$$

The *data-oblivious lower* and *upper bound on the production of a term* s are defined by $\underline{do}_{\mathcal{T}}(s) := \inf(\overline{\underline{do}}_{\mathcal{T}}(s))$ and $\overline{do}_{\mathcal{T}}(s) := \sup(\overline{\underline{do}}_{\mathcal{T}}(s))$, respectively.

The notion of positions of rewrite steps in $\mathcal{T}$ carries over to the ARS $\mathcal{A}_{\mathcal{T},\mathcal{G}}$, since the steps in $\mathcal{A}_{\mathcal{T},\mathcal{G}}$ are projections of steps in $\mathcal{T}$. We adapt the notion of outermost-fairness for our purposes. A rewrite sequence $s_1 \to s_2 \to \ldots$ in $\mathcal{A}_{\mathcal{T},\mathcal{G}}$ is called *outermost-fair* if it is finite or for all $n_0 \in \mathbb{N}$ it holds: if p is an outermost redex position for all terms s_n with $n \geq n_0$, then there exists a step $s_m \to s_{m+1}$ for some $m \geq n_0$ at position p.

Definition 3.4. A stream specification $\mathcal{T}$ is called *data-obliviously productive* (*data-obliviously non-productive*) if $\underline{do}_{\mathcal{T}}(\mathsf{M}_0) = \infty$ (if $\overline{do}_{\mathcal{T}}(\mathsf{M}_0) < \infty$) holds.

Proposition 3.5. *Let $\mathcal{T} = \langle \Sigma, R \rangle$ be a stream specification and $s \in Ter(\Sigma)_S$ a stream term. Then we have*

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

Hence data-oblivious productivity implies productivity and data-oblivious non-productivity implies non-productivity.

We define lower and upper bounds on the data-oblivious consumption/production behavior of stream functions. These bounds are used to reason about data-oblivious (non-)productivity of stream constants, see Sec. 6.

Definition 3.6. Let $\mathcal{T} = \langle \Sigma, R \rangle$ be a stream specification and $g \in \Sigma_{fun}$. We define sets S_n of n-defined (finite) stream terms with $n \in \mathbb{N}$ defined elements:

$$S_n := \{t_1 : \ldots : t_n : \sigma \mid t_1, \ldots, t_n \in Ter(\mathcal{C}(\Sigma_D), \varnothing)\}\,,$$

and sets $G_{n_1,\ldots,n_{ar_s(\mathsf{g})}}$ for $n_i \in \mathbb{N}$ of applications of the stream function g to n_i-defined arguments:

$$G_{n_1,\ldots,n_{ar_s(\mathsf{g})}} := \{\mathsf{g}(t_1, \ldots, t_{ar_s(\mathsf{g})}, u_1, \ldots, u_{ar_d(g)}) \mid \forall i.\, t_i \in S_{n_i},\ \forall j.\, u_j \in Ter(\mathcal{C}(\Sigma_D), \varnothing)\}$$

The *optimal production modulus* ν_{g} *of* g is defined by:

$$\nu_{\mathsf{g}}(n_1, \ldots, n_{ar_s(\mathsf{g})}) := \inf\{\Pi_{\mathcal{T}}(t) \mid t \in G_{n_1,\ldots,n_{ar_s(\mathsf{g})}}\}\,.$$

The *data-oblivious production range of the function* g is:

$$\overline{\underline{do}}_{\mathcal{T}}(\mathsf{g})(n_1, \ldots, n_{ar_s(\mathsf{g})}) := \bigcup\{\overline{\underline{do}}_{\mathcal{T}}(t) \mid t \in G_{n_1,\ldots,n_{ar_s(\mathsf{g})}}\}.$$

The *data-oblivious lower bound on the production of* g is defined by $\underline{do}_{\mathcal{T}}(\mathsf{g}) := \inf(\overline{\underline{do}}_{\mathcal{T}}(\mathsf{g}))$.

Note that even simple stream function specifications can exhibit a complex data-oblivious behavior. For example, consider the following function specification:

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

Fig. 5 (left) shows a (small) selection of the possible function-call traces for f. In particular, it depicts the traces that contribute to the data-oblivious lower bound $\underline{do}_{\mathcal{T}}(\mathsf{f})$. The lower bound $\underline{do}_{\mathcal{T}}(\mathsf{f})$, shown on the right, is a superposition of multiple function-call traces for f. In general $\underline{do}_{\mathcal{T}}(\mathsf{f})$ can even be a superposition of infinitely many traces.

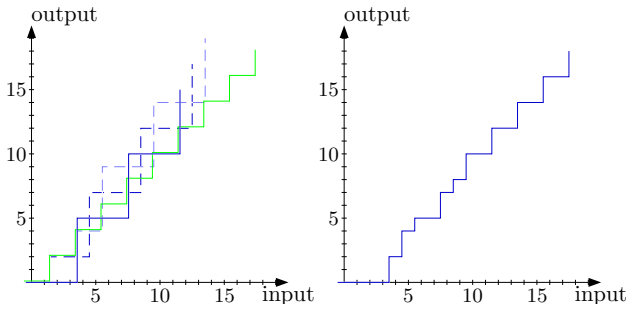

Figure 5: Data-oblivious traces and lower bound

To illustrate the difference between the optimal production modulus ν_{h} and data-oblivious lower bound $\underline{do}_{\mathcal{T}}(\mathsf{h})$, we consider the following stream function specification:

$$\mathsf{h}(0 : x : \sigma) \to x : x : \mathsf{h}(0 : \sigma) \qquad (\rho_{\mathsf{h}0})$$
$$\mathsf{h}(1 : x : \sigma) \to x : \mathsf{h}(0 : \sigma) \qquad (\rho_{\mathsf{h}1})$$

with $\Sigma_D = \{0, 1\}$. Then $\nu_{\mathsf{h}}(n) = 2n \dot{-} 3$ is the optimal production modulus of the stream function h. To obtain this bound one has to take into account that the data element 0 is supplied to the recursive call and conclude that (ρ_{h1}) is only applicable in the first step of a rewrite sequence $\mathsf{h}(u_1 : \ldots : u_n : \sigma) \to \ldots$. However, the data-oblivious lower bound is $\underline{do}_{\mathcal{T}}(\mathsf{h})(n) = n \dot{-} 1$, derived from rule (ρ_{h1}).

4 The Production Calculus

As a means to compute the data-oblivious production behaviour of stream specifications, we introduce a 'production calculus' with periodically increasing functions as its central ingredient.

We use $\overline{\mathbb{N}} := \mathbb{N} \uplus \{\infty\}$, the *extended natural numbers*, with the usual $\leq$, $+$, and we define $\infty - \infty := 0$.

Let $\sigma \in X^\omega$ be an arbitrary sequence. We use $\sigma[n]$ to denote the prefix of length n of σ. We call σ *eventually periodic* if $\sigma = \tau\upsilon\upsilon\upsilon\ldots$ for some $\tau \in X^*$ and $\upsilon \in X^+$.

Definition 4.1. A function $f : \mathbb{N} \to \overline{\mathbb{N}}$ is called *eventually periodic* if the sequence $\langle f(0), f(1), f(2), \ldots\rangle$ is eventually periodic. A function $g : \mathbb{N} \to \overline{\mathbb{N}}$ is called *periodically increasing* if it is non-decreasing and the *derivative of* g, the function $g' : \mathbb{N} \to \overline{\mathbb{N}}$ with $n \mapsto g(n+1) - g(n)$, is eventually periodic. A function $h : \overline{\mathbb{N}} \to \overline{\mathbb{N}}$ is called *periodically increasing* if its restriction to $\mathbb{N}$ is periodically increasing and if $h(\infty) = \lim_{n\to\infty} h(n)$.

Every periodically increasing function can be denoted by its value at 0 followed by a representation of its derivative. For example, $0\overline{312}$ denotes the periodically increasing function $f : \mathbb{N} \to \mathbb{N}$ with values $0, 3, 4, 6, 7, 9, \ldots$. However, we use a finer and more flexible notation over the alphabet $\{-, +\}$ that will be useful in Sec. 5. Formalizing the notation $-{+}{+}{+}\overline{-{+}{-}{+}{+}}$, we denote f as above by the 'io-term' $\langle -{+}{+}{+}, -{+}{-}{+}{+}\rangle$.

Definition 4.2. Let $\pm := \{+, -\}$ and ϵ denote the empty word. An *io-term* is an expression $\langle \alpha, \beta\rangle$ with $\alpha \in \pm^*$ and $\beta \in \pm^+$. The set of io-terms is denoted by $\mathcal{I}$.

For $\langle \alpha, \beta\rangle \in \mathcal{I}$, we define $\pi_{\langle \alpha,\, \beta\rangle} : \overline{\mathbb{N}} \to \overline{\mathbb{N}}$ by:

$$\pi_{\langle \alpha,\, \beta\rangle}(n) = \sup\{\#_+(\sigma[m]) \mid m \in \mathbb{N},\ \#_-(\sigma[m]) \leq n\}$$

for all $n \in \overline{\mathbb{N}}$, and say that $\langle \alpha, \beta\rangle$ *represents* $\pi_{\langle \alpha,\, \beta\rangle}$.

It is easy to verify that, for every $\langle \alpha, \beta\rangle \in \mathcal{I}$, the function $\pi_{\langle \alpha,\, \beta\rangle}$ is periodically increasing and its values can be computed from α and β. Furthermore, every periodically increasing function is represented by an io-term.

Every io-term $\langle \alpha, \beta\rangle$ where $\beta \in \{-\}^*$ can be simplified to $\langle \alpha, \epsilon\rangle$, because $\pi_{\alpha\overline{\beta}} = \pi_{\alpha\epsilon}$. Therefore, when dealing with io-terms, we tacitly assume that this simplification has already been carried out.

Proposition 4.3. *Periodically increasing functions are closed under composition, minimum, and taking fixed point.*

In addition to this, the operations composition, minimum, and taking the fixed point of periodically increasing functions can be computed on representing io-terms: there are computable operations $\circ, \mathit{min} : \mathcal{I} \times \mathcal{I} \to \mathcal{I}$, and $\mathsf{fix} : \mathcal{I} \to \overline{\mathbb{N}}$ such that for all $\sigma, \tau \in \mathcal{I}$ it holds: $\pi_\sigma(\mathsf{fix}(\sigma)) = \mathsf{fix}(\sigma)$, and, for all $n \in \mathbb{N}$, $\pi_{\sigma\circ\tau}(n) = \pi_\sigma(\pi_\tau(n))$ and $\pi_{\mathit{min}(\sigma,\tau)}(n) = \min\{\pi_\sigma(n), \pi_\tau(n)\}$.

We introduce a term syntax for the production calculus and rewrite rules for evaluating closed terms.

Definition 4.4. Let $\mathcal{X}$ be a set of recursion variables. The set of *production terms* $\mathcal{P}$ is generated by:

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

where $x \in \mathcal{X}$, $\sigma \in \mathcal{I}$. and for $k \in \overline{\mathbb{N}}$, the symbol $\underline{k}$ is a *numeral* (a term representation) for k. For $p_1, \ldots, p_n \in \mathcal{P}$ and $n \in \mathbb{N}$, we use $\mathsf{min}_n(p_1, \ldots, p_n)$ as shorthand for the production term $\mathsf{min}(p_1, \mathsf{min}(p_2, \ldots, \mathsf{min}(p_{n-1}, p_n)))$.

The *production* $\Pi(p) \in \overline{\mathbb{N}}$ of a closed production term $p \in \mathcal{P}$ is defined by induction on the term structure, interpreting μ as the least fixed point operator, σ as π_σ, $\underline{k}$ as k, and min as $\min$.

Definition 4.5. Let $r \in \mathbb{N}$ such that $r > 1$. We say that an r-ary function $h : (\overline{\mathbb{N}})^r \to \overline{\mathbb{N}}$ is *periodically increasing* if $h(n_1, ..., n_r) = \min(h_1(n_1), \ldots, h_r(n_r))$ for some unary periodically increasing functions $h_1, \ldots, h_{n_r}$.

For the faithful modeling of data-oblivious lower bounds of stream functions with stream arity r, we employ r-ary periodically increasing functions, denoted as r-ary gates. An *r-ary gate*, abbreviated by $\mathsf{gate}(\sigma_1, \ldots, \sigma_r)$, is a production term context of the form $\mathsf{min}_r(\sigma_1(\Box_1), \ldots, \sigma_r(\Box_r))$, where $\sigma_1, \ldots, \sigma_r \in \mathcal{I}$. We use γ as a syntactic variable for gates. The production function of a gate $\gamma = \mathsf{gate}(\sigma_1, \ldots, \sigma_r)$ is defined as $\pi_\gamma(n_1, \ldots, n_r) := \min(\pi_{\sigma_1}(n_1), \ldots, \pi_{\sigma_r}(n_r))$.

Next, we define a rewrite system for computing the production of a production term. This system uses of the computable operations $\circ$ and fix on io-terms mentioned above.

Definition 4.6. The *reduction relation* $\to_{\mathsf{R}}$ *on production terms* is defined as the compatible closure of the set of rules:

$$\begin{aligned}
\sigma_1(\sigma_2(p)) &\to \sigma_1 \circ \sigma_2(p)\\
\sigma(\mathsf{min}(p_1, p_2)) &\to \mathsf{min}(\sigma(p_1), \sigma(p_2))\\
\mu x.\mathsf{min}(p_1, p_2) &\to \mathsf{min}(\mu x.p_1, \mu x.p_2)\\
\mu x.p &\to p \qquad \text{if } x \notin \mathsf{FV}(p)\\
\mu x.\sigma(x) &\to \underline{\mathsf{fix}(\sigma)}\\
\mu x.x &\to \underline{0}\\
\sigma(\underline{k}) &\to \underline{\pi_\sigma(k)}\\
\mathsf{min}(\underline{k_1}, \underline{k_2}) &\to \underline{\min(k_1, k_2)}
\end{aligned}$$

Based on the following two lemmas, which formulate properties of the rewrite relation $\to_{\mathsf{R}}$, Thm. 4.9 below establishes the usefulness of $\to_{\mathsf{R}}$. It entails that in order to compute the production $\Pi(p)$ of a production term p it suffices to obtain a $\to_{\mathsf{R}}$-normal form of p.

Lemma 4.7. *The rewrite relation $\to_{\mathsf{R}}$ is production preserving, that is, $p \to_{\mathsf{R}} p'$ implies $\Pi(p) = \Pi(p')$.*

Lemma 4.8. *The rewrite relation $\to_{\mathsf{R}}$ is confluent and terminating. Every closed production term has a numeral as its unique normal form.*

Theorem 4.9. *For all $p \in \mathcal{P}$, it holds that $\Pi(p) = k$, where $\underline{k}$ is the uniquely determined $\to_{\mathsf{R}}$-normal form of p.*

5 Translation into Production Terms

In this section we define a translation function that maps, for every flat or friendly nesting stream specification $\mathcal{T}$, the root M_0 of $\mathcal{T}$ to a production term $[\mathsf{M}_0]$ with the property that, in the case that $\mathcal{T}$ is flat (friendly nesting), the data-oblivious production of M_0 in $\mathcal{T}$ equals (is bounded from from below by) the production of $[\mathsf{M}_0]$.

5.1 Function Layer Translation

As a first step of the translation, we describe how for a stream function symbol f in a flat, or friendly nesting, stream specification $\mathcal{T}$ a gate γ_{f} can be obtained that represents the data-oblivious lower bound on the production function of f, a periodically increasing function. This is done by solving, for each argument place in a stream function f, a finite 'io-term specification' that is extracted from an infinite one which precisely determines the data-oblivious lower bound in that argument place.

Definition 5.1. Let $\mathcal{X}$ be a set of variables. The set of *io-term specification expressions over $\mathcal{X}$* is defined by the following grammar:

$$E ::= X \mid -E \mid +E \mid E \wedge E$$

where $X \in \mathcal{X}$. Guardedness is defined by induction: An io-term specification expression is called *guarded* if it is of one of the forms $-E_0$, or $+E_0$, or if it is of the form $E_1 \wedge E_2$ for guarded E_1 and E_2.

Suppose that $\mathcal{X} = \{X_\alpha \mid \alpha \in A\}$ for some (countable) set A. Then by an *io-term specification over (the set of recursion variables) $\mathcal{X}$* we mean a family $\{X_\alpha = E_\alpha\}_{\alpha \in A}$ of *recursion equations*, where, for all $\alpha \in A$, E_α is an io-term specification expression over $\mathcal{X}$. Let $\mathcal{E}$ be an io-term specification. If $\mathcal{E}$ consists of finitely many recursion equations, then it is called *finite*. $\mathcal{E}$ is called *guarded* (*weakly guarded*) if the right-hand side of every recursion equation in $\mathcal{E}$ is guarded (or respectively, can be rewritten, using equational logic and the equations of $\mathcal{E}$, to a guarded io-term specification expression).

For an io-term specification expression $e \in E$ over $\mathcal{X}$ and an evalution $\mathcal{V} : \mathcal{X} \to (\overline{\mathbb{N}} \to \overline{\mathbb{N}})$ we define $e^{\mathcal{V}}$ by:

$$X^{\mathcal{V}} = \mathcal{V}(X) \qquad f \wedge g = \inf(f^{\mathcal{V}}, g^{\mathcal{V}})$$
$$+f = 1 + f^{\mathcal{V}} \qquad -f = f^{\mathcal{V}'}$$

where $\mathcal{V}'(X)(0) = 0$ and $\mathcal{V}'(X)(1+n) := \mathcal{V}(X)(n)$.

Let $\mathcal{E} = \{X_\alpha = E_\alpha\}_{\alpha \in A}$ be an io-term specification and X a recursion variable of $\mathcal{E}$. A function $f : \mathbb{N} \to \overline{\mathbb{N}}$ is called *solution of $\mathcal{E}$ for X* if there exists $\mathcal{V} : \mathcal{X} \to (\mathbb{N} \to \overline{\mathbb{N}})$ such that $\mathcal{V}(X) = f$, and all of the recursion equations in $\mathcal{E}$ are true statements, that is, $\forall \alpha.\ \mathcal{V}(X_\alpha) = E_\alpha^{\mathcal{V}}$.

Lemma 5.2. *For every weakly guarded io-term specification $\mathcal{E}$ and recursion variable X of $\mathcal{E}$, there exists a unique non-decreasing solution $f : \mathbb{N} \to \overline{\mathbb{N}}$ of $\mathcal{E}$ for X. Moreover, if $\mathcal{E}$ is finite then f is periodically increasing, and an io-term representation of f can be computed from $\mathcal{E}$ and X.*

Let $\mathcal{T}$ be an stream specification, and $\mathsf{f} \in \Sigma_S$. By exhaustiveness of $\mathcal{T}$ for f, there is at least one defining rule for f in $\mathcal{T}$. Since $\mathcal{T}$ is a constructor stream TRS, it follows that every defining rule ρ for f is of the form:

$$\mathsf{f}(p_1, \ldots, p_{ar_s(\mathsf{f})}, q_1, \ldots, q_{ar_d(\mathsf{f})}) \to u_1 : \ldots : u_m : u \quad (\rho)$$

with $p_1, \ldots, p_{ar_s(\mathsf{f})} \in Ter(\mathcal{C}(\Sigma))_S$, $q_1, \ldots, q_{ar_d(\mathsf{f})} \in Ter(\mathcal{C}(\Sigma))_D$, $u_1, \ldots, u_m \in Ter(\Sigma)_D$ and $u \in Ter(\Sigma)_S$ where $root(u) \neq$ ':'. We use $out(\rho) := m$ to denote the *production* of ρ. Moreover, p_i is of the form $s_{i,1} : \ldots : s_{i,n_i} : \sigma_i$, and we use $in(\rho, i) := n_i$ to denote the *consumption of ρ at the i-th position*. If ρ is non-nesting then either $u \equiv \sigma_j$ or $u \equiv \mathsf{g}(\vec{t}_1 : \sigma_{\phi(1)}, \ldots, \vec{t}_{ar_s(\mathsf{g})} : \sigma_{\phi(ar_s(\mathsf{g}))}, w_1, \ldots, w_{ar_d(\mathsf{g})})$ where $\vec{t}_i : \sigma_i$ is shorthand for $t_{i,1}:\ldots:t_{i,m_i}:\sigma_i$, and $\phi : \{1, \ldots, ar_s(\mathsf{g})\} \to \{1, \ldots, ar_s(\mathsf{f})\}$ a function used for permuting and replicating stream arguments. We use $su(\rho, i) := m_i$ to denote 'additional supply' to position $1 \leq i \leq ar_s(\mathsf{g})$ of g. If ρ is simple, then $s_{i,j}$ and q_k are data variables.

Definition 5.3. Let $\mathcal{T}$ be a flat stream specification. On Σ_{fun} we define the relation:

$$\rightsquigarrow := \{\langle root(l), root(r)\rangle \mid l \to r \in R_{fun}\} \cap (\Sigma_{fun} \times \Sigma_{fun}).$$

A symbol $\mathsf{f} \in \Sigma_{fun}$ is called *weakly guarded in $\mathcal{T}$* if f is strongly normalising with respect to $\rightsquigarrow$ and called *unguarded*, otherwise.

Definition 5.4. Let $\mathcal{T} = \langle \Sigma, R\rangle$ be a flat stream specification. Based on the set:

$$A := \{\langle - \rangle, \langle + \rangle, \langle -+ \rangle\} \cup \left\{ \begin{matrix} \langle \mathsf{f}, i, q\rangle, \\ \langle \mathsf{f}, i, q, \rho\rangle \end{matrix} \;\middle|\; \begin{matrix} \mathsf{f} \in \Sigma_{fun}, i, q \in \mathbb{N}, 0 \leq i \leq ar_s(\mathsf{f}), \\ \rho \text{ defining rule for } \mathsf{f} \end{matrix} \right\}$$

we define the io-term specification $\mathcal{E}_\mathcal{T} = \{X_\alpha = E_\alpha\}_{\alpha\in A}$ by listing the equations of $\mathcal{E}_\mathcal{T}$. We start with the equations $X_{\langle -\rangle} = -X_{\langle -\rangle}$, $X_{\langle +\rangle} = +X_{\langle +\rangle}$, $X_{\langle -+\rangle} = -+X_{\langle -+\rangle}$. Then let, for all $\mathsf{f} \in \Sigma_{fun}$, $1 \le i \le ar_s(\mathsf{f})$, $q \in \mathbb{N}$:

$$X_{\langle \mathsf{f},i,q\rangle} = \begin{cases} \bigwedge_{\rho\in R^\mathsf{f}} X_{\langle \mathsf{f},i,q,\rho\rangle} & \text{if } \mathsf{f} \text{ is weakly guarded in } \mathcal{T} \\ X_{\langle -\rangle} & \text{if } \mathsf{f} \text{ is unguarded in } \mathcal{T}. \end{cases}$$

For the defining equations for $X_{\langle \mathsf{f},i,q,\rho\rangle}$, we distinguish the two possible forms the rule ρ can have (see page 7):

$$\mathsf{f}(\vec{s}_1{:}\sigma_1, \ldots, \vec{s}_{ar_s(\mathsf{f})}{:}\sigma_{ar_s(\mathsf{f})}, v_1, \ldots, v_{ar_d(\mathsf{f})}) \to u_1{:}\ldots{:}u_m{:}u;$$

where u is of one of the following forms:

$$u \equiv \begin{cases} \mathsf{g}(\vec{t}_1 : \sigma_{\phi(1)}, \ldots, \vec{t}_{ar_s(\mathsf{g})} : \sigma_{\phi(ar_s(\mathsf{g}))}) & \text{case (a),} \\ u \equiv \sigma_j & \text{case (b).} \end{cases}$$

We agree $\bigwedge_{j\in A} E_j$ to be $X_{\langle +\rangle}$ if $A = \varnothing$, and to be E_j if $A = \{j\}$. Then we let:

$$X_{\langle \mathsf{f},i,q,\rho\rangle} = -^{in(\rho,i)\dot{-}q}+^{out(\rho)}$$
$$\begin{cases} \bigwedge_{j\in\phi^{-1}(i)} X_{\langle \mathsf{g},j,(q\dot{-}in(\rho,i))+su(\rho,i)\rangle} & \text{for (a)} \\ +^{q\dot{-}in(\rho,i)}X_{\langle -+\rangle} & \text{for (b) if } j = i \\ X_{\langle +\rangle} & \text{for (b) if } j \ne i \end{cases}$$

Note that the io-term specification $\mathcal{E}_\mathcal{T}$ in the definition is infinite. However, building up this translation can be stopped after finitely many steps: if on a path from $X_{\langle \mathsf{f},i,q\rangle}$ to some $X_{\langle \mathsf{f},i,q'\rangle}$ with $q < q'$ only $+$'s are produced, then stop with $X_{\langle \mathsf{f},i,q'\rangle} = +X_{\langle \mathsf{f},i,q'\rangle}$. It can be shown that on every path such an increasing loop will be encountered.

Definition 5.5. Let $\mathcal{T} = \langle \Sigma, R\rangle$ be a flat stream specification. For each $\mathsf{f} \in \Sigma_{fun}$ we define the *translation of* f as the gate $[\mathsf{f}] := \mathsf{gate}([\mathsf{f}]_1, \ldots, [\mathsf{f}]_{ar_s(\mathsf{f})})$, where, for $1 \le i \le ar_s(\mathsf{f})$, $[\mathsf{f}]_i$ is the io-term representing the unique solution for $X_{\langle \mathsf{f},i,0\rangle}$ of the weakly guarded io-term specification $\mathcal{E}_\mathcal{T}$ defined in Def. 5.4.

Definition 5.6. Let $\mathcal{T}$ be a friendly nesting stream specification. The translation of each $\mathsf{f} \in \Sigma_{fun}$ with friendly nesting defining rules is $[\mathsf{f}] := \mathsf{gate}(\overline{-+}, \ldots, \overline{-+})$, a gate of arity $ar_s(\mathsf{f})$. The remaining symbols in Σ_{fun} are translated according to Def. 5.5.

Since friendly nesting functions are translated into gates the boxes of which contain $\overline{-+}$, their production is bound from below by '$\min$'. These bounds are not tight, but can be used to show productivity of examples like $\mathsf{X} \to \mathsf{0} : \mathsf{f}(\mathsf{X})$ with $\mathsf{f}(x : \sigma) \to x : \mathsf{f}(\mathsf{f}(\sigma))$.

Lemma 5.7. *There is an algorithm that, on the input of a flat, or friendly nesting, stream specification $\mathcal{T}$, a symbol $\mathsf{f} \in \Sigma_{fun}$, and $1 \le i \le ar_s(\mathsf{f})$, computes a weakly guarded, finite io-term specification with $[\mathsf{f}]_i$ as its solution.*

Example 5.8. Consider a flat stream specification consisting of the following rules:

$$\mathsf{f}(x : \sigma) \to x : \mathsf{g}(\sigma, \sigma, \sigma)\,,$$
$$\mathsf{g}(x_1 : x_2 : \sigma, \tau, \upsilon) \to x_1 : \mathsf{g}(x_2 : \tau, x_2 : \upsilon, x_2 : \sigma)\,.$$

The translation of f is $[\mathsf{f}] = \mathsf{gate}([\mathsf{f}]_1)$, where $[\mathsf{f}]_1$ is the unique solution for $X_{\langle \mathsf{f},1,0\rangle}$ of the io-term specification $\mathcal{E}_\mathcal{T}$:

$$X_{\langle \mathsf{f},1,0\rangle} = -+(X_{\langle \mathsf{g},1,0\rangle} \wedge X_{\langle \mathsf{g},2,0\rangle} \wedge X_{\langle \mathsf{g},3,0\rangle})$$
$$X_{\langle \mathsf{g},1,0\rangle} = --+X_{\langle \mathsf{g},3,1\rangle} \quad X_{\langle \mathsf{g},1,1\rangle} = -+X_{\langle \mathsf{g},3,1\rangle}$$
$$X_{\langle \mathsf{g},1,q\rangle} = +X_{\langle \mathsf{g},3,q-1\rangle} \quad (q \ge 2)$$
$$X_{\langle \mathsf{g},2,q\rangle} = +X_{\langle \mathsf{g},1,q+1\rangle} \quad (q \in \mathbb{N})$$
$$X_{\langle \mathsf{g},3,q\rangle} = +X_{\langle \mathsf{g},2,q+1\rangle} \quad (q \in \mathbb{N})$$

By the algorithm referred to in Lem. 5.7 this infinite specification can be turned into a finite one. The 'non-consuming pseudocycle' $X_{\langle \mathsf{g},3,1\rangle} \overset{+++}{\longrightarrow} X_{\langle \mathsf{g},3,2\rangle}$ justifies the modification of $\mathcal{E}_\mathcal{T}$ by setting $X_{\langle \mathsf{g},3,1\rangle} = +X_{\langle \mathsf{g},3,1\rangle}$; likewise we set $X_{\langle \mathsf{g},3,0\rangle} = +X_{\langle \mathsf{g},3,0\rangle}$. Furthermore all equations not reachable from $X_{\langle \mathsf{f},1,0\rangle}$ are removed (garbage collection), and we obtain a finite specification, which, by Lem. 5.2, has a periodically increasing solution, with io-term-representation $[\mathsf{f}]_1 = \langle --+-, +\rangle$. The reader may try to calculate the gate corresponding to g, it is: $[\mathsf{g}] = \mathsf{gate}(--\overline{+}, +-\overline{+}, \overline{+})$.

Second, consider the flat stream specification with data constructor symbols $\mathsf{0}$ and $\mathsf{1}$:

$$\mathsf{f}(\mathsf{0} : \sigma) \to \mathsf{g}(\sigma)\,, \quad \mathsf{f}(\mathsf{1} : x : \sigma) \to x : \mathsf{g}(\sigma)\,,$$
$$\mathsf{g}(x : y : \sigma) \to x : y : \mathsf{g}(\sigma)\,,$$

denoted ρ_{f0}, ρ_{f1}, and ρ_g, respectively. Then, $[\mathsf{f}]_1$ is the solution for $X_{\langle \mathsf{f},1,0\rangle}$ of

$$X_{\langle \mathsf{f},1,0\rangle} = X_{\langle \mathsf{f},1,0,\rho_{\mathsf{f0}}\rangle} \wedge X_{\langle \mathsf{f},1,0,\rho_{\mathsf{f1}}\rangle}$$
$$X_{\langle \mathsf{f},1,0,\rho_{\mathsf{f0}}\rangle} = -X_{\langle \mathsf{g},1,0\rangle} \quad X_{\langle \mathsf{f},1,0,\rho_{\mathsf{f1}}\rangle} = --+X_{\langle \mathsf{g},1,0\rangle}$$
$$X_{\langle \mathsf{g},1,0\rangle} = --++X_{\langle \mathsf{g},1,0\rangle}\,.$$

Given this finite specification, we can compute its io-term-solution as $[\mathsf{f}]_1 = \langle --, -+\rangle$.

The lemma below states that the gate translation $[\mathsf{f}]$ defined in Def. 5.6 of a stream function f indeed models the data-oblivious lower bound on the production function on f in the case of a flat specification, and represents a periodically increasing function lower that the production function of f in case of a friendly nesting specification.

Lemma 5.9. *Let $\mathcal{T}$ be a stream definition, and let $\mathsf{f} \in \Sigma_{fun}$.*

(i) If $\mathcal{T}$ is flat, then it holds: $\pi_{[\mathsf{f}]} = \underline{do}_\mathcal{T}(\mathsf{f})$.
Hence, $\underline{do}_\mathcal{T}(\mathsf{f})$ is a periodically increasing function.

(ii) If $\mathcal{T}$ is friendly nesting, then it holds: $\pi_{[\mathsf{f}]} \le \underline{do}_\mathcal{T}(\mathsf{f})$.

5.2 Stream Layer Translation

In the second step, we define a translation of the stream constants in a flat or friendly nesting stream specification into production terms. Here the idea is that the recursive definition of a stream constant M is unfolded step by step; the terms thus arising are translated according to their structure using the gates defined in Subsec. 5.1 of the stream function symbols encountered; whenever a stream constant is met that has been unfolded before, the translation stops after establishing a binding to a μ-binder created earlier.

Definition 5.10. Let $\mathcal{T}$ be a stream specification, and let $\mathcal{F} = \{\gamma_{\mathsf{f}}\}_{\mathsf{f}\in\Sigma_{fun}}$ a family of gates associated with the symbols in Σ_{fun}. Then, for every $\mathsf{M} \in \Sigma_{str}$ with defining rule $\rho_{\mathsf{M}} \equiv \mathsf{M} \to rhs_{\mathsf{M}}$ and a set α of stream constant symbols, we recursively define the translation of M into a production term $[\mathsf{M}]^{\mathcal{F}}_{\alpha}$ with respect to α and $\mathcal{F}$:

$$[\mathsf{M}]^{\mathcal{F}}_{\alpha} := \begin{cases} \mu M.[rhs_{\mathsf{M}}]^{\mathcal{F}}_{\alpha\cup\{\mathsf{M}\}} & \text{if } \mathsf{M} \notin \alpha \\ M & \text{if } \mathsf{M} \in \alpha \end{cases}$$

$$[t : u]^{\mathcal{F}}_{\alpha} := 1 + [u]^{\mathcal{F}}_{\alpha}$$

$$[\mathsf{f}(u_1, \ldots, u_{ar_s(\mathsf{f})}, t_1, \ldots, t_{ar_d(\mathsf{f})})]^{\mathcal{F}}_{\alpha} := \gamma_{\mathsf{f}}([u_1]^{\mathcal{F}}_{\alpha}, \ldots, [u_{ar_s(\mathsf{f})}]^{\mathcal{F}}_{\alpha})$$

For all $\mathsf{M} \in \Sigma_{str}$, the *production term translation* $[\mathsf{M}]^{\mathcal{F}}$ *of* M *with respect to* $\mathcal{F}$ is defined as: $[\mathsf{M}]^{\mathcal{F}} := [\mathsf{M}]^{\mathcal{F}}_{\varnothing}$.

The following lemmas are the basis of our main results in Sec. 6. The translation of a stream constant M using data-obliviously tight gates yields a production term that exactly represents the data-oblivous production of M.

Lemma 5.11. *Let $\mathcal{T} = \langle\Sigma, R\rangle$ be a stream specification, and let $\mathcal{F} = \{\gamma_{\mathsf{f}}\}_{\mathsf{f}\in\Sigma_{fun}}$ be a family of gates such that for all $\mathsf{f} \in \Sigma_{fun}$: $\pi_{[\mathsf{f}]} = \underline{do}_{\mathcal{T}}(\mathsf{f})$ (that is, the gate γ_{f} models the data-oblivious lower bound of f).*

Then for all $\mathsf{M} \in \Sigma_{str}$ it holds: $\Pi([\mathsf{M}]^{\mathcal{F}}) = \underline{do}_{\mathcal{T}}(\mathsf{M})$. Consequently, $\mathcal{T}$ is data-obliviously productive if and only if $\Pi([\mathsf{M}_0]^{\mathcal{F}}) = \infty$.

Lemma 5.12. *Let $\mathcal{T} = \langle\Sigma, R\rangle$ be a stream specification, and let $\mathcal{F} = \{\gamma_{\mathsf{f}}\}_{\mathsf{f}\in\Sigma_{fun}}$ be a family of gates such that for all $\mathsf{f} \in \Sigma_{fun}$: $\pi_{[\mathsf{f}]} \leq \nu_{\mathsf{f}}$ (that is, the gate γ_{f} bounds the production of f from below)*

Then for all $\mathsf{M} \in \Sigma_{str}$ it holds: $\Pi([\mathsf{M}]^{\mathcal{F}}) \leq \Pi_{\mathcal{T}}(\mathsf{M})$. It follows that $\mathcal{T}$ is productive if $\Pi([\mathsf{M}_0]^{\mathcal{F}}) = \infty$.

6 Deciding Data-Oblivious Productivity

In this subsection we assemble our results concerning decision of data-oblivious productivity, and automatable recognition of productivity. We define methods for:

(DOP) deciding data-oblivious productivity of flat,

(DP) deciding productivity of pure (Def. 6.4 below),

(RP) recognising productivity of friendly nesting

stream specifications, proceeding in the following steps:

(i) Take as input a (DOP) flat, (DP) pure, or (RP) friendly nesting stream specification $\mathcal{T} = \langle\Sigma, R\rangle$.

(ii) Compute the translation of stream function symbols into gates $\mathcal{F} := \{[\mathsf{f}]\}_{\mathsf{f}\in\Sigma_{fun}}$, (Def. 5.5 and 5.6).

(iii) Construct the production term $[\mathsf{M}_0]^{\mathcal{F}}$ of the root M_0 of $\mathcal{T}$ with respect to the family of gates $\mathcal{F}$, see Def. 5.10.

(iv) Compute the production $\underline{k}$ of $[\mathsf{M}_0]^{\mathcal{F}}$ using $\to_{\mathsf{R}}$ (Def. 4.6).

(v) Give the following output:

(DOP) "$\mathcal{T}$ is data-obliviously productive" if $k = \infty$ else "$\mathcal{T}$ is not data-obliviously productive".

(DP) "$\mathcal{T}$ is productive" if $k = \infty$; else "$\mathcal{T}$ is not productive, because M_0 produces k elements only".

(RP) "$\mathcal{T}$ is productive" if $k = \infty$; else "don't know",

Note that all of these steps are automatable.

Our main result states the decidability of data-oblivious productivity for flat stream specifications. It rests on the fact that, as a consequence of Lem. 5.9, (i), and Lem. 5.11, the algorithm DOP obtains the data-oblivious production of M_0 in $\mathcal{T}$ in step (iv), when applied to a flat specification.

Theorem 6.1. *The algorithm* DOP *decides data-oblivious productivity of flat stream specifications.*

Since by Prop. 3.5 data-oblivious productivity implies productivity, we also obtain a computable, data-obliviously optimal, sufficient condition for productivity, which cannot be improved by any other data-oblivious analysis.

Corollary 6.2. *A flat stream specification $\mathcal{T}$ is productive if the algorithm* DOP *recognizes $\mathcal{T}$ as data-obliviously productive.*

Additionally, we obtain a computable, sufficient condition for productivity of friendly nesting stream specifications. This criterion is justified by the fact that due to Lem. 5.9, (ii), and Lem. 5.12, the algorithm RP obtains a lower bound on the production of M_0 in $\mathcal{T}$ in step (iv).

Theorem 6.3. *A friendly nesting stream specification $\mathcal{T}$ is productive if the algorithm* RP *recognizes $\mathcal{T}$ as productive.*

Finally, we extend the decidability result for productivity for 'pure' stream specifications in [2] by relaxing the syntactic constraints imposed on these specifications.

Definition 6.4. A stream specification $\mathcal{T}$ is called *pure* if it is flat and for every stream function symbol $\mathsf{f} \in \Sigma_{fun}$ the abstractions ($[\![\ell]\!] \to [\![r]\!]$) of the defining rules of f coincide (modulo renaming of variables).

The above definition generalizes the specifications called 'pure' in [2] in four ways concerning the defining rules of stream functions: First, the requirement of right-linearity of stream variables is dropped, allowing for rules like $\mathsf{f}(\sigma) \to \mathsf{g}(\sigma, \sigma)$. Second, 'additional supply' to the stream arguments is allowed. For instance, in a rule like $\mathsf{diff}(x{:}y{:}\sigma) \to \mathsf{x}(x, y) : \mathsf{diff}(y : \sigma)$, the variable y is 'supplied' to the recursive call of diff. Third, the use of non-productive functions is allowed now, dismissing the earlier requirement of 'weakly guardedness'. Finally, defining rules for stream function symbols may use a restricted form of pattern matching as long as, for every stream function f, the data-oblivious consumption/production behaviour of all defining rules for f is the same.

Theorem 6.5. *The algorithm* DP *decides productivity of pure stream specifications.*

7 Conclusion and Further Work

In order to formalize quantitative approaches for recognizing productivity of stream specifications, we defined the notion of data-oblivious rewriting and investigated data-oblivious productivity. For the syntactic class of flat stream specifications (that employ pattern matching on data), we devised a decision algorithm for data-oblivious productivity. In this way we settled the productivity recognition problem for flat stream specifications from a data-oblivious perspective. For the even larger class including friendly nesting stream function rules, we obtained a computable sufficient condition for productivity. For the subclass of pure stream specifications (a substantial extension of the class given in [2]) we showed that productivity and data-oblivious productivity coincide, and thereby obtained a decision algorithm for productivity for pure specifications.

We have implemented in Haskell the decision algorithm for data-oblivious productivity. This tool, together with much more information including a manual, examples, our related papers, and a comparison of our criteria with those of [3, 7, 1] can be found at our web page that is available at `http://infinity.few.vu.nl/productivity`. The reader is invited to experiment with our tool.

It is not possible to obtain a data-obliviously optimal criterion for non-productivity of flat specifications in an analogous way to how we established such a criterion for productivity. This is because the upper bounds on the data-oblivious production of stream functions f in flat specifications are in general not of the form $\overline{do}(\mathsf{f}) = \min(\ldots)$ and hence cannot be modelled by gates. Consider the example:

$$\mathsf{f}(x : \sigma, \tau) \to x : \mathsf{f}(\sigma, \tau)\,, \quad \mathsf{f}(\sigma, y : \tau) \to y : \mathsf{f}(\sigma, \tau)\,,$$

with $\overline{do}(\mathsf{f})(n_1, n_2) = n_1 + n_2$ as its data-oblivious upper bound. This example is not orthogonal, but it foreshadows what the following orthogonal example does:

$$\mathsf{f}(0 : x : \sigma, y : \tau) \to x : \mathsf{f}(\sigma, \tau)\,,$$
$$\mathsf{f}(1 : \sigma, x : y : \tau) \to y : \mathsf{f}(\sigma, \tau)\,.$$

Currently we are developing a method that goes beyond a data-oblivious analysis, one that would, e.g., prove productivity of the example B given in the introduction. Moreover, we study a refined production calculus that accounts for the delay of evaluation of stream elements, in order to obtain a faithful modelling of lazy evaluation, needed for example for the example S on page 3, where the first element depends on a 'future' expansion of S.

Acknowledgement. We thank Jan Willem Klop, Vincent van Oostrom, and Roel de Vrijer for useful discussions.

References

[1] W. Buchholz. A Term Calculus for (Co-)Recursive Definitions on Streamlike Data Structures. *Annals of Pure and Applied Logic*, 136(1-2):75–90, 2005.

[2] J. Endrullis, C. Grabmayer, D. Hendriks, A. Isihara, and J.W. Klop. Productivity of Stream Definitions. In *Proceedings of FCT 2007*, number 4639 in LNCS, pages 274–287. Springer, 2007.

[3] J. Hughes, L. Pareto, and A. Sabry. Proving the Correctness of Reactive Systems Using Sized Types. In *POPL '96*, pages 410–423, 1996.

[4] D. Kapur, P. Narendran, Rosenkrantz. D.J., and H. Zhang. Sufficient-completeness, ground-reducibility and their complexity. *Acta Inf.*, 28(4):311–350, 1991.

[5] A. Salomaa. *Jewels of Formal Language Theory*. Pitman Publishing, 1981.

[6] B.A. Sijtsma. On the Productivity of Recursive List Definitions. *ACM Transactions on Programming Languages and Systems*, 11(4):633–649, 1989.

[7] A. Telford and D. Turner. Ensuring Streams Flow. In *AMAST*, pages 509–523, 1997.

[8] Terese. *Term Rewriting Systems*, volume 55 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2003.

Appendix: Examples

Productivity of all of the following examples is recognized fully automatic by a Haskell implementation of our decision algorithm for data-oblivious productivity. The tool and examples can be found at:
`http://infinity.few.vu.nl/productivity`.

Below, we list the output generated by the tool.

7.1 Example Fig. 2

The automated productivity prover has been applied to:

```
-- stream layer --
Q = a:R
R = b:c:f(R)

-- function layer --
f(a:s) = a:b:c:f(s)
f(b:s) = a:c:f(s)
f(c:s) = b:f(s)

-- data layer --
```

This stream specification is 'flat', we can decide data-oblivious productivity.

The translation of f is computed as follows:

$$\begin{aligned}[\mathsf{f}] &= \mathsf{min}([\mathsf{f}]_{1,0}) \\ [\mathsf{f}]_{1,0} &= \mu(\mathsf{f}_{1,0}).\wedge \begin{cases} -{+}{+}{+} \wedge \left\{ \mathsf{f}_{1,0} \right. \\ -{+}{+} \wedge \left\{ \mathsf{f}_{1,0} \right. \\ -{+} \wedge \left\{ \mathsf{f}_{1,0} \right. \end{cases} \\ &= \overline{-+} \end{aligned}$$

We translate Q into a pebbleflow net and collapse it to a source:

$$\begin{aligned}[\mathsf{Q}] &= \mu Q.\bullet(\mu R.\bullet(\bullet(\overline{-+}(R)))) \\ &\twoheadrightarrow_{\mathsf{R}} \mu Q.+\overline{-+}(\mu R.\bullet(\bullet(\overline{-+}(R)))) \\ &\twoheadrightarrow_{\mathsf{R}} \mu Q.+\overline{-+}(\mu R.+\overline{-+}(\bullet(\overline{-+}(R)))) \\ &\twoheadrightarrow_{\mathsf{R}} \mu Q.+\overline{-+}(\mu R.+\overline{-+}(+\overline{-+}(\overline{-+}(R)))) \\ &\twoheadrightarrow_{\mathsf{R}} \mu Q.+\overline{-+}(\mu R.+\overline{+-}(\overline{-+}(R))) \\ &\twoheadrightarrow_{\mathsf{R}} \mu Q.+\overline{-+}(\mu R.+\overline{+-}(R)) \\ &\twoheadrightarrow_{\mathsf{R}} \mu Q.+\overline{-+}(\infty) \\ &\twoheadrightarrow_{\mathsf{R}} \mu Q.\infty \\ &\twoheadrightarrow_{\mathsf{R}} \infty \end{aligned}$$

The specification of Q is productive.

We translate R into a pebbleflow net and collapse it to a source:

$$\begin{aligned}[\mathsf{R}] &= \mu R.\bullet(\bullet(\overline{-+}(R))) \\ &\twoheadrightarrow_{\mathsf{R}} \mu R.+\overline{-+}(\bullet(\overline{-+}(R))) \\ &\twoheadrightarrow_{\mathsf{R}} \mu R.+\overline{-+}(+\overline{-+}(\overline{-+}(R))) \\ &\twoheadrightarrow_{\mathsf{R}} \mu R.+\overline{+-}(\overline{-+}(R)) \\ &\twoheadrightarrow_{\mathsf{R}} \mu R.+\overline{+-}(R) \\ &\twoheadrightarrow_{\mathsf{R}} \infty \end{aligned}$$

The specification of R is productive.

7.2 Example Fig. 3

The automated productivity prover has been applied to:

```
-- stream layer --
Q = diff(M)
M = 0:zip(inv(M),tail(M))

-- function layer --
zip(x:s,t) = x:zip(t,s)
inv(x:s) = i(x):inv(s)
tail(x:s) = s
diff(x:y:s) = X(x,y):diff(y:s)

-- data layer --
i(0) = 1
i(1) = 0
X(0,0) = b
X(0,1) = a
X(1,0) = c
X(1,1) = b
```

This is a pure stream specification, we can decide productivity.

The translation of tail is computed as follows:

$$\begin{aligned}[\mathsf{tail}] &= \mathsf{min}([\mathsf{tail}]_{1,0}) \\ [\mathsf{tail}]_{1,0} &= \mu(\mathsf{tail}_{1,0}).\wedge \left\{ -\mu(x).-{+}x \right. \\ &= -\overline{-+} \end{aligned}$$

The translation of diff is computed as follows:

$$\begin{aligned}[\mathsf{diff}] &= \mathsf{min}([\mathsf{diff}]_{1,0}) \\ [\mathsf{diff}]_{1,0} &= \mu(\mathsf{diff}_{1,0}). \\ &\quad \wedge \left\{ -{-}{+} \wedge \left\{ \mu(\mathsf{diff}_{1,1}).\wedge \left\{ -{+} \wedge \left\{ \mathsf{diff}_{1,1} \right. \right. \right. \\ &= -\overline{-+} \end{aligned}$$

The translation of zip is computed as follows:

$$\begin{aligned}
[\mathsf{zip}] &= \mathsf{min}([\mathsf{zip}]_{1,0}[\mathsf{zip}]_{2,0}) \\
[\mathsf{zip}]_{1,0} &= \mu(\mathsf{zip}_{1,0}).\wedge\left\{-+\wedge\left\{\mu(\mathsf{zip}_{2,0}).\wedge\left\{+\wedge\left\{\mathsf{zip}_{1,0}\right.\right.\right.\right. \\
&= \overline{-++} \\
[\mathsf{zip}]_{2,0} &= \mu(\mathsf{zip}_{2,0}).\wedge\left\{+\wedge\left\{\mu(\mathsf{zip}_{1,0}).\wedge\left\{-+\wedge\left\{\mathsf{zip}_{2,0}\right.\right.\right.\right. \\
&= \overline{+-+}
\end{aligned}$$

The translation of inv is computed as follows:

$$\begin{aligned}
[\mathsf{inv}] &= \mathsf{min}([\mathsf{inv}]_{1,0}) \\
[\mathsf{inv}]_{1,0} &= \mu(\mathsf{inv}_{1,0}).\wedge\left\{-+\wedge\left\{\mathsf{inv}_{1,0}\right.\right. \\
&= \overline{-+}
\end{aligned}$$

We translate Q into a pebbleflow net and collapse it to a source:

$$\begin{aligned}
[\mathsf{Q}] &= \mu Q.-\overline{-+}(\mu M.\bullet(\mathsf{min}(\overline{-++}(\overline{-+}(M)), \\
&\qquad\qquad \overline{+-+}(-\overline{-+}(M))))) \\
&\twoheadrightarrow_{\mathsf{R}} \mu Q.-\overline{-+}(\mu M.\bullet(\mathsf{min}(\overline{-++}(M), \\
&\qquad\qquad +-\overline{-++}(M)))) \\
&\twoheadrightarrow_{\mathsf{R}} \mu Q.-\overline{-+}(\mu M.+\overline{-+}(\mathsf{min}(\overline{-++}(M), \\
&\qquad\qquad +-\overline{-++}(M)))) \\
&\twoheadrightarrow_{\mathsf{R}} \mu Q.-\overline{-+}(\mu M.\mathsf{min}(+\overline{-+}(\overline{-++}(M)), \\
&\qquad\qquad +\overline{-+}(+-\overline{-++}(M)))) \\
&\twoheadrightarrow_{\mathsf{R}} \mu Q.-\overline{-+}(\mu M.\mathsf{min}(\overline{+-+}(M), \\
&\qquad\qquad ++-\overline{-++}(M))) \\
&\twoheadrightarrow_{\mathsf{R}} \mu Q.-\overline{-+}(\mathsf{min}(\mu M.\overline{+-+}(M), \\
&\qquad\qquad \mu M.++-\overline{-++}(M))) \\
&\twoheadrightarrow_{\mathsf{R}} \mu Q.\mathsf{min}(-\overline{-+}(\mu M.\overline{+-+}(M)), \\
&\qquad\qquad -\overline{-+}(\mu M.++-\overline{-++}(M))) \\
&\twoheadrightarrow_{\mathsf{R}} \mathsf{min}(\mu Q.-\overline{-+}(\mu M.\overline{+-+}(M)), \\
&\qquad\qquad \mu Q.-\overline{-+}(\mu M.++-\overline{-++}(M))) \\
&\twoheadrightarrow_{\mathsf{R}} \mathsf{min}(\mu Q.-\overline{-+}(\infty), \mu Q.-\overline{-+}(\infty)) \\
&\twoheadrightarrow_{\mathsf{R}} \mathsf{min}(\mu Q.\infty, \mu Q.\infty) \\
&\twoheadrightarrow_{\mathsf{R}} \mathsf{min}(\infty, \infty) \\
&\twoheadrightarrow_{\mathsf{R}} \infty
\end{aligned}$$

The specification of Q is productive.

We translate M into a pebbleflow net and collapse it to a source:

$$\begin{aligned}
[\mathsf{M}] &= \mu M.\bullet(\mathsf{min}(\overline{-++}(\overline{-+}(M)), \overline{+-+}(-\overline{-+}(M)))) \\
&\twoheadrightarrow_{\mathsf{R}} \mu M.\bullet(\mathsf{min}(\overline{-++}(M), +-\overline{-++}(M))) \\
&\twoheadrightarrow_{\mathsf{R}} \mu M.+\overline{-+}(\mathsf{min}(\overline{-++}(M), +-\overline{-++}(M))) \\
&\twoheadrightarrow_{\mathsf{R}} \mu M.\mathsf{min}(+\overline{-+}(\overline{-++}(M)), +\overline{-+}(+-\overline{-++}(M))) \\
&\twoheadrightarrow_{\mathsf{R}} \mu M.\mathsf{min}(\overline{+-+}(M), ++-\overline{-++}(M)) \\
&\twoheadrightarrow_{\mathsf{R}} \mathsf{min}(\mu M.\overline{+-+}(M), \mu M.++-\overline{-++}(M)) \\
&\twoheadrightarrow_{\mathsf{R}} \mathsf{min}(\infty, \infty) \\
&\twoheadrightarrow_{\mathsf{R}} \infty
\end{aligned}$$

The specification of M is productive.

7.3 Example from Page 5

This example contains no stream specification. Note that the algorithm computes indeed the exact data-oblivious bound of the stream function f.

```
-- stream layer --


-- function layer --
f(s) = g(s,s)
g(0:y:s,x:t) = 0:0:g(s,t)
g(1:s,x1:x2:x3:x4:t) = 0:0:0:0:0:g(s,t)

-- data layer --
```

This is a flat stream specification. We can decide data-oblivious productivity.

The translation of f is computed as follows:

$$\begin{aligned}
[\mathsf{f}] &= \mathsf{min}([\mathsf{f}]_{1,0}) \\
[\mathsf{f}]_{1,0} &= \mu(\mathsf{f}_{1,0}).\wedge\left\{\wedge\left\{\begin{array}{l}\mu(\mathsf{g}_{1,0}).\wedge\left\{\begin{array}{l}--++\wedge\left\{\mathsf{g}_{1,0}\right. \\ -+++++\wedge\left\{\mathsf{g}_{1,0}\right.\end{array}\right. \\ \mu(\mathsf{g}_{2,0}).\wedge\left\{\begin{array}{l}-++\wedge\left\{\mathsf{g}_{2,0}\right. \\ ----+++++\wedge\left\{\mathsf{g}_{2,0}\right.\end{array}\right.\end{array}\right.\right. \\
&= ----++-++-+--++-+\overline{-++-}
\end{aligned}$$

The translation of g is computed as follows:

$$
\begin{aligned}
[\mathsf{g}] &= \mathsf{min}([\mathsf{g}]_{1,0}[\mathsf{g}]_{2,0}) \\
[\mathsf{g}]_{1,0} &= \mu(\mathsf{g}_{1,0}).\wedge \begin{cases} --++ \wedge \left\{ \mathsf{g}_{1,0} \right. \\ -+++++ \wedge \left\{ \mathsf{g}_{1,0} \right. \end{cases} \\
&= \overline{--++} \\
[\mathsf{g}]_{2,0} &= \mu(\mathsf{g}_{2,0}).\wedge \begin{cases} -++ \wedge \left\{ \mathsf{g}_{2,0} \right. \\ ----+++++ \wedge \left\{ \mathsf{g}_{2,0} \right. \end{cases} \\
&= --\overline{--++-++-+}
\end{aligned}
$$