# Part 3: Extended Formats

Jörg Endrullis Clemens Grabmayer Dimitri Hendriks

Vrije Universiteit Amsterdam – Universiteit Utrecht – Vrije Universiteit Amsterdam

ISR 2010, Utrecht University

July 7, 2010

# Overview

1. Extensions of PSF
2. Results for PSF-Extensions
3. Automatic Sequences and Morphic Streams

# Overview

1. Extensions of PSF

2. Results for PSF-Extensions

3. Automatic Sequences and Morphic Streams

# Extending PSF

**Example (poor man's pat-mat)**

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

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

$$\mathsf{morse} \twoheadrightarrow_{\omega} 0 : 1 : 1 : 0 : 1 : 0 : 0 : 1 : 1 : 0 : 0 : 1 : 0 : 1 : 1 : 0 : \ldots$$

# Extending PSF

**Example (translation into PSF)**

| | |
|---|---|
| $\mathsf{morse} \to 0 : 1 : \mathsf{f}(\mathsf{tail}(\mathsf{morse}))$<br>$\mathsf{f}(x : xs) \to \mathsf{a}(x) : \mathsf{b}(x) : \mathsf{f}(xs)$<br>$\mathsf{tail}(x : xs) \to xs$ | *stream layer* |
| $\mathsf{a}(0) \to 0 \qquad \mathsf{b}(0) \to 1$<br>$\mathsf{a}(1) \to 1 \qquad \mathsf{b}(1) \to 0$ | *data layer* |

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

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

# Extending PSF

## Example (translation into PSF)

| | |
|---|---|
| $\mathsf{morse} \to 0 : 1 : \mathsf{f}(\mathsf{tail}(\mathsf{morse}))$<br>$\mathsf{f}(x : xs) \to x : \mathsf{b}(x) : \mathsf{f}(xs)$<br>$\mathsf{tail}(x : xs) \to xs$ | *stream layer* |
| $\mathsf{b}(0) \to 1$<br>$\mathsf{b}(1) \to 0$ | *data layer* |

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

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

# Extending PSF

**Example (translation into PSF)**

$$
\begin{array}{rcl}
\mathsf{morse} & \to & 0 : 1 : \mathsf{f}(\mathsf{tail}(\mathsf{morse})) \\
\mathsf{f}(x : xs) & \to & x : \mathsf{not}(x) : \mathsf{f}(xs) \qquad \textit{stream layer} \\
\mathsf{tail}(x : xs) & \to & xs \\
\hline
\mathsf{not}(0) & \to & 1 \\
\mathsf{not}(1) & \to & 0 \qquad \textit{data layer}
\end{array}
$$

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

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

# Extending PSF (in two steps)

- In pure$^{+}$ specifications, the rules for stream functions allow:
  - a restricted form of exhaustive pattern matching
  - duplication of stream variables
    $\mathsf{f}(xs) \to \mathsf{g}(xs, xs)$.
  - additional supply in stream variables
    $\mathsf{diff}(x : y : xs) \to \mathsf{xor}(x, y) : \mathsf{diff}(y : xs)$
  - use of non-productive stream functions
    $\mathsf{onlyread2}(x : y : xs) \to x : y : \mathsf{idle}(xs) \qquad \mathsf{idle}(xs) \to \mathsf{idle}(xs)$
- In flat specifications:
  - additionally feature: exhaustive pattern matching on constructors

# Extending PSF (in two steps)

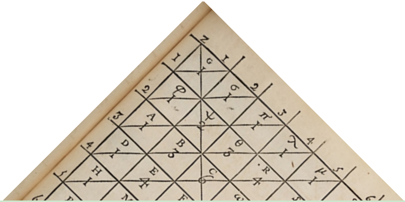
- In pure$^{+}$ specifications, the rules for stream functions allow:
  - a restricted form of exhaustive pattern matching
  - duplication of stream variables

    $\mathsf{f}(xs) \to \mathsf{g}(xs, xs)$.
  - additional supply in stream variables

    $\mathsf{diff}(x : y : xs) \to \mathsf{xor}(x, y) : \mathsf{diff}(y : xs)$
  - use of non-productive stream functions

    $\mathsf{onlyread2}(x : y : xs) \to x : y : \mathsf{idle}(xs) \qquad \mathsf{idle}(xs) \to \mathsf{idle}(xs)$

- In flat specifications:
  - additionally feature: exhaustive pattern matching on constructors



$$
\begin{array}{ccccccccc}
 & & & & 1 & & & & \\
 & & & 1 & & 1 & & & \\
 & & 1 & & 2 & & 1 & & \\
 & 1 & & 3 & & 3 & & 1 & \\
1 & & 4 & & 6 & & 4 & & 1
\end{array}
$$

## Example (Pascal's triangle)

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

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

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

# Stream Specifications

We formalise stream specifications as:

- $\{S, D\}$-sorted, orthogonal, constructor TRSs $\mathcal{R} = \langle \Sigma, R \rangle$
- $\Sigma = \Sigma_S \cup \Sigma_D$, where $\Sigma_S$ stream symbols and $\Sigma_D$ data symbols

## Definition (Stream Specification)

| | |
|---|---|
| $R_S$ | *stream layer* |
| $R_D$ | *data layer* |

- $R = R_S \cup R_D$
- $\mathsf{M}_0 \in \Sigma_S$ with arity 0, the root of $\mathcal{R}$
- $\langle \Sigma_D, R_D \rangle$ is a terminating, $D$-sorted TRS, the data layer of $\mathcal{R}$
- $\mathcal{R}$ is exhaustive

# Flat and pure$^+$ specifications

$\mathcal{R}$ is called flat: in defining rules for every stream function, say f:

- no nested occurrences of stream functions on right-hand sides.
- exhaustive pattern matching: every term $\mathsf{f}(t_1, \ldots, t_n)$ with $t_1, \ldots, t_n$ in constructor normal form is a redex.

$\mathcal{R}$ is called pure$^+$: $\mathcal{R}$ is flat, and for every stream function f:

- all defining rules for f all have the same data abstraction.

## Example

Flat:
$$\begin{aligned} \mathsf{g}(0 : x : xs) &\to x : x : \mathsf{g}(xs) \\ \mathsf{g}(1 : x : xs) &\to x : \mathsf{g}(xs) \\ \mathsf{g}(\bullet : \bullet : xs) &\to \bullet : \bullet : \mathsf{g}(xs) \\ \mathsf{g}(\bullet : \bullet : xs) &\to \bullet : \mathsf{g}(xs) \end{aligned}$$

Pure$^+$:
$$\begin{aligned} \mathsf{inv}(0 : xs) &\to 1 : \mathsf{inv}(xs) \\ \mathsf{inv}(1 : xs) &\to 0 : \mathsf{inv}(xs) \\ \mathsf{inv}(\bullet : xs) &\to \bullet : \mathsf{inv}(xs) \end{aligned}$$

# Flat and pure$^+$ specifications

$\mathcal{R}$ is called flat: in defining rules for every stream function, say f:

- no nested occurrences of stream functions on right-hand sides.
- exhaustive pattern matching: every term $\mathsf{f}(t_1, \ldots, t_n)$ with $t_1, \ldots, t_n$ in constructor normal form is a redex.

$\mathcal{R}$ is called pure$^+$: $\mathcal{R}$ is flat, and for every stream function f:

- all defining rules for f all have the same data abstraction.

## Example

Flat:
$$\mathsf{g}(0 : x : xs) \to x : x : \mathsf{g}(xs)$$
$$\mathsf{g}(1 : x : xs) \to x : \mathsf{g}(xs)$$
$$\mathsf{g}(\bullet : \bullet : xs) \to \bullet : \bullet : \mathsf{g}(xs)$$
$$\mathsf{g}(\bullet : \bullet : xs) \to \bullet : \mathsf{g}(xs)$$

Pure$^+$:
$$\mathsf{inv}(0 : xs) \to 1 : \mathsf{inv}(xs)$$
$$\mathsf{inv}(1 : xs) \to 0 : \mathsf{inv}(xs)$$
$$\mathsf{inv}(\bullet : xs) \to \bullet : \mathsf{inv}(xs)$$

# Flat and pure$^+$ specifications

$\mathcal{R}$ is called flat: in defining rules for every stream function, say f:

- no nested occurrences of stream functions on right-hand sides.
- exhaustive pattern matching: every term $\mathsf{f}(t_1, \ldots, t_n)$ with $t_1, \ldots, t_n$ in constructor normal form is a redex.

$\mathcal{R}$ is called pure$^+$: $\mathcal{R}$ is flat, and for every stream function f:

- all defining rules for f all have the same data abstraction.

## Example

Flat:
$$\begin{aligned}
\mathsf{g}(0 : x : xs) &\to x : x : \mathsf{g}(xs)\\
\mathsf{g}(1 : x : xs) &\to x : \mathsf{g}(xs)\\[4pt]
\mathsf{g}(\bullet : \bullet : xs) &\to \bullet : \bullet : \mathsf{g}(xs)\\
\mathsf{g}(\bullet : \bullet : xs) &\to \bullet : \mathsf{g}(xs)
\end{aligned}$$

Pure$^+$:
$$\begin{aligned}
\mathsf{inv}(0 : xs) &\to 1 : \mathsf{inv}(xs)\\
\mathsf{inv}(1 : xs) &\to 0 : \mathsf{inv}(xs)\\[4pt]
\mathsf{inv}(\bullet : xs) &\to \bullet : \mathsf{inv}(xs)
\end{aligned}$$

# Flat and pure$^+$ specifications

$\mathcal{R}$ is called flat: in defining rules for every stream function, say f:

- no nested occurrences of stream functions on right-hand sides.
- exhaustive pattern matching: every term $\mathsf{f}(t_1, \ldots, t_n)$ with $t_1, \ldots, t_n$ in constructor normal form is a redex.

$\mathcal{R}$ is called pure$^+$: $\mathcal{R}$ is flat, and for every stream function f:

- all defining rules for f all have the same data abstraction.

## Example

Flat:

$$\begin{aligned} \mathsf{g}(0 : x : xs) &\to x : x : \mathsf{g}(xs) \\ \mathsf{g}(1 : x : xs) &\to x : \mathsf{g}(xs) \end{aligned}$$

$$\begin{aligned} \mathsf{g}(\bullet : \bullet : xs) &\to \bullet : \bullet : \mathsf{g}(xs) \\ \mathsf{g}(\bullet : \bullet : xs) &\to \bullet : \mathsf{g}(xs) \end{aligned}$$

Pure$^+$:

$$\begin{aligned} \mathsf{inv}(0{:}xs) &\to 1{:}\mathsf{inv}(xs) \\ \mathsf{inv}(1{:}xs) &\to 0{:}\mathsf{inv}(xs) \end{aligned}$$

$$\mathsf{inv}(\bullet{:}xs) \to \bullet{:}\mathsf{inv}(xs)$$

# Stream Specification (pure)

## Example (Thue–Morse stream)

$$\mathsf{morse} \to 0 : \mathsf{zip}(\mathsf{inv}(\mathsf{morse}), \mathsf{tail}(\mathsf{morse}))$$

$$\mathsf{tail}(x : xs) \to xs$$

$$\mathsf{zip}(x : xs, ys) \to x : \mathsf{zip}(ys, xs)$$

$$\mathsf{inv}(x : xs) \to \mathsf{not}(x) : \mathsf{inv}(xs)$$

*stream layer*

$$\mathsf{not}(0) \to 1 \qquad \mathsf{not}(1) \to 0$$

*data layer*

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

# Stream Specification (pure$^+$)

## Example (Thue–Morse stream)

$$
\begin{aligned}
\mathsf{morse} &\to 0 : 1 : \mathsf{f}(\mathsf{tail}(\mathsf{morse})) \\
\mathsf{f}(0 : xs) &\to 0 : 1 : \mathsf{f}(xs) \\
\mathsf{f}(1 : xs) &\to 1 : 0 : \mathsf{f}(xs) \\
\mathsf{tail}(x : xs) &\to xs
\end{aligned}
$$

*stream layer*

*data layer*

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

# Stream Specification (flat, not pure$^{+}$)

## Example (Ternary Thue–Morse Stream)

$$
\begin{array}{rcl}
\mathsf{Q} & \to & \mathsf{a} : \mathsf{R} \\
\mathsf{R} & \to & \mathsf{b} : \mathsf{c} : \mathsf{f}(\mathsf{R}) \\
\mathsf{f}(\mathsf{a} : xs) & \to & \mathsf{a} : \mathsf{b} : \mathsf{c} : \mathsf{f}(xs) \\
\mathsf{f}(\mathsf{b} : xs) & \to & \mathsf{a} : \mathsf{c} : \mathsf{f}(xs) \\
\mathsf{f}(\mathsf{c} : xs) & \to & \mathsf{b} : \mathsf{f}(xs)
\end{array}
$$

*stream layer*

*data layer*

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

# Stream Specification (flat, not pure$^+$)

**Example (Pascal's triangle stream)**

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

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

# Friendly Nesting Stream Spec's

Friendly (nesting) specifications: the rules for stream functions are flat, or friendly (nesting).

A defining rule $\mathsf{f}(\vec{x}_1 : t_1, \dots, \vec{x}_n : t_n) \to \vec{d} : t$ is called friendly if:

- it consumes in each argument at most one stream element (all $\vec{x}_i$ have length $\leq 1$),
- it produces at least one stream element (length of $\vec{d}$ is $\geq 1$),
- the defining rules of stream function symbols on the rhs (in $t$) are friendly again.

## Example

$$\mathsf{f}(x : xs, ys) \to x : x : \mathsf{g}(\mathsf{f}(xs, x : ys))$$
$$\mathsf{g}(x : xs) \to x : \mathsf{g}(x : \mathsf{f}(xs, xs))$$

## Proposition

*pure* $\subsetneq$ *pure*$^+$ $\subsetneq$ *flat* $\subsetneq$ *friendly nesting*.

# Friendly Nesting Stream Spec's

Friendly (nesting) specifications: the rules for stream functions are flat, or friendly (nesting).

A defining rule $\mathsf{f}(\vec{x}_1 : t_1, \ldots, \vec{x}_n : t_n) \to \vec{d} : t$ is called friendly if:

- it consumes in each argument at most one stream element (all $\vec{x}_i$ have length $\leq 1$),
- it produces at least one stream element (length of $\vec{d}$ is $\geq 1$),
- the defining rules of stream function symbols on the rhs (in $t$) are friendly again.

## Example

$$\mathsf{f}(x : xs, ys) \to x : x : \mathsf{g}(\mathsf{f}(xs, x : ys))$$
$$\mathsf{g}(x : xs) \to x : \mathsf{g}(x : \mathsf{f}(xs, xs))$$

## Proposition

*pure* $\subsetneq$ *pure*$^+$ $\subsetneq$ *flat* $\subsetneq$ *friendly nesting.*

# Friendly Nesting Stream Spec's

Friendly (nesting) specifications: the rules for stream functions are flat, or friendly (nesting).

A defining rule $\mathsf{f}(\vec{x}_1 : t_1, \ldots, \vec{x}_n : t_n) \to \vec{d} : t$ is called friendly if:

- it consumes in each argument at most one stream element (all $\vec{x}_i$ have length $\leq 1$),
- it produces at least one stream element (length of $\vec{d}$ is $\geq 1$),
- the defining rules of stream function symbols on the rhs (in $t$) are friendly again.

## Example

$$\mathsf{f}(x : xs, ys) \to x : x : \mathsf{g}(\mathsf{f}(xs, x : ys))$$
$$\mathsf{g}(x : xs) \to x : \mathsf{g}(x : \mathsf{f}(xs, xs))$$

## Proposition

*pure* $\subsetneq$ *pure*$^+$ $\subsetneq$ *flat* $\subsetneq$ *friendly nesting.*

# Stream Specification (friendly-nesting)

## Example (convolution product $\times$)

$$
\begin{aligned}
\mathsf{nats} &\to 0 : \times(\mathsf{ones}, \mathsf{ones}) \\
\mathsf{ones} &\to \mathsf{s}(0) : \mathsf{ones} \\
\times(x : xs, y : ys) &\to \mathsf{m}(x, y) : \mathsf{add}(\mathsf{times}(ys, x), \times(xs, y : ys)) \\
\mathsf{times}(x : xs, y) &\to \mathsf{m}(x, y) : \mathsf{times}(xs, y) \\
\mathsf{add}(x : xs, y : ys) &\to \mathsf{a}(x, y) : \mathsf{add}(xs, ys)
\end{aligned}
$$

*stream layer*

$$
\begin{array}{ll}
\mathsf{a}(x, 0) \to x & \mathsf{a}(x, \mathsf{s}(y)) \to \mathsf{s}(\mathsf{a}(x, y)) \\
\mathsf{m}(x, 0) \to 0 & \mathsf{m}(x, \mathsf{s}(y)) \to \mathsf{a}(\mathsf{m}(x, y), x)
\end{array}
$$

*data layer*

$\times$ defines the stream operation $\langle xs, ys \rangle \mapsto xs \times ys$:

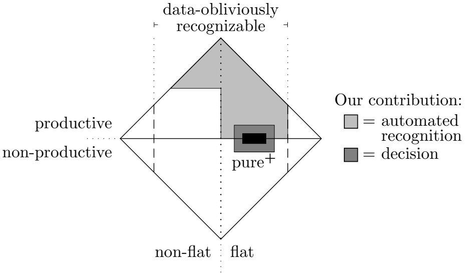
$$
(xs \times ys)(i) = \sum_{j=0}^{i} xs(j) \cdot ys(i - j) \qquad \text{(for all } i \in \mathbb{N}\text{)}
$$

# Overview

1. Extensions of PSF
2. Results for PSF-Extensions
3. Automatic Sequences and Morphic Streams

# Results for PSF-extensions

1. for pure$^{+}$ stream spec's: a decision method for productivity
2. for flat stream spec's: a computable criterion for productivity that is "data-obliviously optimal"
3. for friendly nesting stream spec's: a computable criterion for productivity
4. a productivity prover *ProPro* automating (2), (1), and (3)



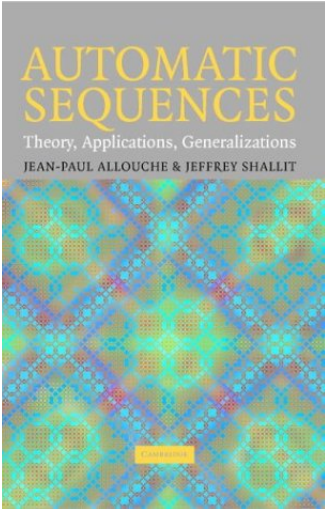
# Overview

1. Extensions of PSF

2. Results for PSF-Extensions

3. Automatic Sequences and Morphic Streams

# Automatic Sequences

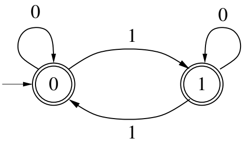


# Automatic Sequences

Property of the Thue–Morse stream:

- The $n$-th element $TM(n)$ is the parity of the number of 1's in the binary expansion of $n$.

Hence $TM(n)$ can be computed on the decimal expansion of $n$ by the following deterministic finite automaton:




$TM$ is called a 2-automatic sequence.

# Morphic streams

The function
$$\phi_0 : \{a, b, c\} \to \{a, b, c\}^*$$
$$a \mapsto a\,b\,c$$
$$b \mapsto a\,c$$
$$c \mapsto b$$

can be extended to a morphism $\phi : \{a, b, c\}^* \to \{a, b, c\}^*$.

Now note that $a \sqsubset abc = \phi(a)$. Then $a$ is prolongable, and it holds:

$$(\phi(a))^n = a\,bc\,\phi(bc)\,\phi^2(bc)\,\phi^3(bc)\,\ldots\,\phi^{n-1}(bc)$$

It follows:

$$(\phi(a))^\omega := \lim_{n\to\infty} \phi^n(bc) = a\,bc\,\phi(bc)\,\phi^2(bc)\,\phi^3(bc)\,\ldots \in \{a, b, c\}^\omega$$

which is called a (pure) morphic stream (generated by morphism $\phi$).

(In this case the ternary Thue–Morse sequence is obtained.)

# Morphic streams

The function
$$\phi_0 : \{a, b, c\} \to \{a, b, c\}^*$$
$$a \mapsto a\,b\,c$$
$$b \mapsto a\,c$$
$$c \mapsto b$$

can be extended to a morphism $\phi : \{a, b, c\}^* \to \{a, b, c\}^*$.

Now note that $a \sqsubset abc = \phi(a)$. Then $a$ is prolongable, and it holds:

$$(\phi(a))^n = a\,bc\,\phi(bc)\,\phi^2(bc)\,\phi^3(bc)\,\ldots\phi^{n-1}(bc)$$

It follows:

$$(\phi(a))^\omega := \lim_{n\to\infty} \phi^n(bc) = a\,bc\,\phi(bc)\,\phi^2(bc)\,\phi^3(bc)\,\ldots \in \{a, b, c\}^\omega$$

which is called a (pure) morphic stream (generated by morphism $\phi$).

(In this case the ternary Thue–Morse sequence is obtained.)

# Morphic streams

The function

$$\phi_0 : \{a, b, c\} \to \{a, b, c\}^*$$
$$a \mapsto a\,b\,c$$
$$b \mapsto a\,c$$
$$c \mapsto b$$

can be extended to a morphism $\phi : \{a, b, c\}^* \to \{a, b, c\}^*$.

Now note that $a \sqsubset abc = \phi(a)$. Then $a$ is prolongable, and it holds:

$$(\phi(a))^n = a\,bc\,\phi(bc)\,\phi^2(bc)\,\phi^3(bc)\,\ldots\,\phi^{n-1}(bc)$$

It follows:

$$(\phi(a))^\omega := \lim_{n\to\infty} \phi^n(bc) = a\,bc\,\phi(bc)\,\phi^2(bc)\,\phi^3(bc)\,\ldots \in \{a, b, c\}^\omega$$

which is called a (pure) morphic stream (generated by morphism $\phi$).

(In this case the ternary Thue–Morse sequence is obtained.)

# Morphic streams

The function
$$\phi_0 : \{a, b, c\} \to \{a, b, c\}^*$$
$$a \mapsto a\,b\,c$$
$$b \mapsto a\,c$$
$$c \mapsto b$$

can be extended to a morphism $\phi : \{a, b, c\}^* \to \{a, b, c\}^*$.

Now note that $a \sqsubset abc = \phi(a)$. Then $a$ is prolongable, and it holds:

$$(\phi(a))^n = a\,bc\,\phi(bc)\,\phi^2(bc)\,\phi^3(bc)\,\ldots\,\phi^{n-1}(bc)$$

It follows:

$$(\phi(a))^\omega := \lim_{n\to\infty} \phi^n(bc) = a\,bc\,\phi(bc)\,\phi^2(bc)\,\phi^3(bc)\,\ldots \in \{a, b, c\}^\omega$$

which is called a (pure) morphic stream (generated by morphism $\phi$).

(In this case the ternary Thue–Morse sequence is obtained.)

# Morphic Streams

**Example (Flat stream spec for ternary Thue–Morse)**

$$
\begin{array}{rcll}
\hline
 & \mathsf{Q} \to \mathsf{a} : \mathsf{R} & & \\
 & \mathsf{R} \to \mathsf{b} : \mathsf{c} : \mathsf{f}(\mathsf{R}) & & \\
\mathsf{f}(\mathsf{a} : xs) & \to & \mathsf{a} : \mathsf{b} : \mathsf{c} : \mathsf{f}(xs) & \textit{stream layer} \\
\mathsf{f}(\mathsf{b} : xs) & \to & \mathsf{a} : \mathsf{c} : \mathsf{f}(xs) & \\
\mathsf{f}(\mathsf{c} : xs) & \to & \mathsf{b} : \mathsf{f}(xs) & \\
\hline
 & & & \textit{data layer} \\
\hline
\end{array}
$$

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

# Morphic streams

- $\phi : \Sigma^* \to \Sigma^*$ is a morphism if for all $u, v \in \Sigma^*$: $\phi(uv) = \phi(u)\phi(v)$. A morphism $\phi$ is a coding if $\phi(a) \in \Sigma$ for all $a \in \Sigma$.
- A stream $\sigma \in \Sigma^\omega$ is called pure morphic if $\sigma = \phi^\omega(a)$ for some morphism $\phi$ and letter $a$ ($k$-uniform if $|\phi(a)| = k$ for all $a \in \Sigma$).
- Morphic streams: images of pure morphic ones under codings.

**Fact**

*For all $k \in \mathbb{N}$: $k$-uniform morphic streams $=$ $k$-automatic sequences.*

**Theorem**

1. *Every automatic/uniform morphic stream can be specified by a (productive) pure$^+$ (and also by a pure) specification.*
2. *Every morphic stream can be specified by a (productive) flat specification.*

# Morphic streams

- $\phi : \Sigma^* \to \Sigma^*$ is a morphism if for all $u, v \in \Sigma^*$: $\phi(uv) = \phi(u)\phi(v)$. A morphism $\phi$ is a coding if $\phi(a) \in \Sigma$ for all $a \in \Sigma$.
- A stream $\sigma \in \Sigma^\omega$ is called pure morphic if $\sigma = \phi^\omega(a)$ for some morphism $\phi$ and letter $a$ ($k$-uniform if $|\phi(a)| = k$ for all $a \in \Sigma$).
- Morphic streams: images of pure morphic ones under codings.

## Fact

*For all $k \in \mathbb{N}$: $k$-uniform morphic streams $=$ $k$-automatic sequences.*

## Theorem

1. *Every automatic/uniform morphic stream can be specified by a (productive) pure$^+$ (and also by a pure) specification.*
2. *Every morphic stream can be specified by a (productive) flat specification.*

# Tomorrow

- Part 4: data-oblivious productivity (C)
- Part 5: productivity of infinite data structures via termination (J)
- Part 6: complexity of productivity (C)
- Practical session (you, with DJC assisting)

# Exercises

Available at:

- the course's page http://www.phil.uu.nl/isr2010/ehg.html
- www.cs.vu.nl/˜diem/ISR-2010/practicum.pdf

Reminder link *ProPro*:

- infinity.few.vu.nl/productivity

# References


- Jörg Endrullis, Clemens Grabmayer, and Dimitri Hendriks.
  Data-Oblivious Stream Productivity.
  In *LPAR*, number 5330 in LNCS, pages 79–96. Springer, 2008.

- Jörg Endrullis, Clemens Grabmayer, and Dimitri Hendriks.
  *ProPro*: an Automated Productivity Prover.
  `http://infinity.few.vu.nl/productivity/`, 2008.

- Jörg Endrullis, Clemens Grabmayer, Dimitri Hendriks, Ariya Isihara, and Jan Willem Klop.
  Productivity of Stream Definitions.
  In *FCT*, number 4639 in LNCS, pages 274–287. Springer, 2007.