Zip-Specifications and Automatic Sequences

Clemens Grabmayer[⋆], Jörg Endrullis[†],
Dimitri Hendriks[†], Jan Willem Klop[†] and Lawrence S. Moss[‡]

[⋆] Universiteit Utrecht
[†] Vrije Universiteit Amsterdam,
[‡] Indiana University

CoiN – Coalgebra in the Netherlands
Radboud Universiteit Nijmegen, June 4, 2012

Overview

- zip-specifications
- observation graphs
- connection with automatic sequences
- mix-automaticity and zip-mix specifications
- dynamic logic representation of automatic sequences

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)

$$\mathsf{L} = 0 : \mathsf{X}$$
$$\mathsf{X} = 1 : \mathsf{zip}(\mathsf{X}, \mathsf{Y})$$
$$\mathsf{Y} = 0 : \mathsf{zip}(\mathsf{Y}, \mathsf{X})$$
$$\mathsf{zip}(x : \sigma, y : \tau) = x : y : \mathsf{zip}(\sigma, \tau)$$

L

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)

$$\mathsf{L} \to 0 : \mathsf{X}$$
$$\mathsf{X} \to 1 : \mathsf{zip}(\mathsf{X}, \mathsf{Y})$$
$$\mathsf{Y} \to 0 : \mathsf{zip}(\mathsf{Y}, \mathsf{X})$$
$$\mathsf{zip}(x : \sigma, y : \tau) \to x : y : \mathsf{zip}(\tau, \sigma)$$

L

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)

$$\begin{aligned}
\mathsf{L} &\to 0 : \mathsf{X} \\
\mathsf{X} &\to 1 : \mathsf{zip}(\mathsf{X}, \mathsf{Y}) \\
\mathsf{Y} &\to 0 : \mathsf{zip}(\mathsf{Y}, \mathsf{X}) \\
\mathsf{zip}(x : \sigma, y : \tau) &\to x : y : \mathsf{zip}(\tau, \sigma)
\end{aligned}$$

$\mathsf{L} \to 0 : \mathsf{X}$

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)

$$\begin{aligned} \mathsf{L} &\to 0 : \mathsf{X} \\ \mathsf{X} &\to 1 : \mathsf{zip}(\mathsf{X}, \mathsf{Y}) \\ \mathsf{Y} &\to 0 : \mathsf{zip}(\mathsf{Y}, \mathsf{X}) \\ \mathsf{zip}(x : \sigma, y : \tau) &\to x : y : \mathsf{zip}(\tau, \sigma) \end{aligned}$$

$$\mathsf{L} \to^* 0 : 1 : \mathsf{zip}(\mathsf{X}, \mathsf{Y})$$

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)

$$\begin{aligned} \mathsf{L} &\to 0 : \mathsf{X} \\ \mathsf{X} &\to 1 : \mathsf{zip}(\mathsf{X}, \mathsf{Y}) \\ \mathsf{Y} &\to 0 : \mathsf{zip}(\mathsf{Y}, \mathsf{X}) \\ \mathsf{zip}(x : \sigma, y : \tau) &\to x : y : \mathsf{zip}(\tau, \sigma) \end{aligned}$$

$$\mathsf{L} \to^* 0 : 1 : \mathsf{zip}(1 : \mathsf{zip}(\mathsf{X}, \mathsf{Y}), \mathsf{Y})$$

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)

$$\mathsf{L} \to 0 : \mathsf{X}$$

$$\mathsf{X} \to 1 : \mathsf{zip}(\mathsf{X}, \mathsf{Y})$$

$$\mathsf{Y} \to 0 : \mathsf{zip}(\mathsf{Y}, \mathsf{X})$$

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

$$\mathsf{L} \to^{*} 0 : 1 : \mathsf{zip}(1 : \mathsf{zip}(\mathsf{X}, \mathsf{Y}), 0 : \mathsf{zip}(\mathsf{Y}, \mathsf{X}))$$

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)

$$\mathsf{L} \to 0 : \mathsf{X}$$
$$\mathsf{X} \to 1 : \mathsf{zip}(\mathsf{X}, \mathsf{Y})$$
$$\mathsf{Y} \to 0 : \mathsf{zip}(\mathsf{Y}, \mathsf{X})$$
$$\mathsf{zip}(x : \sigma, y : \tau) \to x : y : \mathsf{zip}(\tau, \sigma)$$

$$\mathsf{L} \to^* 0 : 1 : 1 : 0 : \mathsf{zip}(\mathsf{zip}(\mathsf{X}, \mathsf{Y}), \mathsf{zip}(\mathsf{Y}, \mathsf{X}))$$

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)

$$\mathsf{L} \to 0 : \mathsf{X}$$
$$\mathsf{X} \to 1 : \mathsf{zip}(\mathsf{X}, \mathsf{Y})$$
$$\mathsf{Y} \to 0 : \mathsf{zip}(\mathsf{Y}, \mathsf{X})$$
$$\mathsf{zip}(x : \sigma, \tau) = x : \mathsf{zip}(\tau, \sigma)$$

$$\mathsf{L} \to^{\omega} 0 : 1 : 1 : 0 : 1 : 0 : 0 : 1 : 1 : 0 : 0 : 1 : 0 : 1 : 1 : 0 : \ldots$$

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)

$$\mathsf{L} \to 0 : \mathsf{X}$$

$$\mathsf{X} \to 1 : \mathsf{zip}(\mathsf{X}, \mathsf{Y})$$

$$\mathsf{Y} \to 0 : \mathsf{zip}(\mathsf{Y}, \mathsf{X})$$

$$\mathsf{zip}(x : \sigma, \tau) \to x : \mathsf{zip}(\tau, \sigma)$$

$\mathsf{L} \to^* 0 : 1 : \mathsf{zip}(\mathsf{X}, \mathsf{Y})$

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)

$$\begin{aligned} \mathsf{L} &\to 0 : \mathsf{X} \\ \mathsf{X} &\to 1 : \mathsf{zip}(\mathsf{X}, \mathsf{Y}) \\ \mathsf{Y} &\to 0 : \mathsf{zip}(\mathsf{Y}, \mathsf{X}) \\ \mathsf{zip}(x : \sigma, \tau) &\to x : \mathsf{zip}(\tau, \sigma) \end{aligned}$$

$$\mathsf{L} \to^* 0 : 1 : \mathsf{zip}(1 : \mathsf{zip}(\mathsf{X}, \mathsf{Y}), \mathsf{Y})$$

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)

$$\begin{aligned}
\mathsf{L} &\to 0 : \mathsf{X} \\
\mathsf{X} &\to 1 : \mathsf{zip}(\mathsf{X}, \mathsf{Y}) \\
\mathsf{Y} &\to 0 : \mathsf{zip}(\mathsf{Y}, \mathsf{X}) \\
\mathsf{zip}(x : \sigma, \tau) &\to x : \mathsf{zip}(\tau, \sigma)
\end{aligned}$$

$$\mathsf{L} \to^{*} 0 : 1 : 1 : \mathsf{zip}(\mathsf{Y}, \mathsf{zip}(\mathsf{X}, \mathsf{Y}))$$

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)

$$\mathsf{L} \to 0 : \mathsf{X}$$

$$\mathsf{X} \to 1 : \mathsf{zip}(\mathsf{X}, \mathsf{Y})$$

$$\mathsf{Y} \to 0 : \mathsf{zip}(\mathsf{Y}, \mathsf{X})$$

$$\mathsf{zip}(x : \sigma, \tau) \to x : \mathsf{zip}(\tau, \sigma)$$

$$\mathsf{L} \to^* 0 : 1 : 1 : \mathsf{zip}(0 : \mathsf{zip}(\mathsf{Y}, \mathsf{X}), \mathsf{zip}(\mathsf{X}, \mathsf{Y}))$$

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)

$$\begin{aligned} \mathsf{L} &\to 0 : \mathsf{X} \\ \mathsf{X} &\to 1 : \mathsf{zip}(\mathsf{X}, \mathsf{Y}) \\ \mathsf{Y} &\to 0 : \mathsf{zip}(\mathsf{Y}, \mathsf{X}) \\ \mathsf{zip}(x : \sigma, \tau) &\to x : \mathsf{zip}(\tau, \sigma) \end{aligned}$$

$$\mathsf{L} \to^* 0 : 1 : 1 : 0 : \mathsf{zip}(\mathsf{zip}(\mathsf{X}, \mathsf{Y}), \mathsf{zip}(\mathsf{Y}, \mathsf{X}))$$

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)

$$\mathsf{L} \to 0 : \mathsf{X}$$
$$\mathsf{X} \to 1 : \mathsf{zip}(\mathsf{X}, \mathsf{Y})$$
$$\mathsf{Y} \to 0 : \mathsf{zip}(\mathsf{Y}, \mathsf{X})$$
$$\mathsf{zip}(x : \sigma, \tau) \to x : \mathsf{zip}(\tau, \sigma)$$

$$\mathsf{L} \to^{\omega} 0 : 1 : 1 : 0 : 1 : 0 : 0 : 1 : 1 : 0 : 0 : 1 : 0 : 1 : 1 : 0 : \ldots$$

Specifying streams

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

$$a_0 : a_1 : a_2 : \dots .$$

Example (Thue–Morse stream)

$$\mathsf{L} \to 0 : \mathsf{X}$$
$$\mathsf{X} \to 1 : \mathsf{zip}(\mathsf{X}, \mathsf{Y})$$
$$\mathsf{Y} \to 0 : \mathsf{zip}(\mathsf{Y}, \mathsf{X})$$
$$\mathsf{zip}(x : \sigma, \tau) \to x : \mathsf{zip}(\tau, \sigma)$$

$$\mathsf{L} \to^{\omega} 0 : 1 : 1 : 0 : 1 : 0 : 0 : 1 : 1 : 0 : 0 : 1 : 0 : 1 : 1 : 0 : \dots$$

Productive stream specification: lazy/fair evaluation of its root L results in an infinite *constructor normal form* (representing a stream).

Streams and zip-specifications

Zip-specifications consist of recursion equations:

$$\mathsf{M}_i = C_i[\mathsf{M}_0, \ldots, \mathsf{M}_{n-1}] \qquad (i = 0, \ldots, n-1)$$

where contexts C_i are built from:

- data constants c_1, c_2, ...
- stream constructor symbol ':'
- the binary stream function symbol zip

$$\mathsf{zip}(x : \sigma, \tau) = x : \mathsf{zip}(\tau, \sigma)$$

Two zip-specifications are equivalent if they define the same stream.

Motivating Question

Is equivalence of zip-specifications decidable?

Related results / existing tools

Equivalence of stream specifications

- Π^0_2-complete (Roșu, 2006)
- proof tools: Circ (Roșu), Stream-Box (E, Zantema)
- Recent: Π^0_1-complete for productive specs (E/H/Bakhshi, 2012)

Productivity of stream specifications

- productivity implies unique solvability (Sijtsma, 1989)
- Π^0_2-complete (Simonsen, E/G/H, 2006),
  undecidable formats (Sattler/Balestrieri, 2012).
- much previous and current work on productivity

  ([Dijkstra], Wadge, Sijtsma, Telford/Turner, Hughes/Pareto/Sabry, Buchholz, E/G/H/K/Isihara, Zantema, Balestrieri)
- Productivity prover *ProPro* (2008) of E/G/H for stream productivity:

  http://infinity.few.vu.nl/productivity/tool.html

Unique Solvability versus Productivity

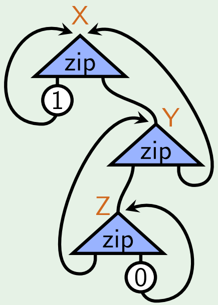
Proposition

For a zip-specification $\mathcal{S}$ the following are equivalent:

- *$\mathcal{S}$ is uniquely solvable,*
- *$\mathcal{S}$ is productive,*
- *$\mathcal{S}$ has a guard on every left-most cycle.*

Example

$$X = \text{zip}(1 : X, Y)$$
$$Y = \text{zip}(Z, X)$$
$$Z = \text{zip}(Y, 0 : Z)$$



Unique Solvability versus Productivity

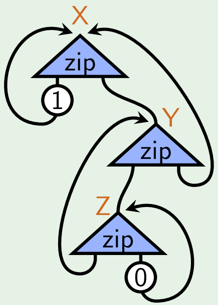
Proposition

For a zip-specification $\mathcal{S}$ the following are equivalent:

- *$\mathcal{S}$ is uniquely solvable,*
- *$\mathcal{S}$ is productive,*
- *$\mathcal{S}$ has a guard on every left-most cycle.*

Example

$$X = \mathsf{zip}(1 : X, Y)$$
$$Y = \mathsf{zip}(Z, X)$$
$$Z = \mathsf{zip}(Y, 0 : Z)$$



Unique Solvability versus Productivity

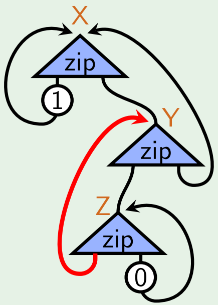
Proposition

For a zip-specification $\mathcal{S}$ the following are equivalent:

- *$\mathcal{S}$ is uniquely solvable,*
- *$\mathcal{S}$ is productive,*
- *$\mathcal{S}$ has a guard on every left-most cycle.*

Example

$$\mathsf{X} = \mathsf{zip}(1 : \mathsf{X}, \mathsf{Y})$$
$$\mathsf{Y} = \mathsf{zip}(\mathsf{Z}, \mathsf{X})$$
$$\mathsf{Z} = \mathsf{zip}(\mathsf{Y}, 0 : \mathsf{Z})$$



No guard on left cycle

$$\mathsf{Y} \to \mathsf{Z} \to \mathsf{Y}$$

Not productive!

Unique Solvability versus Productivity

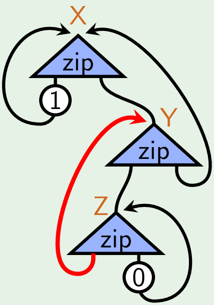
Proposition

For a zip-specification $\mathcal{S}$ the following are equivalent:

- *$\mathcal{S}$ is uniquely solvable,*
- *$\mathcal{S}$ is productive,*
- *$\mathcal{S}$ has a guard on every left-most cycle.*

Example

$$X = \mathsf{zip}(1 : X, Y)$$
$$Y = \mathsf{zip}(Z, X)$$
$$Z = \mathsf{zip}(Y, 0 : Z)$$



No guard on left cycle

$$Y \to Z \to Y$$

Not productive!

Two roads to productivity:

1. $Z = \mathsf{zip}(0 : \mathsf{tl}(Y), 0 : Z)$
2. $Z = \mathsf{zip}(1 : \mathsf{tl}(Y), 0 : Z)$

(tail can be rolled away)

Unique Solvability versus Productivity

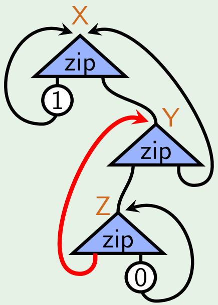
Proposition

For a zip-specification $\mathcal{S}$ the following are equivalent:

- *$\mathcal{S}$ is uniquely solvable,*
- *$\mathcal{S}$ is productive,*
- *$\mathcal{S}$ has a guard on every left-most cycle.*

Example

$$X = \mathrm{zip}(1 : X, Y)$$
$$Y = \mathrm{zip}(Z, X)$$
$$Z = \mathrm{zip}(Y, 0 : Z)$$



No guard on left cycle

$$Y \to Z \to Y$$

Not productive!

Two roads to productivity:

1. $Z = \mathrm{zip}(0 : \mathrm{tl}(Y), 0 : Z)$
2. $Z = \mathrm{zip}(1 : \mathrm{tl}(Y), 0 : Z)$

(tail can be rolled away)

Thus 2 solutions!

Observation graphs

$$\mathsf{L} = 0 : \mathsf{X}$$
$$\mathsf{X} = 1 : \mathsf{zip}(\mathsf{X}, \mathsf{Y})$$
$$\mathsf{Y} = 0 : \mathsf{zip}(\mathsf{Y}, \mathsf{X})$$

Observation graphs

$$\mathsf{L} = 0 : \mathsf{X}$$
$$\mathsf{X} = 1 : \mathsf{zip}(\mathsf{X}, \mathsf{Y})$$
$$\mathsf{Y} = 0 : \mathsf{zip}(\mathsf{Y}, \mathsf{X})$$

We create observation graphs with the stream cobasis $\{\mathsf{hd}, \mathsf{even}, \mathsf{odd}\}$:

$$\mathsf{hd}(x : t) = x \qquad \mathsf{even}(x : t) = x : \mathsf{odd}(t) \qquad \mathsf{odd}(x : t) = \mathsf{even}(t)$$

Observation graphs

$$\mathsf{L} = 0 : \mathsf{X}$$
$$\mathsf{X} = 1 : \mathsf{zip}(\mathsf{X}, \mathsf{Y})$$
$$\mathsf{Y} = 0 : \mathsf{zip}(\mathsf{Y}, \mathsf{X})$$

We create observation graphs with the stream cobasis $\{\mathsf{hd}, \mathsf{even}, \mathsf{odd}\}$:

$$\mathsf{hd}(x : t) = x \qquad \mathsf{even}(x : t) = x : \mathsf{odd}(t) \qquad \mathsf{odd}(x : t) = \mathsf{even}(t)$$

We can observe every element using $\mathsf{hd}(\{\mathsf{even}, \mathsf{odd}\}^*(\sigma))$. E.g.

$$n = 6 = (110)_2 \qquad\qquad \sigma(6) = \mathsf{hd}(\mathsf{odd}(\mathsf{odd}(\mathsf{even}(\sigma))))$$

Observation graphs

$$\begin{aligned} L &= 0 : X \\ X &= 1 : \mathrm{zip}(X, Y) \\ Y &= 0 : \mathrm{zip}(Y, X) \end{aligned}$$

We create observation graphs with the stream cobasis $\{\mathrm{hd}, \mathrm{even}, \mathrm{odd}\}$:

$$\mathrm{hd}(x : t) = x \qquad \mathrm{even}(x : t) = x : \mathrm{odd}(t) \qquad \mathrm{odd}(x : t) = \mathrm{even}(t)$$

We can observe every element using $\mathrm{hd}(\{\mathrm{even}, \mathrm{odd}\}^*(\sigma))$. E.g.

$$n = 6 = (110)_2 \qquad \sigma(6) = \mathrm{hd}(\mathrm{odd}(\mathrm{odd}(\mathrm{even}(\sigma))))$$

The functions even and odd are a form of destructors of zip:

$$\mathrm{even}(\mathrm{zip}(s, t)) = s \qquad \mathrm{odd}(\mathrm{zip}(s, t)) = t$$

Observation graphs

$$\begin{aligned} \mathsf{L} &= 0 : \mathsf{X} \\ \mathsf{X} &= 1 : \mathsf{zip}(\mathsf{X}, \mathsf{Y}) \\ \mathsf{Y} &= 0 : \mathsf{zip}(\mathsf{Y}, \mathsf{X}) \end{aligned}$$

We create observation graphs with the stream cobasis $\{\mathsf{hd}, \mathsf{even}, \mathsf{odd}\}$:

$$\mathsf{hd}(x : t) = x \qquad \mathsf{even}(x : t) = x : \mathsf{odd}(t) \qquad \mathsf{odd}(x : t) = \mathsf{even}(t)$$

Observation graphs

$$\mathsf{L} = 0 : \mathsf{X}$$
$$\mathsf{X} = 1 : \mathsf{zip}(\mathsf{X}, \mathsf{Y})$$
$$\mathsf{Y} = 0 : \mathsf{zip}(\mathsf{Y}, \mathsf{X})$$

We create observation graphs with the stream cobasis $\{\mathsf{hd}, \mathsf{even}, \mathsf{odd}\}$:

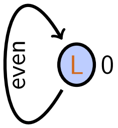
$$\mathsf{hd}(x : t) = x \qquad \mathsf{even}(x : t) = x : \mathsf{odd}(t) \qquad \mathsf{odd}(x : t) = \mathsf{even}(t)$$

Observation graphs

$$\begin{aligned} L &= 0 : X \\ X &= 1 : \mathrm{zip}(X, Y) \\ Y &= 0 : \mathrm{zip}(Y, X) \end{aligned}$$

We create observation graphs with the stream cobasis $\{\mathrm{hd}, \mathrm{even}, \mathrm{odd}\}$:

$$\mathrm{hd}(x : t) = x \qquad \mathrm{even}(x : t) = x : \mathrm{odd}(t) \qquad \mathrm{odd}(x : t) = \mathrm{even}(t)$$



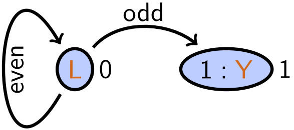
$$\begin{aligned} &\mathrm{even}(L) \\ &= \mathrm{even}(0 : X) \\ &= 0 : \mathrm{odd}(X) \\ &= 0 : \mathrm{odd}(1 : \mathrm{zip}(X, Y)) \\ &= 0 : \mathrm{even}(\mathrm{zip}(X, Y)) \\ &= 0 : X \\ &= L \end{aligned}$$

Observation graphs

$$\mathsf{L} = 0 : \mathsf{X}$$
$$\mathsf{X} = 1 : \mathrm{zip}(\mathsf{X}, \mathsf{Y})$$
$$\mathsf{Y} = 0 : \mathrm{zip}(\mathsf{Y}, \mathsf{X})$$

We create observation graphs with the stream cobasis $\{\mathrm{hd}, \mathrm{even}, \mathrm{odd}\}$:

$$\mathrm{hd}(x : t) = x \qquad \mathrm{even}(x : t) = x : \mathrm{odd}(t) \qquad \mathrm{odd}(x : t) = \mathrm{even}(t)$$



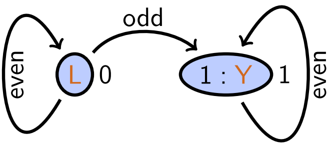
$$\begin{aligned}
&\mathrm{odd}(\mathsf{L}) \\
&= \mathrm{odd}(0 : \mathsf{X}) \\
&= \mathrm{even}(\mathsf{X}) \\
&= \mathrm{even}(1 : \mathrm{zip}(\mathsf{X}, \mathsf{Y})) \\
&= 1 : \mathrm{odd}(\mathrm{zip}(\mathsf{X}, \mathsf{Y})) \\
&= 1 : \mathsf{Y}
\end{aligned}$$

Observation graphs

$$\mathsf{L} = 0 : \mathsf{X}$$
$$\mathsf{X} = 1 : \mathsf{zip}(\mathsf{X}, \mathsf{Y})$$
$$\mathsf{Y} = 0 : \mathsf{zip}(\mathsf{Y}, \mathsf{X})$$

We create observation graphs with the stream cobasis $\{\mathsf{hd}, \mathsf{even}, \mathsf{odd}\}$:

$$\mathsf{hd}(x : t) = x \qquad \mathsf{even}(x : t) = x : \mathsf{odd}(t) \qquad \mathsf{odd}(x : t) = \mathsf{even}(t)$$



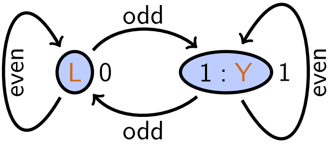
$$\begin{aligned}&\mathsf{even}(1 : \mathsf{Y})\\ &= 1 : \mathsf{odd}(\mathsf{Y})\\ &= 1 : \mathsf{odd}(0 : \mathsf{zip}(\mathsf{Y}, \mathsf{X}))\\ &= 1 : \mathsf{even}(\mathsf{zip}(\mathsf{Y}, \mathsf{X}))\\ &= 1 : \mathsf{Y}\end{aligned}$$

Observation graphs

$$\mathsf{L} = 0 : \mathsf{X}$$
$$\mathsf{X} = 1 : \mathsf{zip}(\mathsf{X}, \mathsf{Y})$$
$$\mathsf{Y} = 0 : \mathsf{zip}(\mathsf{Y}, \mathsf{X})$$

We create observation graphs with the stream cobasis $\{\mathsf{hd}, \mathsf{even}, \mathsf{odd}\}$:

$$\mathsf{hd}(x : t) = x \qquad \mathsf{even}(x : t) = x : \mathsf{odd}(t) \qquad \mathsf{odd}(x : t) = \mathsf{even}(t)$$



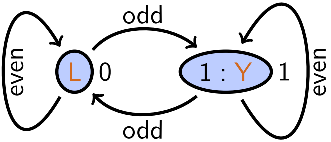
$\mathsf{odd}(1 : \mathsf{Y})$
$= \mathsf{even}(\mathsf{Y})$
$= \mathsf{even}(0 : \mathsf{zip}(\mathsf{Y}, \mathsf{X}))$
$= 0 : \mathsf{odd}(\mathsf{zip}(\mathsf{Y}, \mathsf{X}))$
$= 0 : \mathsf{X}$
$= \mathsf{L}$

Observation graphs

$$\mathsf{L} = 0 : \mathsf{X}$$
$$\mathsf{X} = 1 : \mathsf{zip}(\mathsf{X}, \mathsf{Y})$$
$$\mathsf{Y} = 0 : \mathsf{zip}(\mathsf{Y}, \mathsf{X})$$

We create observation graphs with the stream cobasis $\{\mathsf{hd}, \mathsf{even}, \mathsf{odd}\}$:

$$\mathsf{hd}(x : t) = x \qquad \mathsf{even}(x : t) = x : \mathsf{odd}(t) \qquad \mathsf{odd}(x : t) = \mathsf{even}(t)$$



Theorem

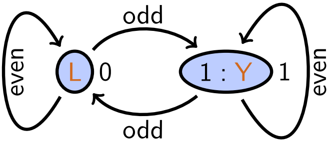
For every productive zip-specification the observation graph is finite.

Observation graphs

$$\mathsf{L} = 0 : \mathsf{X}$$
$$\mathsf{X} = 1 : \mathsf{zip}(\mathsf{X}, \mathsf{Y})$$
$$\mathsf{Y} = 0 : \mathsf{zip}(\mathsf{Y}, \mathsf{X})$$

We create observation graphs with the stream cobasis $\{\mathsf{hd}, \mathsf{even}, \mathsf{odd}\}$:

$$\mathsf{hd}(x : t) = x \qquad \mathsf{even}(x : t) = x : \mathsf{odd}(t) \qquad \mathsf{odd}(x : t) = \mathsf{even}(t)$$



Theorem

For every productive zip-specification the observation graph is finite.

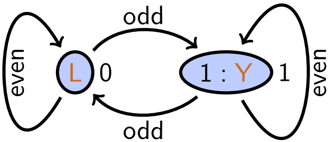
(Remark: zip-free cycles $\mathsf{X} = 1 : 0 : 1 : \mathsf{X}$ need special treatment)

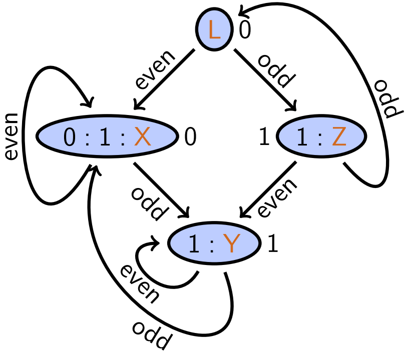
Comparing zip-specifications

$$\mathsf{L} = 0 : \mathsf{X}$$
$$\mathsf{X} = 1 : \mathrm{zip}(\mathsf{X}, \mathsf{Y})$$
$$\mathsf{Y} = 0 : \mathrm{zip}(\mathsf{Y}, \mathsf{X})$$

$$\mathsf{L} = 0 : \mathrm{zip}(1 : \mathsf{Z}, 1 : \mathsf{X})$$
$$\mathsf{X} = 1 : \mathrm{zip}(\mathsf{Y}, \mathsf{X})$$
$$\mathsf{Y} = 0 : \mathrm{zip}(\mathsf{Y}, 1 : \mathsf{X})$$
$$\mathsf{Z} = \mathrm{zip}(\mathsf{L}, \mathsf{Y})$$

Zip-specifications are equal iff their observation graphs are bisimilar



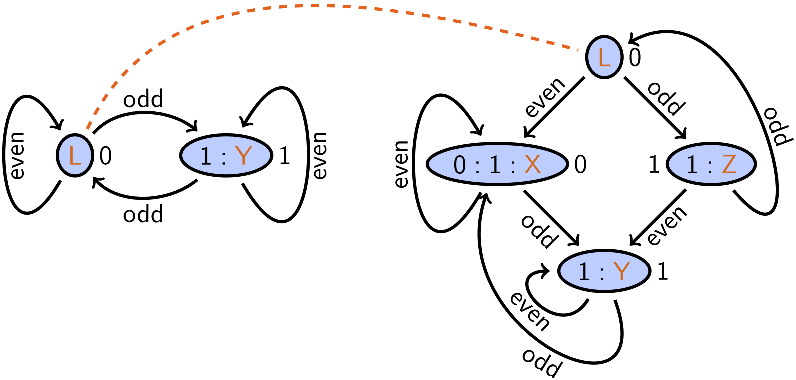


Comparing zip-specifications

$$\begin{aligned} L &= 0 : X \\ X &= 1 : \mathrm{zip}(X, Y) \\ Y &= 0 : \mathrm{zip}(Y, X) \end{aligned}$$

$$\begin{aligned} L &= 0 : \mathrm{zip}(1 : Z, 1 : X) \\ X &= 1 : \mathrm{zip}(Y, X) \\ Y &= 0 : \mathrm{zip}(Y, 1 : X) \\ Z &= \mathrm{zip}(L, Y) \end{aligned}$$

Zip-specifications are equal iff their observation graphs are bisimilar

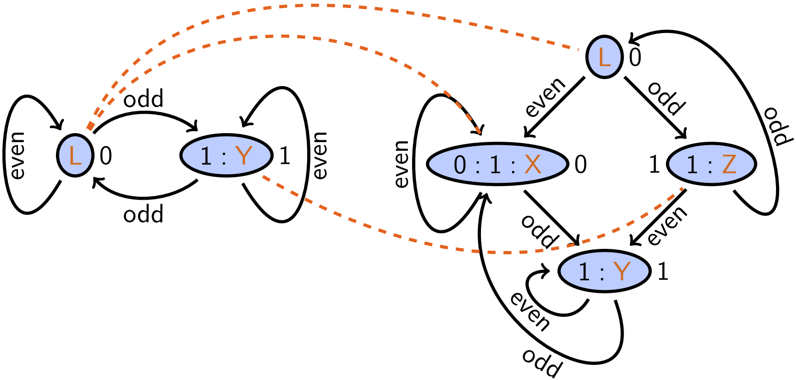


Comparing zip-specifications

$$\mathsf{L} = 0 : \mathsf{X}$$
$$\mathsf{X} = 1 : \mathrm{zip}(\mathsf{X}, \mathsf{Y})$$
$$\mathsf{Y} = 0 : \mathrm{zip}(\mathsf{Y}, \mathsf{X})$$

$$\mathsf{L} = 0 : \mathrm{zip}(1 : \mathsf{Z}, 1 : \mathsf{X})$$
$$\mathsf{X} = 1 : \mathrm{zip}(\mathsf{Y}, \mathsf{X})$$
$$\mathsf{Y} = 0 : \mathrm{zip}(\mathsf{Y}, 1 : \mathsf{X})$$
$$\mathsf{Z} = \mathrm{zip}(\mathsf{L}, \mathsf{Y})$$

Zip-specifications are equal iff their observation graphs are bisimilar

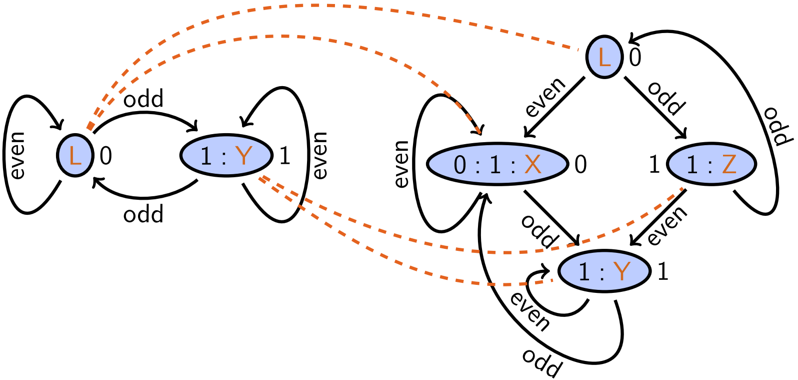


Comparing zip-specifications

$$\begin{aligned} L &= 0 : X \\ X &= 1 : \mathrm{zip}(X, Y) \\ Y &= 0 : \mathrm{zip}(Y, X) \end{aligned}$$

$$\begin{aligned} L &= 0 : \mathrm{zip}(1 : Z, 1 : X) \\ X &= 1 : \mathrm{zip}(Y, X) \\ Y &= 0 : \mathrm{zip}(Y, 1 : X) \\ Z &= \mathrm{zip}(L, Y) \end{aligned}$$

Zip-specifications are equal iff their observation graphs are bisimilar



Generalisations: zip-k-Specifications

Everything generalises to zip-k-specifications, where:

$$\mathrm{zip}_k(x_1 : \sigma_1, x_2 : \sigma_2, \ldots, x_k : \sigma_k) = x_1 : x_2 : \ldots x_k : \mathrm{zip}_k(\sigma_1, \sigma_2, \ldots, \sigma_k)$$

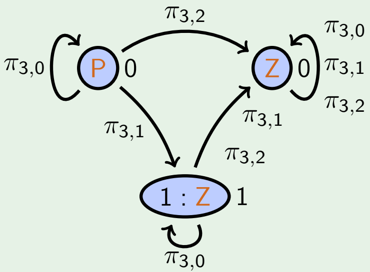
Then we use the cobasis $\{\mathrm{hd}, \pi_{k,0}, \ldots, \pi_{k,k-1}\}$:

$$\pi_{i,k}(x_0 : x_1 : x_2 : \ldots) = x_i : x_{i+k} : x_{i+2k} : \ldots$$

$$\mathsf{P} = 0 : \mathsf{Q}$$

$$\mathsf{Q} = \mathrm{zip}_3(1 : \mathsf{Z}, \mathsf{Z}, \mathsf{Q})$$

$$\mathsf{Z} = 0 : \mathsf{Z}$$



Here P is a stream $0 : 1 : 0 : 1 : 0 : 0 : 0 : 0 : 0 : 1 : \ldots$

$$P(i) = 1 \iff i = 3^n \text{ for some } n \in \mathbb{N}$$

Stream Cobases

A stream cobasis $\mathcal{B} = \langle \mathsf{hd}, \langle \gamma_1, \ldots, \gamma_k \rangle \rangle$ consists of operations $\gamma_i : \Delta^\omega \to \Delta^\omega$ such that, if for all $n \in \mathbb{N}$ and $1 \leq i_1, \ldots, i_n \leq k$:

$$\mathsf{hd}(\gamma_{i_1}(\ldots(\gamma_{i_n}(\sigma))\ldots)) = \mathsf{hd}(\gamma_{i_1}(\ldots(\gamma_{i_n}(\tau))\ldots))$$

holds, then $\sigma = \tau$ follows.

For every $k \geq 2$ we define two stream cobases:

- 'non-orthogonal' basis: $\mathcal{N}_k = \langle \mathsf{hd}, \pi_{0,k}, \ldots, \pi_{k-1,k} \rangle$
- 'orthogonal' basis: $\mathcal{O}_k = \langle \mathsf{hd}, \pi_{1,k}, \ldots, \pi_{k,k} \rangle$

Example

$$\begin{aligned} \mathcal{N}_2 &= \langle \mathsf{hd}, \pi_{0,2}, \pi_{1,2} \rangle \\ &= \langle \mathsf{hd}, \mathsf{even}, \mathsf{odd} \rangle \end{aligned} \qquad \begin{aligned} \mathcal{O}_2 &= \langle \mathsf{hd}, \pi_{1,2}, \pi_{2,2} \rangle \\ &= \langle \mathsf{hd}, \mathsf{odd}, \mathsf{tl}(\mathsf{even}) \rangle \end{aligned}$$

$$\mathcal{N}_3 = \langle \mathsf{hd}, \pi_{0,3}, \pi_{1,3}, \pi_{2,3} \rangle \qquad \mathcal{O}_3 = \langle \mathsf{hd}, \pi_{1,3}, \pi_{2,3}, \pi_{3,3} \rangle$$

Observation graphs

$\mathcal{B} = \langle \mathrm{hd}, \langle \gamma_1, \ldots, \gamma_k \rangle \rangle$ a stream cobasis, F the functor $F(X) = \Delta \times X^k$.
A $\mathcal{B}$-observation graph is an F-coalgebra $\mathcal{G} = \langle S, \langle o, n \rangle \rangle$ with root $r \in S$, such that there exists an F-homomorphism $[\![\cdot]\!] : S \to \Delta^\omega$:

$$\begin{tikzcd} S \arrow[r, "{[\![\cdot]\!]}"] \arrow[d, "{\langle o, n \rangle}"'] & \Delta^\omega \arrow[d, "\mathcal{B}"] \\ \Delta \times S^k \arrow[r, "{\mathrm{id} \times [\![\cdot]\!]^k}"] & \Delta \times (\Delta^\omega)^k \end{tikzcd}$$

- $\mathcal{G}$ defines the stream $[\![r]\!] \in \Delta^\omega$ (is unique!).
- The canonical $\mathcal{B}$-observation graph of $\sigma \in \Delta^\omega$ is the sub-coalgebra of the F-coalgebra $\langle \Delta^\omega, \mathcal{B} \rangle$ generated by σ.
- The set $\partial_{\mathcal{B}}(\sigma)$ of $\mathcal{B}$-derivatives of σ is the set of elements of the canonical observation graph of σ.

Finality

Proposition

The stream coalgebra $\langle \Delta^\omega, \mathcal{O}_k \rangle$ is final for the functor $F(X) = \Delta \times X^k$. (Hence every F-coalgebra is an $\mathcal{O}_k$-observation graph.)

- mentioned by Kupke, Rutten, Niqui (2011)
- Kupke, Rutten (2011, 2012): finality of $\langle \Delta^\omega, \mathcal{N}_k \rangle$ with respect to the class of 'even-consistent' observation graphs.

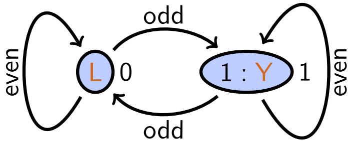
Finite Observation Graphs

Question

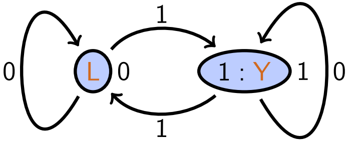
Which streams have finite $\{\mathsf{hd}, \mathsf{even}, \mathsf{odd}\}$ observation graphs?

We consider:

$$\begin{aligned} L &= 0 : X \\ X &= 1 : \mathsf{zip}(X, Y) \\ Y &= 0 : \mathsf{zip}(Y, X) \end{aligned}$$



In the observation graph, we replace even $\mapsto 0$, odd $\mapsto 1$:



DFAO for L reading the binary index from the least significant bit!

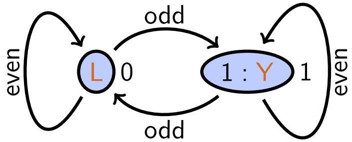
We obtain exactly the 2-automatic sequences.

Connection with DFAO's

Proposition

- *Every $\mathcal{N}_k$-observation graph $\mathcal{G}$ can be viewed as k-DFAO $A(\mathcal{G})$ is invariant under zeros and generates the stream defined by $\mathcal{G}$.*

$\mathcal{G}$

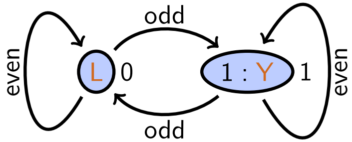


Connection with DFAO's

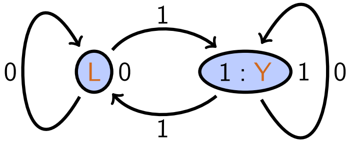
Proposition

- *Every $\mathcal{N}_k$-observation graph $\mathcal{G}$ can be viewed as k-DFAO $A(\mathcal{G})$ is invariant under zeros and generates the stream defined by $\mathcal{G}$.*

$\mathcal{G}$



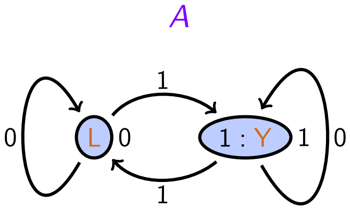
$A(\mathcal{G})$



Connection with DFAO's

Proposition

- *Every k-DFAO A that is invariant under leading zeros can be viewed as $\mathcal{N}_k$-observation graph $\mathcal{G}(A)$ that defines the stream that is generated by A.*

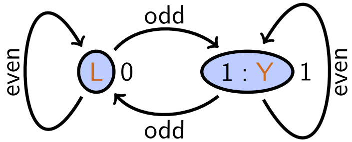


Connection with DFAO's

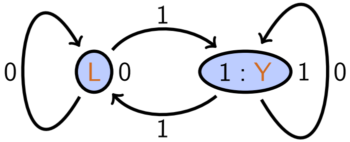
Proposition

- *Every k-DFAO* A *that is invariant under leading zeros can be viewed as* $\mathcal{N}_k$*-observation graph* $\mathcal{G}(A)$ *that defines the stream that is generated by A.*

$\mathcal{G}(A)$



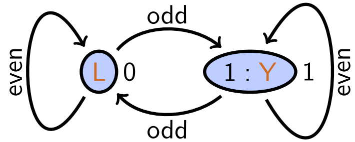
A

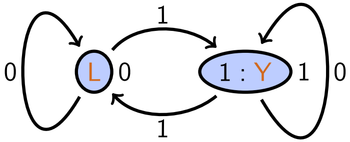


Connection with DFAO's

Proposition

- *Every $\mathcal{N}_k$-observation graph $\mathcal{G}$ can be viewed as k-DFAO $A(\mathcal{G})$ is invariant under zeros and generates the stream defined by $\mathcal{G}$.*
- *Every k-DFAO A that is invariant under leading zeros can be viewed as $\mathcal{N}_k$-observation graph $\mathcal{G}(A)$ that defines the stream that is generated by A.*





Finite observation graphs and zip-specifications

Lemma

1. *The canonical $\mathcal{N}_k$-observation graph of a stream $\sigma \in \Delta^\omega$ is finite if and only if definable by a zip-k spec with equations of form:*

$$\mathsf{X}_i = a_i : \mathsf{X}'_i \qquad \mathsf{X}'_i = \mathsf{zip}_k(\mathsf{X}_{f(i,1)}, \ldots, \mathsf{X}_{f(i,k-1)}, \mathsf{X}'_{f(i,0)})$$

$$\begin{aligned} \mathsf{L} &= 0 : \mathsf{L}' & \mathsf{L}' &= \mathsf{zip}(\mathsf{Z}, \mathsf{L}') \\ \mathsf{Z} &= 0 : \mathsf{Z}' & \mathsf{Z}' &= \mathsf{zip}(\mathsf{L}, \mathsf{Z}') \end{aligned}$$

Finite observation graphs and zip-specifications

Lemma

2. *The canonical $\mathcal{O}_k$-observation graph of a stream $\sigma \in \Delta^\omega$ is finite if and only if definable by a zip-k spec with equations of form:*

$$X_i = a_i : \mathrm{zip}_k(X_{i,1}, X_{i,2}, \ldots, X_{i,k})$$

$$\begin{aligned}
L &= 0 : \mathrm{zip}(X_e, X) \\
X &= 1 : \mathrm{zip}(X, Y) \\
X_e &= 1 : \mathrm{zip}(Y_e, Y) \\
Y &= 0 : \mathrm{zip}(Y, X) \\
Y_e &= 0 : \mathrm{zip}(Y, X)
\end{aligned}$$

Finite observation graphs and zip-specifications

Lemma

1. *The canonical $\mathcal{N}_k$-observation graph of a stream $\sigma \in \Delta^\omega$ is finite if and only if definable by a zip-k spec with equations of form:*

$$\mathsf{X}_i = a_i : \mathsf{X}'_i \qquad \mathsf{X}'_i = \mathsf{zip}_k(\mathsf{X}_{f(i,1)}, \dots, \mathsf{X}_{f(i,k-1)}, \mathsf{X}'_{f(i,0)})$$

2. *The canonical $\mathcal{O}_k$-observation graph of a stream $\sigma \in \Delta^\omega$ is finite if and only if definable by a zip-k spec with equations of form:*

$$\mathsf{X}_i = a_i : \mathsf{zip}_k(\mathsf{X}_{i,1}, \mathsf{X}_{i,2}, \dots, \mathsf{X}_{i,k})$$

$$\begin{aligned} \mathsf{L} &= 0 : \mathsf{L}' & \mathsf{L}' &= \mathsf{zip}(\mathsf{Z}, \mathsf{L}') \\ \mathsf{Z} &= 0 : \mathsf{Z}' & \mathsf{Z}' &= \mathsf{zip}(\mathsf{L}, \mathsf{Z}') \end{aligned}$$

$$\begin{aligned} \mathsf{L} &= 0 : \mathsf{zip}(\mathsf{X_e}, \mathsf{X}) \\ \mathsf{X} &= 1 : \mathsf{zip}(\mathsf{X}, \mathsf{Y}) \\ \mathsf{X_e} &= 1 : \mathsf{zip}(\mathsf{Y_e}, \mathsf{Y}) \\ \mathsf{Y} &= 0 : \mathsf{zip}(\mathsf{Y}, \mathsf{X}) \\ \mathsf{Y_e} &= 0 : \mathsf{zip}(\mathsf{Y}, \mathsf{X}) \end{aligned}$$

Zip-Specifications and automatic sequences

Theorem

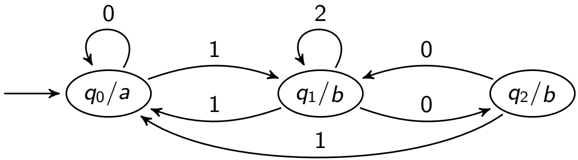
For streams $\sigma \in \Delta^\omega$ the following properties are equivalent:

1. *The stream σ is k-automatic.*
2. *The stream σ can be defined by a zip-k specification.*
3. *The canonical $\mathcal{N}_k$-observation graph of σ is finite.*
4. *The canonical $\mathcal{O}_k$-observation graph of σ is finite.*

(1) $\Leftrightarrow$ (3): independently discovered by Kupke, Rutten (2012).

Mix-Automaticity

Mix-DFAO A:

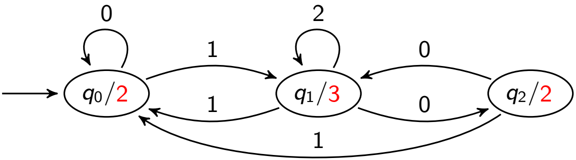


Input in a mix-ary numeration system: A defines admissible input words.

Bringing numbers into mix-ary format w.r.t. A:

Mix-Automaticity

Base determiner for the DFAO A:

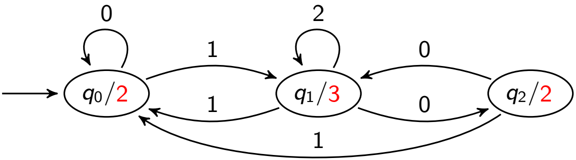


Input in a mix-ary numeration system: A defines admissible input words.

Bringing numbers into mix-ary format w.r.t. A:

Mix-Automaticity

Base determiner for the DFAO A:



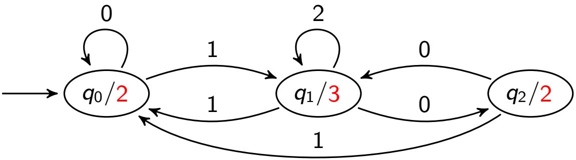
Input in a mix-ary numeration system: A defines admissible input words.

Bringing numbers into mix-ary format w.r.t. A:

$$(5)_{q_0} = ((101)_2)_{q_0} \to ((10)_2)_{q_1}1 = ((2)_3)_{q_1}1 \to (0)_{q_2}21 = 21$$

Mix-Automaticity

Base determiner for the DFAO A:



Input in a mix-ary numeration system: A defines admissible input words.

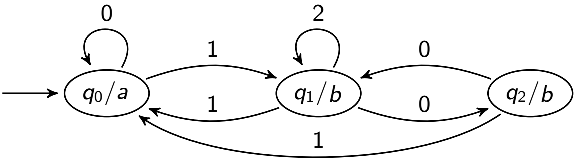
Bringing numbers into mix-ary format w.r.t. A:

$$(5)_{q_0} = ((101)_2)_{q_0} \to ((10)_2)_{q_1}1 = ((2)_3)_{q_1}1 \to (0)_{q_2}21 = 21$$

$$\begin{aligned}(23)_{q_0} &= ((10111)_2)_{q_0} \to ((1011)_2)_{q_1}1 = (11)_{q_1}1 = ((102)_3)_{q_1}1 \\ &\to ((10)_3)_{q_1}21 = (3)_{q_1}21 = ((10)_3)_{q_1}21 \\ &\to ((1)_2)_{q_0}021 \to (0)_{q_1}1021\end{aligned}$$

Mix-Automaticity

Mix-DFAO A:



Input in a mix-ary numeration system: A defines admissible input words.

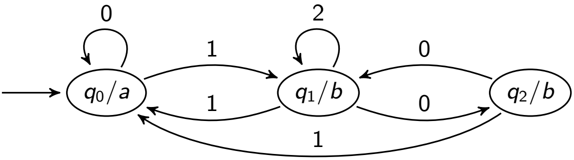
Bringing numbers into mix-ary format w.r.t. A:

$$\begin{aligned}
(5)_{q_0} &= ((101)_2)_{q_0} \to ((10)_2)_{q_1}1 = ((2)_3)_{q_1}1 \to (0)_{q_2}21 = 21 \\
(23)_{q_0} &= ((10111)_2)_{q_0} \to ((1011)_2)_{q_1}1 = (11)_{q_1}1 = ((102)_3)_{q_1}1 \\
&\to ((10)_3)_{q_1}21 = (3)_{q_1}21 = ((10)_3)_{q_1}21 \\
&\to ((1)_2)_{q_0}021 \to (0)_{q_1}1021
\end{aligned}$$

A defines the *mix-automatic* sequence:

$$a : b : b : a : b : b : a : a : b : b : b : a : a : a : a : b : b : b : b : b : b : a : a : a : a : b : a : b : \ldots$$

From mix-automatic to zip-mix specifications



The corresponding zip-mix specification:

$$\begin{aligned} X_0 &= a : X_0' & \qquad X_0' &= \mathrm{zip}_2(X_1, X_0') \\ X_1 &= b : X_1' & \qquad X_1' &= \mathrm{zip}_3(X_0, X_1, X_2') \\ X_2 &= b : X_2' & \qquad X_2' &= \mathrm{zip}_2(X_0, X_1') \end{aligned}$$

Proposition

Mix-automatic sequences can be specified by zip-mix-specifications.

Mix-observation graphs

$$\begin{tikzcd} S \arrow[rr, "\llbracket\cdot\rrbracket"] \arrow[d, "\langle o, n\rangle"'] & & \Delta^\omega \arrow[d, "\mathcal{N}_\kappa"] \\ \sum_{k=2}^{\infty} \Delta \times S^k \arrow[rr, "\sum_{k=2}^{\infty} \mathrm{id} \times \llbracket\cdot\rrbracket^k"] & & \sum_{k=2}^{\infty} \Delta \times (\Delta^\omega)^k \end{tikzcd}$$

Theorem

For streams $\sigma \in \Delta^\omega$ the following are equivalent:

1. *The stream σ is mix-automatic.*
2. *The stream σ can be defined by a zip-mix specification.*
3. *There exists a finite mix-observation graph defining σ.*

Mix-automatic, but not automatic

Proposition

The class of mix-automatic sequences properly extends that of the automatic sequences.

Proof.

We use:

1. Arithmetical subsequences of k-auto sequences are k-auto.
2. Cobham's theorem (1969): Suppose that σ is both k-auto and ℓ-auto for multiplicatively independent $k, \ell \geq 2$ (i.e. $k^i \neq \ell^j$ for all $i, j > 0$). Then σ is eventually periodic.

Let σ k-auto, and τ and ℓ-auto, for multiplicatively independent k en ℓ, and neither is ultimately periodic.If $\mathrm{zip}(\sigma, \tau)$ were m-auto, for some m, then by (1) so would be σ and τ.But then, by (2), $k^{i_1} = m^{j_1}$ and $\ell^{i_2} = m^{j_2}$ for some $i_0, i_1, j_0, j_1 > 0$, which implies the wrong statement: $k^i = \ell^j$ for some $i, j > 0$.Consequently, $\mathrm{zip}(\sigma, \tau)$ is mix-auto, but not automatic. $\square$

Equivalence problem for *zip-mix*-specifications

Question

Is equivalence decidable for streams definable by zip-mix specifications?

Proposition (partial results)

The equivalence problem for defined streams is decidable for:

- *zip_k-specifications versus zip_ℓ-specifications.*
- *zip_k-specifications versus zip-mix-specifications.*

Beyond decidability: zip-mix + $\pi_{i,k}$ specifications

zip$^\pi$-terms over $\langle \Delta, \mathcal{X} \rangle$:

$$Z ::= \mathsf{X} \mid a : Z \mid \mathsf{zip}_k(Z, \ldots, Z) \mid \pi_{i,k}(Z) \qquad (\mathsf{X} \in \mathcal{X},\ a \in \Delta)$$

zip$^\pi$-specifications have equations of the form:

$$\mathsf{X} = t \qquad\qquad (t \text{ a zip}^\pi\text{-term over } \langle \Delta, \mathcal{X} \rangle)$$

Theorem

Deciding equality of streams defined by productive zip$^\pi$-specifications is undecidable, and more precisely, Π^0_1-complete.

Dynamic logic representation

Let $F(X) = \{0,1\} \times X$.
PDL sentences φ and programs π:

$$\varphi ::= 0 \mid 1 \mid \neg\varphi \mid \varphi \wedge \varphi \mid [\pi]\varphi$$
$$\pi ::= \mathsf{even} \mid \mathsf{odd} \mid \pi;\pi \mid \pi \sqcup \pi \mid \pi^*$$

Interpretation of formulas in a F-coalgebra $\mathcal{G} = \langle S, \langle o, n\rangle\rangle$,
or in models $\mathcal{G} = \langle S, \mathbf{0}, \mathbf{1}, \mathbf{even}, \mathbf{odd}\rangle$ where $\mathbf{0}, \mathbf{1} \subseteq S$, $\mathbf{even}, \mathbf{odd} \subseteq S^2$:

$$\begin{aligned}
[\![0]\!] &= \{x \in S : x \in \mathbf{0}\} & [\![\mathsf{even}]\!] &= \mathbf{even} \\
[\![1]\!] &= \{x \in S : x \in \mathbf{1}\} & [\![\mathsf{odd}]\!] &= \mathbf{odd} \\
[\![\varphi \wedge \psi]\!] &= [\![\varphi]\!] \cap [\![\psi]\!] & [\![\pi_1;\pi_2]\!] &= [\![\pi_1]\!];[\![\pi_2]\!] \\
[\![\neg\varphi]\!] &= S \setminus [\![\varphi]\!] & [\![\pi_1 \sqcup \pi_2]\!] &= [\![\pi_1]\!] \cup [\![\pi_2]\!] \\
& & [\![\pi^*]\!] &= [\![\pi]\!]^*
\end{aligned}$$

$$[\![[\pi]\varphi]\!] = \{x : (\forall y)(\langle x, y\rangle \in [\![\pi]\!] \to y \in [\![\varphi]\!])\}$$

Dynamic Logic representation

Proposition (preservation of validity)

If $f : M \to N$ morphism of models and $x \models \varphi$ in M, then $f(x) \models \varphi$ in N.

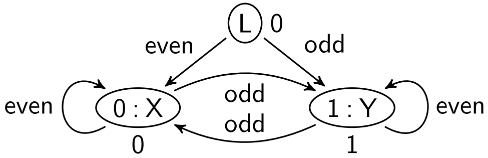
Proposition (characterisation)

For every finite pointed model $\langle \mathcal{G}, x \rangle$ there is a sentence φ_x of PDL so that for all F-coalgebras $\langle \mathcal{H}, y \rangle$, the following are equivalent:

1. *$y \models \varphi_x$ in $\mathcal{H}$.*
2. *There is a bisimulation between $\mathcal{G}$ and $\mathcal{H}$ relating x to y.*

We call φ_x the characterizing sentence of x.

Characterising Thue–Morse



The formula $\varphi_{\mathsf{TM}} = \varphi \wedge [(\mathsf{even} \sqcup \mathsf{odd})^*](\varphi \vee \psi)$ with:

$$\varphi = 0 \wedge \neg 1 \wedge \langle \mathsf{even} \rangle 0 \wedge [\mathsf{even}]0 \wedge \langle \mathsf{odd} \rangle 1 \wedge [\mathsf{odd}]1$$

$$\psi = \neg 0 \wedge 1 \wedge \langle \mathsf{even} \rangle 1 \wedge [\mathsf{even}]1 \wedge \langle \mathsf{odd} \rangle 0 \wedge [\mathsf{odd}]0$$

is a characteristic sentence for the Thue–Morse sequence *TM*:

$$\sigma \models \varphi_{\mathsf{TM}} \iff \sigma = \mathsf{TM}$$

Dynamic Logic representation

Proposition

The following finite model properties hold:

1. *If a sentence φ has a model, it has a finite model (using [Kozen and Parikh, 1981]).*
2. *If φ has a model in which* even *and* odd *are total functions, then it has a finite model with these properties (using [Ben-Ari, Halpern, and Pnueli, 1982]).*

Theorem

The following are equivalent for $\sigma \in \Delta^\omega$:

1. *σ is 2-automatic.*
2. *There is a characterising sentence φ for σ, i.e. for all $\tau \in \Delta^\omega$:*

$$\tau \models \varphi \text{ in } \langle \Delta^\omega, \langle \mathrm{hd}, \mathrm{even}, \mathrm{odd} \rangle \rangle \text{ iff } \tau = \sigma.$$

Our results

- Zip(-k)-stream-specifications
  - equivalence problem is decidable
  - by reduction to bisimilarity of associated observation graphs
  - $\mathcal{O}_k$- and $\mathcal{N}_k$-observation graphs
  - finality of $\langle \Delta^\omega, \mathcal{O}_k \rangle$ for the functor $F(X) = \Delta \times X^k$
- Correspondence with automatic sequences:
  - $\mathcal{N}_k$-observation graphs correspond to zero-consistent DFAO's
  - k-automatic $=$ zip-k-definable
- Mix-DFAO's and mix-automaticity
  - properly extend automatic sequences
  - equivalence problem still decidable?
  - undecidable if projections $\pi_{i,k}$ are added (Π^0_1-complete if productive)
- dynamic logic representation of automatic sequences