Equivalence of Stream Specifications

Jörg Endrullis⋆, Clemens Grabmayer†, Dimitri Hendriks⋆, Jan Willem Klop⋆, and Larry Moss‡

† Universiteit Utrecht
⋆ Vrije Universiteit, Amsterdam
‡ Indiana University

2nd Workshop on Proof Theory and Rewriting
Obergurgl, 29th March 2010

Overview

- Ad: International Summer School Rewriting in Utrecht 3-8 July
  http://www.utrechtsummerschool.nl
- ROS: Realising Optimal Sharing (NWO-project)
- Equivalence of stream specifications
  - stream specifications
  - equivalence of stream specifications
  - productivity vs. unique solvability
  - zip-specifications, Larry Moss' question
  - solution: decidability of equivalence for zip-specs
  - extensions of the result
- Summary

Overview

1. ROS
2. Stream Equality
3. Summary

Overview

Realising Optimal Sharing (ROS)

NWO-Project (2009–2012/13) at Utrecht University linking:

- Dept. of Philosophy (Theor. Philosophy)
- Dept. of Computer Science (Functional Languages)

Aims

- Study optimal-sharing implementations of the λ-calculus
- Try to incorporate optimal-sharing techniques in the Utrecht Haskell Compiler (UHC)

People

- Phil: Vincent van Oostrom (principal investigator), CG (postdoc/3 years)
- CS: Doaitse Swierstra and Atze Dijkstra, Jan Rochel (PhD student/4 years)

Realising Optimal Sharing (ROS)

NWO-Project (2009–2012/13) at Utrecht University linking:

- Dept. of Philosophy (Theor. Philosophy)
- Dept. of Computer Science (Functional Languages)

Aims

- Study optimal-sharing implementations of the λ-calculus
- Try to incorporate optimal-sharing techniques in the Utrecht Haskell Compiler (UHC)

People

- Phil: Vincent van Oostrom (principal investigator),
  CG (postdoc/3 years)
- CS: Doaitse Swierstra and Atze Dijkstra,
  Jan Rochel (PhD student/4 years)

Realising Optimal Sharing (ROS)

NWO-Project (2009–2012/13) at Utrecht University linking:

- Dept. of Philosophy (Theor. Philosophy)
- Dept. of Computer Science (Functional Languages)

Aims

- Study optimal-sharing implementations of the λ-calculus
- Try to incorporate optimal-sharing techniques in the Utrecht Haskell Compiler (UHC)

People

- Phil: Vincent van Oostrom (principal investigator),
  CG (postdoc/3 years)
- CS: Doaitse Swierstra and Atze Dijkstra,
  Jan Rochel (PhD student/4 years)

Research questions

Aims (more detail)

- Theory: contribute to the graph rewrite theory of optimal implementations of rewrite systems, e.g.:
  - refine existing implementations of weak β-reduction by OTRSs
  - refine, adapt for the practice, and compare with other approaches, the LamdaScope optimal implementation of λ-calculus by interaction nets.
  - relation semantics for graph rewrite systems (Birkhoff-theorem?)
- Theory/Practice: gain an overview of existing optimal and non-optimal sharing techniques

Research questions

Aims (more detail)

- Theory: contribute to the graph rewrite theory of optimal implementations of rewrite systems, e.g.:
  - refine existing implementations of weak β-reduction by OTRSs
  - refine, adapt for the practice, and compare with other approaches, the LamdaScope optimal implementation of λ-calculus by interaction nets.
  - relation semantics for graph rewrite systems (Birkhoff-theorem?)
- Theory/Practice: gain an overview of existing optimal and non-optimal sharing techniques

Research questions

Aims (more detail)

- Practice: investigate applications for optimal-sharing techniques for compiler construction
  - find convincing 'real-life' examples in which optimal-sharing algorithms perform better than existing (Haskell) compilers
  - isolate classes of programs where using optimal evaluation leads to speed-up, with the aim of incorporating in UHC of certain Haskell-programs.
  - also interested in applying non-optimal sharing techniques (not already in use)

Overview

Stream Specifications

Example

The specifications:

$$\mathsf{alt} = 0 : 1 : \mathsf{alt}$$

$$\mathsf{alt}_1 = 0 : \mathsf{alt}'_1$$
$$\mathsf{alt}'_1 = 1 : \mathsf{alt}_1$$

define the stream $0 : 1 : 0 : 1 : 0 : 1 : \ldots$.

The same is true for the specification:

$$\mathsf{alt}_2 = \mathsf{zip}(\mathsf{zeros}, \mathsf{ones})$$
$$\mathsf{zeros} = 0 : \mathsf{zeros}$$
$$\mathsf{ones} = 1 : \mathsf{ones}$$
$$\mathsf{zip}(x : \sigma, y : \tau) = x : y : \mathsf{zip}(\sigma, \tau)$$

Stream Specifications

Example

The specifications:

$$\mathsf{alt} = 0 : 1 : \mathsf{alt}$$

$$\mathsf{alt}_1 = 0 : \mathsf{alt}_1' \qquad \mathsf{alt}_1' = 1 : \mathsf{alt}_1$$

define the stream $0 : 1 : 0 : 1 : 0 : 1 : \ldots$.

The same is true for the specification:

$$\mathsf{alt}_2 = \mathsf{zip}(\mathsf{zeros}, \mathsf{ones})$$
$$\mathsf{zeros} = 0 : \mathsf{zeros}$$
$$\mathsf{ones} = 1 : \mathsf{ones}$$
$$\mathsf{zip}(x : \sigma, y : \tau) = x : y : \mathsf{zip}(\sigma, \tau)$$

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)

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

$$L \twoheadrightarrow 0 : 1 : 1 : 0 : 1 : 0 : 0 : 1 : 1 : 0 : 0 : 1 : 0 : 1 : 1 : 0 : \ldots$$

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)

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

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

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

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

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

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)

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

$$L \twoheadrightarrow 0 : 1 : 1 : 0 : 1 : 0 : 0 : 1 : 1 : 0 : 0 : 1 : 0 : 1 : 1 : 0 : \dots$$

Specifying Streams

Example (Thue–Morse stream)

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

one finds: T

A stream specification is productive if lazy/fair evaluation of its root M_0 results in an infinite constructor normal form (representing a stream).

Specifying Streams

Example (Thue–Morse stream)

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

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

A stream specification is productive if lazy/fair evaluation of its root M_0 results in an infinite constructor normal form (representing a stream).

Specifying Streams

Example (Thue–Morse stream)

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

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

A stream specification is productive if lazy/fair evaluation of its root M_0 results in an infinite constructor normal form (representing a stream).

Specifying Streams

Example (Thue–Morse stream)

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

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

A stream specification is productive if lazy/fair evaluation of its root M_0 results in an infinite constructor normal form (representing a stream).

Specifying Streams

Example (Thue–Morse stream)

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

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

A stream specification is productive if lazy/fair evaluation of its root M_0 results in an infinite constructor normal form (representing a stream).

Specifying Streams

Example (Thue–Morse stream)

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

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

A stream specification is productive if lazy/fair evaluation of its root M_0 results in an infinite constructor normal form (representing a stream).

Specifying Streams

Example (Thue–Morse stream)

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

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

A stream specification is productive if lazy/fair evaluation of its root M_0 results in an infinite constructor normal form (representing a stream).

Specifying Streams

Example (Thue–Morse stream)

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

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

A stream specification is productive if lazy/fair evaluation of its root M_0 results in an infinite constructor normal form (representing a stream).

zip-specifications: motivating question

Consider zip-specifications formed with zip-terms built from:

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

and with defining equations:

$$M_i = C_i[M_1, \ldots, M_n] \qquad (i = 0, \ldots, n)$$

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

where C_i are zip-term contexts with n holes.

Question

Is equivalence of specified stream decidable for zip-specifications?

zip-specifications: motivating question

Consider zip-specifications formed with zip-terms built from:

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

and with defining equations:

$$M_i = C_i[M_1, \ldots, M_n] \qquad (i = 0, \ldots, n)$$

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

where C_i are zip-term contexts with n holes.

Question

Is equivalence of specified stream decidable for zip-specifications?

zip-specifications: motivating question

Consider zip-specifications formed with zip-terms built from:

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

and with defining equations:

$$M_i = C_i[M_1, \ldots, M_n] \qquad (i = 0, \ldots, n)$$

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

where C_i are zip-term contexts with n holes.

Question

Is equivalence of specified stream decidable for zip-specifications?

zip-specifications: motivating question

Consider zip-specifications formed with zip-terms built from:

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

and with defining equations:

$$M_i = C_i[M_1, \ldots, M_n] \qquad (i = 0, \ldots, n)$$

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

where C_i are zip-term contexts with n holes.

Question

Is equivalence of specified stream decidable for zip-specifications?

Some known results / existing tools

Equivalence of stream specifications

- Π^0_2-complete (Roşu, 2006)
- Proof Tool *Circ* of Roşu for stream equivalence.

Productivity of stream specifications

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

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

  `infinity.few.vu.nl/productivity/tool.html`

Some known results / existing tools

Equivalence of stream specifications

- Π^0_2-complete (Roşu, 2006)
- Proof Tool *Circ* of Roşu for stream equivalence.

Productivity of stream specifications

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

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

  `infinity.few.vu.nl/productivity/tool.html`

Some known results / existing tools

Equivalence of stream specifications

- Π^0_2-complete (Roşu, 2006)
- Proof Tool *Circ* of Roşu for stream equivalence.

Productivity of stream specifications

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

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

  `infinity.few.vu.nl/productivity/tool.html`

Roadmap to a decidability result

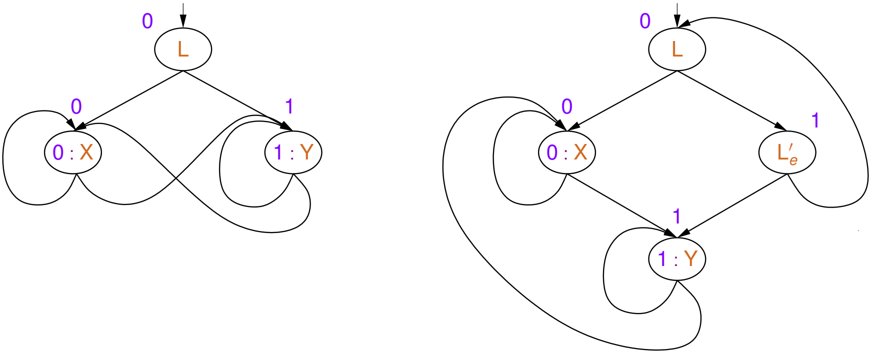
- unique solvability versus productivity for zip-specs
- transformation into 'zip-guarded', and 'flat' zip-specs
- 'observation graphs' of flat zip-specs
  - using a rewrite system that employs the ⟨head, even, odd⟩-cobasis for streams
- link between:
  - equivalence of zip-specs, and
  - bisimilarity of associated observation graphs
- using bisimilarity-checking to decide equivalence of zip-specs

Roadmap: uphill to observation graphs

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

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

Roadmap: uphill to observation graphs



Unique Solvability versus Productivity

Proposition

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

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

Hence: Productivity is decidable for zip-specifications.

Example

- $Z = \mathsf{zip}(Z, \mathsf{zip}(Z, 0 : Z))$ is neither productive nor uniquely solvable.
- $Z = \mathsf{zip}(0 : Z, \mathsf{zip}(Z, 0 : Z))$ is productive and uniquely solvable.

Unique Solvability versus Productivity

Proposition

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

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

Hence: Productivity is decidable for zip-specifications.

Example

- $\mathsf{Z} = \mathsf{zip}(\mathsf{Z}, \mathsf{zip}(\mathsf{Z}, 0 : \mathsf{Z}))$ is neither productive nor uniquely solvable.
- $\mathsf{Z} = \mathsf{zip}(0 : \mathsf{Z}, \mathsf{zip}(\mathsf{Z}, 0 : \mathsf{Z}))$ is productive and uniquely solvable.

zip-guarded zip-specifications

A zip-specification $\mathcal{S}$ is called zip-guarded if every cycle in $\mathcal{S}$ contains an occurrence of zip.

Non-Example/Example

Non-Example

$$\mathsf{alt}_2 = \mathsf{zip}(\mathsf{zeros}, \mathsf{ones})$$
$$\mathsf{zeros} = 0 : \mathsf{zeros}$$
$$\mathsf{ones} = 1 : \mathsf{ones}$$
$$\mathsf{zip}(x : \sigma, \tau) = x : \mathsf{zip}(\tau, \sigma)$$

Example

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

zip-guarded zip-specifications

Lemma

Every productive zip-specification can be transformed into an equivalent zip-guarded and productive zip-specification.

Idea of Proof: Remove cycles that specify periodic streams.

Every cycle $M = c : M$ of length 1 can be replaced by:

$$M = c : \mathrm{zip}(M, M) \;;$$

A cycle $M = a : b : M$ of length 2 can be replaced by the spec:

$$M = \mathrm{zip}(M_a, M_b) \qquad M_a = a : \mathrm{zip}(M_a, M_a) \qquad M_b = b : \mathrm{zip}(M_b, M_b) \;;$$

A cycle $M = a : b : c : M$ of length 3 by the specification:

$$M_{abc} = \mathrm{zip}(a : c : M_{bac}, M_{bac}) \qquad M_{bac} = \mathrm{zip}(b : c : M_{abc}, M_{abc}) \;.$$

cycles of even length: split into cycles of odd length;
cycles of odd length n: idea as for length 3 applies. □

zip-guarded zip-specifications

Lemma

Every productive zip-specification can be transformed into an equivalent zip-guarded and productive zip-specification.

Idea of Proof: Remove cycles that specify periodic streams.

Every cycle $\mathsf{M} = \mathsf{c} : \mathsf{M}$ of length 1 can be replaced by:

$$\mathsf{M} = \mathsf{c} : \mathsf{zip}(\mathsf{M}, \mathsf{M}) \ ;$$

A cycle $\mathsf{M} = \mathsf{a} : \mathsf{b} : \mathsf{M}$ of length 2 can be replaced by the spec:

$$\mathsf{M} = \mathsf{zip}(\mathsf{M}_a, \mathsf{M}_b) \qquad \mathsf{M}_a = \mathsf{a} : \mathsf{zip}(\mathsf{M}_a, \mathsf{M}_a) \qquad \mathsf{M}_b = \mathsf{b} : \mathsf{zip}(\mathsf{M}_b, \mathsf{M}_b) \ ;$$

A cycle $\mathsf{M} = \mathsf{a} : \mathsf{b} : c : \mathsf{M}$ of length 3 by the specification:

$$\mathsf{M}_{abc} = \mathsf{zip}(\mathsf{a} : \mathsf{c} : \mathsf{M}_{bac}, \mathsf{M}_{bac}) \qquad \mathsf{M}_{bac} = \mathsf{zip}(\mathsf{b} : \mathsf{c} : \mathsf{M}_{abc}, \mathsf{M}_{abc}) \ .$$

cycles of even length: split into cycles of odd length;
cycles of odd length *n*: idea as for length 3 applies. □

zip-guarded zip-specifications

Lemma

Every productive zip-specification can be transformed into an equivalent zip-guarded and productive zip-specification.

Idea of Proof: Remove cycles that specify periodic streams.

Every cycle $\mathsf{M} = \mathsf{c} : \mathsf{M}$ of length 1 can be replaced by:

$$\mathsf{M} = \mathsf{c} : \mathsf{zip}(\mathsf{M}, \mathsf{M}) \; ;$$

A cycle $\mathsf{M} = \mathsf{a} : \mathsf{b} : \mathsf{M}$ of length 2 can be replaced by the spec:

$$\mathsf{M} = \mathsf{zip}(\mathsf{M}_a, \mathsf{M}_b) \qquad \mathsf{M}_a = \mathsf{a} : \mathsf{zip}(\mathsf{M}_a, \mathsf{M}_a) \qquad \mathsf{M}_b = \mathsf{b} : \mathsf{zip}(\mathsf{M}_b, \mathsf{M}_b) \; ;$$

A cycle $\mathsf{M} = \mathsf{a} : \mathsf{b} : c : \mathsf{M}$ of length 3 by the specification:

$$\mathsf{M}_{abc} = \mathsf{zip}(\mathsf{a} : \mathsf{c} : \mathsf{M}_{bac}, \mathsf{M}_{bac}) \qquad \mathsf{M}_{bac} = \mathsf{zip}(\mathsf{b} : \mathsf{c} : \mathsf{M}_{abc}, \mathsf{M}_{abc}) \; .$$

cycles of even length: split into cycles of odd length;
cycles of odd length n: idea as for length 3 applies. □

zip-guarded zip-specifications

Lemma

Every productive zip-specification can be transformed into an equivalent zip-guarded and productive zip-specification.

Idea of Proof: Remove cycles that specify periodic streams.

Every cycle $\mathsf{M} = \mathsf{c} : \mathsf{M}$ of length 1 can be replaced by:

$$\mathsf{M} = \mathsf{c} : \mathsf{zip}(\mathsf{M}, \mathsf{M}) \;;$$

A cycle $\mathsf{M} = \mathsf{a} : \mathsf{b} : \mathsf{M}$ of length 2 can be replaced by the spec:

$$\mathsf{M} = \mathsf{zip}(\mathsf{M}_a, \mathsf{M}_b) \qquad \mathsf{M}_a = \mathsf{a} : \mathsf{zip}(\mathsf{M}_a, \mathsf{M}_a) \qquad \mathsf{M}_b = \mathsf{b} : \mathsf{zip}(\mathsf{M}_b, \mathsf{M}_b) \;;$$

A cycle $\mathsf{M} = \mathsf{a} : \mathsf{b} : c : \mathsf{M}$ of length 3 by the specification:

$$\mathsf{M}_{abc} = \mathsf{zip}(\mathsf{a} : \mathsf{c} : \mathsf{M}_{bac}, \mathsf{M}_{bac}) \qquad \mathsf{M}_{bac} = \mathsf{zip}(\mathsf{b} : \mathsf{c} : \mathsf{M}_{abc}, \mathsf{M}_{abc}) \,.$$

cycles of even length: split into cycles of odd length;
cycles of odd length n: idea as for length 3 applies. □

Flat zip-specifications

A zip-guarded spec $\mathcal{S}$ is called flat if its equations are of the form:

$$\mathsf{M}_i = \mathsf{c}_{i,1} : \ldots : \mathsf{c}_{i,m_i} : \mathsf{zip}(\mathsf{M}_{i,1}, \mathsf{M}_{i,2}) \qquad \text{for } i = 0, \ldots, n$$

Proposition

Every zip-guarded specification $\mathcal{S}$ can be transformed into a flat zip-specification $\mathcal{S}'$ with the same solutions.

Idea of Proof. Introduce new recursion variables. E.g., the spec:

$$\mathsf{M} = 0 : \mathsf{zip}(1 : \mathsf{zip}(\mathsf{M}, \mathsf{M}), 0 : \mathsf{M})$$

can be transformed into the spec:

$$\mathsf{M} = 0 : \mathsf{zip}(\mathsf{M}_1, \mathsf{M}_2)$$
$$\mathsf{M}_1 = 1 : \mathsf{zip}(\mathsf{M}, \mathsf{M})$$
$$\mathsf{M}_2 = 0 : 0 : \mathsf{zip}(\mathsf{M}_1, \mathsf{M}_2)$$

Flat zip-specifications

A zip-guarded spec $\mathcal{S}$ is called flat if its equations are of the form:

$$\mathsf{M}_i = \mathsf{c}_{i,1} : \ldots : \mathsf{c}_{i,m_i} : \mathsf{zip}(\mathsf{M}_{i,1}, \mathsf{M}_{i,2}) \qquad \text{for } i = 0, \ldots, n$$

Proposition

Every zip-guarded specification $\mathcal{S}$ can be transformed into a flat zip-specification $\mathcal{S}'$ with the same solutions.

Idea of Proof. Introduce new recursion variables. E.g., the spec:

$$\mathsf{M} = 0 : \mathsf{zip}(1 : \mathsf{zip}(\mathsf{M}, \mathsf{M}), 0 : \mathsf{M})$$

can be transformed into the spec:

$$\mathsf{M} = 0 : \mathsf{zip}(\mathsf{M}_1, \mathsf{M}_2)$$
$$\mathsf{M}_1 = 1 : \mathsf{zip}(\mathsf{M}, \mathsf{M})$$
$$\mathsf{M}_2 = 0 : 0 : \mathsf{zip}(\mathsf{M}_1, \mathsf{M}_2)$$

Flat zip-specifications

A zip-guarded spec $\mathcal{S}$ is called flat if its equations are of the form:

$$\mathsf{M}_i = \mathsf{c}_{i,1} : \ldots : \mathsf{c}_{i,m_i} : \mathsf{zip}(\mathsf{M}_{i,1}, \mathsf{M}_{i,2}) \qquad \text{for } i = 0, \ldots, n$$

Proposition

Every zip-guarded specification $\mathcal{S}$ can be transformed into a flat zip-specification $\mathcal{S}'$ with the same solutions.

Idea of Proof. Introduce new recursion variables. E.g., the spec:

$$\mathsf{M} = 0 : \mathsf{zip}(1 : \mathsf{zip}(\mathsf{M}, \mathsf{M}), 0 : \mathsf{M})$$

can be transformed into the spec:

$$\mathsf{M} = 0 : \mathsf{zip}(\mathsf{M}_1, \mathsf{M}_2)$$
$$\mathsf{M}_1 = 1 : \mathsf{zip}(\mathsf{M}, \mathsf{M})$$
$$\mathsf{M}_2 = 0 : 0 : \mathsf{zip}(\mathsf{M}_1, \mathsf{M}_2)$$

Flat zip-specifications

A zip-guarded spec $\mathcal{S}$ is called flat if its equations are of the form:

$$\mathsf{M}_i = \mathsf{c}_{i,1} : \ldots : \mathsf{c}_{i,m_i} : \mathsf{zip}(\mathsf{M}_{i,1}, \mathsf{M}_{i,2}) \qquad \text{for } i = 0, \ldots, n$$

Proposition

Every zip-guarded specification $\mathcal{S}$ can be transformed into a flat zip-specification $\mathcal{S}'$ with the same solutions.

Idea of Proof. Introduce new recursion variables. E.g., the spec:

$$\mathsf{M} = 0 : \mathsf{zip}(1 : \mathsf{zip}(\mathsf{M}, \mathsf{M}), 0 : \mathsf{M})$$

can be transformed into the spec:

$$\mathsf{M} = 0 : \mathsf{zip}(\mathsf{M}_1, \mathsf{M}_2)$$
$$\mathsf{M}_1 = 1 : \mathsf{zip}(\mathsf{M}, \mathsf{M})$$
$$\mathsf{M}_2 = 0 : 0 : \mathsf{zip}(\mathsf{M}_1, \mathsf{M}_2)$$

Flat zip-specifications

A zip-guarded spec $\mathcal{S}$ is called flat if its equations are of the form:

$$\mathsf{M}_i = \mathsf{c}_{i,1} : \ldots : \mathsf{c}_{i,m_i} : \mathsf{zip}(\mathsf{M}_{i,1}, \mathsf{M}_{i,2}) \qquad \text{for } i = 0, \ldots, n$$

Proposition

Every zip-guarded specification $\mathcal{S}$ can be transformed into a flat zip-specification $\mathcal{S}'$ with the same solutions.

Idea of Proof. Introduce new recursion variables. E.g., the spec:

$$\mathsf{M} = 0 : \mathsf{zip}(1 : \mathsf{zip}(\mathsf{M}, \mathsf{M}), 0 : \mathsf{M})$$

can be transformed into the spec:

$$\mathsf{M} = 0 : \mathsf{zip}(\mathsf{M}_1, \mathsf{M}_2)$$
$$\mathsf{M}_1 = 1 : \mathsf{zip}(\mathsf{M}, \mathsf{M})$$
$$\mathsf{M}_2 = 0 : 0 : \mathsf{zip}(\mathsf{M}_1, \mathsf{M}_2)$$

Flat zip-specifications

A zip-guarded spec $\mathcal{S}$ is called flat if its equations are of the form:

$$\mathsf{M}_i = \mathsf{c}_{i,1} : \ldots : \mathsf{c}_{i,m_i} : \mathsf{zip}(\mathsf{M}_{i,1}, \mathsf{M}_{i,2}) \qquad \text{for } i = 0, \ldots, n$$

Proposition

Every zip-guarded specification $\mathcal{S}$ can be transformed into a flat zip-specification $\mathcal{S}'$ with the same solutions.

Idea of Proof. Introduce new recursion variables. E.g., the spec:

$$\mathsf{M} = 0 : \mathsf{zip}(1 : \mathsf{zip}(\mathsf{M}, \mathsf{M}), 0 : \mathsf{M})$$

can be transformed into the spec:

$$\mathsf{M} = 0 : \mathsf{zip}(\mathsf{M}_1, \mathsf{M}_2)$$
$$\mathsf{M}_1 = 1 : \mathsf{zip}(\mathsf{M}, \mathsf{M})$$
$$\mathsf{M}_2 = 0 : 0 : \mathsf{zip}(\mathsf{M}_1, \mathsf{M}_2)$$

Flat zip-specifications

Example (Thue–Morse)

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

Rewriting zip-terms

For a zip-spec $\mathcal{S}$, the zip-terms over $\mathcal{S}$ are defined by the grammar:

$$Z ::= \mathsf{M}_i \mid \mathsf{c} : Z \mid \mathsf{zip}(Z, Z)$$

Definition

Let $\mathcal{S}$ be a zip-spec. The TRS R on zip-terms over $\mathcal{S}$ has the rules:

$$\mathsf{head}(x : t) \to x \qquad \mathsf{head}(\mathsf{zip}(s, t)) \to \mathsf{head}(s)$$
$$\mathsf{even}(x : t) \to x : \mathsf{odd}(t) \qquad \mathsf{even}(\mathsf{zip}(s, t)) \to s$$
$$\mathsf{odd}(x : t) \to \mathsf{even}(t) \qquad \mathsf{odd}(\mathsf{zip}(s, t)) \to t$$

and, in addition, for each equation $\mathsf{M}_i = t$ of $\mathcal{S}$, rules:

$$\mathsf{head}(\mathsf{M}_i) \to \mathsf{head}(t) \qquad \mathsf{even}(\mathsf{M}_i) \to \mathsf{even}(t) \qquad \mathsf{odd}(\mathsf{M}_i) \to \mathsf{odd}(t)$$

By $t{\downarrow}$ we denote the normal form of t with respect to R.

R is orthogonal, hence CR. If $\mathcal{S}$ is product., R is terminating, thus UN.

Rewriting zip-terms

For a zip-spec $\mathcal{S}$, the zip-terms over $\mathcal{S}$ are defined by the grammar:

$$Z ::= \mathsf{M}_i \mid \mathsf{c} : Z \mid \mathsf{zip}(Z, Z)$$

Definition

Let $\mathcal{S}$ be a zip-spec. The TRS R on zip-terms over $\mathcal{S}$ has the rules:

$$\begin{aligned} \mathsf{head}(x : t) &\to x & \mathsf{head}(\mathsf{zip}(s, t)) &\to \mathsf{head}(s) \\ \mathsf{even}(x : t) &\to x : \mathsf{odd}(t) & \mathsf{even}(\mathsf{zip}(s, t)) &\to s \\ \mathsf{odd}(x : t) &\to \mathsf{even}(t) & \mathsf{odd}(\mathsf{zip}(s, t)) &\to t \end{aligned}$$

and, in addition, for each equation $\mathsf{M}_i = t$ of $\mathcal{S}$, rules:

$$\mathsf{head}(\mathsf{M}_i) \to \mathsf{head}(t) \quad \mathsf{even}(\mathsf{M}_i) \to \mathsf{even}(t) \quad \mathsf{odd}(\mathsf{M}_i) \to \mathsf{odd}(t)$$

By $t{\downarrow}$ we denote the normal form of t with respect to R.

R is orthogonal, hence CR. If $\mathcal{S}$ is product., R is terminating, thus UN.

Rewriting zip-terms

For a zip-spec $\mathcal{S}$, the zip-terms over $\mathcal{S}$ are defined by the grammar:

$$Z ::= \mathsf{M}_i \mid \mathsf{c} : Z \mid \mathsf{zip}(Z, Z)$$

Definition

Let $\mathcal{S}$ be a zip-spec. The TRS R on zip-terms over $\mathcal{S}$ has the rules:

$$\mathsf{head}(x : t) \to x \qquad \mathsf{head}(\mathsf{zip}(s, t)) \to \mathsf{head}(s)$$
$$\mathsf{even}(x : t) \to x : \mathsf{odd}(t) \qquad \mathsf{even}(\mathsf{zip}(s, t)) \to s$$
$$\mathsf{odd}(x : t) \to \mathsf{even}(t) \qquad \mathsf{odd}(\mathsf{zip}(s, t)) \to t$$

and, in addition, for each equation $\mathsf{M}_i = t$ of $\mathcal{S}$, rules:

$$\mathsf{head}(\mathsf{M}_i) \to \mathsf{head}(t) \quad \mathsf{even}(\mathsf{M}_i) \to \mathsf{even}(t) \quad \mathsf{odd}(\mathsf{M}_i) \to \mathsf{odd}(t)$$

By $t{\downarrow}$ we denote the normal form of t with respect to R.

R is orthogonal, hence CR. If $\mathcal{S}$ is product., R is terminating, thus UN.

(even, odd)-Derivatives

Definition

Let $\mathcal{S}$ be a zip-guarded zip-specification. Let t a zip-term over $\mathcal{S}$.

(even, odd)-derivatives of t (w.r.t. $\mathcal{S}$) are defined inductively:

- $t{\downarrow}$ is an (even, odd)-derivative of t;
- if s is an (even, odd)-der. of t, then so are even$(s){\downarrow}$ and odd$(s){\downarrow}$.

By $\partial_{\mathcal{S}}(t)$ we denote the set of (even, odd)-derivatives of t.

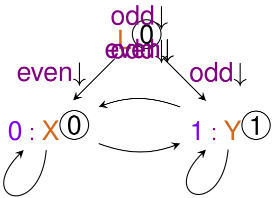
Observation graphs

Definition

Let $\mathcal{S}$ be a zip-guarded, productive zip-specification.

The ((even, odd)-)observation graph $\mathcal{O}(\mathcal{S})$ of $\mathcal{S}$:

- its root node is M_0;
- every node t is labelled with $\mathsf{head}(t)\!\downarrow$;
- every node t has two outgoing edges, even and odd, to the nodes $\mathsf{even}(t)\!\downarrow$, and $\mathsf{odd}(t)\!\downarrow$, resp. .



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

$(\mathsf{ev}, \mathsf{od})$-derivatives *versus* observation graphs

Proposition

Let $\mathcal{S}$ be a zip-guarded, productive zip-specification.

The set of nodes of $\mathcal{O}(\mathcal{S})$ coincides with the set $\partial_{\mathcal{S}}(\mathsf{M}_0)$ of $(\mathsf{even}, \mathsf{odd})$-derivatives of the root M_0 of $\mathcal{S}$.

Hence (at least) for flat specs, the observation graph of $\mathcal{S}$ is finite.

Finiteness of (even, odd)-derivatives

Main Lemma

Let $\mathcal{S}$ be a flat zip-specification.
The set $\partial_{\mathcal{S}}(\mathsf{M}_0)$ of $(\mathsf{even}, \mathsf{odd})$-derivatives of the root M_0 of $\mathcal{S}$ is finite.

Proof.

Since $\mathcal{S}$ is flat, its equations are of the form:

$$\mathsf{M}_i = \mathsf{c}_{i,1} : \ldots : \mathsf{c}_{i,m_i} : \mathsf{zip}(\mathsf{M}_{i,1}, \mathsf{M}_{i,2}) \qquad \text{for } i = 0, \ldots, n$$

Let $m := \max_{0 \le i \le n} m_i$.

It suffices to show that every $t \in \partial_{\mathcal{S}}(\mathsf{M}_0)$ is of the form:

$$\mathsf{c}_1 : \ldots : \mathsf{c}_k : \mathsf{M}_i \tag{1}$$

where $k \le m$, $c_1, \ldots, c_k$ are constants, and M_i a rec. var. of $\mathcal{S}$.

Finiteness of (even, odd)-derivatives

Main Lemma

Let $\mathcal{S}$ be a flat zip-specification.
The set $\partial_{\mathcal{S}}(\mathsf{M}_0)$ of (even, odd)-derivatives of the root M_0 of $\mathcal{S}$ is finite.

Proof.

Since $\mathcal{S}$ is flat, its equations are of the form:

$$\mathsf{M}_i = \mathsf{c}_{i,1} : \ldots : \mathsf{c}_{i,m_i} : \mathsf{zip}(\mathsf{M}_{i,1}, \mathsf{M}_{i,2}) \qquad \text{for } i = 0, \ldots, n$$

Let $m := \max_{0 \le i \le n} m_i$.

It suffices to show that every $t \in \partial_{\mathcal{S}}(\mathsf{M}_0)$ is of the form:

$$\mathsf{c}_1 : \ldots : \mathsf{c}_k : \mathsf{M}_i \tag{1}$$

where $k \le m$, $c_1, \ldots, c_k$ are constants, and M_i a rec. var. of $\mathcal{S}$.

Finiteness of (even, odd)-derivatives (Proof)

Proof (Continued).

We use induction on the definition of derivatives of M_0 over $\mathcal{S}$.

- **Base case.** We note that $\mathsf{M}_0{\downarrow} = \mathsf{M}_0$, and hence (**??**) holds.
- **Induction Step.** Let $t \in \partial_{\mathcal{S}}(\mathsf{M}_0)$ be arbitrary.

  By induction hypothesis, t is of the form (**??**), that is:

  $$t = c_1 : \ldots : c_k : \mathsf{M}_i$$

  for some $k \leq m$, c_1, constants $\ldots$, c_k are constants, and a rec. var. M_i of $\mathcal{S}$.

  We have to show that $\mathsf{even}(t){\downarrow}$ and $\mathsf{odd}(t){\downarrow}$ are again of the form (**??**).

  We treat only check the case of $\mathsf{even}(t){\downarrow}$;
  the case of $\mathsf{odd}(t){\downarrow}$ can be established analogously.

Finiteness of (even, odd)-derivatives (Proof)

Proof (Continued).

We use induction on the definition of derivatives of M_0 over $\mathcal{S}$.

- **Base case.** We note that $\mathsf{M}_0{\downarrow} = \mathsf{M}_0$, and hence (**??**) holds.
- **Induction Step.** Let $t \in \partial_{\mathcal{S}}(\mathsf{M}_0)$ be arbitrary.

  By induction hypothesis, t is of the form (**??**), that is:

  $$t = \mathsf{c}_1 : \ldots : \mathsf{c}_k : \mathsf{M}_i$$

  for some $k \leq m$, c_1, constants $\ldots$, c_k are constants, and a rec. var. M_i of $\mathcal{S}$.

  We have to show that $\mathsf{even}(t){\downarrow}$ and $\mathsf{odd}(t){\downarrow}$ are again of the form (**??**).

  We treat only check the case of $\mathsf{even}(t){\downarrow}$;
  the case of $\mathsf{odd}(t){\downarrow}$ can be established analogously.

Finiteness of (even, odd)-derivatives (Proof)

Proof (Continued).

We use induction on the definition of derivatives of M_0 over $\mathcal{S}$.

- Base case. We note that $\mathsf{M}_0\!\downarrow = \mathsf{M}_0$, and hence (**??**) holds.
- Induction Step. Let $t \in \partial_{\mathcal{S}}(\mathsf{M}_0)$ be arbitrary.

  By induction hypothesis, t is of the form (**??**), that is:

  $$t = \mathsf{c}_1 : \ldots : \mathsf{c}_k : \mathsf{M}_i$$

  for some $k \leq m$, c_1, constants $\ldots$, c_k are constants, and a rec. var. M_i of $\mathcal{S}$.

  We have to show that $\mathsf{even}(t)\!\downarrow$ and $\mathsf{odd}(t)\!\downarrow$ are again of the form (**??**).

  We treat only check the case of $\mathsf{even}(t)\!\downarrow$;
  the case of $\mathsf{odd}(t)\!\downarrow$ can be established analogously.

Finiteness of (even, odd)-derivatives (Proof)

Proof (Continued).

We use induction on the definition of derivatives of M_0 over $\mathcal{S}$.

- Base case. We note that $\mathsf{M}_0{\downarrow} = \mathsf{M}_0$, and hence (**??**) holds.
- Induction Step. Let $t \in \partial_{\mathcal{S}}(\mathsf{M}_0)$ be arbitrary.

  By induction hypothesis, t is of the form (**??**), that is:

  $$t = \mathsf{c}_1 : \ldots : \mathsf{c}_k : \mathsf{M}_i$$

  for some $k \leq m$, c_1, constants $\ldots$, c_k are constants, and a rec. var. M_i of $\mathcal{S}$.

  We have to show that $\mathsf{even}(t){\downarrow}$ and $\mathsf{odd}(t){\downarrow}$ are again of the form (**??**).

  We treat only check the case of $\mathsf{even}(t){\downarrow}$;
  the case of $\mathsf{odd}(t){\downarrow}$ can be established analogously.

Finiteness of (even, odd)-derivatives (Proof)

Proof (Continued).

Induction Step (Continued). We find:

$$\underbrace{\mathsf{even}(\mathsf{c}_1 : \ldots : \mathsf{c}_k : \mathsf{M}_i)}_{=\,\mathsf{even}(t)} \twoheadrightarrow \begin{cases} \mathsf{c}_1 : \mathsf{c}_3 : \ldots : \mathsf{c}_{k-1} : \mathsf{c}_{i,1} : \mathsf{c}_{i,3} : \ldots : \mathsf{M}_{i,1} \\ \qquad \ldots \text{ for } k \text{ even, } \mathsf{M}_i \text{ even} \\ \mathsf{c}_1 : \mathsf{c}_3 : \ldots : \mathsf{c}_{k-1} : \mathsf{c}_{i,1} : \mathsf{c}_{i,3} : \ldots : \mathsf{M}_{i,2} \\ \qquad \ldots \text{ for } k \text{ even, } \mathsf{M}_i \text{ odd} \\ \mathsf{c}_1 : \mathsf{c}_3 : \ldots : \mathsf{c}_{k} : \mathsf{c}_{i,2} : \mathsf{c}_{i,4} : \ldots : \mathsf{M}_{i,1} \\ \qquad \ldots \text{ for } k \text{ odd, } \mathsf{M}_i \text{ even} \\ \mathsf{c}_1 : \mathsf{c}_3 : \ldots : \mathsf{c}_{k} : \mathsf{c}_{i,2} : \mathsf{c}_{i,4} : \ldots : \mathsf{M}_{i,2} \\ \qquad \ldots \text{ for } k \text{ odd, } m_i \text{ odd} \end{cases}$$

The terms on the right have data prefixes of length $\leq m$, and are normal forms w.r.t. R. Hence even(t) is again of the form (**??**). □

Finiteness of (even, odd)-derivatives (Proof)

Proof (Continued).

Induction Step (Continued). We find:

$$\underbrace{\mathsf{even}(c_1 : \ldots : c_k : M_i)}_{= \mathsf{even}(t)} \twoheadrightarrow \begin{cases} c_1 : c_3 : \ldots : c_{k-1} : c_{i,1} : c_{i,3} : \ldots : M_{i,1} \\ \quad \ldots \text{ for } k \text{ even, } M_i \text{ even} \\ c_1 : c_3 : \ldots : c_{k-1} : c_{i,1} : c_{i,3} : \ldots : M_{i,2} \\ \quad \ldots \text{ for } k \text{ even, } M_i \text{ odd} \\ c_1 : c_3 : \ldots : c_k : c_{i,2} : c_{i,4} : \ldots : M_{i,1} \\ \quad \ldots \text{ for } k \text{ odd, } M_i \text{ even} \\ c_1 : c_3 : \ldots : c_k : c_{i,2} : c_{i,4} : \ldots : M_{i,2} \\ \quad \ldots \text{ for } k \text{ odd, } m_i \text{ odd} \end{cases}$$

The terms on the right have data prefixes of length $\leq m$, and are normal forms w.r.t. R. Hence $\mathsf{even}(t)$ is again of the form (**??**). □

Finiteness of (even, odd)-derivatives (Proof)

Proof (Continued).

Induction Step (Continued). We find:

$$\underbrace{\mathsf{even}(c_1 : \ldots : c_k : M_i)}_{=\,\mathsf{even}(t)} \twoheadrightarrow \begin{cases} c_1 : c_3 : \ldots : c_{k-1} : c_{i,1} : c_{i,3} : \ldots : M_{i,1} \\ \qquad \ldots \text{ for } k \text{ even, } M_i \text{ even} \\ c_1 : c_3 : \ldots : c_{k-1} : c_{i,1} : c_{i,3} : \ldots : M_{i,2} \\ \qquad \ldots \text{ for } k \text{ even, } M_i \text{ odd} \\ c_1 : c_3 : \ldots : c_k : c_{i,2} : c_{i,4} : \ldots : M_{i,1} \\ \qquad \ldots \text{ for } k \text{ odd, } M_i \text{ even} \\ c_1 : c_3 : \ldots : c_k : c_{i,2} : c_{i,4} : \ldots : M_{i,2} \\ \qquad \ldots \text{ for } k \text{ odd, } m_i \text{ odd} \end{cases}$$

The terms on the right have data prefixes of length $\leq m$, and are normal forms w.r.t. R. Hence $\mathsf{even}(t)$ is again of the form (**??**). □

Equivalence of zip-specs via bisimilarity

Proposition

Two productive, zip-guarded zip-specifications are equivalent if and only if their observation graphs are bisimilar.

Proof (Idea).

$\langle$head, odd, even$\rangle$ is a cobasis of the coalgebra of streams.
That is, 'experiments' built from these operations can be used to observe every element of a stream σ:

- $\sigma(0)$ by head(σ), and head(even^i(σ)) for all i;
- $\sigma(1)$ by head(odd(σ)), and head(even^i(odd(σ))) for all i;
- $\sigma(2)$ by head(odd(even(σ))), and head(even^i(odd(even(σ))));
- $\sigma(3)$ by head(odd(odd(σ))), and head(even^i(odd(odd(σ))));
- …

Carrying out the same 'experiment' at bisimilar observation graphs leads to the same observation. □

Equivalence of zip-specs via bisimilarity

Proposition

Two productive, zip-guarded zip-specifications are equivalent if and only if their observation graphs are bisimilar.

Proof (Idea).

$\langle$head, odd, even$\rangle$ is a cobasis of the coalgebra of streams.
That is, 'experiments' built from these operations can be used to observe every element of a stream σ:

- $\sigma(0)$ by head(σ), and head(even^i(σ)) for all i;
- $\sigma(1)$ by head(odd(σ)), and head(even^i(odd(σ))) for all i;
- $\sigma(2)$ by head(odd(even(σ))), and head(even^i(odd(even(σ))));
- $\sigma(3)$ by head(odd(odd(σ))), and head(even^i(odd(odd(σ))));
- ...

Carrying out the same 'experiment' at bisimilar observation graphs leads to the same observation. □

Equivalence of zip-specs via bisimilarity

Proposition

Two productive, zip-guarded zip-specifications are equivalent if and only if their observation graphs are bisimilar.

Proof (Idea).

$\langle$head, odd, even$\rangle$ is a cobasis of the coalgebra of streams.
That is, 'experiments' built from these operations can be used to observe every element of a stream σ:

- $\sigma(0)$ by head(σ), and head(even^i(σ)) for all i;
- $\sigma(1)$ by head(odd(σ)), and head(even^i(odd(σ))) for all i;
- $\sigma(2)$ by head(odd(even(σ))), and head(even^i(odd(even(σ))));
- $\sigma(3)$ by head(odd(odd(σ))), and head(even^i(odd(odd(σ))));
- ...

Carrying out the same 'experiment' at bisimilar observation graphs leads to the same observation. □

Equivalence of zip-specs via bisimilarity

Proposition

Two productive, zip-guarded zip-specifications are equivalent if and only if their observation graphs are bisimilar.

Proof (Idea).

$\langle$head, odd, even$\rangle$ is a cobasis of the coalgebra of streams.
That is, 'experiments' built from these operations can be used to observe every element of a stream σ:

- $\sigma(0)$ by head(σ), and head(even^i(σ)) for all i;
- $\sigma(1)$ by head(odd(σ)), and head(even^i(odd(σ))) for all i;
- $\sigma(2)$ by head(odd(even(σ))), and head(even^i(odd(even(σ))));
- $\sigma(3)$ by head(odd(odd(σ))), and head(even^i(odd(odd(σ))));
- ...

Carrying out the same 'experiment' at bisimilar observation graphs leads to the same observation. □

Equivalence of zip-specs via bisimilarity

Proposition

Two productive, zip-guarded zip-specifications are equivalent if and only if their observation graphs are bisimilar.

Proof (Idea).

$\langle$head, odd, even$\rangle$ is a cobasis of the coalgebra of streams.
That is, 'experiments' built from these operations can be used to observe every element of a stream σ:

- $\sigma(0)$ by head(σ), and head(even^i(σ)) for all i;
- $\sigma(1)$ by head(odd(σ)), and head(even^i(odd(σ))) for all i;
- $\sigma(2)$ by head(odd(even(σ))), and head(even^i(odd(even(σ))));
- $\sigma(3)$ by head(odd(odd(σ))), and head(even^i(odd(odd(σ))));
- ...

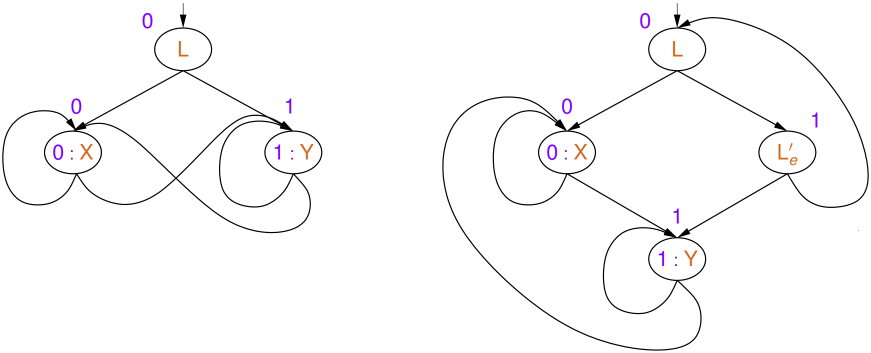
Carrying out the same 'experiment' at bisimilar observation graphs leads to the same observation. □

Bisimilarity of observation graphs (downhill)

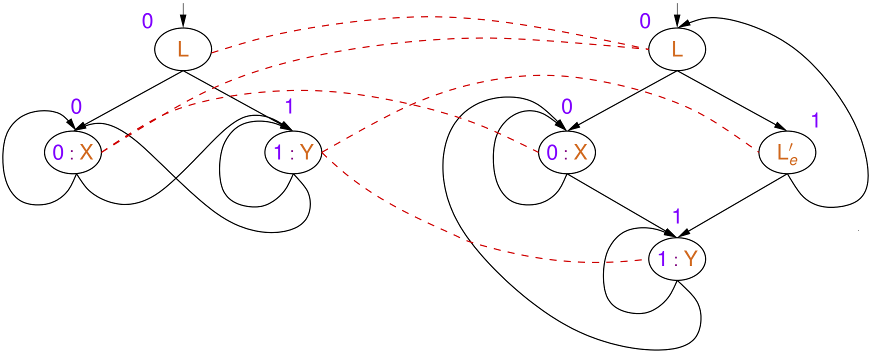
$$\mathsf{L} = 0 : \mathsf{X}$$
$$\mathsf{X} = 1 : \mathsf{zip}(\mathsf{X}, \mathsf{Y})$$
$$\mathsf{Y} = 0 : \mathsf{zip}(\mathsf{Y}, \mathsf{X})$$
$$\mathsf{zip}(x : \sigma, \tau) = x : \mathsf{zip}(\tau, \sigma)$$

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

Bisimilarity of observation graphs (downhill)



Bisimilarity of observation graphs (downhill)



Bisimilarity of observation graphs

Proposition

Bisimilarity of pairs of (even, odd)*-observation graphs* $\mathcal{O}(\mathcal{S})$ *with* $\leq n$ *vertices is decidable in time* $O(n)$.

Proof.

Bisimilarity of finite transition systems with n states and m transitions can be decided in:

- $O(mn + n^2)$ time (Kannellakis–Smolka),
- $O(m \log n)$ time (Tarjan–Paige),

which implies $O(n \log n)$ (with T/P) for obs. graphs with n vertices.

However: For deterministic transition systems with n states, bisimilarity coincides with trace (language) equivalence, which can be decided in time:

- $O(n)$ time (Hopcroft–Karp).

□

Bisimilarity of observation graphs

Proposition

Bisimilarity of pairs of (even, odd)-observation graphs $\mathcal{O}(\mathcal{S})$ with $\leq n$ vertices is decidable in time $O(n)$.

Proof.

Bisimilarity of finite transition systems with n states and m transitions can be decided in:

- $O(mn + n^2)$ time (Kannellakis–Smolka),
- $O(m \log n)$ time (Tarjan–Paige),

which implies $O(n \log n)$ (with T/P) for obs. graphs with n vertices.

However: For deterministic transition systems with n states, bisimilarity coincides with trace (language) equivalence, which can be decided in time:

- $O(n)$ time (Hopcroft–Karp).

□

Bisimilarity of observation graphs

Proposition

Bisimilarity of pairs of (even, odd)-observation graphs $\mathcal{O}(\mathcal{S})$ with $\leq n$ vertices is decidable in time $O(n)$.

Proof.

Bisimilarity of finite transition systems with n states and m transitions can be decided in:

- $O(mn + n^2)$ time (Kannellakis–Smolka),
- $O(m \log n)$ time (Tarjan–Paige),

which implies $O(n \log n)$ (with T/P) for obs. graphs with n vertices.

However: For deterministic transition systems with n states, bisimilarity coincides with trace (language) equivalence, which can be decided in time:

- $O(n)$ time (Hopcroft–Karp).

□

Bisimilarity of observation graphs

Proposition

Bisimilarity of pairs of (even, odd)-observation graphs $\mathcal{O}(\mathcal{S})$ with $\leq n$ vertices is decidable in time $O(n)$.

Proof.

Bisimilarity of finite transition systems with n states and m transitions can be decided in:

- $O(mn + n^2)$ time (Kannellakis–Smolka),
- $O(m \log n)$ time (Tarjan–Paige),

which implies $O(n \log n)$ (with T/P) for obs. graphs with n vertices.

However: For deterministic transition systems with n states, bisimilarity coincides with trace (language) equivalence, which can be decided in time:

- $O(n)$ time (Hopcroft–Karp).

□

Decidability Result

Theorem

Equivalence of zip-specifications is decidable.

Proof (Putting things together).

1. Unique solvability of zip-specs is equivalent to productivity.
2. Productivity of zip-specs is (easily) decidable. Hence it suffices to decide equivalence of productive zip-specs.
3. Every productive zip-spec S can be transformed into a flat zip-spec S' that specifies/computes that same stream.
4. Observation graphs of flat zip-specs are finite.
5. Two productive, flat specifications are equivalent if and only if the associated observation graphs are bisimilar.
6. Given two productive zip-specs S_1 and S_2, their equivalence can be decided by obtaining flat forms S_1' and S_2', and deciding bisimilarity for the observation graphs $\mathcal{O}(S_1')$ and $\mathcal{O}(S_2')$.

Decidability Result

Theorem

Equivalence of zip-specifications is decidable.

Proof (Putting things together).

1. Unique solvability of zip-specs is equivalent to productivity.
2. Productivity of zip-specs is (easily) decidable. Hence it suffices to decide equivalence of productive zip-specs.
3. Every productive zip-spec $\mathcal{S}$ can be transformed into a flat zip-spec $\mathcal{S}'$ that specifies/computes that same stream.
4. Observation graphs of flat zip-specs are finite.
5. Two productive, flat specifications are equivalent if and only if the associated observation graphs are bisimilar.
6. Given two productive zip-specs $\mathcal{S}_1$ and $\mathcal{S}_2$, their equivalence can be decided by obtaining flat forms $\mathcal{S}_1'$ and $\mathcal{S}_2'$, and deciding bisimilarity for the observation graphs $\mathcal{O}(\mathcal{S}_1')$ and $\mathcal{O}(\mathcal{S}_2')$.

Decidability Result

Theorem

Equivalence of zip-specifications is decidable.

Proof (Putting things together).

1. Unique solvability of zip-specs is equivalent to productivity.
2. Productivity of zip-specs is (easily) decidable. Hence it suffices to decide equivalence of productive zip-specs.
3. Every productive zip-spec $\mathcal{S}$ can be transformed into a flat zip-spec $\mathcal{S}'$ that specifies/computes that same stream.
4. Observation graphs of flat zip-specs are finite.
5. Two productive, flat specifications are equivalent if and only if the associated observation graphs are bisimilar.
6. Given two productive zip-specs $\mathcal{S}_1$ and $\mathcal{S}_2$, their equivalence can be decided by obtaining flat forms $\mathcal{S}_1'$ and $\mathcal{S}_2'$, and deciding bisimilarity for the observation graphs $\mathcal{O}(\mathcal{S}_1')$ and $\mathcal{O}(\mathcal{S}_2')$.

Decidability Result

Theorem

Equivalence of zip-specifications is decidable.

Proof (Putting things together).

1. Unique solvability of zip-specs is equivalent to productivity.
2. Productivity of zip-specs is (easily) decidable. Hence it suffices to decide equivalence of productive zip-specs.
3. Every productive zip-spec $\mathcal{S}$ can be transformed into a flat zip-spec $\mathcal{S}'$ that specifies/computes that same stream.
4. Observation graphs of flat zip-specs are finite.
5. Two productive, flat specifications are equivalent if and only if the associated observation graphs are bisimilar.
6. Given two productive zip-specs $\mathcal{S}_1$ and $\mathcal{S}_2$, their equivalence can be decided by obtaining flat forms $\mathcal{S}_1'$ and $\mathcal{S}_2'$, and deciding bisimilarity for the observation graphs $\mathcal{O}(\mathcal{S}_1')$ and $\mathcal{O}(\mathcal{S}_2')$.

Decidability Result

Theorem

Equivalence of zip-specifications is decidable.

Proof (Putting things together).

1. Unique solvability of zip-specs is equivalent to productivity.
2. Productivity of zip-specs is (easily) decidable. Hence it suffices to decide equivalence of productive zip-specs.
3. Every productive zip-spec $\mathcal{S}$ can be transformed into a flat zip-spec $\mathcal{S}'$ that specifies/computes that same stream.
4. Observation graphs of flat zip-specs are finite.
5. Two productive, flat specifications are equivalent if and only if the associated observation graphs are bisimilar.
6. Given two productive zip-specs $\mathcal{S}_1$ and $\mathcal{S}_2$, their equivalence can be decided by obtaining flat forms $\mathcal{S}'_1$ and $\mathcal{S}'_2$, and deciding bisimilarity for the observation graphs $\mathcal{O}(\mathcal{S}'_1)$ and $\mathcal{O}(\mathcal{S}'_2)$.

Decidability Result

Theorem

Equivalence of zip-specifications is decidable.

Proof (Putting things together).

1. Unique solvability of zip-specs is equivalent to productivity.
2. Productivity of zip-specs is (easily) decidable. Hence it suffices to decide equivalence of productive zip-specs.
3. Every productive zip-spec $\mathcal{S}$ can be transformed into a flat zip-spec $\mathcal{S}'$ that specifies/computes that same stream.
4. Observation graphs of flat zip-specs are finite.
5. Two productive, flat specifications are equivalent if and only if the associated observation graphs are bisimilar.
6. Given two productive zip-specs $\mathcal{S}_1$ and $\mathcal{S}_2$, their equivalence can be decided by obtaining flat forms $\mathcal{S}_1'$ and $\mathcal{S}_2'$, and deciding bisimilarity for the observation graphs $\mathcal{O}(\mathcal{S}_1')$ and $\mathcal{O}(\mathcal{S}_2')$.

Decidability Result

Theorem

Equivalence of zip-specifications is decidable.

Proof (Putting things together).

1. Unique solvability of zip-specs is equivalent to productivity.
2. Productivity of zip-specs is (easily) decidable. Hence it suffices to decide equivalence of productive zip-specs.
3. Every productive zip-spec $\mathcal{S}$ can be transformed into a flat zip-spec $\mathcal{S}'$ that specifies/computes that same stream.
4. Observation graphs of flat zip-specs are finite.
5. Two productive, flat specifications are equivalent if and only if the associated observation graphs are bisimilar.
6. Given two productive zip-specs $\mathcal{S}_1$ and $\mathcal{S}_2$, their equivalence can be decided by obtaining flat forms $\mathcal{S}_1'$ and $\mathcal{S}_2'$, and deciding bisimilarity for the observation graphs $\mathcal{O}(\mathcal{S}_1')$ and $\mathcal{O}(\mathcal{S}_2')$.

PTIME-decidability result

Remember:

Main Lemma

Let $\mathcal{S}$ be a flat zip-specification.
The set $\partial_{\mathcal{S}}(\mathsf{M}_0)$ of $(\mathsf{even}, \mathsf{odd})$-derivatives of the root M_0 of $\mathcal{S}$ is finite.

It can be strengthened:

Main Lemma Plus

Let $\mathcal{S}$ be a flat zip-specification with n recursion variables, c stream constants, and m the longest stream prefix in $\mathcal{S}$. Then it holds:

$$|\partial_{\mathcal{S}}(\mathsf{M}_0)| \leq 2 \cdot (c + 1) \cdot m \cdot n + 4 \cdot m$$

Theorem

Equivalence of zip-specifications is decidable in PTIME.

PTIME-decidability result

Remember:

Main Lemma

Let $\mathcal{S}$ be a flat zip-specification.
The set $\partial_{\mathcal{S}}(\mathsf{M}_0)$ of $(\mathsf{even}, \mathsf{odd})$-derivatives of the root M_0 of $\mathcal{S}$ is finite.

It can be strengthened:

Main Lemma Plus

Let $\mathcal{S}$ be a flat zip-specification with n recursion variables, c stream constants, and m the longest stream prefix in $\mathcal{S}$. Then it holds:

$$|\partial_{\mathcal{S}}(\mathsf{M}_0)| \leq 2 \cdot (c+1) \cdot m \cdot n + 4 \cdot m$$

Theorem

Equivalence of zip-specifications is decidable in PTIME.

PTIME-decidability result

Remember:

Main Lemma

Let $\mathcal{S}$ be a flat zip-specification.
The set $\partial_{\mathcal{S}}(\mathsf{M}_0)$ of $(\mathsf{even}, \mathsf{odd})$-derivatives of the root M_0 of $\mathcal{S}$ is finite.

It can be strengthened:

Main Lemma Plus

Let $\mathcal{S}$ be a flat zip-specification with n recursion variables, c stream constants, and m the longest stream prefix in $\mathcal{S}$. Then it holds:

$$|\partial_{\mathcal{S}}(\mathsf{M}_0)| \leq 2 \cdot (c+1) \cdot m \cdot n + 4 \cdot m$$

Theorem

Equivalence of zip-specifications is decidable in PTIME.

Extensions of the decidability result

- zip-inv-tail-specifications:

$$\mathsf{inv}(0 : \sigma) \to 1 : \mathsf{inv}(\sigma) \qquad \mathsf{tail}(x : \sigma) \to \sigma$$
$$\mathsf{inv}(1 : \sigma) \to 0 : \mathsf{inv}(\sigma) \qquad \mathsf{zip}(x : \sigma, \tau) \to x : \mathsf{zip}(\tau, \sigma)$$

- zip_n-specs for $n \in \mathbb{N}$, $n > 2$, where zip_n is defined by:

$$\mathsf{zip}_n(x : \sigma_1', \sigma_2, \ldots, \sigma_n) \to x : \mathsf{zip}_n(\sigma_2, \ldots, \sigma_n, \sigma_1')$$

- zip_n-specs versus zip_m-specs for $m, n \geq 2$, $m \neq n$.
- zip_n-specs versus zip_i-mix-specs (all of zip_i, $i \geq 2$, may be used).

Extensions of the decidability result

- zip-inv-tail-specifications:

$$\mathsf{inv}(0 : \sigma) \to 1 : \mathsf{inv}(\sigma) \qquad \mathsf{tail}(x : \sigma) \to \sigma$$
$$\mathsf{inv}(1 : \sigma) \to 0 : \mathsf{inv}(\sigma) \qquad \mathsf{zip}(x : \sigma, \tau) \to x : \mathsf{zip}(\tau, \sigma)$$

- zip_n-specs for $n \in \mathbb{N}$, $n > 2$, where zip_n is defined by:

$$\mathsf{zip}_n(x : \sigma'_1, \sigma_2, \ldots, \sigma_n) \to x : \mathsf{zip}_n(\sigma_2, \ldots, \sigma_n, \sigma'_1)$$

- zip_n-specs versus zip_m-specs for $m, n \geq 2$, $m \neq n$.
- zip_n-specs versus zip_i-mix-specs (all of zip_i, $i \geq 2$, may be used).

Extensions of the decidability result

- zip-inv-tail-specifications:

$$\mathsf{inv}(0 : \sigma) \to 1 : \mathsf{inv}(\sigma) \qquad \mathsf{tail}(x : \sigma) \to \sigma$$
$$\mathsf{inv}(1 : \sigma) \to 0 : \mathsf{inv}(\sigma) \qquad \mathsf{zip}(x : \sigma, \tau) \to x : \mathsf{zip}(\tau, \sigma)$$

- zip_n-specs for $n \in \mathbb{N}$, $n > 2$, where zip_n is defined by:

$$\mathsf{zip}_n(x : \sigma_1', \sigma_2, \ldots, \sigma_n) \to x : \mathsf{zip}_n(\sigma_2, \ldots, \sigma_n, \sigma_1')$$

- zip_n-specs versus zip_m-specs for $m, n \geq 2$, $m \neq n$.
- zip_n-specs versus zip_i-mix-specs (all of zip_i, $i \geq 2$, may be used).

Extensions of the decidability result

- zip-inv-tail-specifications:

$$\mathsf{inv}(0 : \sigma) \to 1 : \mathsf{inv}(\sigma) \qquad \mathsf{tail}(x : \sigma) \to \sigma$$
$$\mathsf{inv}(1 : \sigma) \to 0 : \mathsf{inv}(\sigma) \qquad \mathsf{zip}(x : \sigma, \tau) \to x : \mathsf{zip}(\tau, \sigma)$$

- zip_n-specs for $n \in \mathbb{N}$, $n > 2$, where zip_n is defined by:

$$\mathsf{zip}_n(x : \sigma_1', \sigma_2, \ldots, \sigma_n) \to x : \mathsf{zip}_n(\sigma_2, \ldots, \sigma_n, \sigma_1')$$

- zip_n-specs versus zip_m-specs for $m, n \geq 2$, $m \neq n$.
- zip_n-specs versus zip_i-mix-specs (all of zip_i, $i \geq 2$, may be used).

Overview

1. ROS
2. Stream Equality
3. Summary

Summary

- Ad for ISR'2010 in Utrecht.
- ROS: Realising Optimal Sharing (NWO-project)
- Equivalence of stream specifications
  - stream specifications
  - equivalence of stream specifications versus productivity and unique solvability
  - zip-specifications, Larry Moss' question
  - solution: decidability of equivalence for zip-specs
    - zip-guarded, and flat specs
    - observation graphs of zip-guarded specs
    - reducing equivalence to checking bisimilarity of obs. graphs
  - extensions of the result