Complexity of Fractran and Productivity

Jörg Endrullis Clemens Grabmayer Dimitri Hendriks

Vrije Universiteit Amsterdam – Universiteit Utrecht – Vrije Universiteit Amsterdam
The Netherlands

CADE 2009, McGill University, Montreal

6 Aug 2009

Overview

- Introduction
  - stream specifications:
    productivity,
    'pure' and 'lazy' stream specification formats (PSF, LSF),
    decidability of productivity for PSF: productivity prover *ProPro*
  - Fractran
  - the arithmetical and analytical hierarchies
- Complexity of Fractran
- Complexity of LSF-specifications
- Complexity of productivity (in TRSs), and of variant definitions
- Summary

Overview

1. Introduction
2. Complexity of Fractran
3. Complexity of productivity for LSF-spec's
4. Complexity of productivity (in TRS's), and of variant definitions
5. Summary

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 (Specification of the Thue–Morse stream)

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

one finds: T

Specifying streams

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

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

Example (Specification of the Thue–Morse stream)

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

one finds: $\underline{\mathsf{T}}$

Specifying streams

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

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

Example (Specification of the Thue–Morse stream)

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

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

Specifying streams

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

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

Example (Specification of the Thue–Morse stream)

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

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

Specifying streams

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

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

Example (Specification of the Thue–Morse stream)

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

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

Specifying streams

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

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

Example (Specification of the Thue–Morse stream)

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

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

Specifying streams

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

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

Example (Specification of the Thue–Morse stream)

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

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

Specifying streams

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

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

Example (Specification of the Thue–Morse stream)

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

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

Specifying streams

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

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

Example (Specification of the Thue–Morse stream)

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

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

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 (Data-abstraction of a productive stream specification)

$\mathsf{T} \to \bullet : \bullet : \mathsf{F}(\mathsf{tail}(\mathsf{T}))$	*stream constant*
$\mathsf{F}(x : \sigma) \to x : \mathsf{inv}(x) : \mathsf{F}(\sigma)$ $\mathsf{tail}(x : \sigma) \to \sigma$	*stream functions*
$\mathsf{inv}(\bullet) \to \bullet$	*data functions*

one finds: $\mathsf{T} \twoheadrightarrow \bullet : \bullet : \bullet : \bullet : \bullet : \bullet : \bullet : \bullet : \bullet : \bullet : \bullet : \bullet : \bullet : \bullet : \bullet : \bullet : \ldots$

Pure Stream Format (PSF) [FCT'07, LPAR'08]

Pure stream specifications are of the form:

$$\mathsf{M} \to C[\mathsf{M}]$$

where C is a context consisting of pure stream functions such as:

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

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

with the properties (FCT'07, LPAR'08):

- no nesting of stream functions on the right-hand side (flatness)
- every stream function is defined by a rule scheme (pureness)

Excluded: stream-dependent data functions like: $\mathsf{head}(x : \sigma) \to x$.

Productivity of stream spec's. Decidability for PSF.

Definition

A stream specification $\mathcal{S} = \{M \to C[M], \ldots\}$ is productive for M if outermost-fair evaluation of M w.r.t. $\mathcal{S}$ results in an infinite constructor normal form:

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

A not productive spec: $J \to 0 : 1 : \mathrm{odd}(J)$. Production stops:

$$J \twoheadrightarrow 0 : 1 : 0 : 0 : \mathrm{odd}(\mathrm{odd}(\ldots))$$

Theorem (FCT'07, LPAR'08)

For PSF-specifications, productivity is decidable.

Productivity Prover *ProPro*

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

Productivity of stream spec's. Decidability for PSF.

Definition

A stream specification $\mathcal{S} = \{\mathsf{M} \to C[\mathsf{M}], \ldots\}$ is productive for M if outermost-fair evaluation of M w.r.t. $\mathcal{S}$ results in an infinite constructor normal form:

$$\mathsf{M} \twoheadrightarrow a_0 : a_1 : a_2 : \ldots\ .$$

A not productive spec: $\mathsf{J} \to 0 : 1 : \mathsf{odd}(\mathsf{J})$. Production stops:

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

Theorem (FCT'07, LPAR'08)

For PSF-specifications, productivity is decidable.

Productivity Prover *ProPro*

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

Productivity of stream spec's. Decidability for PSF.

Definition

A stream specification $\mathcal{S} = \{\mathsf{M} \to C[\mathsf{M}], \ldots\}$ is productive for M if outermost-fair evaluation of M w.r.t. $\mathcal{S}$ results in an infinite constructor normal form:

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

A not productive spec: $\mathsf{J} \to 0 : 1 : \mathsf{odd}(\mathsf{J})$. Production stops:

$$\mathsf{J} \twoheadrightarrow_{\omega} 0 : 1 : 0 : 0 : \mathsf{odd}(\mathsf{odd}(\ldots))$$

Theorem (FCT'07, LPAR'08)

For PSF-specifications, productivity is decidable.

Productivity Prover *ProPro*

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

Lazy Stream Format (LSF)

Lazy stream specifications are of the form:

$$\mathsf{M} \to C[\mathsf{M}]$$

where C is a finite context built up from:

- a single data-element symbol $\bullet$
- the stream constructor ':'
- stream function symbols head, tail, mod_n and zip_n ($n \geq 1$)

together with the defining rules for the occurring stream functions:

$$\begin{aligned}
\mathsf{head}(x : \sigma) &\to x \\
\mathsf{tail}(x : \sigma) &\to \sigma \\
\mathsf{mod}_n(\sigma) &\to \mathsf{head}(\sigma) : \mathsf{mod}_n(\mathsf{tail}^n(\sigma)) \\
\mathsf{zip}_n(\sigma_1, \sigma_2 \ldots, \sigma_n) &\to \mathsf{head}(\sigma_1) : \mathsf{zip}_n(\sigma_2, \ldots, \sigma_n, \mathsf{tail}(\sigma_1))
\end{aligned}$$

LSF, example: Collatz

$$\mathsf{C} \to \bullet : \mathsf{zip}_2(\mathsf{C}, \mathsf{mod}_3(\mathsf{tail}^4(\mathsf{C})))$$

where (by the rules of the prev. slide)

$$\begin{aligned}
\mathsf{zip}_2(\sigma,\tau) &\twoheadrightarrow \sigma(1) : \tau(1) : \sigma(2) : \tau(2) : \sigma(3) : \ldots \\
\mathsf{mod}_3(\sigma) &\twoheadrightarrow \sigma(1) : \sigma(4) : \sigma(7) : \ldots \\
\mathsf{tail}^4(\sigma) &\twoheadrightarrow \sigma(5) : \sigma(6) : \sigma(7) : \ldots \\
\mathsf{mod}_3(\mathsf{tail}^4(\mathsf{C})) &\twoheadrightarrow \mathsf{C}(5) : \mathsf{C}(8) : \mathsf{C}(11) : \ldots \\
\mathsf{C} &\twoheadrightarrow \bullet : \mathsf{C}(1) : \mathsf{C}(5) : \mathsf{C}(2) : \mathsf{C}(8) : \mathsf{C}(3) : \mathsf{C}(11) : \ldots \\
&\twoheadrightarrow \ldots
\end{aligned}$$

where we write $\sigma(n)$ for $\mathsf{head}(\mathsf{tail}^n(\sigma))$.

Fractran

Fractran is a theoretical programming language using arithmetic (1986) by John Horton Conway.

A Fractran program P is an ordered list of fractions

$$P = \frac{p_1}{q_1}, \frac{p_2}{q_2}, \ldots, \frac{p_k}{q_k}$$

Given a natural number $N \geq 1$, P repeats, until termination, the step:

St:
- Suppose that $\frac{p_i}{q_i}$ be the first fraction from left in P such that $N \cdot \frac{p_i}{q_i} \in \mathbb{N}$. Then set $N := N \cdot \frac{p_i}{q_i}$.
- If no such fraction exists, terminate.

Idea: view the primes occurring in P as storage registers $r_2, r_3, r_5, \ldots$. If the current working number is

$$N = 2^a \, 3^b \, 5^c \, \ldots$$

then $r_2 = a$, $r_3 = b$, $r_5 = c$, ...

Fractran, example 1

$$P = \frac{2}{3}$$

can be used to add the contents of register r_3 to register r_2.

Starting with

$$N = 2^n \, 3^m \ ,$$

in each step r_3 is decremented while r_2 is incremented.

Execution terminates when

$$N = 2^{n+m} \, 3^0 \ .$$

Fractran, example 2

$$P = \frac{5 \times 11}{2 \times 7}, \frac{7}{11}, \frac{13}{7}, \frac{2 \times 17}{3 \times 13}, \frac{13}{17}, \frac{1}{13}, \frac{3}{5}$$

P swaps the contents of registers r_2 and r_3: $\quad 2^n\, 3^m\, 7 \twoheadrightarrow_P 2^m\, 3^n$

$$\begin{aligned}
2^2\,3^1\,7^1 \;=\;& 2^2\,3^1\,5^0\,7^1\,11^0\,13^0\,17^0 \to_P 2^1\,3^1\,5^1\,7^0\,11^1\,13^0\,17^0 \\
\to_P\;& 2^1\,3^1\,5^1\,7^1\,11^0\,13^0\,17^0 \to_P 2^0\,3^1\,5^2\,7^0\,11^1\,13^0\,17^0 \\
\to_P\;& 2^0\,3^1\,5^2\,7^1\,11^0\,13^0\,17^0 \to_P 2^0\,3^1\,5^2\,7^0\,11^0\,13^1\,17^0 \\
\to_P\;& 2^1\,3^0\,5^2\,7^0\,11^0\,13^0\,17^1 \to_P 2^1\,3^0\,5^2\,7^0\,11^0\,13^1\,17^0 \\
\to_P\;& 2^1\,3^0\,5^2\,7^0\,11^0\,13^0\,17^0 \to_P 2^1\,3^1\,5^1\,7^0\,11^0\,13^0\,17^0 \\
\to_P\;& 2^1\,3^2\,5^0\,7^0\,11^0\,13^0\,17^0 \;=\; 2^1\,3^2
\end{aligned}$$

Fractran, example 2

$$P = \frac{5 \times 11}{2 \times 7}, \frac{7}{11}, \frac{13}{7}, \frac{2 \times 17}{3 \times 13}, \frac{13}{17}, \frac{1}{13}, \frac{3}{5}$$

P swaps the contents of registers r_2 and r_3: $2^n\,3^m\,7 \twoheadrightarrow_P 2^m\,3^n$

$$\begin{aligned}
2^2\,3^1\,7^1 \;=\; & 2^2\,3^1\,5^0\,7^1\,11^0\,13^0\,17^0 \to_P 2^1\,3^1\,5^1\,7^0\,11^1\,13^0\,17^0 \\
& \to_P 2^1\,3^1\,5^1\,7^1\,11^0\,13^0\,17^0 \to_P 2^0\,3^1\,5^2\,7^0\,11^1\,13^0\,17^0 \\
& \to_P 2^0\,3^1\,5^2\,7^1\,11^0\,13^0\,17^0 \to_P 2^0\,3^1\,5^2\,7^0\,11^0\,13^1\,17^0 \\
& \to_P 2^1\,3^0\,5^2\,7^0\,11^0\,13^0\,17^1 \to_P 2^1\,3^0\,5^2\,7^0\,11^0\,13^1\,17^0 \\
& \to_P 2^1\,3^0\,5^2\,7^0\,11^0\,13^0\,17^0 \to_P 2^1\,3^1\,5^1\,7^0\,11^0\,13^0\,17^0 \\
& \to_P 2^1\,3^2\,5^0\,7^0\,11^0\,13^0\,17^0 \;=\; 2^1\,3^2
\end{aligned}$$

Fractran, example 2

$$P = \frac{5 \times 11}{2 \times 7}, \frac{7}{11}, \frac{13}{7}, \frac{2 \times 17}{3 \times 13}, \frac{13}{17}, \frac{1}{13}, \frac{3}{5}$$

P swaps the contents of registers r_2 and r_3: $2^n\, 3^m\, 7 \twoheadrightarrow_P 2^m\, 3^n$

$$2^2\, 3^1\, 7^1 = 2^2\, 3^1\, 5^0\, 7^1\, 11^0\, 13^0\, 17^0 \to_P 2^1\, 3^1\, 5^1\, 7^0\, 11^1\, 13^0\, 17^0$$
$$\to_P 2^1\, 3^1\, 5^1\, 7^1\, 11^0\, 13^0\, 17^0 \to_P 2^0\, 3^1\, 5^2\, 7^0\, 11^1\, 13^0\, 17^0$$
$$\to_P 2^0\, 3^1\, 5^2\, 7^1\, 11^0\, 13^0\, 17^0 \to_P 2^0\, 3^1\, 5^2\, 7^0\, 11^0\, 13^1\, 17^0$$
$$\to_P 2^1\, 3^0\, 5^2\, 7^0\, 11^0\, 13^0\, 17^1 \to_P 2^1\, 3^0\, 5^2\, 7^0\, 11^0\, 13^1\, 17^0$$
$$\to_P 2^1\, 3^0\, 5^2\, 7^0\, 11^0\, 13^0\, 17^0 \to_P 2^1\, 3^1\, 5^1\, 7^0\, 11^0\, 13^0\, 17^0$$
$$\to_P 2^1\, 3^2\, 5^0\, 7^0\, 11^0\, 13^0\, 17^0 = 2^1\, 3^2$$

Fractran, example 2

$$P = \frac{5 \times 11}{2 \times 7}, \frac{7}{11}, \frac{13}{7}, \frac{2 \times 17}{3 \times 13}, \frac{13}{17}, \frac{1}{13}, \frac{3}{5}$$

P swaps the contents of registers r_2 and r_3: $2^n\,3^m\,7 \twoheadrightarrow_P 2^m\,3^n$

$$\begin{aligned}
2^2\,3^1\,7^1 \;=\; & 2^2\,3^1\,5^0\,7^1\,11^0\,13^0\,17^0 \to_P 2^1\,3^1\,5^1\,7^0\,11^1\,13^0\,17^0 \\
\to_P\; & 2^1\,3^1\,5^1\,7^1\,11^0\,13^0\,17^0 \to_P 2^0\,3^1\,5^2\,7^0\,11^1\,13^0\,17^0 \\
\to_P\; & 2^0\,3^1\,5^2\,7^1\,11^0\,13^0\,17^0 \to_P 2^0\,3^1\,5^2\,7^0\,11^0\,13^1\,17^0 \\
\to_P\; & 2^1\,3^0\,5^2\,7^0\,11^0\,13^0\,17^1 \to_P 2^1\,3^0\,5^2\,7^0\,11^0\,13^1\,17^0 \\
\to_P\; & 2^1\,3^0\,5^2\,7^0\,11^0\,13^0\,17^0 \to_P 2^1\,3^1\,5^1\,7^0\,11^0\,13^0\,17^0 \\
\to_P\; & 2^1\,3^2\,5^0\,7^0\,11^0\,13^0\,17^0 \;=\; 2^1\,3^2
\end{aligned}$$

Fractran, example 2

$$P = \frac{5 \times 11}{2 \times 7}, \frac{7}{11}, \frac{13}{7}, \frac{2 \times 17}{3 \times 13}, \frac{13}{17}, \frac{1}{13}, \frac{3}{5}$$

P swaps the contents of registers r_2 and r_3: $2^n\,3^m\,7 \twoheadrightarrow_P 2^m\,3^n$

$$\begin{aligned}
2^2\,3^1\,7^1 \;=\; & 2^2\,3^1\,5^0\,7^1\,11^0\,13^0\,17^0 \to_P 2^1\,3^1\,5^1\,7^0\,11^1\,13^0\,17^0 \\
\to_P\; & 2^1\,3^1\,5^1\,7^1\,11^0\,13^0\,17^0 \to_P 2^0\,3^1\,5^2\,7^0\,11^1\,13^0\,17^0 \\
\to_P\; & 2^0\,3^1\,5^2\,7^1\,11^0\,13^0\,17^0 \to_P 2^0\,3^1\,5^2\,7^0\,11^0\,13^1\,17^0 \\
\to_P\; & 2^1\,3^0\,5^2\,7^0\,11^0\,13^0\,17^1 \to_P 2^1\,3^0\,5^2\,7^0\,11^0\,13^1\,17^0 \\
\to_P\; & 2^1\,3^0\,5^2\,7^0\,11^0\,13^0\,17^0 \to_P 2^1\,3^1\,5^1\,7^0\,11^0\,13^0\,17^0 \\
\to_P\; & 2^1\,3^2\,5^0\,7^0\,11^0\,13^0\,17^0 \;=\; 2^1\,3^2
\end{aligned}$$

Fractran, example 2

$$P = \frac{5 \times 11}{2 \times 7}, \frac{7}{11}, \frac{13}{7}, \frac{2 \times 17}{3 \times 13}, \frac{13}{17}, \frac{1}{13}, \frac{3}{5}$$

P swaps the contents of registers r_2 and r_3: $2^n\,3^m\,7 \twoheadrightarrow_P 2^m\,3^n$

$$\begin{aligned}
2^2\,3^1\,7^1 \;=\; & 2^2\,3^1\,5^0\,7^1\,11^0\,13^0\,17^0 \to_P 2^1\,3^1\,5^1\,7^0\,11^1\,13^0\,17^0 \\
\to_P\; & 2^1\,3^1\,5^1\,7^1\,11^0\,13^0\,17^0 \to_P 2^0\,3^1\,5^2\,7^0\,11^1\,13^0\,17^0 \\
\to_P\; & 2^0\,3^1\,5^2\,7^1\,11^0\,13^0\,17^0 \to_P 2^0\,3^1\,5^2\,7^0\,11^0\,13^1\,17^0 \\
\to_P\; & 2^1\,3^0\,5^2\,7^0\,11^0\,13^0\,17^1 \to_P 2^1\,3^0\,5^2\,7^0\,11^0\,13^1\,17^0 \\
\to_P\; & 2^1\,3^0\,5^2\,7^0\,11^0\,13^0\,17^0 \to_P 2^1\,3^1\,5^1\,7^0\,11^0\,13^0\,17^0 \\
\to_P\; & 2^1\,3^2\,5^0\,7^0\,11^0\,13^0\,17^0 \;=\; 2^1\,3^2
\end{aligned}$$

Fractran, example 2

$$P = \frac{5 \times 11}{2 \times 7}, \frac{7}{11}, \frac{13}{7}, \frac{2 \times 17}{3 \times 13}, \frac{13}{17}, \frac{1}{13}, \frac{3}{5}$$

P swaps the contents of registers r_2 and r_3: $2^n 3^m 7 \twoheadrightarrow_P 2^m 3^n$

$$\begin{aligned}
2^2 3^1 7^1 \; = \; & 2^2 3^1 5^0 7^1 11^0 13^0 17^0 \to_P 2^1 3^1 5^1 7^0 11^1 13^0 17^0 \\
\to_P \; & 2^1 3^1 5^1 7^1 11^0 13^0 17^0 \to_P 2^0 3^1 5^2 7^0 11^1 13^0 17^0 \\
\to_P \; & 2^0 3^1 5^2 7^1 11^0 13^0 17^0 \to_P 2^0 3^1 5^2 7^0 11^0 13^1 17^0 \\
\to_P \; & 2^1 3^0 5^2 7^0 11^0 13^0 17^1 \to_P 2^1 3^0 5^2 7^0 11^0 13^1 17^0 \\
\to_P \; & 2^1 3^0 5^2 7^0 11^0 13^0 17^0 \to_P 2^1 3^1 5^1 7^0 11^0 13^0 17^0 \\
\to_P \; & 2^1 3^2 5^0 7^0 11^0 13^0 17^0 \; = \; 2^1 3^2
\end{aligned}$$

Fractran, example 2

$$P = \frac{5 \times 11}{2 \times 7}, \frac{7}{11}, \frac{13}{7}, \frac{2 \times 17}{3 \times 13}, \frac{13}{17}, \frac{1}{13}, \frac{3}{5}$$

P swaps the contents of registers r_2 and r_3: $2^n\,3^m\,7 \twoheadrightarrow_P 2^m\,3^n$

$$\begin{aligned}
2^2\,3^1\,7^1 \;=\; & 2^2\,3^1\,5^0\,7^1\,11^0\,13^0\,17^0 \to_P 2^1\,3^1\,5^1\,7^0\,11^1\,13^0\,17^0 \\
\to_P\; & 2^1\,3^1\,5^1\,7^1\,11^0\,13^0\,17^0 \to_P 2^0\,3^1\,5^2\,7^0\,11^1\,13^0\,17^0 \\
\to_P\; & 2^0\,3^1\,5^2\,7^1\,11^0\,13^0\,17^0 \to_P 2^0\,3^1\,5^2\,7^0\,11^0\,13^1\,17^0 \\
\to_P\; & 2^1\,3^0\,5^2\,7^0\,11^0\,13^0\,17^1 \to_P 2^1\,3^0\,5^2\,7^0\,11^0\,13^1\,17^0 \\
\to_P\; & 2^1\,3^0\,5^2\,7^0\,11^0\,13^0\,17^0 \to_P 2^1\,3^1\,5^1\,7^0\,11^0\,13^0\,17^0 \\
\to_P\; & 2^1\,3^2\,5^0\,7^0\,11^0\,13^0\,17^0 \;=\; 2^1\,3^2
\end{aligned}$$

Fractran, example 2

$$P = \frac{5 \times 11}{2 \times 7}, \frac{7}{11}, \frac{13}{7}, \frac{2 \times 17}{3 \times 13}, \frac{13}{17}, \frac{1}{13}, \frac{3}{5}$$

P swaps the contents of registers r_2 and r_3: $2^n\,3^m\,7 \twoheadrightarrow_P 2^m\,3^n$

$$\begin{aligned}
2^2\,3^1\,7^1 \;=\; & 2^2\,3^1\,5^0\,7^1\,11^0\,13^0\,17^0 \to_P 2^1\,3^1\,5^1\,7^0\,11^1\,13^0\,17^0 \\
\to_P\; & 2^1\,3^1\,5^1\,7^1\,11^0\,13^0\,17^0 \to_P 2^0\,3^1\,5^2\,7^0\,11^1\,13^0\,17^0 \\
\to_P\; & 2^0\,3^1\,5^2\,7^1\,11^0\,13^0\,17^0 \to_P 2^0\,3^1\,5^2\,7^0\,11^0\,13^1\,17^0 \\
\to_P\; & 2^1\,3^0\,5^2\,7^0\,11^0\,13^0\,17^1 \to_P 2^1\,3^0\,5^2\,7^0\,11^0\,13^1\,17^0 \\
\to_P\; & 2^1\,3^0\,5^2\,7^0\,11^0\,13^0\,17^0 \to_P 2^1\,3^1\,5^1\,7^0\,11^0\,13^0\,17^0 \\
\to_P\; & 2^1\,3^2\,5^0\,7^0\,11^0\,13^0\,17^0 \;=\; 2^1\,3^2
\end{aligned}$$

Fractran, example 2

$$P = \frac{5 \times 11}{2 \times 7}, \frac{7}{11}, \frac{13}{7}, \frac{2 \times 17}{3 \times 13}, \frac{13}{17}, \frac{1}{13}, \frac{3}{5}$$

P swaps the contents of registers r_2 and r_3: $2^n\,3^m\,7 \twoheadrightarrow_P 2^m\,3^n$

$$\begin{aligned}
2^2\,3^1\,7^1 \;=\; & 2^2\,3^1\,5^0\,7^1\,11^0\,13^0\,17^0 \to_P 2^1\,3^1\,5^1\,7^0\,11^1\,13^0\,17^0 \\
\to_P\; & 2^1\,3^1\,5^1\,7^1\,11^0\,13^0\,17^0 \to_P 2^0\,3^1\,5^2\,7^0\,11^1\,13^0\,17^0 \\
\to_P\; & 2^0\,3^1\,5^2\,7^1\,11^0\,13^0\,17^0 \to_P 2^0\,3^1\,5^2\,7^0\,11^0\,13^1\,17^0 \\
\to_P\; & 2^1\,3^0\,5^2\,7^0\,11^0\,13^0\,17^1 \to_P 2^1\,3^0\,5^2\,7^0\,11^0\,13^1\,17^0 \\
\to_P\; & 2^1\,3^0\,5^2\,7^0\,11^0\,13^0\,17^0 \to_P 2^1\,3^1\,5^1\,7^0\,11^0\,13^0\,17^0 \\
\to_P\; & 2^1\,3^2\,5^0\,7^0\,11^0\,13^0\,17^0 \;=\; 2^1\,3^2
\end{aligned}$$

Fractran, example 2

$$P = \frac{5 \times 11}{2 \times 7}, \frac{7}{11}, \frac{13}{7}, \frac{2 \times 17}{3 \times 13}, \frac{13}{17}, \frac{1}{13}, \frac{3}{5}$$

P swaps the contents of registers r_2 and r_3: $2^n\,3^m\,7 \twoheadrightarrow_P 2^m\,3^n$

$$2^2\,3^1\,7^1 = 2^2\,3^1\,5^0\,7^1\,11^0\,13^0\,17^0 \to_P 2^1\,3^1\,5^1\,7^0\,11^1\,13^0\,17^0$$
$$\to_P 2^1\,3^1\,5^1\,7^1\,11^0\,13^0\,17^0 \to_P 2^0\,3^1\,5^2\,7^0\,11^1\,13^0\,17^0$$
$$\to_P 2^0\,3^1\,5^2\,7^1\,11^0\,13^0\,17^0 \to_P 2^0\,3^1\,5^2\,7^0\,11^0\,13^1\,17^0$$
$$\to_P 2^1\,3^0\,5^2\,7^0\,11^0\,13^0\,17^1 \to_P 2^1\,3^0\,5^2\,7^0\,11^0\,13^1\,17^0$$
$$\to_P 2^1\,3^0\,5^2\,7^0\,11^0\,13^0\,17^0 \to_P 2^1\,3^1\,5^1\,7^0\,11^0\,13^0\,17^0$$
$$\to_P 2^1\,3^2\,5^0\,7^0\,11^0\,13^0\,17^0 = 2^1\,3^2$$

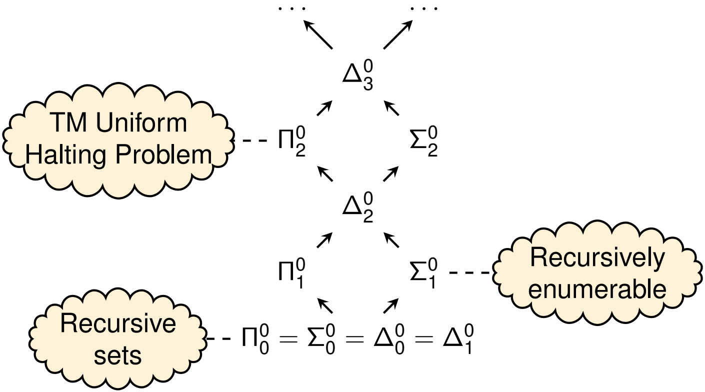
Fractran, example 2

$$P = \frac{5\times 11}{2\times 7}, \frac{7}{11}, \frac{13}{7}, \frac{2\times 17}{3\times 13}, \frac{13}{17}, \frac{1}{13}, \frac{3}{5}$$

P swaps the contents of registers r_2 and r_3: $2^n\, 3^m\, 7 \twoheadrightarrow_P 2^m\, 3^n$

$$\begin{aligned}
2^2\, 3^1\, 7^1 &= 2^2\, 3^1\, 5^0\, 7^1\, 11^0\, 13^0\, 17^0 \to_P 2^1\, 3^1\, 5^1\, 7^0\, 11^1\, 13^0\, 17^0 \\
&\to_P 2^1\, 3^1\, 5^1\, 7^1\, 11^0\, 13^0\, 17^0 \to_P 2^0\, 3^1\, 5^2\, 7^0\, 11^1\, 13^0\, 17^0 \\
&\to_P 2^0\, 3^1\, 5^2\, 7^1\, 11^0\, 13^0\, 17^0 \to_P 2^0\, 3^1\, 5^2\, 7^0\, 11^0\, 13^1\, 17^0 \\
&\to_P 2^1\, 3^0\, 5^2\, 7^0\, 11^0\, 13^0\, 17^1 \to_P 2^1\, 3^0\, 5^2\, 7^0\, 11^0\, 13^1\, 17^0 \\
&\to_P 2^1\, 3^0\, 5^2\, 7^0\, 11^0\, 13^0\, 17^0 \to_P 2^1\, 3^1\, 5^1\, 7^0\, 11^0\, 13^0\, 17^0 \\
&\to_P 2^1\, 3^2\, 5^0\, 7^0\, 11^0\, 13^0\, 17^0 = 2^1\, 3^2
\end{aligned}$$

The arithmetical hierarchy



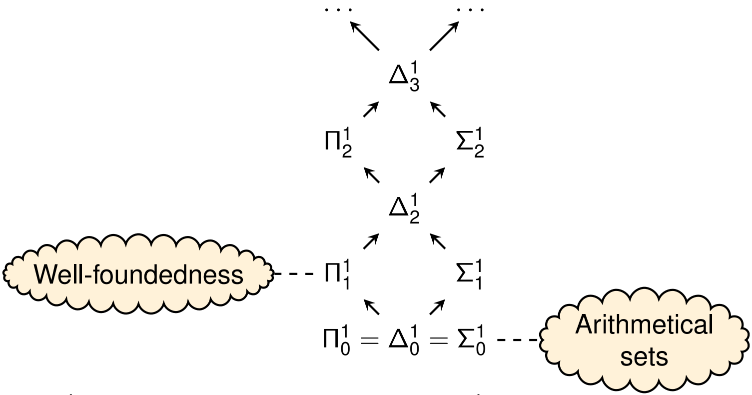
$\mathbf{\Pi^0_0} := \mathbf{\Sigma^0_0} :=$ 1st-order arithmetic formulas with bounded quantifiers

$$\mathbf{\Sigma^0_{n+1}} := \{\exists x_1 \ldots \exists x_k \Psi \mid \Psi \in \mathbf{\Pi^0_n}\}$$

$$\mathbf{\Pi^0_{n+1}} := \{\forall x_1 \ldots \forall x_k \Psi \mid \Psi \in \mathbf{\Sigma^0_n}\}$$

$\Sigma^0_n(\Pi^0_n) :=$ interpretations of formulas in $\mathbf{\Sigma^0_n}(\mathbf{\Pi^0_n})$ over $\mathbb{N}$ $\qquad \Delta^0_n := \Sigma^0_n \cap \Pi^0_n$

The analytical hierarchy



$\mathbf{\Pi^1_0} := \mathbf{\Sigma^1_0} :=$ 2nd-order arithm. formulas without set quantifiers

$\mathbf{\Sigma^1_{n+1}} := \{\exists X_1 \ldots \exists X_k \Psi \mid \Psi \in \mathbf{\Pi^1_n}\}$

$\mathbf{\Pi^1_{n+1}} := \{\forall X_1 \ldots \forall X_k \Psi \mid \Psi \in \mathbf{\Sigma^1_n}\}$

$\Sigma^1_n(\Pi^1_n) :=$ interpretations of formulas in $\mathbf{\Sigma^1_n}(\mathbf{\Pi^1_n})$ over $\mathbb{N}$

$\Delta^1_n := \Sigma^1_n \cap \Pi^1_n$

Overview

1. Introduction
2. Complexity of Fractran
3. Complexity of productivity for LSF-spec's
4. Complexity of productivity (in TRS's), and of variant definitions
5. Summary

Complexity of Fractran (I) [well-known]

Fractran is Turing-complete (Conway). An easy consequence:

Proposition

The halting problem for Fractran programs is Σ^0_1-complete.

HALTING PROBLEM FOR FRACTRAN PROGRAMS

Instance: Code $\ulcorner P \urcorner$ of a Fractran program P, a positive integer n.
Question: Does P holds for starting value n?
Problem: $\{\langle \ulcorner P \urcorner, n \rangle \mid P \text{ holds for starting value } n\}$

Proof.

Σ^0_1-hardness: reducing the halting problem for Turing-machines (Σ^0_1-complete) to the halting problem for Fractran programs by a 'folklore' encoding of Turing-machines as Fractran programs. □

Complexity of Fractran (I) [well-known]

Fractran is Turing-complete (Conway). An easy consequence:

Proposition

The halting problem for Fractran programs is Σ_1^0-complete.

HALTING PROBLEM FOR FRACTRAN PROGRAMS

Instance: Code $\ulcorner P \urcorner$ of a Fractran program P, a positive integer n.
Question: Does P holds for starting value n?
Problem: $\{\langle \ulcorner P \urcorner, n\rangle \mid P$ holds for starting value $n\}$

Proof.

Σ_1^0-hardness: reducing the halting problem for Turing-machines (Σ_1^0-complete) to the halting problem for Fractran programs by a 'folklore' encoding of Turing-machines as Fractran programs. □

Complexity of Fractran (II)

Theorem

The uniform halting problem for Fractran programs is Π^0_2-complete.

UNIFORM HALTING PROBLEM FOR FRACTRAN PROGRAMS

Instance: Encoding $\ulcorner P \urcorner$ of a Fractran program P.

Question: Does P holds for every starting value $n \in \mathbb{N}_{>0}$?

Proof.

We show Π^0_2-hardness by reducing the Π^0_2-complete

– uniform halting problem for Turing-machines (halting on all config's)

to the

– uniform halting problem for Fractran programs

by a refined encoding of Turing-machines as Fractran programs. □

Complexity of Fractran (II)

Theorem

The uniform halting problem for Fractran programs is Π^0_2-complete.

UNIFORM HALTING PROBLEM FOR FRACTRAN PROGRAMS
Instance: Encoding $\ulcorner P \urcorner$ of a Fractran program P.
Question: Does P holds for every starting value $n \in \mathbb{N}_{>0}$?

Proof.

We show Π^0_2-hardness by reducing the Π^0_2-complete
– uniform halting problem for Turing-machines (halting on all config's)
to the
– uniform halting problem for Fractran programs
by a refined encoding of Turing-machines as Fractran programs. □

Overview

1. Introduction
2. Complexity of Fractran
3. Complexity of productivity for LSF-spec's
4. Complexity of productivity (in TRS's), and of variant definitions
5. Summary

Encoding Collatz conjecture in LSF (1)

Conjecture (Collatz Conjecture, $3n+1$-problem)

For all $N \geq 1$, let $$g(N) := \begin{cases} \frac{N}{2} & N \text{ is even} \\ 3N+1 & N \text{ is odd}\,. \end{cases}$$

It holds: $(\forall N \geq 1)\,(\exists i \in \mathbb{N})\,(g^i(N) = 1)$.

Writing '$\bullet$' for successful termination, and dividing $3N+1$ immediately by 2 in case N is odd, the Collatz conjecture can be reformulated as

$$(\forall N \geq 1)\,(\exists i \in \mathbb{N})\,(F^i(N) = \bullet)$$

for $F : \mathbb{N} \cup \{\bullet\} \rightharpoonup \mathbb{N} \cup \{\bullet\}$ defined by:

$$\begin{aligned} F(1) &= \bullet \\ F(2n) &= n && (n > 0) \\ F(2n+1) &= 3n+2 && (n > 0) \end{aligned}$$

Encoding Collatz conjecture in LSF (2)

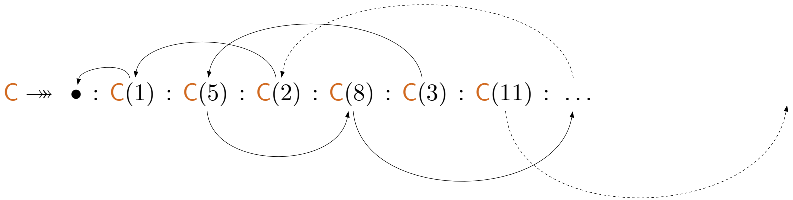
We have seen

$$\begin{aligned} \mathsf{C} &\to \bullet : \mathsf{zip}_2(\mathsf{C}, \mathsf{mod}_3(\mathsf{tail}^4(\mathsf{C}))) \\ &\twoheadrightarrow \bullet : \mathsf{C}(1) : \mathsf{C}(5) : \mathsf{C}(2) : \mathsf{C}(8) : \mathsf{C}(3) : \mathsf{C}(11) : \ldots \\ &\twoheadrightarrow \ldots \end{aligned}$$

resembling Collatz function F written as a stream:

$$F = \bullet : 1 : 5 : 2 : 8 : 3 : 11 : \ldots$$

Picturing the 'runs' through C, we get



Encoding Collatz conjecture in LSF (3)

$$\mathsf{C} \to \bullet : \mathsf{zip}_2(\mathsf{C}, \mathsf{mod}_3(\mathsf{tail}^4(\mathsf{C})))$$

Proposition

Collatz conjecture is true

$\iff$ *all runs are finite (ending in $\bullet$)*

$\iff$ *the specification for* C *is productive:*
i.e. $\mathsf{C} \twoheadrightarrow \bullet : \bullet : \bullet : \bullet : \ldots$

Complexity of productivity for LSF-specifications

Theorem

Productivity for LSF-specifications is Π^0_2-hard.

Proof.

Reducing the uniform halting problem for Fractran prog's to the productivity problem for LSF-spec's:

For every Fractran program P construct an LSF-spec $\mathcal{R}_P$ with:

$$P \text{ terminates on all } N > 0 \iff \mathcal{R}_P \text{ is productive.}$$

□

Fractran to LSF, example

The Fractran program $P = \frac{2}{3}$ is translated into the lazy specification $\mathcal{R}_P$:

$$\mathsf{M}_P \to \bullet : \bullet : \mathsf{mod}_2(\mathsf{tail}(\mathsf{M}_P))$$
$$\mathsf{mod}_2(\sigma) \to \mathsf{head}(\sigma) : \mathsf{mod}_2(\mathsf{tail}^2(\sigma))$$
$$\mathsf{tail}(x : \sigma) \to \sigma$$

We get that

$$\mathsf{M}_P \twoheadrightarrow \bullet : \bullet : \mathsf{M}_P(2) : \bullet : \bullet : \mathsf{M}_P(4) : \bullet : \bullet : \mathsf{M}_P(6) : \ldots :$$

This is a productive specification: all runs are finite.
Hence the rewrite sequence has $\bullet^\omega$ as its normal form:

$$\ldots \twoheadrightarrow \bullet : \bullet : \bullet : \ldots$$

Overview

Productivity and variants

1. $\mathsf{zeros} \to 0 : \mathsf{zeros}$
   - productive: there is only one maximal rewrite sequence: $\mathsf{zeros} \to 0 : \mathsf{zeros} \to 0 : 0 : \mathsf{zeros} \to \ldots \twoheadrightarrow 0 : 0 : 0 : \ldots$

2. $\mathsf{zeros} \to 0 : \mathsf{id}(\mathsf{zeros}) \qquad \mathsf{id}(\sigma) \to \sigma$
   - $\mathsf{zeros} \twoheadrightarrow 0 : \mathsf{id}(0 : \mathsf{id}(0 : \mathsf{id}(\ldots)))$
   - still productive, since for all max. outermost-fair rewrite sequences: $\mathsf{zeros} \twoheadrightarrow 0 : 0 : 0 : \ldots$

Even for well-behaved spec's (orthogonal TRSs), productivity should be based on a fair treatment of outermost redexes.

Productivity and variants

1 $$\mathsf{zeros} \to 0 : \mathsf{zeros}$$

- productive: there is only one maximal rewrite sequence:
  $\mathsf{zeros} \to 0 : \mathsf{zeros} \to 0 : 0 : \mathsf{zeros} \to \ldots \twoheadrightarrow 0 : 0 : 0 : \ldots$

2 $$\mathsf{zeros} \to 0 : \mathsf{id}(\mathsf{zeros}) \qquad \mathsf{id}(\sigma) \to \sigma$$

- $\mathsf{zeros} \twoheadrightarrow 0 : \mathsf{id}(0 : \mathsf{id}(0 : \mathsf{id}(\ldots)))$
- still productive, since for all max. outermost-fair rewrite sequences:
  $\mathsf{zeros} \twoheadrightarrow 0 : 0 : 0 : \ldots$

Even for well-behaved spec's (orthogonal TRSs), productivity should be based on a fair treatment of outermost redexes.

Productivity and variants

1

$$\mathsf{zeros} \to 0 : \mathsf{zeros}$$

- productive: there is only one maximal rewrite sequence:
  $\mathsf{zeros} \to 0 : \mathsf{zeros} \to 0 : 0 : \mathsf{zeros} \to \ldots \twoheadrightarrow 0 : 0 : 0 : \ldots$

2

$$\mathsf{zeros} \to 0 : \mathsf{id}(\mathsf{zeros}) \qquad \mathsf{id}(\sigma) \to \sigma$$

- $\mathsf{zeros} \twoheadrightarrow 0 : \mathsf{id}(0 : \mathsf{id}(0 : \mathsf{id}(\ldots)))$
- still productive, since for all max. outermost-fair rewrite sequences:
  $\mathsf{zeros} \twoheadrightarrow 0 : 0 : 0 : \ldots$

Even for well-behaved spec's (orthogonal TRSs), productivity should be based on a fair treatment of outermost redexes.

Productivity and variants

3 $\mathsf{maybe} \to 0 : \mathsf{maybe} \qquad \mathsf{maybe} \to \mathsf{sink} \qquad \mathsf{sink} \to \mathsf{sink}$

- productive or not, dependent on the chosen strategy
- ‘weakly productive’: $\mathsf{maybe} \twoheadrightarrow 0 : 0 : 0 : \ldots$
- not ‘strongly productive’: e.g. $\mathsf{maybe} \to \mathsf{sink} \to \mathsf{sink} \to \ldots$

4 $\mathsf{bitstream} \to 0 : \mathsf{bitstream} \qquad \mathsf{bitstream} \to 1 : \mathsf{bitstream}$

- productive independent of the strategy chosen
- ‘weakly’ and ‘strongly productive’
- infinite normal forms not unique

Productivity and variants

3 $\quad \mathsf{maybe} \to 0 : \mathsf{maybe} \qquad \mathsf{maybe} \to \mathsf{sink} \qquad \mathsf{sink} \to \mathsf{sink}$

- productive or not, dependent on the chosen strategy
- 'weakly productive': $\mathsf{maybe} \twoheadrightarrow 0 : 0 : 0 : \ldots$
- not 'strongly productive': e.g. $\mathsf{maybe} \to \mathsf{sink} \to \mathsf{sink} \to \ldots$

4 $\quad \mathsf{bitstream} \to 0 : \mathsf{bitstream} \qquad \mathsf{bitstream} \to 1 : \mathsf{bitstream}$

- productive independent of the strategy chosen
- 'weakly' and 'strongly productive'
- infinite normal forms not unique

Definition of productivity in general TRSs

We think:

- For non-well-behaved spec's (non-orthogonal TRSs), productivity has to be defined relative a given rewrite strategy.
- Strategy-independent variants (strong, weak productivity) are of limited general interest.
- Uniqueness of (infinite) normal form UN^{∞} should be considered to be a separate property, independent of productivity. (In orthogonal TRSs, UN^{∞} is guaranteed.)

Productivity w.r.t. computable strategies

A strategy for a rewrite relation $\to_R$ is a relation $\rightsquigarrow \subseteq \to_R$ with the same normal forms as $\to_R$.

Definition

A term t is called productive w.r.t. a strategy $\rightsquigarrow$ if all maximal $\rightsquigarrow$-rewrite sequences starting from t end in a constructor normal form.

- (TRS-indexed) family of strategies $\mathcal{S}$: a function that assigns to every TRS R set $\mathcal{S}(R)$ of strategies for R.
- such a family $\mathcal{S}$ is admissible: if R is orthogonal, $\mathcal{S}(R) \neq \emptyset$.

PRODUCTIVITY PROBLEM w.r.t. a family $\mathcal{S}$ of computable strategies

Instance: Encodings of a finite TRS R, a strategy $\rightsquigarrow \in \mathcal{S}(R)$, and a term t in R.

Question: Is t productive w.r.t. $\rightsquigarrow$?

Productivity w.r.t. computable strategies

Theorem

For every family of admissible, computable strategies $\mathcal{S}$, the productivity problem w.r.t. $\mathcal{S}$ is Π^0_2-complete.

Proof.

Contained in Π^0_2: a term t is productive w.r.t. $\rightsquigarrow \in \mathcal{S}(R)$ iff

$$\left.\begin{array}{l}\forall d \in \mathbb{N}.\ \exists n \in \mathbb{N}.\ \text{every } n\text{-step } \rightsquigarrow\text{-reduct of } t \\ \qquad\qquad\qquad \text{is a constructor normal form up to depth } d\end{array}\right\} \in \Pi^0_2$$

Π^0_2-complete: By reducing the totality problem for Turing-machines, which is Π^0_2-complete, to the productivity problem here. □

Corollary

In orthogonal TRSs, productivity w.r.t. lazy (outermost-fair) evaluation is Π^0_2-complete.

Productivity w.r.t. computable strategies

Theorem

For every family of admissible, computable strategies $\mathcal{S}$, the productivity problem w.r.t. $\mathcal{S}$ is Π^0_2-complete.

Proof.

Contained in Π^0_2: a term t is productive w.r.t. $\rightsquigarrow \in \mathcal{S}(R)$ iff

$$\left.\begin{array}{l}\forall d \in \mathbb{N}.\ \exists n \in \mathbb{N}.\ \text{every } n\text{-step} \rightsquigarrow\text{-reduct of } t \\ \qquad\qquad\qquad\qquad \text{is a constructor normal form up to depth } d\end{array}\right\} \in \mathbf{\Pi^0_2}$$

Π^0_2-complete: By reducing the totality problem for Turing-machines, which is Π^0_2-complete, to the productivity problem here. □

Corollary

In orthogonal TRSs, productivity w.r.t. lazy (outermost-fair) evaluation is Π^0_2-complete.

Strong and weak productivity

A term t is called

- strongly productive: all maximal outermost-fair rewrite sequences starting from t end in a constructor normal form.
- weakly productive: if there exists a rewrite sequence starting from t that ends in a constructor normal form.

Theorem

The recognition problem for

- *strong productivity is Π^1_1-complete;*
- *weak productivity is Σ^1_1-complete.*

Proof (Idea).

Π^1_1-hardness (Σ^1_1-hardness): reducing the

– recognition problem for well-founded (for non-well-founded) binary relations over $\mathbb{N}$, which is Π^1_1-complete (Σ^1_1-complete), to the

– to the recognition problem of strong (weak) productivity. □

Strong and weak productivity

A term t is called

- strongly productive: all maximal outermost-fair rewrite sequences starting from t end in a constructor normal form.
- weakly productive: if there exists a rewrite sequence starting from t that ends in a constructor normal form.

Theorem

The recognition problem for

- *strong productivity is Π^1_1-complete;*
- *weak productivity is Σ^1_1-complete.*

Proof (Idea).

Π^1_1-hardness (Σ^1_1-hardness): reducing the
– recognition problem for well-founded (for non-well-founded) binary relations over $\mathbb{N}$, which is Π^1_1-complete (Σ^1_1-complete), to the
– to the recognition problem of strong (weak) productivity. □

Uniqueness of infinite normal form

Theorem

The problem of recognising, for TRSs R and terms t in R, whether t has a unique (finite or infinite) normal form is Π^1_1-complete.

Changes due to adding the condition uniqueness of normal form:

(i) w.r.t. family of strategies:
- uniqueness of normal forms w.r.t. $\leadsto$: Π^0_2-complete.
- uniqueness of normal forms generally: Π^1_1-complete.

(ii) strong productivity: Π^1_1-complete

(iii) weak productivity: now $(\Pi^1_1 \cup \Sigma^1_1)$-hard

Uniqueness of infinite normal form

Theorem

The problem of recognising, for TRSs R and terms t in R, whether t has a unique (finite or infinite) normal form is Π^1_1-complete.

Changes due to adding the condition uniqueness of normal form:

(i) w.r.t. family of strategies:
- uniqueness of normal forms w.r.t. $\rightsquigarrow$: Π^0_2-complete.
- uniqueness of normal forms generally: Π^1_1-complete.

(ii) strong productivity: Π^1_1-complete

(iii) weak productivity: now $(\Pi^1_1 \cup \Sigma^1_1)$-hard

Overview

Summary: Results

- Uniform halting problem for Fractran is Π^0_2-complete
- Productivity problem for LSF-specifications is Π^0_2-complete

  (decidable for PSF-specifications, see [FCT'07, LPAR'08])
- Complexity of productivity in TRS's, and variant definitions:
  - productivity w.r.t. computable strategies: Π^0_2-complete
  - strong productivity: Π^1_1-complete
  - weak productivity: Σ^1_1-complete
  - unique infinite normal forms: Π^1_1-complete

Summary: Results