Optimization of string transducers

PhD Defense

Gaëtan Douéneau-Tabot

Under the supervision of **Olivier Carton** and **Emmanuel Filiot**

November 23rd, 2023

Université Paris Cité

INSTITUT DE RECHERCHE EN INFORMATIQUE FONDAMENTALE

DGA
DIRECTION GÉNÉRALE DE L'ARMEMENT

Motivation: program optimization

Summer “holidays” activities

- finishing PhD manuscript
- biking from Prague to Verona

Motivation: program optimization

Summer "holidays" activities

- finishing PhD manuscript
- biking from Prague to Verona

↓

Problem: using a phone as a GPS viewer, without reaching  .

Solution A: more battery $\implies$ more space + more weight.

Motivation: program optimization

Summer “holidays” activities

- finishing PhD manuscript
- biking from Prague to Verona

↓

Problem: using a phone as a GPS viewer, without reaching  .

Solution A: more battery $\implies$ more space + more weight.

Solution B: more energy efficient GPS program.

Motivation: program optimization

Summer "holidays" activities

- finishing PhD manuscript
- biking from Prague to Verona

↓

Problem: using a phone as a GPS viewer, without reaching 

~~Solution A: more battery $\implies$ more space + more weight.~~

Solution B: more energy efficient GPS program.

↓

Motivation of this thesis: program optimization

Given a program, automatically build a more efficient (with respect to resources consumption) equivalent program.

→ Useful for systems with limited resources (bike?, satellite, etc.).

Example: optimization of nested loops

- Input: a 0/1 sequence denoted list.
- Output: number of pairs $i \geq j$ such that list[i] = list[j] $= 0$.

Computing pairs

```
n := 0
for i from 1 to length(list) do
    for j from 1 to i do
        if list[i] =list[j] = 0 then
            n := n+1
    end
end
return n
```

Execution time $\sim$ length(list)^2

Example: optimization of nested loops

- Input: a 0/1 sequence denoted list.
- Output: number of pairs $i \geq j$ such that list[i] = list[j] $= 0$.

Computing pairs	Doing a product

n := 0
for i **from** 1 **to** length(list) **do**
  for j **from** 1 **to** i **do**
    if list[i] =list[j] = 0 **then**
      n := n+1
  end
end
return n
	n := 0
	for i **from** 1 **to** length(list) **do**
	  if list[i] = 0 **then** n := n+1
	end
	return n(n+1)/2

Execution time ~ length(list)^2	Execution time ~ length(list)

Example: optimization of nested loops

- Input: a 0/1 sequence denoted list.
- Output: number of pairs $i \geq j$ such that list[i] = list[j] $= 0$.

<table>
<tr><th>Computing pairs</th><th>Doing a product</th></tr>
<tr><td>

```
n := 0
for i from 1 to length(list) do
    for j from 1 to i do
        if list[i] =list[j] = 0 then
            n := n+1
    end
end
return n
```

</td><td>

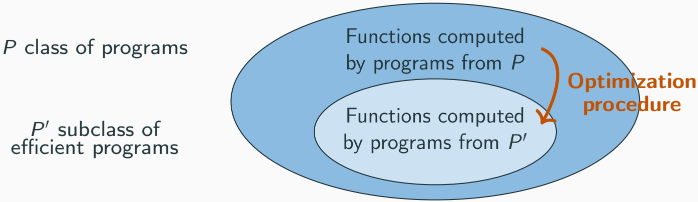
```
n := 0
for i from 1 to length(list) do
    if list[i] = 0 then  n := n+1
end
return n(n+1)/2
```

</td></tr>
<tr><td>Execution time ~ length(list)^2</td><td>Execution time ~ length(list)</td></tr>
<tr><td>Pebble transducer</td><td>Blind pebble transducer</td></tr>
</table>

Optimization procedure [Manuscript, Chapter 6]

Formalization: class membership problems

Class membership problem from P to P'



- Input: Program from P.
- Question: Does an equivalent program from P' exist ?
  + Effectively build this program.

Example: Removing nested loops

P programs with 2 nested loops, P' programs without nested loops.

Formalization: class membership problems for transducers

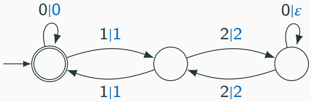
Class membership problems are often challenging

- Depend on the semantic and not to the syntax of programs.
- Quickly undecidable for classes of expressive programs.
- Heuristic approaches are used to avoid undecidability/complexity.

→ **In this thesis:** optimal results for classes of restricted programs.

Finite-state transducer

- finite state program;
- input: string, output: string.



→ **In this thesis:** restricted programs = extended transducers.

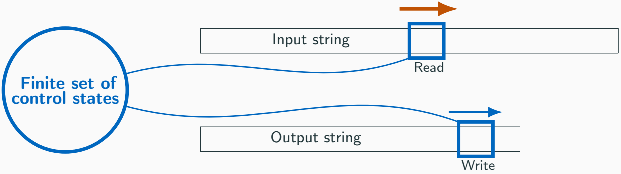
Outline

1. Background: transducers of finite strings
2. Nesting optimization within subclasses of pebble transducers
3. Pebble transducers with commutative output
4. Determinization for transducers of infinite strings
5. Outlook

Background: transducers of finite strings

One-way transducers

Definition: one-way deterministic transducer



→ Compute the class of sequential functions from strings to strings.

Example: first letter to last position $\mathbf{1}2345 \mapsto 2345\mathbf{1}$

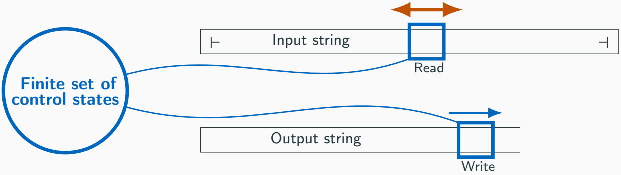
Definition: (functional) one-way non-deterministic transducer

→ Compute the class of rational functions from strings to strings.

Example: last letter to first position $1234\mathbf{5} \mapsto \mathbf{5}1234$

Two-way transducers

Definition: **two-way deterministic transducer**



→ Compute the class of regular functions from strings to strings.

Example: duplicating the input $12345 \mapsto 12345\#12345$

Example: reversing the input $12345 \mapsto 54321$

Remark: non-determinism does not increase expressive power.

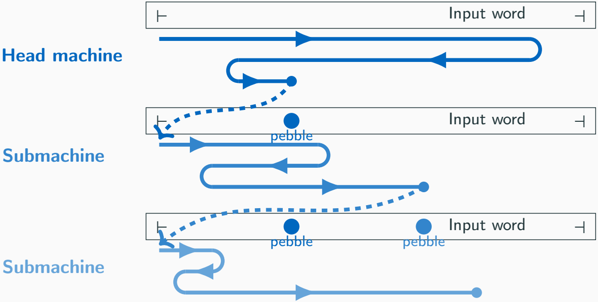
Pebble transducers

Definition: k-pebble transducers

- Nested two-way transducers with nesting depth k.
- A pebble is added to the input when doing a nested call.

→ Compute the class of polyregular functions.

Behavior of a 3-pebble transducer



Pebble transducers

Example: square $1234 \mapsto 1234\#1234\#1234\#1234$

can be computed by a 2-pebble transducer.

Example: unary product $\underbrace{1\dots1}_{n}\#\underbrace{1\dots1}_{n'}\#\underbrace{1\dots1}_{n''} \mapsto \underbrace{1\dots1}_{n\times n'\times n''}$

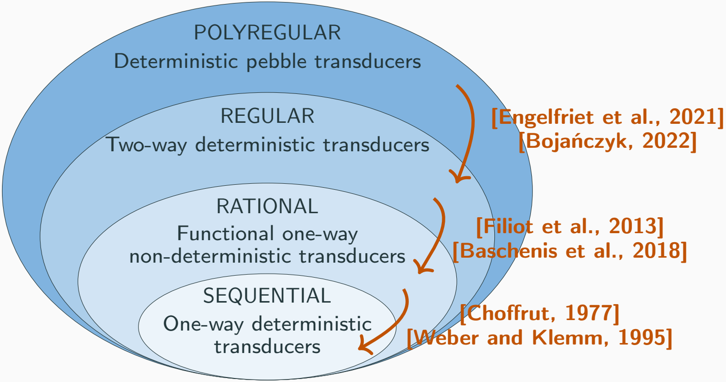
can be computed by a 3-pebble transducer.

Asymptotic growth of the output

- A k-pebble transducer can be understood...
  - either a program with functions calls of nesting depth k;
  - or as a program with k nested (two-way) for loops.
- If a k-pebble transducer computes a function f, then:

$$|f(u)| = \mathcal{O}(|u|^k).$$

Known membership results



- All the statements are decidable + effective.
- The proofs are rather technical and use disparate methods.

Membership and output growth

Theorem: [Engelfriet et al., 2021, Bojańczyk, 2022]

Let f be a polyregular function, then $|f(u)| = \mathcal{O}(|u|) \iff f$ is regular.

→ More generally, do we (effectively) have:
$|f(u)| = \mathcal{O}(|u|^k) \iff f$ can be computed by a k-pebble transducer?

Counterexample [Bojańczyk, 2023]

For all $k \geq 3$, there exists a polyregular function f such that $|f(u)| = \mathcal{O}(|u|^2)$ but cannot be computed with less than k pebbles.

→ **First part of this thesis:** subclasses of pebble transducers where:
$|f(u)| = \mathcal{O}(|u|^k) \iff f$ can be computed by with k nested layers.
+ Decidable + Effective (nested loop optimization).

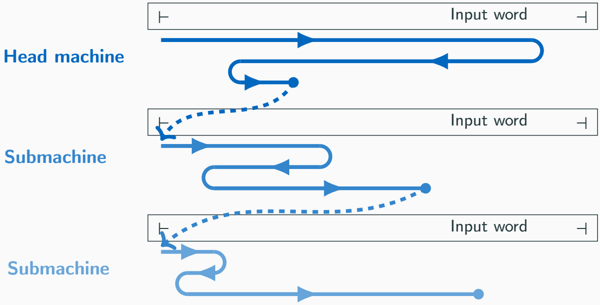
Nesting optimization within subclasses of pebble transducers

Blind pebble transducers [Nguyên et al., 2021]

Definition: blind k-pebble transducer

Submachines have no information about the positions of the calls.

Behavior of a blind 3-pebble transducer



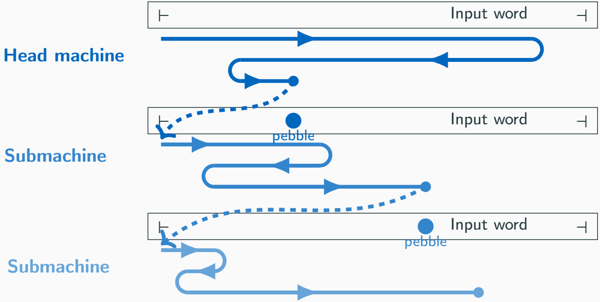
Example: square $1234 \mapsto 1234\#1234\#1234\#1234$

Last pebble transducers [Engelfriet et al., 2007]

Definition: last k-pebble transducer

Submachines can only see the position in which they are called.

Behavior of a last 3-pebble transducer



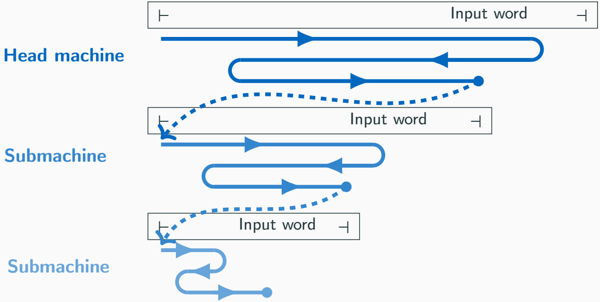
Example: marked square $1234 \mapsto \underline{1}234\#1\underline{2}34\#12\underline{3}4\#123\underline{4}$

Marble transducers [Engelfriet et al., 1999]

Definition: k**-marble transducer**

Submachines are called on a prefix of the input.

Behavior of a 3-marble transducer



Example: prefixes $1234 \mapsto 1\#12\#123\#1234$

Nesting optimization [Chapters 3 and 4]

Theorem: nesting optimization

Let $1 \leq \ell \leq k$. The following are (effectively) equivalent:

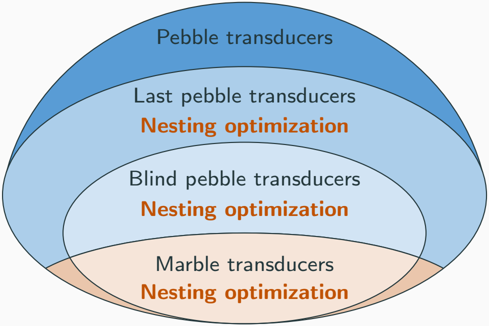
1. f is computed by a blind k-pebble / last k-pebble / k-marble transducer and $|f(u)| = \mathcal{O}(|u|^{\ell})$;
2. f is computed by a blind ℓ-pebble / last ℓ-pebble / ℓ-marble.

+ Decidable membership problem.

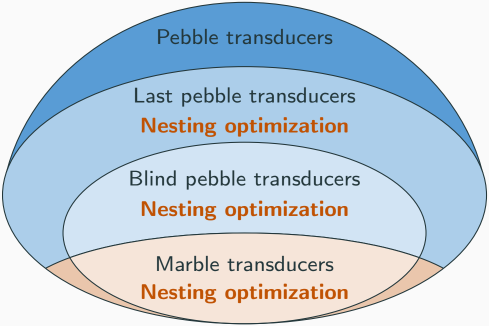
Main proof ideas

- For blind pebble / last pebble transducers: transition monoids + factorization forests + pumping arguments.
- For marble transducers: correspondence with *streaming string transducers* + classical techniques for *weighted automata*.

Overview: nesting optimization [Chapters 3 and 4]



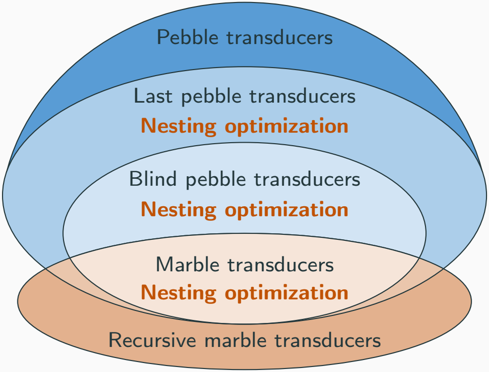
Overview: nesting optimization [Chapters 3 and 4]



Can we go beyond using output growth?

- No for other subclasses of pebble transducers.

Overview: nesting optimization [Chapters 3 and 4]



Can we go beyond using output growth?

- No for other subclasses of pebble transducers.
- Yes for models with unbounded nesting depth (≡ recursion): shown for marbles + conjectured for last pebbles.

Pebble transducers with commutative output

Pebble transducers with output in $\mathbb{N}$ / $\mathbb{Z}$ [Chapter 5]

Definition: transducers with outputs in $\mathbb{N}$ / $\mathbb{Z}$

- case of $\mathbb{N}$: output alphabet is $\{1\}$, result is the length/sum;
- case of $\mathbb{Z}$: output alphabet is $\{\pm 1\}$, result is the sum.

Examples: pebble transducers with output in $\mathbb{Z}$

- $u \mapsto (|u|_0 - |u|_1)^2$ is computed by a (blind) 2-pebble transducer;
- $u \mapsto (-1)^{|u|}|u|^3$ is computed by a (blind) 3-pebble transducer.

Theorem: pebble $\equiv$ last pebble $\equiv$ marble

For $k \geq 1$, k-pebble, last k-pebble and k-marble transducers with output in $\mathbb{N}$ / $\mathbb{Z}$ (effectively) compute the same classes of functions.

Pebble transducers with output in $\mathbb{N}$ / $\mathbb{Z}$ [Chapter 5]

Theorem: subclass of rational series [implicit in folklore]

The following are (effectively) equivalent:

1. f is a $\mathbb{N}$- / $\mathbb{Z}$-rational series and $|f(u)| = \mathcal{O}(|u|^k)$ for some $k \geq 1$;
2. f is computed by a pebble transducer with output in $\mathbb{N}$ / $\mathbb{Z}$.

\+ Decidable membership problem.

Theorem: nesting optimization

Let $1 \leq \ell \leq k$. The following are (effectively) equivalent:

1. f is computed by a k-pebble transducer in $\mathbb{N}/\mathbb{Z}$ and $|f(u)| = \mathcal{O}(|u|^\ell)$;
2. f is computed by an ℓ-pebble transducer in $\mathbb{N}/\mathbb{Z}$.

\+ Decidable membership problem.

Main proof ideas

For $\mathbb{Z}$: tuples in factorization forests + multivariate polynomials.

Blind pebble transducers with output in $\mathbb{N}$ / $\mathbb{Z}$ [Chapter 6]

Example: squaring blocks $1^{n_1} \# 1^{n_2} \# \cdots \# 1^{n_m} \mapsto \sum_{i=1}^{m} n_i^2$ cannot be computed by a blind pebble transducer.

→ Blind are less expressive than pebble ≡ last pebble ≡ marble.

Theorem: blind membership

The following are (effectively) equivalent:

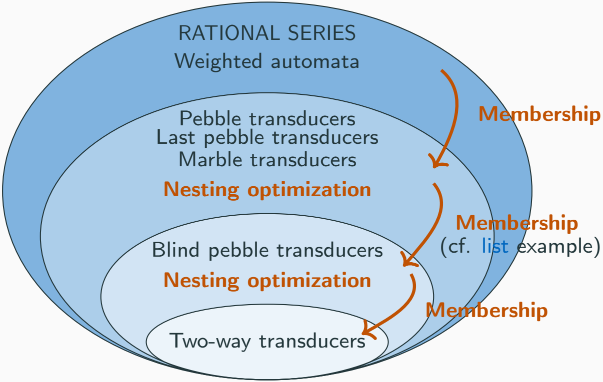
1. f is computed by a pebble transducer in $\mathbb{N}/\mathbb{Z}$ and is repetitive;
2. f is computed by a blind pebble transducer in $\mathbb{N}/\mathbb{Z}$.

+ Decidable membership problem + Commutes with optimization.

Main proof ideas

Previous tools + inductive techniques on polyregular functions.

Overview: transducers with output in $\mathbb{N}$ / $\mathbb{Z}$ [Chapters 5 and 6]



+ Multiple characterizations as subclasses of rational series.

Aperiodic automata and transducers

Definition: aperiodic automata/transducer

An automaton/transducer is aperiodic if its transition monoid is so.

→ Motivated by strong connections to logics/expressions since the study of star-free expressions [Schützenberger, 1965].

Generic question: aperiodic class membership

Given a function, can it be computed by an aperiodic transducer?

→ Results for string-to-string sequential or rational functions [Filiot et al., 2019], partial results for regular [Bojańczyk, 2014].

Example: pebble transducers $u \mapsto (-1)^{|u|} \times |u|$

cannot be computed by an aperiodic pebble transducer.

Aperiodic pebble transducers with output in $\mathbb{Z}$ [Chapter 7]

Definition: smooth function

f is smooth if $X \mapsto f(uv^X w)$ is a polynomial for X large enough.

Example: $u \mapsto (-1)^{|u|} \times |u|$ is not smooth.

Theorem: aperiodic membership

The following conditions are (effectively) equivalent:

1. f is computed by a pebble transducer with output in $\mathbb{Z}$ and smooth;
2. f is computed by an aperiodic pebble transducer with output in $\mathbb{Z}$.

\+ Decidable membership problem + Commutes with optimization.

Main proof ideas

Build by residuation (crucial: $\mathbb{Z}$ is a group) a nested canonical object
\+ inductively translate smoothness into an aperiodicity property.

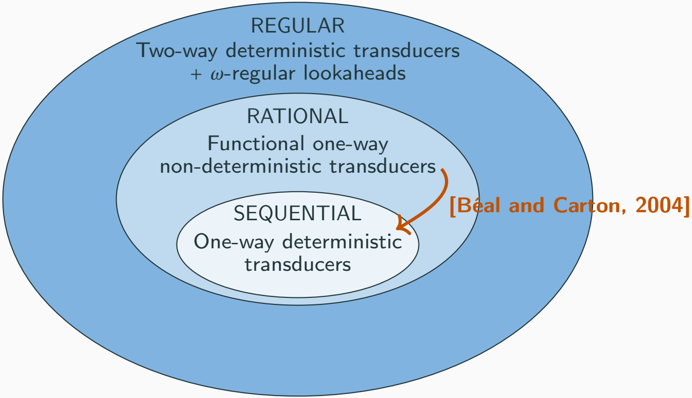
Determinization for transducers of infinite strings

Transducers of infinite strings

Definition: transducers of infinite strings

- Input: infinite string, output: infinite string.
- Infinite execution + Büchi/Muller/parity acceptance conditions.

→ Motivation: transducers of infinite strings ≡ streaming algorithms.



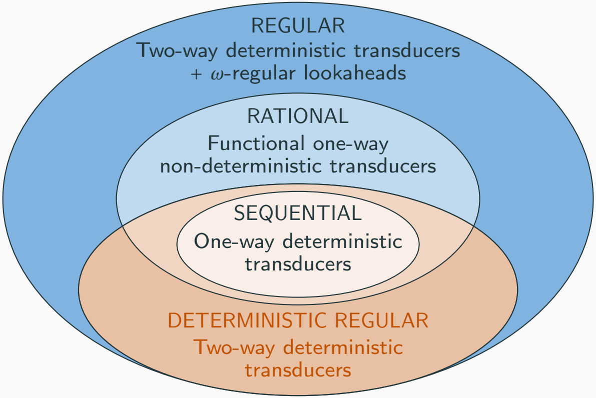
Transducers of infinite strings

Definition: deterministic regular functions

Computed by two-way deterministic transducers.

Example: normalization in base 10

$09999\cdots \mapsto 100000\cdots$ is rational but not deterministic regular.



Two-way determinization of rational functions [Chapter 10]

Theorem: two-way determinization

The following are (effectively) equivalent:

1. f is rational and continuous;
2. f is rational and deterministic regular.

+ Decidable membership problem.

Main proof arguments

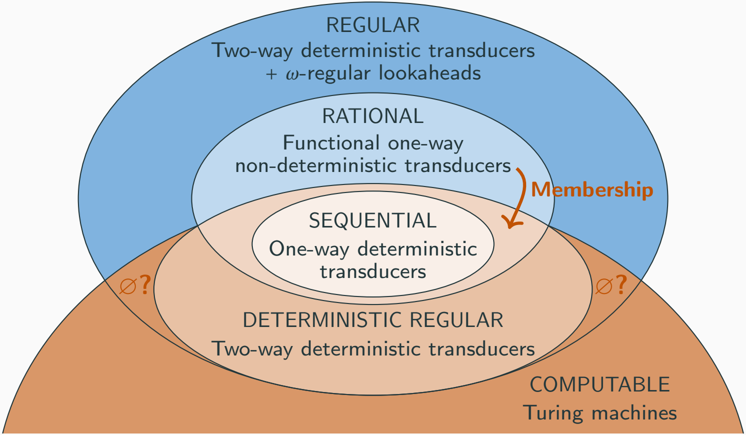
Composition of deterministic regular functions + Equivalence with *streaming string transducers* + Original tree-based constructions.

Theorem: Continuity = computability [Dave et al., 2020]

Regular $\cap$ computable = regular $\cap$ continuous.

→ Rational $\cap$ deterministic regular = rational $\cap$ computable/continuous.
→ **Conjecture:** deterministic regular = regular $\cap$ computable/continuous.

Overview: transducers of infinite strings [Chapters 9 and 10]



+ Multiple characterizations of deterministic regular functions.

Outlook

Overview of contributions

Finite strings		Infinite strings
Nesting optimization for models of nested two-way transducers	Membership problems for nested transducers with output in $\mathbb{N}$ or $\mathbb{Z}$	Determinization result + study of deterministic two-way transducers
[Manuscript, Part I] [D-T, Filiot, Gastin, 2020], [D-T, 2023]	**[Manuscript, Part II]** [D-T, 2021] [D-T, 2022] [Colcombet, D-T, Lopez, 2023]	**[Manuscript, Part III]** [Carton, D-T, 2022] [Carton, D-T, Filiot, Winter, 2023]

+ Semantic and syntactic characterizations of the classes.

+ Effective translations between several transducer models.

Present and future

Present: a toolbox for solving membership problems

- High-level strategies (syntax vs semantics).
- Low-level techniques (factorization forests, inductive methods for polyregular functions, determinization constructions, etc.).

Future: research directions

- Over infinite strings, do we have:
  deterministic regular $\equiv$ regular $\cap$ continuous?
- Are canonical models really necessary for solving class membership problems? In particular to study aperiodicity.

\+ Multiple low-hanging conjectures available.

Thank you!