# Fonctions récursives sur les mots. Complexité et constructions dépourvues de concaténation

Jérôme Durand-Lose

Laboratoire d'Informatique Fondamentale d'Orléans
ÉA 4022
Université d'Orléans, Orléans, FRANCE


Journées GAMoC 2023 — 6 juin 2023

Funded by ANR $\partial$IFFERENCE (ANR-20-CE40-0002)

LABORATOIRE D'INFORMATIQUE FONDAMENTALE D'ORLEANS

LABORATOIRE D'INFORMATIQUE FONDAMENTALE D'ORLEANS

# Well-known: Classical recursion (on natural numbers)

## Functions from $\mathbb{N}^k$ to $\mathbb{N}$ constructed from

- constant 0 function,
- successor function
- projections $(\pi_i^n)$
- composition $\mathbf{Comp}(g, (h_i)_{1 \leq i \leq k})$
- recursion $f = \mathbf{Rec}(g, h)$ defined by:

$$\begin{aligned} f(0, \vec{y}) &= g(\vec{y}) \qquad \text{and} \\ f(n+1, \vec{y}) &= h(n, f(n, \vec{y}), \vec{y}) \end{aligned}$$

- (add minimisation to get all recursive functions)

## Pros

- simple
- relate to arithmetic

## Cons

- unfit for symbolic manipulation
- complexity blowup

# Recursion on string/words

- $\Sigma = \{a_1, a_2, \cdots, a_r\}$
- $\varepsilon$ empty word

## Functions from $(\Sigma^*)^k$ to $\Sigma^*$ constructed from

- constant $\widehat{\varepsilon}$,
- all left concatenation by one letter/symbol ${}_{a}\cdot(w) = a \cdot w = aw$
- projections $(\pi_i^n)$
- composition $\mathbf{Comp}(g, (h_i)_{1 \leq i \leq k})$
- (left) recursion $f = \mathbf{Rec}\big(g, (h_a)_{a \in \Sigma}\big)$ defined by:

$$\begin{array}{rrcl} & f(\varepsilon, \vec{y}) & = & g(\vec{y}) \qquad \text{and} \\ \forall a \in \Sigma, & f(a \cdot w, \vec{y}) & = & h_a(w, f(w, \vec{y}), \vec{y}) \end{array}$$

- (what minimisation to get all recursive functions?)

# Observations

## 1 letter alphabet corresponds to $\mathbb{N}$ (in unary)

- everything matches

## *r-adic* encoding function from $\Sigma^*$ to $\mathbb{N}$

- $\Sigma = \{a_1, a_2, \cdots, a_r\}$
- $\langle \varepsilon \rangle = 0$
- $a_k \cdot w$, $\langle a_k \cdot w \rangle = k + r \cdot \langle w \rangle$
- division, modulo, multiplication, addition...
  are primitive recursive (on $\mathbb{N}$)

## Since the functions are the same (up to some encoding)...

- Why bother?

# Why bother? indeed

## Tropism

- culture and education stress on numbers, symbols are only to write sentences with
- proof by recursion and not induction
  (up to introducing measures like depth to do recursion)

## Symbols are what is relevant

- in nowadays computations, computers...
- natural numbers are represented by *sequences of symbols*

## Computability...

- is about symbol manipulation
- not natural numbers
- the term *recursive* is getting replaced by *computable* (Soare, 2007)

# State of the art... ancient and number oriented — 1

- *recursion on string*, *recursion on word*,
  *recursive string-functions*, *recursive word-functions*
- *recursion on representation*: representation of natural numbers by words in shortlex/military order, *non-trivial successor word-function*
- peak in the 1960's
- Most papers deal with hierarchies and is number-centric

**Cook and Kapron (2017)... notes from late 1960's**

- $m$-adic notation of numbers (digits exlude 0) and relations on weak classes
- primitives $\{n \mapsto 10\,n + i\}_{0 \leq i \leq 9}$

# State of the art... ancient and number oriented — 2

## von Henke et al. (1975)

- survey on counterparts on words of classical results for primitive recursion on numbers

## Variations

- infinite alphabet (Vučkovi, 1970), computation over finite sequences of numbers encoded by numbers
- restriction to unitary word-functions is considered in (Asser, 1987; Sântean, 1990; Calude and Sântean, 1990)
- the nowhere defined function is added to primitive recursive word-functions in Khachatryan (2015)

# Word recursion formalisation found. . .

### . . . in some textbook

- Machtey and Young (1978)
- Gallier and Quaintance (2022)

### . . . in articles

- Leivant (1994); Leivant and Marion (2020) (ramification)
- (primitive recursion over free algebras)

LABORATOIRE D'INFORMATIQUE FONDAMENTALE D'ORLEANS

# Complexity measure

## Needed

- formalism defines functions, not evaluation!
- what is the computation?
- what is the measure?

## An evaluation is a direct acyclic graph

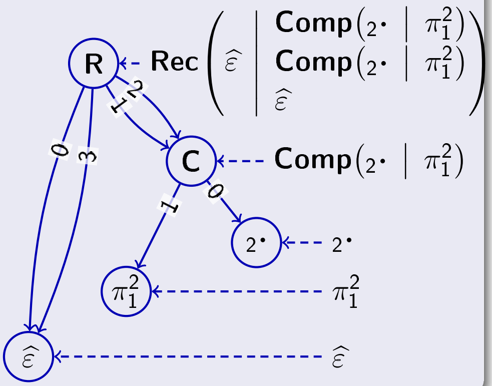
- started from the function definition and the input
- completed in a deterministic way

## Delayed evaluation

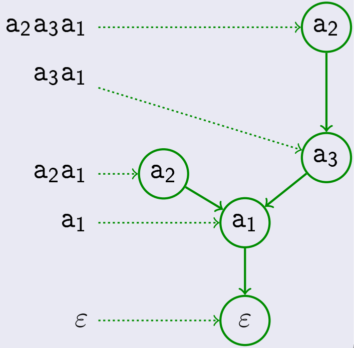
- compute a value only when needed
- call by name

# Examples of functions and values encoding

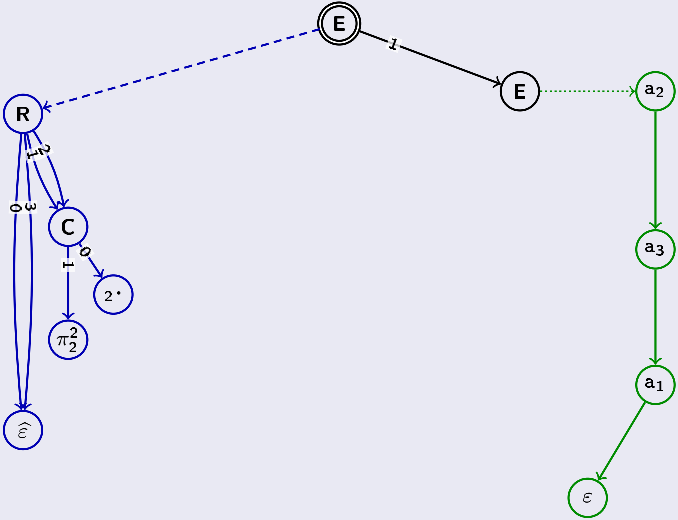
## Encoding functions



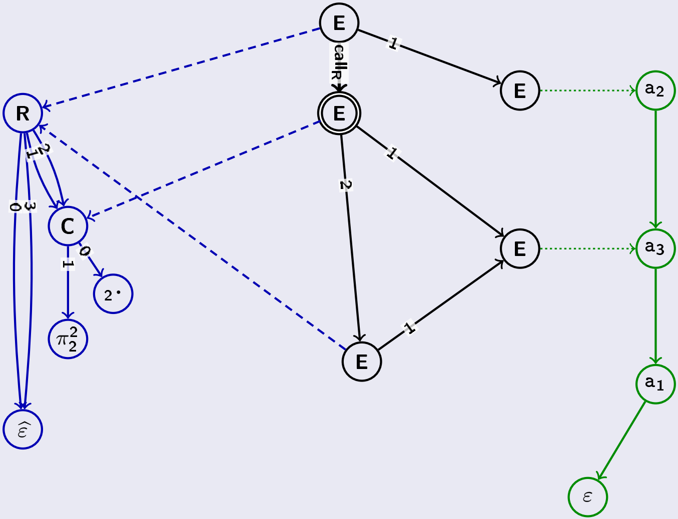

## Encoding values



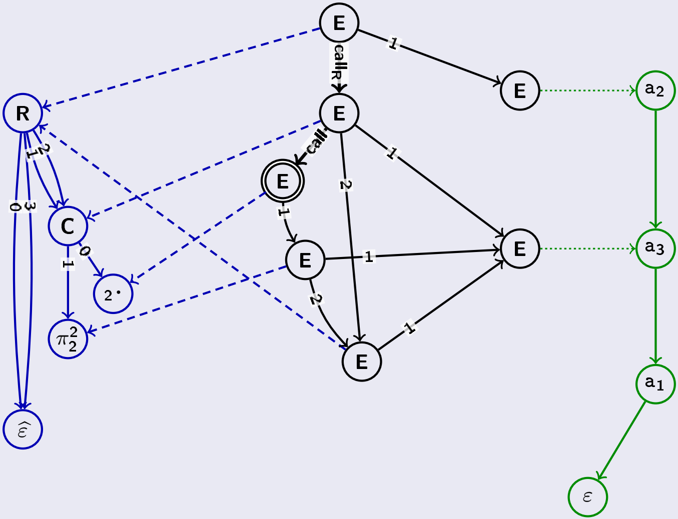
## Example of a computation



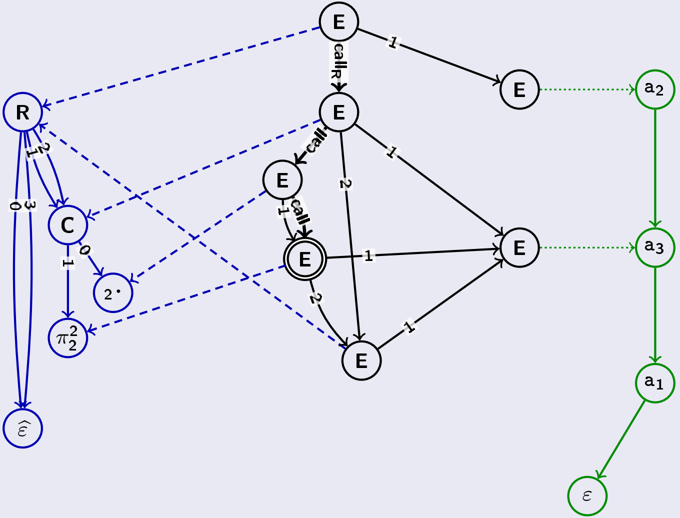
## Example of a computation



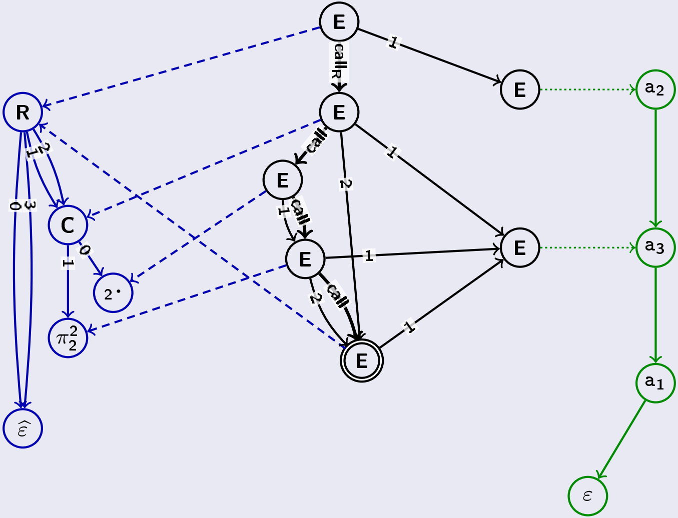
## Example of a computation



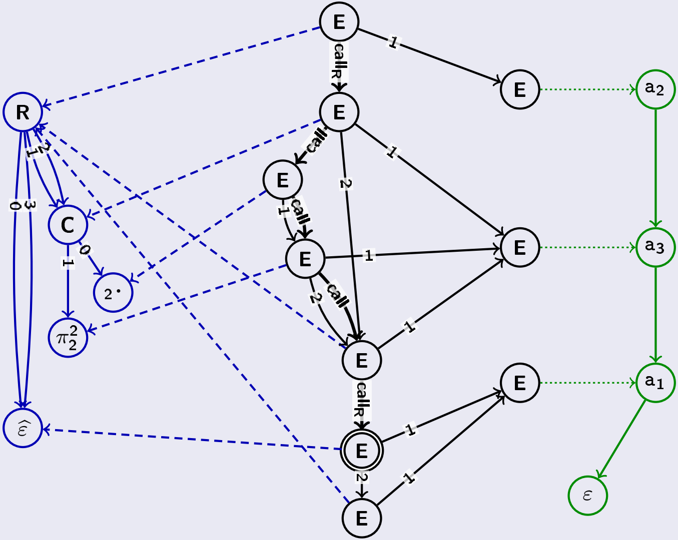
## Example of a computation



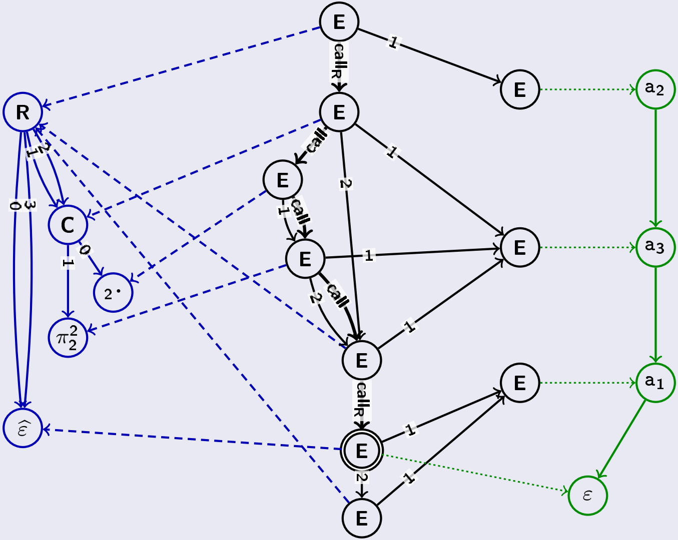
## Example of a computation



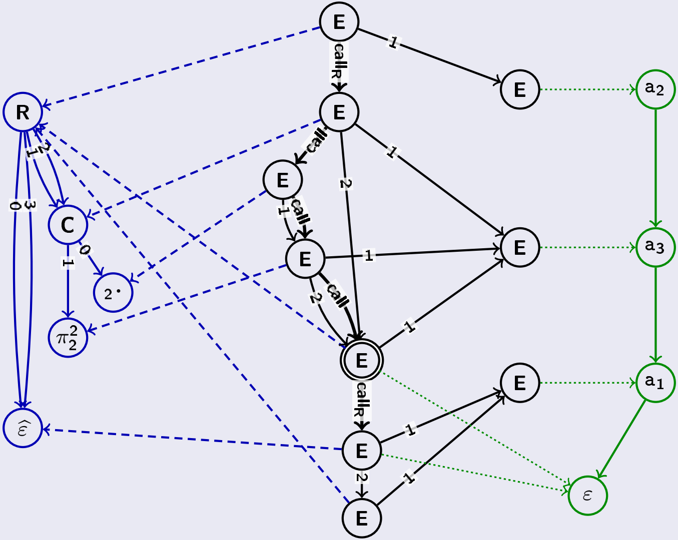
## Example of a computation



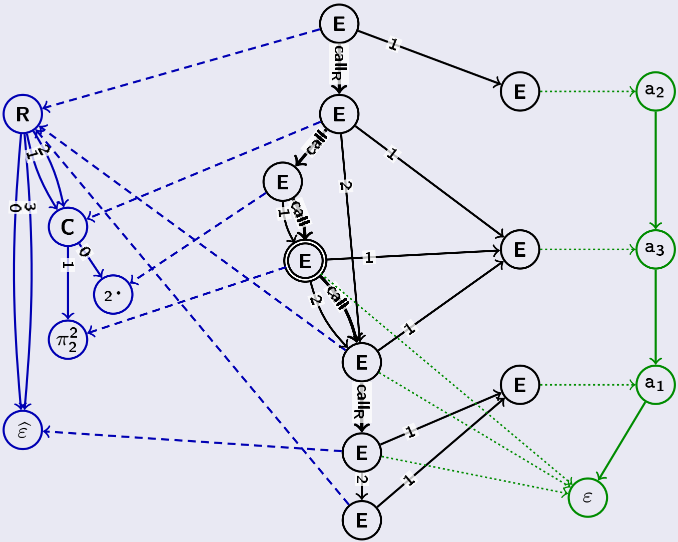
## Example of a computation



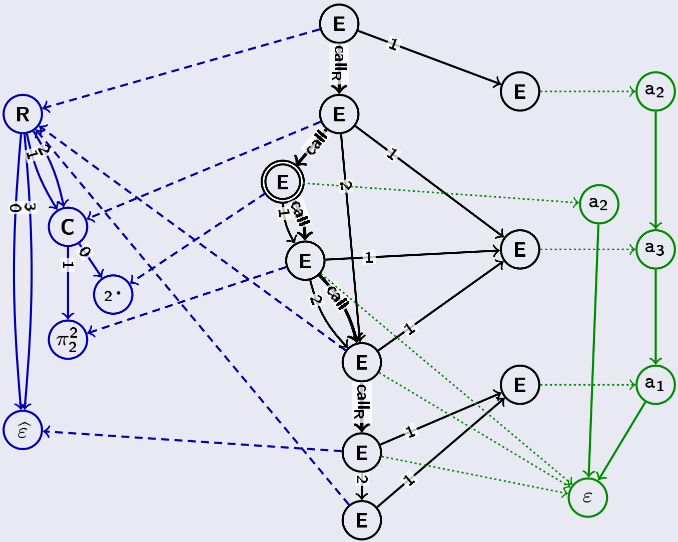
## Example of a computation



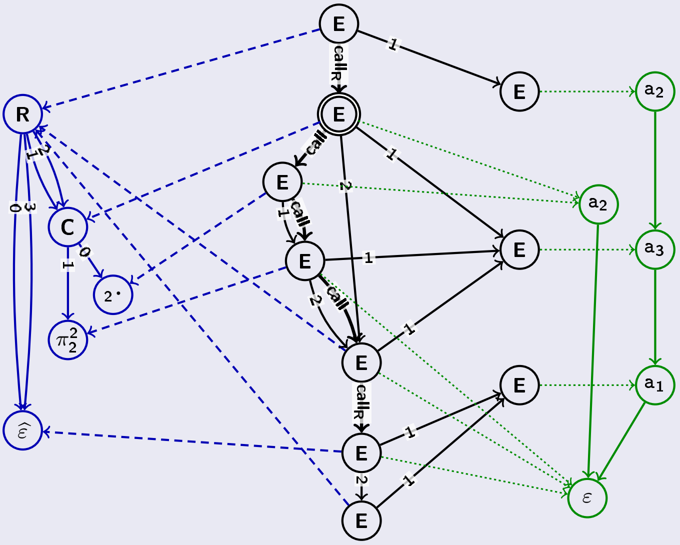
## Example of a computation



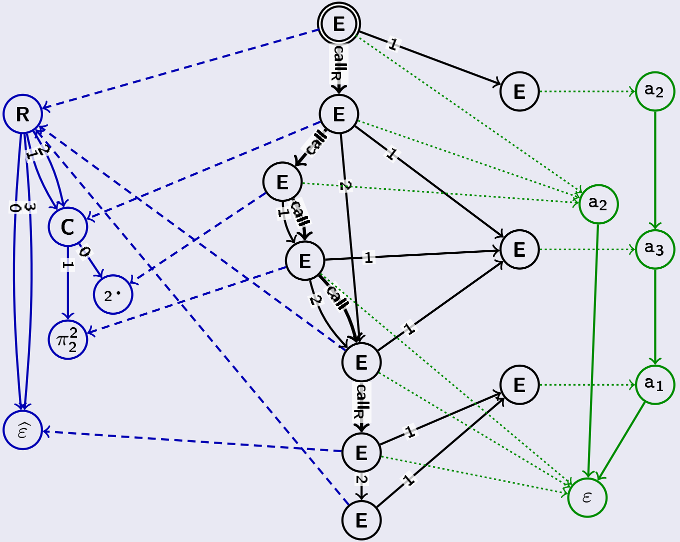
## Example of a computation



## Example of a computation



## Example of a computation



# Complexity classes

## Constructing the DAG (with your favourite language)

- update rules:
  - finitely many
  - local
  - deterministic activation
- size linear in the number of updates
- manipulation of graph: logarithmic overhead

## Simulation of a Turing machine

- encoding: state \$ read symbol \$ word on left \$ word on right
- one update in linear time

# Complexity classes

## Class P is the same

- similar definition
- (one way) simulation of a Turing machine
- (other way) construction of the DAG in quasi-linear time

## Similarly for higher classes

- NP (with certificate)
- EXP time...

LABORATOIRE D'INFORMATIQUE FONDAMENTALE D'ORLEANS

# Strong limitation

## Lemma

- the output is a suffix of an input

## Corollary

- paring is not possible anymore!
- indeed $\{\varepsilon, \mathrm{a}, \mathrm{aa}\} \times \{\varepsilon, \mathrm{a}, \mathrm{aa}\}$ has to be mapped one-to-one into $\{\varepsilon, \mathrm{a}, \mathrm{aa}\}$

## Equality test on numbers leads to a reverse test

$$\mathbf{Comp}\left(\mathbf{Rec}\left(\pi^2_1 \;\middle|\; \begin{array}{l} \mathbf{Comp}\left(\mathbf{Rec}\left(\mathrm{id} \;\middle|\; \begin{array}{c}\pi^3_1\\ \pi^3_3\end{array}\right) \;\middle|\; \begin{array}{c}\pi^4_2\\ \pi^4_4\end{array}\right) \\ \mathbf{Comp}\left(\mathbf{Rec}\left(\mathrm{id} \;\middle|\; \begin{array}{c}\pi^3_3\\ \pi^3_1\end{array}\right) \;\middle|\; \begin{array}{c}\pi^4_2\\ \pi^4_4\end{array}\right) \end{array}\right) \;\middle|\; \begin{array}{c}\pi^2_1\\ \pi^2_2\\ \pi^2_1\end{array}\right)$$

- test if on is the reverse of the other!
- $\rightsquigarrow$ palindrome test
- algebraic language, non-ambiguous but not deterministic

## Equality... more involving

- test if both first letter are similar
- remove $k$ first letters where $k$ is the size of an argument
- loop on $k$

# Language decision

- $L = f^{-1}(\{\varepsilon\})$

## Regular languages

- $(Q, \delta, q_0, A)$ some DFA
- $\vec{c}$ vector of distinct words, one for each state of the FDA
- ${}^{\mathrm{cf}}\mathrm{test}_A(x, \vec{c})$ returns true iff $x$ equals some word encoding an accepting state
- transition function:
  $\forall i, 0 \leq i \leq n, \;\; f_{\delta_i}(\mathrm{a}_i.w, x, \vec{c}) = c_k$ s.t. $x = c_j \wedge \delta(q_j, \mathrm{a}_i) = q_k$

- All regular languages are decidable (concatenation-free)

# Boolean operators — closure properties

- $\top$ identified with $\varepsilon$ ($\perp$ identify with every non-$\varepsilon$ word)

## Ternary operator / test function

- ${}^{\mathrm{cf}}\mathrm{if}_\varepsilon = \mathbf{Rec}\left(\pi^2_1, (\pi^4_4, \pi^4_4)\right)$

## Conjunction and disjunction

- $\wedge$ is ${}^{\mathrm{cf}}\mathrm{and} = \mathbf{Comp}\left({}^{\mathrm{cf}}\mathrm{if}_\varepsilon, (\pi^2_1, \pi^2_2, \pi^2_1)\right)$
- $\vee$ is ${}^{\mathrm{cf}}\mathrm{or} = \mathbf{Comp}\left({}^{\mathrm{cf}}\mathrm{if}_\varepsilon, (\pi^2_1, \widehat{\varepsilon}, \pi^2_2)\right)$

## Negation — a non-$\varepsilon$ argument is needed

- $\neg$ is $\mathbf{Comp}\left({}^{\mathrm{cf}}\mathrm{if}_\varepsilon, (\pi^2_1, \pi^2_2, \widehat{\varepsilon})\right)$ — arity is 2

## Algebraic languages

- $a_1^n a_2^n$
  - non-ambiguous, deterministic
  - read $a_1$ and stack functions to remove $a_2$
- $a_1^n a_2^n a_1^m \cup a_1^n a_2^m a_1^m$
  - ambiguous (non-deterministic)

## Non-algebraic languages

- $a_1^n a_2^n a_1^n = a_1^n a_2^n a_1^m \cap a_1^n a_2^m a_1^m$
- $a_1^n a_2^{P(n)}$ with $P$ polynomial with positive coefficients
- any boolean combination of these

### Problem: non-$\varepsilon$ output

Input: A concatenation-free primitive recursive function

Question: Is there an input such that the ouput is not $\varepsilon$?

### Or equivalently

- is the function the constant function $\varepsilon$?

### Theorem

- This problem is not decidable.

- Function are total but not all computable total functions
- Rice's theorem is not applicable

# Reduction from the halting problem

- Fix a TM and an input
- Provide an encoding of sequences of configurations
- Ouput the input if the input is a legal sequence of configurations from the TM-input to halting stage
- Otherwise output $\varepsilon$

**Construction similar to...**

- Post correspondence problem
- Ambiguity of a context-free grammar

LABORATOIRE D'INFORMATIQUE FONDAMENTALE D'ORLEANS

# Results

## With concatenation

- computability: identical
- complexity: compatible (P and above)

## Without concatenation, decides...

- all rational languages
- some algebraic (deterministic/non ambiguous/ambiguous)
- some non algebraic
- languages with polynomial conditions on exponents/repetitions (unary encoding of natural numbers)

LABORATOIRE D'INFORMATIQUE FONDAMENTALE D'ORLEANS

# Perspectives — concatenation-less

- Polynomials in many variables, negative coefficients
- All algebraic languages (deterministic, non-ambiguous, ambiguous)

- Condition for not computability/decision

LABORATOIRE D'INFORMATIQUE FONDAMENTALE D'ORLEANS

Asser, G. (1987). Primitive recursive word-functions of one variable. In Börger, E., editor, *Computation Theory and Logic, In Memory of Dieter Rödding*, volume 270 of *LNCS*, pages 14–19. Springer.

Calude, C. and Sântean, L. (1990). On a theorem of günter asser. *Math. Log. Q.*, 36(2):143–147.

Cook, S. A. and Kapron, B. M. (2017). A survey of classes of primitive recursive functions. *Electron. Colloquium Comput. Complex.*, page 1.

Durand-Lose, J. (2022). On the power of recursive word-functions without concatenation. In Han, Y. and Vaszil, G., editors, *Descriptional Complexity of Formal Systems - 24th IFIP WG 1.02 International Conference, DCFS 2022, Debrecen, Hungary*, volume 13439 of *LNCS*, pages 30–42. Springer.

Gallier, J. and Quaintance, J. (2022). Proofs, computability, undecidability, complexity, and the lambda calculus an introduction.

Khachatryan, M. H. (2015). On generalized primitive recursive string functions. *Mathematical Problems of Computer Science*, 43:42–46.

Leivant, D. (1994). A foundational delineation of poly-time. *Inf. Comput.*, 110(2):391–420.

Leivant, D. and Marion, J. (2020). Primitive recursion in the abstract. *Math. Struct. Comput. Sci.*, 30(1):33–43.

Machtey, M. and Young, P. (1978). *An Introduction to the General Theory of Algorithms*. Elsevier North-Holland.

Sântean, L. (1990). A hierarchy of unary primitive recursive string-functions. In Dassow, J. and Kelemen, J., editors, *Aspects and Prospects of Theoretical Computer Science, 6th International Meeting of Young Computer Scientists, Smolenice, Czechoslovakia, November 19-23, 1990, Proceedings*, volume 464 of *LNCS*, pages 225–233. Springer.

Soare, R. I. (2007). Computability and incomputability. In Cooper, S. B., Löwe, B., and Sorbi, A., editors, *Computation and Logic in the Real World, Third Conference on Computability in Europe, CiE 2007, Siena, Italy, June 18-23, 2007, Proceedings*, volume 4497 of *LNCS*, pages 705–715. Springer.

von Henke, F. W., Rose, G., Indermark, K., and Weihrauch, K. (1975). On primitive recursive wordfunctions. *Computing*, 15(3):217–234.

Vučkovi, V. (1970). Recursive word-functions over infinite alphabets. *Mathematical Logic Quarterly*, 13(2):123–138.

LABORATOIRE D'INFORMATIQUE FONDAMENTALE D'ORLEANS