On the power of recursive word-functions without concatenation

Jérôme Durand-Lose

LABORATOIRE D'INFORMATIQUE FONDAMENTALE D'ORLEANS

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

L'UNIVERSITÉ N'EST PAS UNE ENTREPRISE
LE SAVOIR N'EST PAS UNE MARCHANDISE

August 30h, juillet 2022 — DCFS — Debrecen, Hungary

1. Introduction
2. Complexity
3. Computing without concatenation
4. Conclusion

LABORATOIRE D'INFORMATIQUE FONDAMENTALE D'ORLEANS

1 Introduction

2 Complexity

3 Computing without concatenation

4 Conclusion

Well-known: Classical recursion (on natural numbers)

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

- constant 0 function,
- successor function
- projections (π_n^i)
- 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

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

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

- constant $\widehat{\varepsilon}$,
- all left concatenation by one letter/symbol ${}_{a}\cdot(w) = a \cdot w = aw$
- projections (π_n^i)
- 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:

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

- (what minimisation to get all recursive functions?)

Observations

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

- everything matches

r-adic encoding function from Σ^* 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)

- 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; Santean, 1990; Calude and Sântean, 1990)
- the nowhere defined function is added to primitive recursive word-functions in Khachatryan (2015)

LABORATOIRE D'INFORMATIQUE FONDAMENTALE D'ORLEANS

1 Introduction

2 Complexity

3 Computing without concatenation

4 Conclusion

Complexity measure

Needed

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

Dynamical computation

- store every result of evaluation
- do not recompute

Delayed evaluation

- compute value when need
- call by name

Complexity classes

Simulation of a Turing machine

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

Class P is the same

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

Same for higher classes

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

1 Introduction

2 Complexity

3 Computing without concatenation

4 Conclusion

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 $\{\varepsilon, \mathrm{a}, \mathrm{aa}\}$

Language decision

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

LABORATOIRE D'INFORMATIQUE FONDAMENTALE D'ORLEANS

Multiple recursion

Multiple recursion

- usually done with pairing
- add operator: The $(k+1)$-ary functions

$$(f_i)_{1\leq i\leq m} = \mathbf{Rec}^m\left((g_i)_{1\leq i\leq m}, (h_{\mathtt{a},i})_{\mathtt{a}\in\Sigma,1\leq i\leq m}\right)$$

are uniquely defined by $\forall i, 1 \leq i \leq m$:

$$\begin{array}{rrcl} & f_i(\varepsilon,\vec{y}) & = & g_i(\vec{y}) \quad \text{and} \\ \forall \mathtt{a}\in\Sigma, & f_i(\mathtt{a}\cdot w,\vec{y}) & = & h_{\mathtt{a},i}(w, f_1(w,\vec{y}),\cdots,f_m(w,\vec{y}),\vec{y}) \end{array}$$

Regular languages

- decided with this extra operator
- scheme: one function for each state

Boolean operators — closure properties

- $\top$ identified with ε

Ternary operator / test function

- $\mathrm{if}_\varepsilon = \mathbf{Rec}\big(\pi^1_2, (\pi^4_4, \pi^4_4)\big)$

Conjunction and disjunction

- $\wedge$ is $\mathrm{and}_\varepsilon = \mathbf{Comp}\big(\mathrm{if}_\varepsilon, (\pi^1_2, \pi^2_2, \pi^1_2)\big)$
- $\vee$ is $\mathrm{or}_\varepsilon = \mathbf{Comp}\big(\mathrm{if}_\varepsilon, (\pi^1_2, \widehat{\varepsilon}, \pi^2_2)\big)$

Negation — non-ε argument is needed

- $\neg$ is $\mathbf{Comp}\big(\mathrm{if}_\varepsilon, (\pi^1_2, \pi^2_2, \widehat{\varepsilon})\big)$ — arity is 2

Equality test to palindrome decision

$$\mathbf{Comp}\left(\mathbf{Rec}\left(\pi_2^1 \;\middle|\; \begin{matrix}\mathbf{Comp}\left(\mathbf{Rec}\left(\mathrm{id} \;\middle|\; \begin{matrix}\pi_3^1\\ \pi_3^3\end{matrix}\right) \;\middle|\; \begin{matrix}\pi_4^2\\ \pi_4^4\end{matrix}\right)\\ \mathbf{Comp}\left(\mathbf{Rec}\left(\mathrm{id} \;\middle|\; \begin{matrix}\pi_3^3\\ \pi_3^1\end{matrix}\right) \;\middle|\; \begin{matrix}\pi_4^2\\ \pi_4^4\end{matrix}\right)\end{matrix}\right) \;\middle|\; \begin{matrix}\pi_2^1\\ \pi_2^2\\ \pi_2^1\end{matrix}\right)$$

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

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 the latter ones

- with prefixes and suffixes a_3^*

1 Introduction

2 Complexity

3 Computing without concatenation

4 Conclusion

Results

With concatenation

- computability identical
- complexity compatible (P and above)

Without concatenation

- decide all rational languages with multiple recursion
- decide languages with polynomial conditions on exponents / repetitions
  (unary encoding of natural numbers)

LABORATOIRE D'INFORMATIQUE LIFO FONDAMENTALE D'ORLEANS

Perspectives — concatenation-less

- Test identity
- Polynomials in many variables, negative coefficients
- Regular languages without multiple recursion
- All algebraic languages (deterministic, non-ambiguous)

- Condition for not computability/decision

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.

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

Santean, 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.