A remaining issue after Turing's work : formalize the notion of algorithm

Serge Grigorieff

LIAFA, CNRS & Université Denis Diderot-Paris 7

Colloquium Polaris, Lille, 31 mai 2012

(Année du centenaire de Turing)

Two questions by Turing
in his celebrated paper (1936)

1) What is an algorithm?

2) What is computable?

The successful answer to the second question
long overshadowed the first question.

Turing's First Question:
What is computable?

The *scope of the question* is in the title
of his momentous paper, 1936

On computable numbers, with an application
to the Entscheidungsproblem

(numbers = real numbers)

From the start,
Turing considers computability
over finite and infinite objects

Turing's First Question: What is computable?

"Although the subject of this paper
is ostensibly the computable [real] numbers,
it is almost equally easy to define and investigate
computable functions of an integral variable
or a real or computable variable.
[...] The fundamental problems involved are,
however, the same in each case,..."

Turing, 1936, page 1, line 3-7

$\boxed{\textit{On this point, Turing was wrong}}$

Computability over finite and over infinite objects
are NOT equally easy: For sequences of reals and
functions over reals, computability is representation
dependent (Mostowski, 1957). (cf. Computable Analysis)

Turing's First Question: What is computable?

"Let us suppose that we are supplied with
some unspecified means
of solving number theoretic problems;
a kind of oracle as it were. [...]
With the help of the oracle we could form a new
kind of machine (call them o-machine),[...]"

Turing's Thesis, 1938, page 18
"Systems of logic based on ordinals"

Turing 1938 introduces
Computability with an oracle
Computability as a relative notion

What has become of Turing 1st question?

$\boxed{\text{Computability on } \mathbb{N}}$

Herbrand-Gödel

Turing, Church,

Kleene,

. . . a cornucopia of computation models

$\boxed{\text{Computability on } \mathbb{R}}$

Turing,

Grzegorczyk, Lacombe,

Shannon's General Purpose Analogic Computer

Shannon takes a completely different approach:

(cf. Olivier Bournez and Daniel Graça's papers)

Turing's Second Question: What is an algorithm?

Turing's concern for the question

The real question at issue is "What are the possible processes which can be carried out in computing a [real] number?"

Turing, 1936 (page 20)

Donald Knuth's concern for the same question

The notion of an algorithm is basic to all of computer programming, so we should begin with a careful analysis of this concept.

(The Art of Computer Programming, vol. 1, p.1, §1.1 "Algorithms")

Let be naive for a while. . .

Non formal definitions of algorithms

What dictionaries tell about algorithms?

Antoine Furetière , *Dictionnaire universel* (1690)

algorisme. s.m. est un mot arabe dont plusieurs Auteurs se sont servis, & sur tout les Espagnols, pour signifier la science des nombres.

Dictionnaire de L'Académie française

Le mot *algorithme* **ne figure pas** dans
la 1ère édition (1694),
ni la 5ème (1798),
ni la 6ème (1835),
ni la 8ème (1935)

Figure dans la 4ème Édition (1762)
et la 9ème Édition (tome 1, 1994)

Dictionnaire de L'Académie française

4ème Éd. (1762)

ALGORITHME. s.m. Terme didactique. L'art de calculer.

L'Algorithme des entiers. L'Algorithme des fractions.

9ème Édition (tome 1, 1994)

ALGORITHME. n. m. XIIIe siècle, augorisme. Altération, sous l'influence du grec arithmos, ≪ nombre ≫, d'algorisme, qui, par l'espagnol, remonte à l'arabe Al-Khuwarizmi, surnom d'un mathématicien.

MATH. Méthode de calcul qui indique la démarche à suivre pour résoudre une série de problèmes équivalents en appliquant dans un ordre précis une suite finie de règles. L'algorithme de la multiplication de nombres à plusieurs chiffres.

Littré, *Dictionnaire de la langue française, 1872*

algorithme. al-go-ri-tm'/ s.m. 1. En termes d'algèbre, procédé de calcul.
2. Genre particulier de notations. Algorithme différentiel.
Hist. Cette senefiance est apelée algorisme de le quele nous usons de tels figures: 9, 8, 7, 6, 5, 4, 3, 2, 1... Quatre parties sont d'angorisme, assembler, abatre, dividier, multeplier. *Comput, XIIIe s. , manuscrit de la Bibliothèque Nationale.*
Arismetique est science de gecter et compter par le nombre de angorisme et autre nombre commun. *Eustache Deschamps, XVe s. , Art de faire chansons.*

Avecques eulx [les Vénitiens] leur duc serenissime, Qu'on peut juger un chiffre [zéro] en algorisme *Jean Marot, XVIe s. , in Oeuvres de Clément Marot*

Le Robert, *Dict. alphabétique et analogique de la langue française*

algorithme n.m. (du bas latin *algorismus*; arabe *al-khowarezmi*). Ensemble de symboles, de procédés de calcul. Algorithme d'Euclide

James & James, *Mathematics dictionary, 3rd ed. 1968*

al'go-rithm, n. Some special process of solving a certain type of problem, particularly a method that continually repeats some basic process. Division algorithm. Euclid's algorithm (...)

How some mathematicians define algorithms?

Marvin Minsky, *Finite and infinite machines,*

1967 (p.105)

The idea of an algorithm or effective procedure arises whenever we are presented with a set of instructions about how to behave. This happens when, in the course of working on a problem, we discover that a certain procedure, if properly carried out, will end up giving us the answer. Once we make such a discovery, the task of finding the solution is reduced from a matter of intellectual discovery to a mere matter of effort; of carrying out the discovered procedure – obeying the specified instructions.

Donald E. Knuth,

The art of computer programming, 1968

Knuth puts a temptative “axiomatic approach” where it has to be, at the start: volume 1, p.4

The modern meaning for algorithm is quite similar to that of recipe, process,method, technique, procedure, routine, except that the word "algorithm" connotes something just a little different. Besides merely being a finite set of rules which gives a sequence of operations for solving a specific type of problem,
an algoritm has five important features:

1) Finiteness. An algorithm must always terminate after a finite number of steps (...)
2) Definiteness. Each step of an algorithm must be precisely defined; the actions to be carried out must be rigorously and unambiguously specified for each case (...)
3) Input. An algorithm has zero or more inputs, i.e. quantities which are given to it initially before the algorithm begins. These inputs are taken from specified sets of objects (...)
4) Output. An algorithm has zero or more outputs, i.e. quantities which have a specified relation to the inputs (...).

5) Effectiveness. An algorithm is generally expected to be effective. This means that all operations to be performed in the algorithm must be sufficiently basic that they can in principle be done exactly and in a finite length of time by a man using pencil and paper (...)

But how does one tell, given what appears to be a set of instructions, that we really have been told exactly what to do ? How can we be sure that we can hencefoth effectively act, in accord with the "rules", without ever having to make any further choice or innovation of our own ?(...)
The position we will take is this : If the procedure can be carried out by some very simple machine, so that there can be no question of or need for "innovation or intelligence", then we can be sure that the specification is complete and that we have an "effective procedure".

We expect no quarrel with this.

Towards
formal definitions of algorithms

After Turing & al.'s work,
What is still missing?

Two aspects of algorithms

$$\begin{array}{rcl} \text{Denotational} & = & \textbf{What does it do?} \\ & = & \textbf{Input/Output behaviour} \end{array}$$

$$\begin{array}{rcl} \text{Operational} & = & \textbf{How does it do?} \end{array}$$

= **step by step behaviour + environment**

Many computation models

$\equiv$ **particular classes of algorithms**

Most proved to denotationally coincide

1936, Church thesis:

$\boxed{\textbf{They capture ALL computable functions}}$

= they are denotationally complete

Denotationally, OK… But Operationally?

Each one of these models captures ALL computable functions

but is there any model capturing ALL algorithms? ($\equiv$ operationally complete)

- **Until 1984, NONE proved to be so**
- **1984, Gurevich's Abstract State Machines**
- **2010, Lambda-calculus proved operationally complete** (Marie Ferbus & SG)

Resource complexity = operational feature
Complexity theory proves operational gaps in some computation models

Example: Palindrome recognition

- Quadratic time required with 1-tape Turing machines (Hennie, 1966)
- $O(n^2/\log(n))$ time required with multidimensional 1-tape Turing machines (Biedl & al., 2003)
- Linear time with two 1-dimensional tapes

Let us insist:
The theory of computable functions
IS NOT
a genuine theory of algorithms

Many textbooks intitled
"Theory of algorithms"

FAKE TITLE!

they are not about a genuine theory of algorithms:

- they explicit algorithms for some problems

- or develop the theory of computable functions. . .
(which is, by the way, the theory of partial computable functions)

Preliminary cautious question: Can there be a notion of algorithm?

WANT to catch ALL algorithms!

PROBLEM: there are

- "sequential" algorithms,
- parallel algorithms,
- interactive algorithms,
- distributed algorithms,
- analog algorithms,
- quantum algorithms,
- ...

An expanding notion!

a positive general solution seems desperate

Algorithms may have unspecified actions!!!

Does it contradict the idea of algorithm? No

- Unspecified choice $\equiv$ non determinism

  “To start, pick any vertex in the graph…”

- Unspecified management of recursive calls

  Fibonacci: $F_0 = F_1 = 1\ ,\ F_{n+2} = F_{n+1} + F_n$

  (bad implementation $\Rightarrow$ repeated computations
  $\Rightarrow$ Fibonaccian computation time)

Formalize algorithms $\Longrightarrow$

formalize wild unspecified stuff!!!

Let be less demanding:

WANTED:

A computation model catching ALL algorithms

- evolving in discrete sequential time
- with parallelism reduced to
  vector assignment of type 0 objects
- totally specified or “tame” unspecified part
(in particular, tell how recursive calls (if any) are managed)

In particular, we want to catch
ALL (non parallel) MACHINE MODELS

Capture ALL such algorithms?

Kolmogorov tried ca 1953:

Kolmogorov-Uspensky Machines

Schönhage tried ca 1970:

Storage Modification Machines

They failed but were rather close…

Abstract State Machines
aka Evolving algebras

Gurevich's formalization
of the notion of algorithm

Gurevich Abstract State Machines, 1984

Gurevich's Sequential Thesis

Every algorithm

- evolving in discrete sequential time
- with parallelism reduced to

  vector assignment of type 0 objects
- with tame or no non determinism

is matched step-by-step
by an Abstract State Machine

ASMs are operationally complete

Gurevich's Sequential Thesis

similar to Church-Turing Thesis

No formal proof is possible

But $\boxed{\text{the Thesis has been successfully tested}}$

with all existing computation models s.t.

- evolution in discrete sequential time
- strictly bounded parallelism
- tame or no non determinism

ASMs match step-by-step ALL sequential algorithms
What does mean "match step-by-step"?

Match = Simulation

Lockstep simulation: 1 step simulated by $\leq k$ ones (k is fixed)

Strict lockstep simulation: 1 step simulated by exactly k ones (k is fixed)

Fact. Abstract State Machines match by strict lockstep simulation with $k = 1$

(The other known operationally complete model, Λ-calculus matches by strict lockstep)

What is an Abstract State Machine?

The ASM concept is built from
Gurevich's fine analysis
of what is an algorithm

Takes features from Knuth's informal analysis
plus some more subtle ones

What is an Abstract State Machine?

- $\boxed{\text{MULTIDOMAIN}}$

  Finitely many sorts $M_1, \ldots, M_n$

  Contain the objects involved in the algorithm

  (Example: vector spaces involve 2 sorts: scalars + vectors)

- $\boxed{\text{STATIC PRIMITIVES}}$

  A first order structure on $(M_1, \ldots, M_n)$

  Equality on some sorts

  Finitely many functions $M_{i_1} \times \ldots \times M_{i_\ell} \to M_k$

  (predicates viewed as Boolean valued (sort BOOL) functions)

- $\boxed{\text{DYNAMIC VOCABULARY}}$

  Finite vocabulary of functions typed with the M_i's

- $\boxed{\text{ASM PROGRAM}}$

ASM: static multisort algebra

An algorithm relies on some
"primitive" operations (possibly quite complex)

$\boxed{\text{This "oracle" framework = static multisort algebra}}$

The NOTION OF ALGORITHM is
intrinsically ORACULAR

Though there is an absolute notion of computability
algorithms are intrinsically oracular!

$\boxed{\boxed{\text{This is one of Gurevich's crucial ideas}}}$

Premiss of this idea in Turing oracular computability

No ABSOLUTE notion of algorithm?

“Absolute algorithms”

$=$ “Effective algorithms”

$=$ algorithms with computable oracles (as their static part)

ASM: Why such a static part?

An algorithm decomposes its global input/output action into a succession of atomic steps.

Example. A single transition of a Turing machine involves the following atomic steps:

1) read the scanned cell,
2) overwrite the scanned cell,
3) move the head,
4) change state.

These atomic steps are for free in the TM
BUT when programming a TM in any language,
these are not for free: they require lines of code

ASM: static multisort algebra

An algorithm relies on some "primitive" operations (possibly quite complex)

ALGORITHMS INTRINSICALLY ORACULAR

$\boxed{\text{This "oracle" framework} = \text{static multisort algebra}}$

- Euclid's algorithm for the gcd relies on
- the $x = 0$ test on $\mathbb{N}$
- the remainder in Euclidean division on $\mathbb{N}$

Static part $=$ algebra $(\mathbb{N}; 0, \mathrm{mod})$

- Turing machine The tape is the sort $\mathbb{Z}$

Move head $=$ static operations $\begin{cases} x \mapsto x+1 \\ x \mapsto x-1 \end{cases}$

(there are other operations: cf. next slides)

ASM: dynamic vocabulary

An algorithm
modifies an environment

The dynamic vocabulary allows to
name this dynamic environment

Finitely many

- objects in the sorts
- functions over the sorts typed with the sorts

Abstract State Machines: ASM PROGRAM

The algorithms we consider
manipulate their environment
(named by the dynamic vocabulary)
in a very gentle way:

for some k, at each step,

- they bring at most k modifications
  to the current environment
- to do so, they query at most k values of
  the current environment

Abstract State Machines: ASM PROGRAM

The ASM program is built with:

- Conditional: `IF` u=v `THEN` instruction `ELSE` instr.
- Vector assignment: $\left\{\begin{array}{lcl} f_1(\vec{s}_1) & := & t_1 \\ & \vdots & \\ f_\ell(\vec{s}_\ell) & := & t_\ell \end{array}\right.$

$f_1, \ldots, f_\ell$ are dynamic functions

u, v, $\vec{s}_i$'s, t_i's: terms with static and dynamic funct.

The f_i's can occur in the terms $\vec{s}_j$'s, t_j's BUT

assignment such that $\boxed{(\text{NEW } f_i)(\text{OLD } \vec{s}_i) = \text{OLD } t_i}$

NO LOOP INSTRUCTION

SOLE (META) LOOP = repeated execution of the program

= ASM computation

Run an ASM $\equiv$ a functional

One step of a run of an ASM amounts to the update of the dynamic objects:

$$\begin{array}{rcl} (f_1, \ldots, f_\ell) & \longmapsto & (f'_1, \ldots, f'_\ell) \\ \Psi : T_1 \times \ldots \times T_\ell & \longrightarrow & T_1 \times \ldots \times T_\ell \end{array}$$

- T_i space of all functions $f_i : (\prod_{j \in J_i} M_j) \longrightarrow M_m$
  J_i finite, $\quad M_1, \ldots, M_n$ sorts
- f'_i coincides with f except on at most k points
- $\boxed{f'_i(\vec{x}) \text{ depends on at most } k \text{ values of the } f_j\text{'s}}$

Gurevich's Sequential Thesis

Gurevich's Sequential Thesis

Every algorithm

- evolving in discrete sequential time
- with parallelism reduced to
  vector assignment of type 0 objects
- with tame or no non determinism

is matched step-by-step
by an Abstract State Machine

ASMs are operationally complete

What about a 2d order Sequential Thesis?

Could the class of ASMs matching the algorithms of any given sequential computation model be of a remarkably "simple" form?

Proved true by Gurevich & al. for Schönhage Storage Modification Machines: they correspond to unary ASMs

Can we have this in full generality?

What about a 2d order Sequential Thesis?

Possible for computation models closed under constant speed-up

(SG & Pierre Valarcher, 2010)

One can view closure under constant speed-up as removing contingencies of the computation model while preserving the core paradigms

Constant speed-up and Time Unit

Any choice of a time unit is usually quite arguable

Turing machine: 1 transition = 1 step
but 1 transition can also can be viewed as 6 steps:

1) read the scanned cell,
2) decide how to overwrite its contents,
3) overwrite the scanned cell,
4) decide how to move the head,
5) move the head,
6) change state.

No *genuine* time unit   Up to a constant time unit

A 2d order Sequential Thesis

Computation models (closed on constant speed-up)
are obtained by fixing
the three first constituents of ASMs:

- the data structures,
- the static framework,
- the dynamic vocabulary

No constraint on the ASM program
except for type constraint in the construction of terms.
(SG & Pierre Valarcher, 2010)

No possible formal proof   Successfully tested
on "all" possible computation models

Turing machines with constant speed-up

Contingencies of Turing machines

The head	moves by one cell	Why not two, three?
	scans one cell	Why not a window?

Constant speed-up removes these contingencies

Core paradigm of Turing machines

- **Local work and local move**
- **Number of tapes and/or heads**

(this is some elementary parallelism)

Turing machines as a class of ASMs

Theorem. "Finite windows" TMs with one bi-infinite tape are literally identical to the ASMs s.t.

$\boxed{\text{Static multialgebra}}$

$$\langle(\mathbb{Z}; Succ, Pred), (\Sigma; a)_{a\in\Sigma}, (Q; q)_{q\in Q}, (S; s)_{s\in S}\rangle$$

where Q, Σ are infinite sets,

$S = \{\text{Go, HaltAccept, HaltReject}\}$

Dynamic vocabulary	$\pi{:}\mathbb{Z}$ position	$\sigma{:}Q$ state	$\eta{:}S$ status	$\omega{:}\mathbb{Z} \to \Sigma$ contents
Initial	$[\![\pi]\!], [\![\sigma]\!], [\![\eta]\!]$ defined, $[\![\omega]\!]$ total $(=$ input$)$			

$\boxed{\text{Program: No condition (but typing constraints)}}$

NB: an ASM program will involve finitely many states q's and symbols σ's

Turing machines / ASMs: sketch of proof

$\boxed{\textit{Litteral identity: static multialgebra}}$

Tape+moves	$\equiv$	algebra $(\mathbb{Z}; Succ, Pred)$
Alphabet	$\equiv$	algebra $(\Sigma; a)_{a\in\Sigma}$
States, Statuses	$\equiv$	algebras $(Q; q)_{q\in Q}$, $(S; s)_{s\in S}$

$\boxed{\textit{Litteral identity: dynamic vocabulary}}$

Current state, status, position, contents, window

$\equiv [\![\sigma]\!], [\![\eta]\!], [\![\pi]\!], [\![\omega]\!], \quad [\![\omega]\!] \restriction \{[\![\pi]\!] - k, \ldots, [\![\pi]\!] + k\}$

$\boxed{\textit{Litteral identity: functional}}$

ASM program definable+Typing $\Rightarrow$ ‖ Degenerated functional $\equiv$ *Trans. function*

$$\Sigma^{2k+1} \times Q \quad \to \quad \Sigma^{2k+1} \times Q \times S \times \{-\ell, \ldots, \ell\}$$

OLD window, σ NEW window, σ, η, move

CARE: benign component $\Rightarrow$ big effect

Add the 0 constant to the static multialgebra

$$\langle(\mathbb{Z}; 0, Succ, Pred), (\Sigma; a)_{a\in\Sigma}, (Q; q)_{q\in Q}, (S; s)_{s\in S}\rangle$$

Then any integer can be “named” by some term

Such ASMs characterize “reset” Turing machines

s.t.
- the head “knows” if it is on cell i
- the head can also jump to cell i

$(-\ell \leq i \leq \ell)$

$$\delta : \Sigma^{2k+1} \times Q \times (\{-\ell, \ldots, \ell\} \cup \{outside\}) \\ \longrightarrow \Sigma^{2k+1} \times Q \times S \times \{-\ell, \ldots, \ell\} \times \{\alpha, \beta\}$$

$\delta(\ldots) = (\ldots, i, \alpha)$ means $NEW[\![\pi]\!] = i$

$\delta(\ldots) = (\ldots, i, \beta)$ means $NEW[\![\pi]\!] = OLD[\![\pi]\!] + i$

RAMs with constant speed-up

Contingencies of Random Access Machines

- Only one register can be accessed at each step,
- no iterated indirect access in a single step
- particular set of instructions for RAM programs

Constant speed-up removes these contingencies

Core paradigms of Random Access Machines

- **Registers containing arbitrarily large integers**
- **Indirect access**
- **Set of operations on register contents**

RAMs as a class of ASMs

Theorem. "Finitely iterative transition" RAMs are literally identical to the ASMs such that

Static multialgebra	(Q, S idem Turing machines)

$$\langle(\mathbb{N}, 0, 1, +); \mathbb{N}^{address}, (Q; q)_{q\in Q}, (S; s)_{s\in S}), \text{cast}\rangle$$

- $\mathbb{N}^{address}$ is a copy of $\mathbb{N}$ with no structure,
- $cast : \mathbb{N} \to \mathbb{N}^{\text{address}}$ is identity as cast function

Dynamic vocabulary	$\sigma{:}Q$	$\eta{:}S$	$\omega{:}\mathbb{N}^{\text{address}} \to \mathbb{N}$
	state	status	contents
Initial	$[\![\sigma]\!], [\![\eta]\!]$ defined, $[\![\omega]\!]$ total ($=$ input)		

Program: No condition (but typing constraints)

RAMs / ASMs: sketch of proof

$\boxed{\textit{Litteral identity: static multialgebra}}$

States, Statuses	$\equiv$	algebras $(Q;q)_{q\in Q}$, $(S,s)_{s\in S}$
Registers	$\equiv$	$\mathbb{N}^{address}$
Contents+operations	$\equiv$	algebra $(\mathbb{N},0,1,+)$
Indirect access	$\equiv$	$cast : \mathbb{N} \to \mathbb{N}^{\text{address}}$

$\boxed{\textit{Litteral identity: dynamic vocabulary}}$

Current state, status, reg. contents $\equiv [\![\sigma]\!], [\![\eta]\!], [\![\omega]\!]$

$\boxed{\textit{Litteral identity: functional}}$

Quantifier-free definable+Typing $\Rightarrow$

$$Q \times \mathbb{N}^{k(1+\ell)} \to Q \times S \times \mathbb{N}^{k(1+\ell)}$$

OLD NEW

σ, η and accessed registers contents

k first registers $+$ ℓ iterations of indirect access

$\boxed{\begin{array}{c}\text{Degenerated}\\ \text{functional}\\ \equiv\\ \textit{Trans. funct.}\end{array}}$

It works for

- Automata
- Stack automata
- Schönhage machines
  (with an extended typing: CORRELATION)
- ...

CORRELATION

DO NOT identify

$\alpha = (\alpha, \beta){:}X \to Y \times Z$ (CORRELATED functions)

and (= simultaneous functions)

$(\alpha{:}X \to Y, \beta{:}X \to Z)$

CORRELATION IS A CONSTRAINT

- When firing $(\alpha, \beta){:}X \to Y \times Z$
  α and β are fired on the same arguments
- There are less possible terms

Schönhage machines: contingencies versus core ideas

SMM = Turing but the tape is a dynamic oriented graph with bounded out-degree

(intuition: arcs = pointers. Few pointers out of a node but arbitrarily many can point to it)

Contingencies of Schönhage SMMs

- One modification per step of the graph of pointers

  Why not 2, 3,. . . ?
- Particular set of instructions for SMM programs

Core paradigm of Schönhage SMMs

- **graph of pointers + local action**

Schönhage machines as a class of ASMs

Theorem. SMMs are literally identical to ASMs s.t.

$\boxed{\text{Static multialgebra}}$ (Q, S idem Turing machines)

$$\langle X, \mathcal{P}_{fin}(X), new : \mathcal{P}_{fin}(X) \to X \times \mathcal{P}_{fin}(X),$$
$$(\Sigma^*; Last, Pred, a)_{a\in\Sigma}, (Q; q)_{q\in Q}, (S, s)_{s\in S})\rangle$$

X infinite set of nodes Σ input/output alphabet

CORRELATED new$(A) = (a, A \cup \{a\})$ **with** $a \notin A$

Dynamic vocab.	$\sigma{:}Q, \eta{:}S$	$\omega{:}\Sigma^*$	$U{:}\mathcal{P}_{fin}(X)$	$\nu{:}X$
		unread	used nodes	center
	pointers $f_i : X \to X$ partial function			
Initial	$[\![\sigma]\!], [\![\eta]\!], [\![\nu]\!], [\![\omega]\!]$ defined ($[\![\omega]\!] =$ input)			

$$[\![U]\!] = dom([\![f_i]\!]) = \{[\![\nu]\!]\} \quad [\![f_i]\!]([\![\nu]\!]) = [\![\nu]\!]$$

$\boxed{\text{Program: No condition (but typing constraints)}}$

Schönhage machines / ASMs: sketch of proof

$\boxed{\textit{Litteral identity: static multialgebra}}$

Nodes, States, Statuses $\equiv$ $\mathbb{N}, (Q; q)_{q \in Q}, (S, s)_{s \in S}$

Input set, read, move $\equiv$ $(\Sigma^*; Last, Pred, a)_{a \in \Sigma}$

$\boxed{\textit{Litteral identity: dynamic vocabulary}}$

Current state, status, pointers, center node, input

$$\equiv [\![\sigma]\!], [\![\eta]\!], ([\![f_i]\!])_{i=1,\ldots,k}, [\![\nu]\!], [\![\omega]\!]$$

$\boxed{\textit{Litteral identity: functional}}$

$\boxed{\text{Degenerated functional} \equiv \text{Trans. funct.}}$

Quantifier-free definable+Typing $\Rightarrow$

$$Q \times \mathbb{N}^{\{1,\ldots,k\}^{\leq \ell}} \times \Sigma \cup \{\varepsilon\} \longrightarrow$$

$$Q \times S \times \mathbb{N}^{\{1,\ldots,k\}^{\leq \ell}} \times \Sigma \cup \{\varepsilon\} \times \{1, \ldots, k\}^{\leq \ell}$$

OLD state, pointers around center node, input letter

NEW $[\![\sigma]\!]$, $[\![\eta]\!]$, pointers, c. node, output, path move

Characterizing the dynamic of algorithms

From an ASM functional to $\mathbb{N}^{\mathbb{N}} \to \mathbb{N}$

- $T_i \;=\; (\prod_{j\in J_i} M_j) \longrightarrow M_m \quad\equiv\quad \mathbb{N}^{\mathbb{N}}$ or $\mathbb{N}$
- The ASM functional space
  (evolution of dynamic functions)
- $T_1 \times \ldots \times T_\ell \longrightarrow T_1 \times \ldots \times T_\ell$

$$\begin{aligned}
&\equiv\; (\mathbb{N}^{\mathbb{N}})^{\ell'} \times \mathbb{N}^{\ell''} \to (\mathbb{N}^{\mathbb{N}})^{\ell'} \times \mathbb{N}^{\ell''}\\
&\equiv\; \mathbb{N}^{\mathbb{N}} \to \mathbb{N}^{\mathbb{N}}\\
&\equiv\; (\mathbb{N}^{\mathbb{N}}) \times \mathbb{N} \to \mathbb{N}\\
&\equiv\; \mathbb{N}^{\mathbb{N}\cup\{\top\}} \to \mathbb{N}\\
&\equiv\; \mathbb{N}^{\mathbb{N}} \to \mathbb{N}
\end{aligned}$$

Is there a topological characterization
of ASM functionals?

k-bounded continuity

Let $\Phi : \mathbb{N}^{\mathbb{N}} \to \mathbb{N}$ be continuous

Then every $f \in \mathbb{N}^{\mathbb{N}}$ is in some clopen

$$[u] = \{g \in \mathbb{N}^{\mathbb{N}} \mid g \textit{ extends } u\}$$

- $u : X \to \mathbb{N}$ where X is finite
- Φ is constant on the clopen set $[u]$

k-continuity $=$ $\Big|$ *THE SIZE OF X IS BOUNDED by k*

Example where the size of X is 3-bounded

$$\Phi : \mathbb{N}^{\mathbb{N}} \to \mathbb{N} \qquad \Phi(f) = f(\,\alpha(f(0), f(1))\,)$$

To compute $\Phi(f)$ we only need 3 values of f,
namely those at $\boxed{0, \quad 1, \quad \alpha(f(0), f(1))}$
Any $f \in \mathbb{N}^{\mathbb{N}}$ is in some

$$[u] = \{g \in \mathbb{N}^{\mathbb{N}} \mid g \textit{ extends } u\}$$

- $u : X \to \mathbb{N}$ where X has ≤ 3 elements
- Φ is constant on the clopen set $[u]$

The sets $[\{(0,\ x),\ (1,\ y),\ (\alpha(x, y),\ z)\}],\ \ x, y, z \in \mathbb{N}$
partition $\mathbb{N}^{\mathbb{N}}$ in clopen sets on which Φ is constant
We show **This is the sole kind of example**

Variations around continuity of $\Phi : \mathbb{N}^{\mathbb{N}} \to \mathbb{N}$

$[u] = \{f \in \mathbb{N}^{\mathbb{N}} \mid f \textit{ extends } u\}$ $\qquad u : X \to \mathbb{N}$

(1_k) k-continuous functional $\qquad \exists\ (\Phi, k)$-covering

$$\mathbb{N}^{\mathbb{N}} = \bigcup_{i \in \mathbb{N}} [u_i] \quad \text{where} \quad \begin{cases} |dom(u_i)| \leq k \\ \Phi \textit{ constant on } [u_i] \end{cases}$$

(2_k) unambiguous k-continuous functional

$\equiv \quad (1_k) +$ the $[u_i]$'s are pairwise disjoint

Does this help to compute Φ ? $\qquad$ A priori, NO

(3_k) Non deterministic k-query functional

Exists non det. algo. with oracle $\omega : \mathbb{N} \to \mathbb{N}$
querying $\leq k$ values of f to compute $\Phi(f)$

$(3_k) \Leftrightarrow (1_k)$ straightforward $\qquad (2_k) \Rightarrow (1_k)$ trivial

Variations around continuity of $\Phi : \mathbb{N}^{\mathbb{N}} \to \mathbb{N}$

$[u] = \{f \in \mathbb{N}^{\mathbb{N}} \mid f \textit{ extends } u\}$ $\qquad u : X \to \mathbb{N}$

(3_k) Non deterministic k-query functional
Exists non det. algo. with oracle $\omega : \mathbb{N} \to \mathbb{N}$
querying k values of f to compute $\Phi(f)$

(4_k) Deterministic k-query functional
$\equiv \quad (3_k)$ with a deterministic algorithm

(5_k) k-(term query) functional
$\equiv \ (4_k)$ where the k queries are done via terms

$$\begin{aligned}\Phi(f) \ = \ & \alpha(f(a_1), \ldots, f(a_m), \\ & \quad f(\beta_1(f(\vec{a})), \ldots, \beta_n(f(\vec{a}))), \ldots)\end{aligned}$$

(means Φ comes from some Abstract State Machine)

$(5_k) \Rightarrow (4_k) \Rightarrow (3_k)$ trivial

Main results

(1_k) k-continuous functional $\Phi : \mathbb{N}^{\mathbb{N}} \to \mathbb{N}$

$$\mathbb{N}^{\mathbb{N}} = \bigcup_{i\in\mathbb{N}}[u_i] \quad \text{where} \quad \begin{cases} |dom(u_i)| \leq k \\ \Phi \text{ constant on } [u_i] \end{cases}$$

(5_k) k-(term query) functional

$\equiv$ (4_k) where the k queries are done via terms

$$\Phi(f) = \alpha(f(a_1), \ldots, f(a_m), f(\beta_1(f(\vec{a}))), \ldots, \beta_n(f(\vec{a}))), \ldots)$$

$\alpha, \beta_1, \ldots, \beta_n, \ldots$ fixed oracles

Theorem (SG & Pierre Valarcher, 2012).

- $(1_k) \Rightarrow (5_{k^2})$ k^2 is optimal
- $(1_k^{effective}) \Rightarrow (5_{2k^2}^{effective})$

Topological interpretation

k-continuity looks like uniform continuity

$\boxed{\text{Bounded Uniformity on } \mathbb{N}^{\mathbb{N}}}$

Base of entourages: the $\bigcup_{i \in \mathbb{N}} [u_i] \times [u_i]$
all unambiguous k-coverings $(u_i)_{i \in \mathbb{N}}$, all k
($\equiv$ all deterministic term-query k-coverings)

- The associated topology on $\mathbb{N}^{\mathbb{N}}$ is the Baire one
- The Bounded Uniformity is transitive,
  refines the uniformity of the usual Baire distance
- and does not come from a metric

Theorem. $\Phi : \mathbb{N}^{\mathbb{N}} \to \mathbb{N}$ is k-continuous for some k
$\iff$ Φ is uniformly continuous
(wrt bounded uniformity on $\mathbb{N}^{\mathbb{N}}$, discrete one on $\mathbb{N}$)

Application: the dynamic of algorithms

Assuming Gurevich Sequential Thesis

Functionals $\Psi : (f_1, \ldots, f_\ell) \longmapsto (f'_1, \ldots, f'_\ell)$ giving the evolution of the environment of algorithms

- evolving in discrete sequential time
- with parallelism reduced to

  vector assignment of type 0 objects
- telling how recursive calls (if any) are managed

$$\equiv$$

Functionals Ψ such that, for some k,

(1) $\boxed{\Psi \text{ is } k\text{-continuous}}$ (2) f'_i coincides with f_i except on a set $X_i(\vec{f})$ of at most k points

$\boxed{(3)\ \vec{f} \longmapsto X_i(\vec{f}) \text{ is } k\text{-continuous}}$

Thank you for your attention