Contrat Doctoral — ED Galilée

Titre du sujet : Algorithms, data, and logic

➢ Unité de recherche : LIPN, UMR7030

➢ Discipline : Informatique

➢ Direction de thèse : Thomas Seiller (CR CNRS, LIPN, 80%), Alberto Naibo (MCF, IHPST, 20%)

➢ Contact : thomas.seiller@cnrs.fr

➢ Domaine de recherche : Computability, Logic

➢ Mots-clés : Algorithms, Data structures, Logic

Description

This projects takes its foundation within the "Mathematical Informatics" framework introduced by T. Seiller [16], which is based on a clear and precise distinction between three notions that are frequently wrongly coalesced, namely that of *computation*, *programs*, and *algorithms*. Computation denotes here the physical process of computation, whether it is e.g. mechanical, electronic, quantum. A computation takes place each time a computer program is run. It is bound to a physical theory ; it is also deterministic by nature. Conceived as a physical process a computation is very similar to the notion of controlled experiment in physics [2]. This is the starting point of an analysis justifying the use of a mathematical model for studying computation similar to what is used in mathematical physics : dynamical systems [1] [5, 18].

A program is then bound to a *model of computation* which abstracts the physical devices on which computations are performed : abstraction here allows one to (separate) from the worldly constraints such as finiteness of the computing devices. Models of computations are abstractly as monoid actions [9], which mathematically formalises how the *instructions* (generating the monoid) act on the set of configurations (the space acted upon). A program then describes a family of possible computations based on these specific instructions, and is naturally formalised as an α-graphing [2], a generalisation of dynamical systems that allows for e.g. non-determinism, probabilities, while keeping a finite description as a generalised graph.

An example. We here illustrate how monoid actions and graphings formalise the notions of model of computation and program. We consider Turing machines, and represent it mathematically as the following monoid action. We consider the space of configurations $X = \{\star, 0, 1\}^{|\mathbb{Z}|}$ of $\mathbb{Z}$-indexed sequences of symbols $\star, 0, 1$ that are almost always equal to $\star$, i.e. an infinite tape with only a finite number of symbols 0 and 1 written on it. A given point in this configuration space, extended with a chosen *control state*, describes a configuration of a Turing machine. Now, instructions allowed in the model are represented as maps from X to X as follows : for instance moving the working head to the right can be represented as $\texttt{right} : X \to X$, $(a_i)_{i\in\mathbf{Z}} \mapsto (a_{i+1})_{i\in\mathbf{Z}}$. The set of instructions then generates a monoid action $M \curvearrowright X$. A graphing is then a collection of *edges* consisting of a source (a subspace of X) and a realiser (an element of the monoid M). The instruction "if in control state a and the head is reading a 0 or a 1, move to the right and move to control state b" is represented as an edge of source the subspace $\{(a_i)_{i\in\mathbf{Z}} \in X \mid a_0 \neq \star\} \times \{a\}$ and realised by the map $\texttt{right} \times (a \mapsto b)$. A Turing machine is then a partial dynamical system $f : X \hookrightarrow X$ in the *full group* of

1. Physicists describe dynamical systems by partial differential equations (PDEs) derived from general theories. Dynamical systems and PDEs are two sides of the same coin, but we do not know a generalisation of PDEs comparing in scope with the generalisation of dynamical systems offered by graphings.
2. The notion first appeared in ergodic theory [1, 3, 4] and introduced in computer science by Seiller [10, 13, 11, 12, 14, 15, 17].

the monoid action[3], i.e. whose graph is contained in the preorder $\mathcal{P}(\alpha) = \{(x, y) \mid \exists m \in M, m \cdot x = y\}$.

These formalised notions of models of computation and programs then lead to a formal definition of *algorithms*. This notion is defined by means of *abstract data structures* which describe the mathematical basis on which the algorithm act. A typical example of an abstract data structure is that of *unary integers* defined as the set of natural numbers $\mathbf{N}$ together with the following *structural maps* : the successor $S : n \mapsto n+1$ and tests to zero defined as partial maps $\text{iszero} : 0 \mapsto 0$ (undefined otherwise) and $\text{isnonzero} : x \mapsto x$ (undefined at $x = 0$). More complex data structures can be considered, such as binary lists (singly chained, or doubly chained), pairs of a binary integer and a boolean, etc. An algorithm is then defined as a directed graph with labelled edges, whose labels are mapped to « structural maps » (those maps defined by the abstract data structure). One can then define programs that *implement* the algorithm as those that are equal to a glueing of programs implementing the structural maps along the algorithm A (while the definition is technical, we illustrate the operation in Figure 1).

The proposed definitions of algorithm is the starting point of the thesis topic, which will exploit those in several directions.

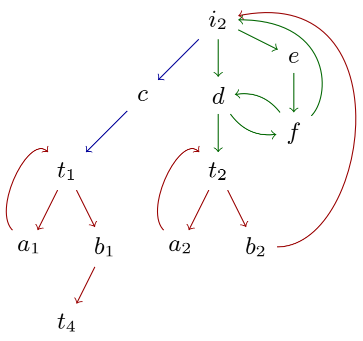
Objectives.

1. **Algorithms : a quest for absolute definitions.** There are two main other proposals for defining algorithms : Gurevich's abstract state machines [7] and Moschovakis' theory of recursors [8]. This new approach should be properly compared to those, both technically and conceptually. Moreover, it should be understood if the notion defined captures the notion of algorithm from computer science or that of mathematics ; unless both notions coincide ? In fact two slightly different notions of algorithms can be defined : the one described above is bound to a specific data domain (e.g. the integers), while the other one [16] is based on a notion of *logical* data structure which describes data structures as models of a first order theory : it is then possible to write down a single algorithm accounting for the calculation of the gcd on integers, polynomials, or any euclidean ring. The distinction between those two notions may shed light on this question. These investigations include philosophical matters which will be explored in collaboration with A. Naibo.
2. **Abstract data structures and logic.**
   (a) Among the models of computation represented in the above framework, one can find functional models. Those can be characterised as obtained by lifting an initial action $\alpha : \text{Mon}(I) \curvearrowright \mathbf{X}$ to an action $\bar{\alpha} : \text{Mon}(I) \curvearrowright \text{Programs}(\alpha)$ on the set of α-programs[4]. As part of the proofs-as-programs correspondence, it is known that some formulas represent data. For instance, the type $\forall X, (X \Rightarrow X) \Rightarrow ((X \Rightarrow X) \Rightarrow (X \Rightarrow X))$ defines binary lists [6]. This leads to the following questions. Does every logical formula give rise to an abstract data structure ? If not, is it possible to characterise those that do ? Conversely : can every abstract data structure be obtained from a logical formula and, if not, is it possible to describe those that can ? This last question should have connection with descriptive complexity, where specific data structures (related to the formulas / logic considered) are used in the proofs of characterisations of complexity classes.

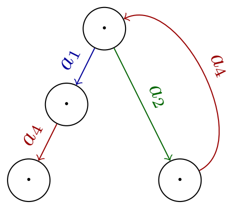
   (b) The question of approximation is also important when one departs from the usual discrete data structures. For instance, different models of computation on the reals use sometimes incompatible representations of real numbers. Roughly, when the Blum-Shub-Smale (BSS) model considers reals as given from the definition of the model (and can therefore compute with non-computable

3. The name is chosen because it generalises the usual notion of full group for measure-preserving (countable) group actions ; in this specific case, the preorder $\mathcal{P}(\alpha)$ is a *Borel equivalence relation.*

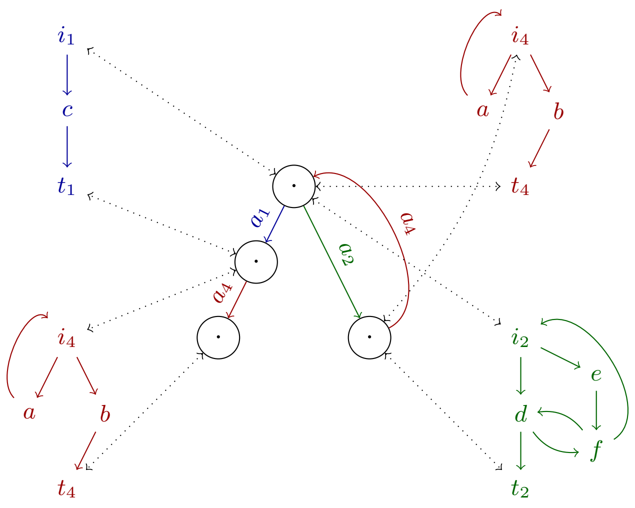
4. An immediate consequence is that the set of $\bar{\alpha}$-programs is in bijection with the set of α-programs, which implies the usual conflation of programs with data in functional programming.



(a) A program M : bound to a model of computation $\alpha : \mathrm{Mon}(I) \curvearrowright \mathbf{X}$, its edges are elements of $\mathrm{Mon}(I)$.



(b) An algorithm A : it is a labelled graph, with labels associated to structural maps



(c) The program M can seen as a glueing along an algorithm A when each label in A can be ≪ replaced ≫ by a program implementing the corresponding structural map

FIGURE 1 – Example of glueing

reals), computable analysis considers real numbers to be given as the output of a Turing machine. Algorithm computing on the reals, such as root-finding algorithms, can thus be considered in a *idealised* version (in the BSS model) or in a more realistic model based on approximations. This notion of approximation should be found and will be studied at the level of abstract data structures.

3. **Algorithms, metrics, and approximations.**
   (a) Having a formal, mathematical, notion of algorithms can be exploited to define a notion of distance between those. More importantly, the candidate will work on defining a notion of approximate implementation capturing the fact that a program *almost* (i.e. up to some small $\epsilon > 0$) implements a given algorithm. This notion is essential as it can be used to define a notion of convergence and formally establish that a sequence of programs *converges* to an implementation of some algorithm.
   (b) This theoretical investigation will be coupled with some small experiments to understand if this notion of convergence can be witnessed in simple exemples of learning. In short : if one trains a

neural network to perform a simple task, such as multiplying integers, is it possible to define/find an algorithm towards which the sequence of programs considered in the training process ? While some similar experiments were already performed, they only distinguished algorithms through their complexity which I believe is based on incorrect assumptions [5].

5. My assumption is that an algorithm is not bound to some complexity : only programs are. In particular, different implementations of the same algorithm may have different complexities when implemented by different programs (possibly in different models of computation).

Références

[1] S. Adams. Trees and amenable equivalence relations. *Ergodic Theory and Dynamical Systems*, 10 :1–14, 1990.

[2] G. Dowek. The physical church thesis as an explanation of the galileo thesis. *Natural Computing*, 11(2) :247–251, 2012.

[3] D. Gaboriau. Coût des relations d'équivalence et des groupes. *Inventiones Mathematicae*, 139 :41–98, 2000.

[4] D. Gaboriau. Invariants ℓ^2 de relations d'équivalence et de groupes. *Publ. Math. Inst. Hautes Études Sci*, 95(93-150) :15–28, 2002.

[5] R. Gandy. Church's thesis and principles for mechanisms. In J. Barwise, H. J. Keisler, and K. Kunen, editors, *The Kleene Symposium*, volume 101 of *Studies in Logic and the Foundations of Mathematics*, pages 123–148. Elsevier, 1980.

[6] J.-Y. Girard, Y. Lafont, and P. Taylor. *Proofs and types.* CUP, 1989.

[7] Y. Gurevich. *What Is an Algorithm ?*, pages 31–42. Springer Berlin Heidelberg, 2012.

[8] Y. Moschovakis. What is an algorithm ? In *Mathematics Unlimited — 2001 and beyond.* 2001.

[9] L. Pellissier and T. Seiller. Semantics, entropy and complexity lower bounds. Research Report, 2018.

[10] T. Seiller. Interaction graphs : Full linear logic. In *IEEE/ACM Logic in Computer Science (LICS)*, 2016.

[11] T. Seiller. Interaction graphs : Graphings. *Annals of Pure and Applied Logic*, 168(2) :278–320, 2017.

[12] T. Seiller. Interaction graphs : Nondeterministic automata. *ACM Transaction in Computational Logic*, 19(3), 2018.

[13] T. Seiller. Interaction graphs : Exponentials. *Logical Methods in Computer Science*, 15(3), 2019.

[14] T. Seiller. Probabilistic complexity classes through semantics. *CoRR*, abs/2002.00009, 2020.

[15] T. Seiller. Zeta functions and the (linear) logic of markov processes. Under revision for publication in Logical Methods in Computer Science, `https://hal.archives-ouvertes.fr/hal-02458330`, 2022.

[16] T. Seiller. Mathematical informatics, 2024. Habilitation thesis (`https://www.seiller.org/HdR.pdf`).

[17] T. Seiller, L. Pellissier, and U. Léchine. Unifying lower bounds for algebraic machines, semantically. Under revision for publication in Information and Computation, `https://hal.archives-ouvertes.fr/hal-01921942`, 2022.

[18] W. Sieg. On computability. *Philosophy of mathematics*, 4 :549–630, 2009.