Computation, information and the arrow of time

Pieter Adriaans[1,2] and Peter van Emde Boas[3,4]

[1] Adriaans ADZA Beheer B.V.
[2] FNWI
University of Amsterdam,
Science Park 107
1098 XG Amsterdam,
The Netherlands.
[3] Bronstee.com B.V., Heemstede
[4] ILLC, FNWI, University of Amsterdam
Postbus 94242,
1090 GE Amsterdam
pieter@pieter-adriaans.com; peter@bronstee.com

Abstract. In this paper we investigate the relation between information and computation under time symmetry. We show that there is a class of non-deterministic automata, the *quasi-reversible automata* (QRTM), that is the class of classical deterministic Turing machines operating in negative time, and that computes all the languages in NP. The class QRTM is isomorphic to the class of standard deterministic Turing machines TM, in the sense that for every $M \in TM$ there is a M^{-1} in QRTM such that each computation on M is mirrored by a computation on M^{-1} with the arrow of time reversed. This suggests that non-deterministic computing might be more aptly described as deterministic computing in negative time. If M_i is deterministic then M_i^{-1} is non deterministic. If M is information discarding then M^{-1} 'creates' information. The two fundamental complexities involved in a deterministic computation are Program Complexity and Program Counter Complexity. Programs can be classified in terms of their 'information signature' with pure counting programs and pure information discarding programs as two ends of the spectrum. The paper provides a formal basis for a further analysis of such diverse domains as learning, creative processes, growth and the study of the interaction between computational processes and thermodynamics.

1 Introduction

The motivation behind this research is expressed in a childhood memory of one of the authors: "When I was a toddler my father was an enthusiastic 8-mm movie amateur. The events captured in these movies belong to my most vivid memories. One of the things that fascinated me utterly was the fact that you could reverse the time. In my favorite movie I was eating a plate of French fries. When played forward one saw the fries vanish in my mouth one by one, but when played backward a miracle happened. Like a magician pulling a rabbit out

of a hat I was pulling undamaged fries out of my mouth. The destruction of fries in positive time was associated with the creation of fries in negative time."

This is a nice example of the kind of models we have been discussing when we were working on the research for this paper. It deals with computation and the growth and destruction of information. Deterministic computation seems to be incapable of creating new information. In fact most recursive functions are non-reversible. They discard information. If one makes a calculation like $a+b=c$ then the input contains roughly $(\log a + \log b)$ bits of information whereas the answer contains $\log(a+b)$ bits which is in general much less. Somewhere in the process of transforming the input to the output we have lost bits. The amount of information we have lost is exactly the information needed to separate c in to a and b. There are many ways to select two numbers a and b that add up to c. So there are many inputs that could create the output. The information about the exact history of the computation is discarded by the algorithm. This leaves us with an interesting question: *if there is so much information in the world and computation does not generate information, then, where does the information come from?*

Things get more fascinating if we consider the Turing machine version of the French fries example above. Suppose we make a Turing machine that only erases its input and we make a movie of its execution and play it backwards. What would we see? We see a machine creating information out of nothing, just the same way the toddler in the reversed movie was pulling neat French fries out of his mouth. So also in this case, if we reverse the arrow of time, destruction of information becomes creation and vice versa. In previous papers the first author has investigated the relation between learning and data compression ([4; 2]). Here we are interested in the converse problem: how do data-sets from which we can learn something emerge in the world? What processes grow information?

There is a class of deterministic processes that discard or destroy information. Examples are: simple erasure of bits, (lossy) data compression and learning. There is another class of processes that seem to create information: coin-flipping, growth, evolution. In general, stochastic processes create information, exactly because we are uncertain of their future, and deterministic processes discard information, precisely because the future of the process is known. The basic paradigm of a stochastic information generating process is coin flipping. If we flip a coin in such a way that the probability of head is equal to the probability of tails, and we note the results as a binary string, then with high probability this string is random and incompressible. The string will then have maximal Kolmogorov complexity, i.e. a program that generates the string on a computer will be at least as long as the string itself ([8]). On the other hand if we generate a string by means of a simple deterministic program (say "`For x = 1 to k print("1")`") then the string is highly compressible and by definition has a low Kolmogorov complexity which approximates $\log k$ for large enough k. In the light of these observations one could formulate the following research question: *given the fact that creation and destruction of information seem to be symmetrical over the time axis, could one develop a time-invariant description of computational*

processes for which creation of information is the same process as destruction of information with the time arrow reversed? A more concise version of the same question is: *are destruction and creation of information computationally symmetrical in time?* The main part of this paper is dedicated to a positive answer to this question.

Prima facie it seems that we compute to get new information. So if we want to know what the exact value of 10! is, then answer 3628800 really contains information for us. It tells us something we did not know. We also have the intuition, that the harder it is to compute a function, the more value (i.e. information) the answer contains. So 10! in a way contains more information than 10^2. Yet from a mathematical point of view 10! and 3628800 are just different descriptions of the same number. The situation becomes even more intriguing if we turn our intention to the simulation of processes on a computer that really seem to create new information like the growth of a tree, game playing or the execution of a genetic algorithm. What is happening here if computation cannot generate information? What is the exact relation between information generating processes that we find in our world and our abstract models of computation?

In most curricula theories about information and computation are treated in isolation. That is probably the reason why the rather fundamental question studied in this paper up till now has received little attention in computer science: *What is the interaction between information and computation?* Samson Abramsky has posed this question in a recent publication with some urgency (without offering a definitive answer): *We compute in order to gain information, but how is this possible logically or thermodynamically? How can it be reconciled with the point of view of Information Theory? How does information increase appear in the various extant theories?* ([1]), pg. 487). Below we will formulate a partial answer to this question by means of an analysis of time invariant descriptions of computational processes.

2 A formal framework: meta-computational space

In order to study the interplay between entropy, information and computation we need to develop a formal framework. For this purpose we develop the notion of meta-computational space in this section: formally the space of the graphs of all possible computations of all possible Turing machines. The physical equivalent would be the space of all possible histories of all possible universes.

$C(x)$ will be the classical Kolmogorov complexity of a binary string x, i.e. the length of the shortest program p that computes x on a reference universal Turing machine U. Given the correspondence between natural numbers and binary strings, $\mathcal{M}$ consists of an enumeration of all possible self-delimiting programs for a preselected arbitrary universal Turing machine U. Let x be an arbitrary bit string. The shortest program that produces x on U is $x^* = argmin_{M \in \mathcal{M}}(U(M) = x)$ and the Kolmogorov complexity of x is $C(x) = |x^*|$. The conditional Kolmogorov complexity of a string x given a string y is $C(x|y)$, this can be interpreted as the length of a program for x given input y. A string is defined to

be *random* if $C(x) \geq |x|$. $I(x)$ is the classical integer complexity function that assigns to each integer x another integer $C(x)$[8].

We will follow the standard textbook of Hopcroft, Motwani and Ullman for the basic definitions ([7]). A *Turing machine* (TM) is described by a 7-tuple

$$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$$

Here, as usual, Q is the finite set of states, Σ is the finite set of input symbols with $\Sigma \subset \Gamma$, where Γ is the complete set of tape symbols, δ is a transition function such that $\delta(q, X) = (p, Y, D)$, if it is defined, where $p \in Q$ is the next state, $X \in \Gamma$ is the symbol read in the cell being scanned, $Y \in \Gamma$ is the symbol written in the cell being scanned, $D \in \{L, R\}$ is the direction of the move, either left or right, $q_0 \in Q$ is the start state, $B \in \Gamma - \Sigma$ is the blank default symbol on the tape and $F \subset Q$ is the set of accepting states. A *move* of a TM is determined by the current content of the cell that is scanned and the state the machine is in. It consists of three parts:

1. Change state
2. Write a tape symbol in the current cell
3. Move the read-write head to the tape cell on the left or right

A *nondeterministic Turing machine* (NTM) is equal to a deterministic TM with the exception that the range of the transition function consists of sets of triples:

$$\delta(q, X) = \{(p_1, Y_1, D_1), (p_2, Y_2, D_2), ..., (p_k, Y_k, D_k)\}$$

A TM is a *reversible Turing machine* (RTM) if the transition function $\delta(q, X) = (p, Y, D)$ is one-to-one, with the additional constraint that the movement D of the read-write head is uniquely determined by the target state p.

Definition 1. *An* Instantaneous Description *(ID) of a TM during its execution is a string $X_1X_2...X_{i-1}qX_iX_{i+1}...X_n$ in which q is the state of the TM, the tape head is scanning the i-th head from the left, $X_1X_2...X_n$ is the portion of the tape between the leftmost and the rightmost blank. Given an Instantaneous Description $X_1X_2...X_{i-1}qX_iX_{i+1}...X_n$ it will be useful to define an* Extensional Instantaneous Description *(EID) $X_1X_2...X_{i-1}X_iX_{i+1}...X_n$, that only looks at the content of the tape and ignores the internal state of the machine and an* Intensional Instantaneous Description *(IID) qX_iD, that only looks at the content of the current cell of the tape, the internal state of the machine and the direction D in which the head will move.*

We make the jump from an object- to a meta-level of descriptions of computations by means of considering the set of all possible transitions between instantaneous descriptions.

Definition 2. *Let $< \mathcal{ID}_M, \vdash_M >$ be* the configuration graph *of all possible transformations of a machine M, i.e. $\mathcal{ID}_M$ is the countable set of all possible instantaneous descriptions and for $ID_{i,j} \in \mathcal{ID}_M$:*

$$ID_i \vdash_M ID_j$$

iff TM can reach ID_j *in one move from* ID_i. ID_m *is* reachable *from* ID_i *iff there exists a sequence of transformations from one to the other:*

$$(ID_i \vdash_M^* ID_m) \Leftrightarrow ID_i \vdash_M ID_j \vdash_M ID_k ... ID_l \vdash_M ID_m$$

The intensional description of such a transformation will be: $(IID_i \vdash_M^* IID_m)$. *The extensional description will be:* $(EID_i \vdash_M^* EID_m)$

Note that two machines can perform computations that are extensionally isomorphic without intensional isomorphism and vice-versa. We refer here to transformations rather than computations since, in most cases, only a subpart of the configuration graph represents valid computations that start with a start state and end in an accepting state. Note that the class of all possible instantaneous descriptions for a certain machine contains for each possible tape configuration, at each possible position of the head on the tape an instance for each possible internal state. Most of these configurations will only be the result, or lead to, fragments of computations. On the other hand all valid computations, that start with a start state and either continue forever or end in an accepting state will be represented in the configuration graph.

Note that there is a strict relation between the structure of the transition function δ and the configuration graph: for a deterministic machine the configuration graph has only one outgoing edge for each configuration, for a nondeterministic machine the configuration graph can have multiple outgoing edges per ID, for a reversible machine, the graph consists only of a number of linear paths without bifurcations either way.

Lemma 1. *Let* M *be a Turing machine. We have* $C(< \mathcal{ID}_M, \vdash_M >) < C(M) + O(1)$

Proof: Given M the graph $< \mathcal{ID}_M, \vdash_M >$ can be constructed by the following algorithm: create $\mathcal{ID}_M$ by enumerating the language of all possible ID's, at each step of this process run M for one step on all ID's created sofar and add appropriate edges to $\vdash_M$ when M transforms ID_i in ID_j.

The finite object M and the infinite object $< \mathcal{ID}_M, \vdash_M >$ identify the same structure. We use here two variants of the Kolmogorov complexity: the complexity finite object M is defined by the smallest program that computes the object on a universal Turing machine and then halts, the complexity of $< \mathcal{ID}_M, \vdash_M >$ is given by the shortest program that creates the object in an infinite run.

Definition 3. *Given an enumeration of Turing machines the* meta-computational space *is defined as the disjunct sum of all configuration graphs* $< \mathcal{ID}_{M_i}, \vdash_{M_i} >$ *for* $i \in \mathbb{N}$.

The meta-computational space is a very rich object in which we can study a number of fundamental questions concerning the interaction between information and computation. We can also restrict ourselves to the study of either extensional or intensional descriptions of computations and this will prove useful, e.g. when we want to study the class of all computational histories that have descriptions with isomorphic pre- or suffixes. For the moment we want to concentrate on time symmetries in meta-computational space.

3 Time symmetries in meta-computational space

In this paragraph we study the fact that some well known classes of computational processes can be interpreted as each others symmetrical images in time, i.e. processes in one class can be described as processes in the other class with the time arrow reversed, or to say it differently as processes taking place in negative time. We can reverse the time arrow for all possible computations of a certain machine by means of reversing all the edges in the computational graph. This motivates the following notation:

Definition 4.

$$(ID_i \vdash ID_j) \Leftrightarrow (ID_j \dashv ID_i)$$

$$(ID_i \vdash^* ID_k) \Leftrightarrow (ID_k \dashv^* ID_i)$$

The analysis of valid computations of TM can now be lifted to the study of reachability in the configuration graph. The introduction of such a meta-computational model allows us to study a much more general class of computations in which the arrow of time can be reversed. We will introduce the following shorthand notation that allows us to say that M^{-1} is the same machine as M with the arrow of time reversed:

$$M =< \mathcal{ID}_M, \vdash_M > \Leftrightarrow M^{-1} =< \mathcal{ID}_M, \dashv_M >$$

Intuitively the class of languages that is 'computed' in negative time by a certain Turing machine is the class of accepting tape configurations that can be reached from a start state. We have to stress however, that moving back in time in the configuration graph describes a process that is fundamentally different from the standard notion of 'computation' as we know it. We give some differences:

- The standard definition of a Turing machine knows only one starting state and possibly several accepting states. Computing in negative time will trace back from several accepting states to one start state.
- The interpretation of the δ-function or relation is different. In positive time we use the δ-function to decide which action to take given a certain state-symbol combination. In negative time this situation is reversed: which state-symbol-move combination could have lead to a certain action.
- At the start of a computation there could be a lot of rubbish on the tape that is simply not used during the computation. All computations starting with arbitrary rubbish are in the configuration graph. We want to exclude these from our definitions and stick to some minimal definition of the input of a computation in negative time.

In order to overcome these difficulties the following lemma will be useful:

Lemma 2 (Minimal Input-Output Reconstruction). *If an intensional description of a fragment of a (deterministic or non-deterministic) computation of a machine M: $(IID_i \vdash_M^* IID_m)$ can be interpreted as the trace of a valid computation then there exist a* minimal input configuration ID_i *and a* minimal output configuration ID_m *for which M will reach ID_m starting at ID_i. Otherwise the minimal input and output configuration are undefined.*

Proof: The proof first gives a construction for the minimal output in a positive sweep and then the minimal input in a negative sweep.

Positive sweep: Note that $(IID_i \vdash_M^* IID_m)$ consists of a sequence of descriptions: $q_iX_iD_i \vdash q_{i+1}X_{i+1}D_{i+1} \vdash ... \vdash q_mX_mD_m$. Reconstruct a computation in the following way: Start with an infinite tape for which all of the symbols are unknown. Position the read-write head at an arbitrary cell and perform the following interpretation operation: interpret this as the state-symbol-move configuration $q_iX_iD_i$. Now we know the contents of the cell X_i, the state q_i and the direction D of the move of the read-write head. The action will consist of writing a symbol in the current cell and moving the read-write head left or write. Perform this action. The content of one cell is now fixed. Now there are two possibilities:

1. We have the read write head in a new cell with unknown content. From the intensional description we know that the state symbol combination is $q_{i+1}X_{i+1}D_{i+1}$, so we can repeat the interpretation operation for the new cell.
2. We have visited this cell before in our reconstruction and it already contains a symbol. From the intensional description we know that the state symbol combination should be $q_{i+1}X_{i+1}D_{i+1}$. If this is inconsistent with the content of the current cell, the reconstruction stops and the minimal output is undefined. If not, we can repeat the interpretation operation for the new cell.

Repeat this operation till the intensional description is exhausted. Cells on the tape that still have unknown content have not been visited by the computational process: we may consider them to contain blanks. We now have the minimal output configuration on the tape ID_m.

Negative sweep: start with the minimal output configuration ID_m. We know the current location of the read write head and the content of the cell. From the intensional description $(IID_i \vdash_M^* IID_m)$ we know which state symbol combination $q_mX_mD_m$ has lead to ID_m: from this we can construct ID_{m-1}. Repeat this process till the intensional description is exhausted and we read ID_i which is the minimal input configuration.

Lemma 2 gives us a way to tame the richness of the configuration graphs: we can restrict ourselves to the study of computational processes that are intensionally equivalent, specifically intensionally equivalent processes that start with a starting state and end in an accepting state. This facilitates the following definition:

Definition 5. *If* $(IID_i \vdash_M^* IID_m)$ *is an intensional description of a computation then*

$$\texttt{INPUT}(IID_i \vdash_M^* IID_m) = x$$

gives the minimal input x *and*

$$\texttt{OUTPUT}(IID_i \vdash_M^* IID_m) = y$$

gives the minimal output y*. With some abuse of notation we will also apply these functions to histories of full ID's.*

Definition 6. *Given a Turing machine* M *the language recognized by its counterpart* M^{-1} *in negative time is the set of minimal output configurations associated with intensional descriptions of computations on* M *that start in a start state and end in an accepting state.*

Definition 7. *The class* P^{-1} *is the class of languages that are recognized by an* M_i^{-1} *with* $i \in \mathbb{N}$ *in time polynomial to the length of minimal input configuration.*

Note that, after a time reversal operation, the graph of a deterministic machine is transformed in to a specific non-deterministic graph with the characteristic that each ID has only one incoming edge. We will refer to such a model of computation as *quasi-reversible*. The essence of this analysis is that, given a specific machine M, we can study its behavior under reversal of the arrow of time.

We can use the symmetry between deterministic and quasi-reversible computing in proofs. Whatever we prove about the execution of a program on M also holds for M^{-1} with the time reversed and vice versa.

Let $QRTM$ be the class of quasi-reversible non-deterministic machines that are the mirror image in time of the class of deterministic machines TM, and QRP be the class of languages that can be recognized by $QRTM$ in polynomial time. The lemma below is at first sight quite surprising. The class of languages that we can recognize nondeterministically in polynomial time is the same class as the class of polynomial quasi-reversible languages:

Lemma 3. *The class* L_{QRP} *of languages recognized by a* $QRTM$ *in polynomial time is NP.*

Proof:1) $L_{QRP} \subseteq NP$: The class of languages recognized by quasi-reversible machines is a subclass of the class of languages recognized by non-deterministic machines. This is trivial since there is a non-deterministic machine that produces any $\{0,1\}^{\leq k}$ in time k.

2) $NP \subseteq L_{QRP}$: The class NP is defined in a standard way in terms of a checking relation $R \subseteq \Sigma^* \times \Sigma_1^*$ for some finite alphabets Σ^* and Σ_1^*. We associate with each such relation R a language L_R over $\Sigma^* \cup \Sigma_1^* \cup \#$ defined by

$$L_R = \{w\#y | R(w,y)\}$$

where the symbol # is not in Σ. We say that R is polynomial-time iff $L_R \in P$. Now we define the class NP of languages by the condition that a language L over Σ is in NP iff there is $k \in \mathbb{N}$ and a polynomial-time checking relation R such that for all $w \in \Sigma^*$.,

$$w \in L \Leftrightarrow \exists y(|y| < |w|^k \ \& \ R(w, y))$$

where $|w|$ and $|y|$ denote the lengths of w and y, respectively. Suppose that M implements a polynomial-time checking relation for R. Adapt M to form M' that takes $R(w, y)$ as input and erases y from the tape after checking the relation, the transformation of M to M^{-1} is polynomial. The corresponding QRTM M'^{-1} will start with guessing a value for y non-deterministically and will finish in a configuration for which the relation $R(w, y)$ holds in polynomial time since $|y| < |w|^k$ and the checking relation R is polynomial.

We can formulate the following result:

Theorem 1. $NP = P^{-1}$

Proof: immediate consequence of lemma 3 and definition 7.

NP is the class of languages that can be recognized by deterministic Turing machines in negative time. This shows that quasi-reversible computing is in a way a more natural model of non-deterministic computing than the classical full-blown non-deterministic model. The additional power is unnecessary.

4 The interplay of computation and information

We now look at the interplay between information and computation. The tool we use will be the study of the changes in $C(ID_t)$, i.e. changes in the Kolmogorov complexity of instantaneous descriptions over time. We make some observations:

- If $ID_i \vdash_M ID_j$ then the information distance between the instantaneous descriptions ID_i and ID_j is $\log k + 1$ at most where k is the number of internal states of M.
- If $EID_i \vdash_M EID_j$ then the information distance between the extensional descriptions EID_i and EID_j is 1 bit at most.
- If $IID_i \vdash_M IID_j$ then the information distance between the intensional descriptions IID_i and IID_j is $\log k + 2$ at most where k is the number of internal states of M.
- Let x be the minimal input of a computational fragment $(IID_i \vdash_M^* IID_m)$ and let y be the minimal output. We have

$$C(x|IID_i \vdash_M^* IID_m) = C(y|IID_i \vdash_M^* IID_m) = O(1)$$

This is an immediate consequence of lemma 2.

We can now identify some interesting typical machines:

– No machine can produce information faster than 1 bit per computational step. There is indeed a non-deterministic machine that reaches this 'speed': the non-deterministic 'coin-flip' automaton that writes random bits. For such an automaton we have with high probability $C(ID_t) \approx t$. In negative time this machine is the maximal eraser. It erases information with the maximum 'speed' of 1 bit per computational step.
– A unary counting machine produces information with a maximum speed of $\log t$. Note that $C(t) = I(t)$, i.e. the complexity at time t is equal to the value of the integer complexity function. The function $I(x)$ has indefinite 'dips' in complexity, i.e. at those places where it approaches a highly compressible number. When t approaches such a dip the information produced by a unary counting machine will drop as the machine continues to write bits. The counter part of the unary counter in negative time is the unary eraser. It erases information with the maximal speed of $\log t$, although at times it will create information by erasing bits.
– The slowest information producer for its size is the busy-beaver function. When it is finished it will have written a enormous amount of bits with a conditional complexity of $O(1)$. Its counter part in negative time will be a busy-glutton automaton that 'eats' an enormous amount of bits of an exact size.

These insights allow us to draw a picture that tells us how information and computation are intertwined in a deterministic process.

The complexity of the history of a computation is related to the complexity of the input given the output. There are two forms of complexity involved in a deterministic computation:

– Program Complexity: this is the complexity of the input and its subsequent configurations during the process. It can not grow during the computation. Most computations reduce program complexity.
– Program Counter complexity: This is the descriptive complexity of the program counter during the execution of the process. It is 0 at the beginning, grows to $\log a$ in the middle and reduces to 0 again at the end of the computation.

The relationship between these forms of complexity is given by the following theorem:

Theorem 2 (Information exchange in Deterministic Computing). *Suppose M is a deterministic machine and $ID_i \vdash_M ID_a$ is a fragment of an accepting computation, where ID_m contains an accepting state. For every $i \leq k \leq a$ we have:*

1. *Determinism: $C(ID_{i+k+1} \vdash_M ID_a | M, ID_{i+k}) = O(1)$, i.e. at any moment of time if we have the present configuration and the definition of M then the future of the computation is known.*
2. *Program counter complexity from the start: $C(ID_t | ID_0, M) < (\log k) + O(1)$, this constraint is known during the computation.*

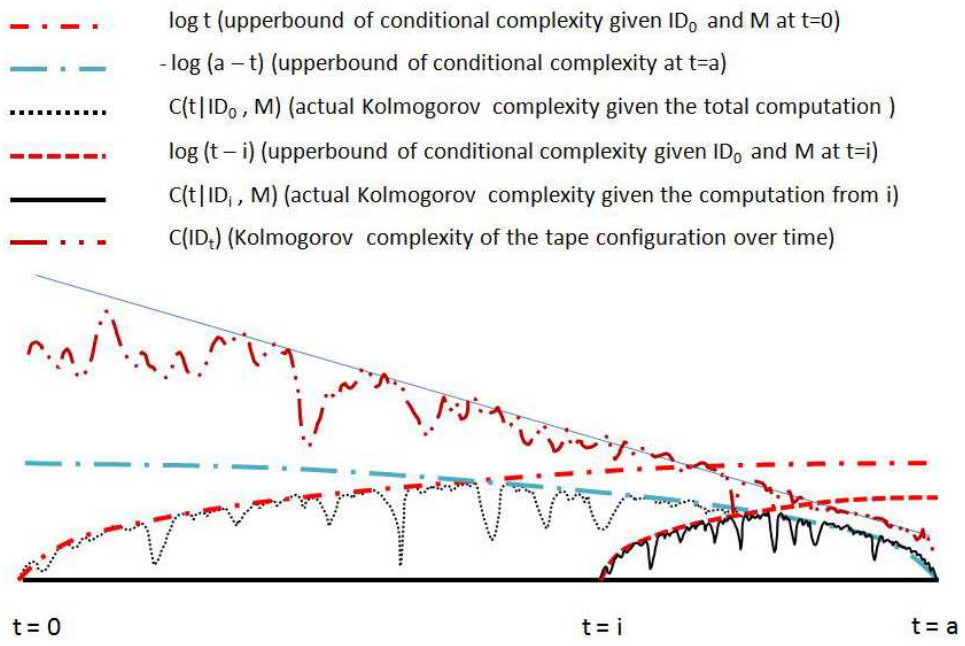

Fig. 1. Schematic representation of the various types of complexity estimates involved in a deterministic computation

3. *Program counter complexity from the end:* $C(ID_t|ID_0, M) < (\log a - k) + O(1)$*, this constraint is not known during the computation.*
4. *Program complexity:*

$$C((IID_{i+k} \vdash_M^* IID_a)|M) = C(\texttt{INPUT}(IID_{i+k} \vdash_M^* IID_a)|M) + O(1)$$

Proof:

1. Trivial, since M is deterministic.
2. Any state ID_k at time k can be identified by information of size $\log k$ if the initial configuration and M are known.
3. Any state ID_k at time k can be identified by information of size $\log(a-k)$ if the total description of the accepting computational process and M are known.
4. By the fact that the computation is deterministic it can be reconstructed from the minimal input, given M. By lemma 2, given M, the minimal input can be reconstructed from $(IID_i \vdash_M^* IID_a)$. This gives the equality modulo $O(1)$.

We cannot prove such a nice equality for the minimal output. Note that even if the following inequality holds:

$$C((IID_i \vdash_M^* IID_a)|M) \geq C((IID_{i+k} \vdash_M^* IID_a)|M) + O(1)$$

this does not imply that:

$$C(\texttt{OUTPUT}(IID_i \vdash_M^* IID_a)|M) \geq C(\texttt{OUTPUT}(IID_{i+k} \vdash_M^* IID_a)|M) + O(1)$$

As a counter example: observe that a program that erases a random string has a string of blanks as minimal output. A longer string still can have a lower Kolmogorov complexity.

In computations that use counters, program complexity and program counter complexity are mixed up during the execution. In fact one can characterize various types of computations by means of their 'information signature'. Informally, at extremes of the spectrum, one could distinguish:

- *Pure Information Discarding Processes*: in such processes the program counter does not play any role. They reach an accepting state by means of systematically reducing the input. Summation of a set of numbers, or erasing of a string are examples.
- *Pure Counting Processes*: `For x=1 to i write("1")`: The conditional complexity of the tape configuration grows from 0 to $\log i$ and then diminishes to 0 again.
- *Pure Search Processes*: In such processes the input is not reduced but is kept available during the whole process. The information in the program counter is used to explore the search space. Standard decision procedures for NP-hard programs, where the checking function is tested on an enumeration of all possible solutions, are an example.

A deeper analysis of various information signatures of computational processes and their consequences for complexity theory is a subject of future work.

5 Discussion

We can draw some conclusions and formulate some observations on the basis of the analysis given above.

1) Erasing and creating information are indeed, as suggested in the introduction, from a time invariant computational point of view the same processes: The quasi-reversible machine that is associated with a simple deterministic machine that erases information is a non-deterministic machine writing arbitrary bit-strings on the tape. This symmetry also implies that creation of information in positive time involves destruction of information in negative time.

2) The class of quasi-reversible machines indeed describes the class of data sets from which we can learn something in the following way: If L is the language accepted by M then M^{-1} generates L. M^{-1} is an *informer* for L in the sense

of [6], every sentence in L will be non-deterministically produced by M^{-1} in the limit. $QRTM$ is the class of all informers for type-0 languages.

3) These insights suggests that we can describe stochastic processes in the real world as deterministic processes in negative time: e.g. throwing a dice in positive time is erasing information about its 'future' in negative time, the evolution of species in positive time could be described as the 'deterministic' computation of their ancestor in negative time. A necessary condition for the description of such growth processes as computational processes is that the number of bits that can be produced per time unit is restricted. A stochastic interpretation of a QRTM can easily be developed by assigning a set of probabilities to each split in the δ relation. The resulting stochastic-QRTM is a *sufficient statistic* for the data sets that are generated.

4) The characterization of the class NP in terms of quasi-reversible computing seems to be more moderate than the classical description in terms of full non-deterministic computing. The full power of non-deterministic computing is never realized in a system with only one time direction.

5) Processes like game playing and genetic algorithms seem to be meta-computational processes in which non-deterministic processes (throwing a dice, adding mutations) seem to be intertwined with deterministic phases (making moves, checking the fitness function)

6) The time-symmetry has consequences for some philosophical positions. The idea that the evolution of our universe can be described as a deterministic computational process has been proposed by several authors (Zuse, Bostrom, Schmidthuber, Wolfram [10], Lloyd [9], etc.). It nowadays is referred to as pan-computationalism [5]. If deterministic computation is an information discarding process then it implies that the amount of information in the universe rapidly decreases. This contradicts the second law of thermodynamics. On the other hand, if the universe evolves in a quasi-reversible way, selecting possible configurations according to some quasi-reversible computational model, it computes the big bang in negative time. The exact implications of these observations can only be explained by means of the notion of facticity [3], but that is another discussion. The concept of quasi-reversible computing seems to be relevant for these discussions [2].

6 Conclusion

Computing is moving through meta-computational space. For a fixed Turing machine M_i such movement is confined to one local infinite graph $< \mathcal{ID}_{M_i}, \vdash_{M_i} >$. If M_i is deterministic then M_i^{-1} is non deterministic. If M is information discarding then M^{-1} 'creates' information. The two fundamental complexities involved in a deterministic computation are Program Complexity and Program Counter Complexity. Programs can be classified in terms of their 'information signature' with pure counting programs and pure information discarding programs as two ends of the spectrum. The class NP is simply the class of polynomial deterministic time calculations in negative time. Thinking in terms of meta-computational

space allows us to conceptualize computation as movement in a certain space and is thus a source of new intuitions to study computation. Specifically a deeper analysis of various information signatures of computational (and other) processes is a promising subject for further study.

Bibliography

[1] S. Abramsky, Information, Processes Games, in Handbook of the philosophy of information, P.W.Adriaans, J.F.A.K. van Benthem eds. in Handbook of the philosophy of science, Series edited by D. M. Gabbay, P. Thagard and J. Woods, eds. Elsevier, pg. 483, 550, 2009.

[2] Handbook of the philosophy of information, P.W.Adriaans, J.F.A.K. van Benthem eds. in Handbook of the philosophy of science, Series edited by D. M. Gabbay, P. Thagard and J. Woods, eds. Elsevier, 2009.

[3] Adriaans , P.W. , (2009) Between Order and Chaos: The Quest for Meaningful Information, Theory of Computing Systems, Volume 45 , Issue 4 (July 2009), Special Issue: Computation and Logic in the Real World; Guest Editors: S. Barry Cooper, Elvira Mayordomo and Andrea Sorbi, 650-674.

[4] Adriaans, P. Vitányi, P., *Approximation of the Two-Part MDL Code*, Comput. Sci. Dept., Univ. of Amsterdam, Amsterdam; Information Theory, IEEE Transactions on, Jan. 2009 Volume: 55, Issue: 1 On page(s): 444-457.

[5] Floridi, L. (2008) Trends in the Philosophy of information, in Handbook of Philosophy of Information, P.W.Adriaans, J.F.A.K. van Benthem (eds.), Elseviers Science Publishers.

[6] Gold, E. Mark, Language Identification in the Limit, Information and Control, pg. 447,474,1967.

[7] J.E. Hopcroft, Rajeev Motwani, Jeffrey D. Ullman, Introduction to Automata Theory, Languages, and Computation Second Edition. Addison-Wesley, 2001.

[8] Ming Li and Paul Vitányi, An Introduction to Kolmogorov Complexity and Its Applications, Third Edition, Springer Verlag, 2008.

[9] S. Lloyd, "Ultimate physical limits to computation". Nature 406: 10471054, 2000.

[10] S. Wolfram, A new kind of science, Wolfram Media Inc., 2002.