Apples and Oranges? Comparing Unconventional Computers

ED BLAKEY
Oxford University Computing Laboratory
Wolfson Building, Parks Road, Oxford, OX1 3QD
UNITED KINGDOM
edward.blakey@queens.ox.ac.uk
http://users.ox.ac.uk/~quee1871/

Abstract: Complexity theorists routinely compare—via the pre-ordering induced by asymptotic $O$-notation—the efficiency of computers so as to ascertain which offers the most efficient solution to a given problem. Tacit in this statement, however, is that the computers conform to a standard computational model: that is, they are Turing machines, random-access machines or similar. However, whereas meaningful comparison between these conventional computers is well understood and correctly practised, that of non-standard machines (such as quantum, chemical and optical computers) is rarely even attempted and, where it is, is often attempted under the typically false assumption that the conventional-computing approach to comparison is adequate in the unconventional-computing case. We discuss in the present paper a computational-model-independent approach to the comparison of computers’ complexity (and define the corresponding complexity classes). Notably, the approach allows meaningful comparison between an unconventional computer and an existing, digital-computer benchmark that solves the same problem.

Key-Words: Computational complexity, Unconventional computation, Theoretical computer science, Comparison, Computational resource, Complexity class, Dominance, Asymptotic notation.

1 Introduction

Computational complexity theory has as one of its chief aims the quantification of mathematical problems’ difficulty: complexity theorists wish to make statements of the form ‘solving problem $X$ (e.g., the Travelling Salesperson Problem, factorization or addition) requires $O(f)$ time, $O(g)$ space, etc.’. However, it is possible directly to measure the complexity not of problems but only of methods that solve these problems: whereas one would like to demonstrate that ‘problem $X$ requires $O(f)$ time’, it is usually forthcoming only that ‘problem $X$ can be solved by algorithm $Y$, which requires $O(f)$ time’. Typically, then, all that is known about a problem’s complexity is that it is bounded above by that of the most efficient, known solution method for the problem.

It is clear from this that the ability to compare computers’ efficiency—and thereby to ascertain which computer offers the most efficient solution to a problem—is of the utmost importance. It is unsurprising, then, that, when the computers in question are ‘standard’ (e.g., when they are modelled as Turing machines), the method by which they may be compared is well understood; we now recap this method, by first recalling the auxiliary notions of $O$-notation and the pre-ordering induced thereby.

Definition 1

1. Let $O(g(n))$ denote the class of all functions $f(n)$ such that there exist a threshold $n_0 \in \mathbb{N}$ and constant $c \in \mathbb{R}$ satisfying $|f(n)| \leq c|g(n)|$ for all natural numbers $n$ such that $n > n_0$.
2. Write ‘$f \lesssim g$’ for ‘$f \in O(g)$’ and ‘$f \preceq g$’ for ‘$f \not\in O(g)$’.

It is trivial that $\lesssim$ is both reflexive (since threshold $0$ and constant $1$ witness that $f \lesssim f$) and transitive (since, if threshold-constant pair $(n_0, c)$ witnesses that $f \lesssim g$ and $(n'_0, c')$ that $g \lesssim h$, then $(\max \{n_0, n'_0\}, cc')$ witnesses that $f \lesssim h$); hence, $\lesssim$ is a pre-ordering of functions. Of specific interest here is that $\lesssim$ may be used as a pre-ordering of complexity functions (which we define in Definition 3). We can now describe the method by which the efficiency of standard computers is compared.

Suppose that (conventional, Turing-machine-style) computers $\Phi$ and $\Psi^2$ have respective time-

---

1The ‘etc.’ here refers to computational resources other than run-time and memory space—see [1], [4] and Sect. 2.1.
complexity functions $T_\Phi^s$ and $T_\Psi^s$ (that is, given an arbitrary input value of size $n$, $\Phi$ requires at most $T_\Phi^s(n)$ time steps to return the corresponding output value; similarly $\Psi$—see Definition 3). One may utilize the pre-ordering $\preceq$ to determine which (if either) of $\Phi$ and $\Psi$ is the more efficient, in the obvious way:

- if $T_\Phi^s \preceq T_\Psi^s \preceq T_\Phi^s$, then $\Phi$ is deemed the more efficient computer;
- if $T_\Phi^s \preceq T_\Psi^s \preceq T_\Phi^s$, then $\Psi$ is deemed the more efficient;
- if $T_\Phi^s \preceq T_\Psi^s \preceq T_\Phi^s$, then $\Phi$ and $\Psi$ are deemed equally efficient; and
- if $T_\Phi^s \preceq T_\Psi^s \preceq T_\Phi^s$, then $\Phi$ and $\Psi$ are deemed incomparably efficient.

Note that we tacitly identify (overall) efficiency with time efficiency in particular. This is because, in the case of conventional, Turing-style computers, the only computational resources that need be considered are time and space, and, of these, time is always consumed in greater quantity (the space complexity $S_\Phi^s$ of a conventional computer $\Phi$ is always a smaller function than the time complexity $T_\Phi^s$—in that $S_\Phi^s(n) \leq T_\Phi^s(n)$ for all $n$—, since writing to a tape cell takes one time step).

However, there exist unconventional computers (conforming, for example, to quantum, chemical or analogue paradigms), for which this is not the case: such computers may consume resources other than time and space, whence it is no longer valid to identify time efficiency with overall efficiency; consequently, one cannot simply apply the pre-ordering $\preceq$ to these unconventional computers’ time-complexity functions in order to determine which is the most efficient. For example, we describe in [2] an analogue computer that can factorize natural numbers in time and space polynomial in the size of the input value; since the true (exponential) complexity of the system lies in its precision complexity (which notion is introduced in [3] and discussed in [2]), an exclusively time-based consideration—à la standard, Turing-machine complexity theory—of the system leads to an overly generous quantification of its complexity and to misleading comparisons with other (e.g., Turing-machine) solutions to the problem of factorization.

In summary, then, the difficulty is that unconventional computers may consume unconventional resources (in addition to the standard resources of time and space), which leads to each computer’s having many complexity functions. The comparison of the efficiency of two such computers, then, is not merely a case of applying the pre-ordering $\preceq$ to the respective time-complexity functions. We describe in this paper a more suitable method whereby such computers’ efficiency can be meaningfully compared.

2 Comparison of Computers

2.1 Resource

We mention above the widely considered computational resources of time and space. We also hint that certain (unconventional) computers consume other resources (such as precision). We defer to [1] and [4] a full motivation and discussion of resources, making instead the following definition, which is adequate for present purposes.

Definition 2

- We model a resource as a function (denoted by an upper-case letter $A$, $B$, $C$, etc.) that depends upon the choice of computational system (shown as a subscript to the function) and that maps each input value to a natural number, which can be thought of as the corresponding number of units of the resource consumed by the system in processing that value.
- Let $T$ denote the resource of time, $S$ space and $P$ precision.

Hence, for computer $\Phi$ and input value $x$, $A_\Phi(x)$ denotes the amount of resource $A$ consumed by $\Phi$ in processing $x$. Specific and notable examples are $T_\Phi(x)$, which denotes the number of units of time (e.g., the number of time steps if $\Phi$ is a Turing machine) taken by $\Phi$ to process $x$, and $S_\Phi(x)$, the number of units of space occupied (e.g., the number of tape cells used if $\Phi$ is a Turing machine); the resource of precision is discussed in, for example, [3].

2.2 Complexity

Given a computer, and considering a specific resource, we may ask how this resource scales. In particular, we may be interested not in the resource used/required by the computer given one specific input value (that is, in some $A_\Phi(x)$), but in the resource used as a function of the size of the input value—this is what is meant by the complexity function corresponding to the resource. Specifically, we have the following.
Definition 3 For resource $A$, the corresponding complexity function $A^*$ is defined by

$$A^*_A (n) := \sup \{ A^*_A (x) \mid |x| = n \}.$$ 

In this definition, $|x|$ stands for the size of input value $x$. If, for example, $x$ is a natural number expressed in binary, then we can take as its size the number of bits, excluding leading zeros (or, as is sufficient for virtually all complexity-theoretic purposes, the approximation $\log_2 (x)$ to this number of bits).

Note that our usage in Sect. 1 of ‘$T^*$’ and ‘$S^*$’ conforms with the notation of Definition 3.

2.3 Dominance Motivated

We see in Sect. 1 that $O$-notation, and in particular the pre-ordering $\preceq$ induced thereby, allows comparison of the respective time-complexity functions of standard (e.g., Turing-machine) computers (in this context, the comparison of $\text{time-efficiency}$ actually offers an assessment of the relative, overall efficiency of the computers being analysed, as we comment in Sect. 1). More generally, the pre-ordering allows ‘apples-to-apples’ comparison, in that computers $\Phi$ and $\Psi$ may be compared with respect to the same (arbitrary) resource $A$.

However, when one considers unconventional (optical, quantum, DNA, etc.) computers, which may consume many unconventional resources (precision, energy, thermodynamic cost, etc.), it is no longer generally true that an ‘apples-to-apples’ comparison with respect to any given resource equates to a fair comparison of the computers’ overall efficiency. We now illustrate this phenomenon by recalling an instance from [3].

Supposing that we wish to find the greatest common divisor of two given, natural numbers (with a combined length of $n$ digits, say), we have available (amongst others) two solution methods:

- Euclid’s Algorithm (which we denote by ‘E’), which has time and space complexities polynomial, and precision complexity constant, in $n$ (that is, $T^*_E \in O (n^k)$ (Lemma 11.7 of [5]), $S^*_E \in O (n^k)$ (since $S^*_E (n) \leq T^*_E (n)$ for all $n$) and $P^*_E \in O (1)$ (Theorem 1 of [3]) ($k$ constant)); and
- an analogue system (which we denote by ‘B’, and of which we defer description to [3]), which has time and space complexities constant, and precision complexity exponential, in $n$ (that is, $T^*_B, S^*_B \in O (1)$ and $P^*_B \in O (l^n)$ (see [3]) ($l$ constant)).

One may describe the methods $E$ and $B$ respectively as polynomial- and constant-time (and infer by ‘apples-to-apples’ comparison that $B$ is the more time-efficient); however, it is intuitively more insightful to describe the latter as an $\text{exponential-precision}$ (and, hence, less efficient overall), rather than $\text{constant-time}$, method, since the former description focuses on the more ‘relevant’ resource. We now define dominance so as to formalize this notion of ‘relevance’, and to allow meaningful comparison between computers regardless of how many different resources they consume.

2.4 Dominance Defined

The intent of dominance is that the complexity functions corresponding to resources deemed to be dominant should be $\preceq$-greater than those corresponding to other resources; non-dominant resources, then, are negligible in the sense that their asymptotic contribution to a computer’s overall resource consumption is dwarfed by that of dominant resources. Accordingly, we make the following tentative definition.

Definition 4 (provisional—see Definition 5) A dominant resource for a computer $\Phi$ is a resource $A$ such that, for any resource $B$, $B^*_\Phi \preceq A^*_\Phi$.

This definition requires modification for the following two reasons.

- First, $A$’s dominance is over every other resource $B$. It is not sufficient (in order that, for example, precision is shown to be dominant according to Definition 4) to show that precision complexity $\preceq$-exceeds time and space complexity; rather, precision complexity must be shown to $\preceq$-exceed time, space, and all other conceivable resources’ complexity, of which resources there are indeterminately many—this is clearly a futile task and a worthless definition. Accordingly, we redefine dominance below relative to an explicit set of resources.

- Secondly, Definition 4 does not for every computer guarantee the existence of a dominant resource (much as we should like one), since there exist pairs of functions $f$ and $g$ such that $f \not\preceq g \not\preceq f$. We weaken accordingly the definition of dominance (essentially from ‘$A^* \preceq$-exceeds all other complexity functions’ to ‘$A^* \preceq$-exceeds all other complexity functions with which it is $\preceq$-comparable’).

Definition 5 (to replace Definition 4) Let $\Phi$ be a computer, and let $R$ be a finite, non-empty set of resources consumed by $\Phi$. An $R$-dominant resource
for \( \Phi \) is a resource \( A \in \mathcal{R} \) such that, for any resource \( B \in \mathcal{R} \) satisfying \( A_{\Phi}^* \lesssim B_{\Phi}^* \), we have that \( B_{\Phi}^* \lesssim A_{\Phi}^* \).

We note in passing that \( \mathcal{R} \)-dominance has the following ‘all-or-nothing’ property.

**Proposition 6** Let \( \Phi \) and \( \mathcal{R} \) be as in Definition 5, and let \( X, Y \in \mathcal{R} \). Suppose that \( X_{\Phi}^* \lesssim Y_{\Phi}^* \lesssim X_{\Phi}^* \). Then \( X \) is \( \mathcal{R} \)-dominant for \( \Phi \) if and only if \( Y \) is.

**Proof:** Suppose that \( X, Y \in \mathcal{R} \) are such that \( X_{\Phi}^* \lesssim Y_{\Phi}^* \lesssim X_{\Phi}^* \).

(We prove first the ‘\( X \) dominant \( \implies \) \( Y \) dominant’ direction.) Suppose that \( X \) is \( \mathcal{R} \)-dominant. Then, by Definition 5, \( X_{\Phi}^* \lesssim B_{\Phi}^* \Rightarrow B_{\Phi}^* \lesssim X_{\Phi}^* \) for all \( B \in \mathcal{R} \).

If, for some \( B \in \mathcal{R} \), \( Y_{\Phi}^* \lesssim B_{\Phi}^* \), then \( X_{\Phi}^* \lesssim B_{\Phi}^* \) (by the fact that \( X_{\Phi}^* \lesssim Y_{\Phi}^* \) and by transitivity of \( \lesssim \)), whence \( B_{\Phi}^* \lesssim X_{\Phi}^* \) (by \( \mathcal{R} \)-dominance of \( X \)). Since this holds for arbitrary \( B \in \mathcal{R} \), we have that \( Y \) is \( \mathcal{R} \)-dominant.

(‘\( Y \) dominant \( \implies \) \( X \) dominant’ direction.) The converse argument differs only in that the roles of \( X \) and \( Y \) are switched, which is valid since the hypotheses of the proposition are symmetrical in \( X \) and \( Y \). \( \square \)

It is at this juncture natural to introduce the following complexity classes.

**Definition 7** Let \( \mathcal{R} \) be as in Definition 5.

- For \( A \in \mathcal{R} \) and function \( f \), let \( C_{\mathcal{R}} (f, A) \) denote the complexity class of problems solved by some deterministic\(^4\) computer \( \Phi \) with \( \mathcal{R} \)-dominant resource \( A \) such that \( A_{\Phi}^* \lesssim f \).
- Let \( NC_{\mathcal{R}} (f, A) \) denote the analogous non-deterministic class.
- Let \( C_{\mathcal{R}} (f) = \bigcup_{A \in \mathcal{R}} C_{\mathcal{R}} (f, A) \).
- Let \( NC_{\mathcal{R}} (f) = \bigcup_{A \in \mathcal{R}} NC_{\mathcal{R}} (f, A) \).

We define also a notion of ‘overall complexity’.

**Definition 8** Let \( \Phi \) and \( \mathcal{R} \) be as in Definition 5.

- Define \( B_{\mathcal{R}, \Phi} \) by

\[
B_{\mathcal{R}, \Phi} (n) := \sum_{A \text{ is } \mathcal{R} \text{-dominant}} A_{\Phi}^* (n) .
\]

*Call this the \( \mathcal{R} \)-complexity of \( \Phi \); it is intended to capture, in a single complexity function, the ‘overall complexity’ of \( \Phi \).*

- Let \( B_{\mathcal{R}} (f) \) denote the complexity class of problems for which there exists a deterministic computer \( \Phi \) with \( B_{\mathcal{R}, \Phi} (n) \leq f (n) \) for all \( n \).
- Let \( NB_{\mathcal{R}} (f) \) denote the analogous non-deterministic class.

3 **Complexity Class Inclusions**

So as to give a flavour of the structure of the hierarchy of complexity classes (namely, \( C_{\mathcal{R}} (f, A) \), \( C_{\mathcal{R}} (f) \), \( B_{\mathcal{R}} (f) \), and their non-deterministic counterparts) defined in Definitions 7 and 8, we state (without proof, which is deferred to future work for reasons of space) the following theorems.

We note first (in Theorem 9) the connection between *determinism* and *non-determinism* (specifically, that deterministic computers are a special case of non-deterministic computers).

**Theorem 9** If \( \mathcal{R} \) is a set of resources with \( A \in \mathcal{R} \), and if \( f \) is a function, then

- \( C_{\mathcal{R}} (f, A) \subseteq NC_{\mathcal{R}} (f, A) \).
- \( C_{\mathcal{R}} (f) \subseteq NC_{\mathcal{R}} (f) \) and
- \( B_{\mathcal{R}} (f) \subseteq NB_{\mathcal{R}} (f) \).

We consider now the dependency of our classes \( C_{\mathcal{R}} (f, A) \), etc. upon their resource-set parameter \( \mathcal{R} \), first (in Theorem 10, Corollary 11 and Theorem 12) by contrasting a subset \( \mathcal{R} \) against a superset \( \mathcal{S} \),

- secondly (in Theorem 13) by contrasting disjoint sets \( \mathcal{R} \) and \( \mathcal{S} \), and

- thirdly (in Theorem 14) by contrasting non-disjoint, non-enveloping sets \( \mathcal{R} \) and \( \mathcal{S} \).

**Theorem 10** If \( \mathcal{S} \) is a resource set and \( A \in \mathcal{R} \subseteq \mathcal{S} \), then

- \( C_{\mathcal{S}} (f, A) \subseteq C_{\mathcal{R}} (f, A) \) and
- \( NC_{\mathcal{S}} (f, A) \subseteq NC_{\mathcal{R}} (f, A) \).

As a consequence, we have the following.

**Corollary 11** If \( \mathcal{S} \) is a resource set and \( \mathcal{R} \subseteq \mathcal{S} \), then

- \( C_{\mathcal{S}} (f) \subseteq C_{\mathcal{R}} (f) \) and
- \( NC_{\mathcal{S}} (f) \subseteq NC_{\mathcal{R}} (f) \).

**Theorem 12** If \( \mathcal{S} \) is a resource set, and if \( \mathcal{R} \subseteq \mathcal{S} \), then \( B_{\mathcal{R}, \Phi} \lesssim B_{\mathcal{S}, \Phi} \) for all computers \( \Phi \). It does not necessarily hold, however, that \( B_{\mathcal{S}} (f) \subseteq B_{\mathcal{R}} (f) \) for bounding function \( f \), and neither do we have the converse class inclusion; similarly, it necessarily holds neither that \( NB_{\mathcal{S}} (f) \subseteq NB_{\mathcal{R}} (f) \) nor conversely.
Theorem 13 Let $\mathcal{R}$ and $\mathcal{S}$ be disjoint resource sets. Then no relation of class inclusion (neither $\subseteq$ nor $\supseteq$) holds in generality between
\begin{itemize}
  \item $\mathcal{C}_\mathcal{R}(f, A)$ and $\mathcal{C}_\mathcal{S}(f, A)$;
  \item $\mathcal{NC}_\mathcal{R}(f, A)$ and $\mathcal{NC}_\mathcal{S}(f, A)$;
  \item $\mathcal{C}_\mathcal{R}(f, A)$ and $\mathcal{C}_\mathcal{S}(f, A)$;
  \item $\mathcal{NC}_\mathcal{R}(f, A)$ and $\mathcal{NC}_\mathcal{S}(f, A)$;
  \item $\mathcal{B}_\mathcal{R}(f, A)$ and $\mathcal{B}_\mathcal{S}(f, A)$;
  \item $\mathcal{NB}_\mathcal{R}(f, A)$ and $\mathcal{NB}_\mathcal{S}(f, A)$.
\end{itemize}

Neither is it generally the case that $\mathcal{B}_\mathcal{R} \subseteq \mathcal{B}_\mathcal{S}$ or vice versa, nor that $\mathcal{B}_\mathcal{R} \leq \mathcal{B}_\mathcal{S}^5$ or vice versa.

Theorem 14 Let $\mathcal{R}$ and $\mathcal{S}$ be such that $\mathcal{R} \not\subseteq \mathcal{S} \subseteq \mathcal{R}$ and $\mathcal{R} \cap \mathcal{S} \neq \emptyset$. Then no relation of class inclusion (neither $\subseteq$ nor $\supseteq$) holds in generality between
\begin{itemize}
  \item $\mathcal{C}_\mathcal{R}(f, A)$ and $\mathcal{C}_\mathcal{S}(f, A)$;
  \item $\mathcal{NC}_\mathcal{R}(f, A)$ and $\mathcal{NC}_\mathcal{S}(f, A)$;
  \item $\mathcal{C}_\mathcal{R}(f, A)$ and $\mathcal{C}_\mathcal{S}(f, A)$;
  \item $\mathcal{NC}_\mathcal{R}(f, A)$ and $\mathcal{NC}_\mathcal{S}(f, A)$;
  \item $\mathcal{B}_\mathcal{R}(f, A)$ and $\mathcal{B}_\mathcal{S}(f, A)$;
  \item $\mathcal{NB}_\mathcal{R}(f, A)$ and $\mathcal{NB}_\mathcal{S}(f, A)$.
\end{itemize}

Neither is it generally the case that $\mathcal{B}_\mathcal{R} \subset \mathcal{B}_\mathcal{S}$ or vice versa, nor that $\mathcal{B}_\mathcal{R} \leq \mathcal{B}_\mathcal{S}$ or vice versa.

Theorem 17 Let $f$ and $g$ satisfy $f \preceq g$. Then no relation of class inclusion (neither $\subseteq$ nor $\supseteq$) holds in generality between
\begin{itemize}
  \item $\mathcal{B}_\mathcal{R}(f, g)$;
  \item $\mathcal{B}_\mathcal{R}(f, g)$;
  \item $\mathcal{B}_\mathcal{R}(f, g)$;
  \item $\mathcal{NB}_\mathcal{R}(f, g)$.
\end{itemize}

However, the following restriction yields a more definite result.

Theorem 18 If $f \preceq g$, then
\begin{itemize}
  \item $\mathcal{B}_\mathcal{R}(f, g)$;
  \item $\mathcal{NB}_\mathcal{R}(f, g)$.
\end{itemize}

Finally, we consider (in Theorem 19) the dependency of our classes $\mathcal{C}_\mathcal{R}(f, A)$ and $\mathcal{NC}_\mathcal{R}(f, A)$ upon their resource parameter $A$, by contrasting two resources of which one is always pairwise dominant.

Theorem 19 Suppose that $(\mathcal{R}	ext{-dominant})$ resources $A$ and $B$ are such that $A^*_\Phi \preceq B^*_\Phi$ for all computers $\Phi$ (notably, this inclusion is certainly true when $A^*_\Phi \leq B^*_\Phi$ for all $\Phi$). Then
\begin{itemize}
  \item $\mathcal{C}_\mathcal{R}(f, B) \subseteq \mathcal{C}_\mathcal{R}(f, A)$ and
  \item $\mathcal{NC}_\mathcal{R}(f, B) \subseteq \mathcal{NC}_\mathcal{R}(f, A)$.
\end{itemize}

4 Discussion

Our notion of dominance formalizes resources’ relevance to computational processes: resources that are dominant impose the greatest asymptotic cost, so much so that non-dominant resources can be disregarded as irrelevant. Thus, much as the pre-ordering $\preceq$ can be used to compare the respective time-complexity functions of Turing machines (or similar) and thereby compare their overall efficiency, $\preceq$ can also be used to compare the respective $\mathcal{R}$-complexity functions of arbitrary-paradigm computers and thereby compare their overall efficiency.

Furthermore, we have specified (in Definitions 7 and 8) complexity classes in which are categorized problems according to their cost in terms of relevant (i.e., dominant) resources. Consequently, we have a framework in which meaningful, consistent comparison of model-heterogenous sets of computers is possible; the framework’s complexity classes can accommodate computers conforming to various computational paradigms, and can provide structure reflecting the cost of computation in terms of various resources.

The model-heterogeneity of our framework offers an immediate and important advantage: a problem’s
completeness, which is the most commonly sought object in computational complexity theory, is bounded above by the complexity of the most efficient, known solution method for the problem; the ability to compare model-heterogeneous—and, hence, larger—sets of solution methods results in a lower minimal complexity of methods, and, hence, tighter upper bounds on the complexity of problems themselves.

A further advantage (especially for the unconventional-computation community) of the definitions proposed in the present paper is that a newly-designed, non-standard computer that solves a problem can be meaningfully compared with the benchmark of an existing, standard computer that solves the same problem.

We hope that this work, and the above-mentioned advantages thereof, are of interest and use to practitioners of complexity theory and unconventional computing, and to participants of ISTASC ’10.

Acknowledgements: We thank Bob Coecke and Joël Ouaknine for their continued support, supervision and suggestions; participants of conferences attended/organized by the author for their encouraging feedback and useful discussion; and reviewers of publications to which the author has contributed for their detailed comments. We acknowledge the generous support of the EPSRC: this work is funded by the grant Complexity and Decidability in Unconventional Computational Models. Finally, we thank the organizers of ISTASC ’10 for their kind invitation to present this work.

References: