

An Introduction to post-Newtonian and non-Turing computation

Mike Stannett, Hon Visiting Research Fellow
Dept of Computer Science, University of Sheffield, United Kingdom
Sheffield, July 2000

mike@noisefactory.co.uk

Author's note: *This document reproduces Research Report CS-91-02, An Introduction to Post-Newtonian and non-Turing Computation, Formal Methods Group, Department of Computer Science, University of Sheffield, United Kingdom. Copyright ©1991 Mike Stannett. All Rights Reserved. The report is no longer available in its original format, and is reproduced here with minor corrections. The AXM semantics presented here comprise an early, very loose, sketch – the AXM concept has evolved radically since this report was originally published. Except where indicated, all footnotes are those of the original report.*

Keywords: Hypercomputation, computability theory, Turing machine, super-Turing, X-Machine, Universal Quantum Computer; historical introduction.

ABSTRACT

In our paper [Sta90], we described a model of computation – the Analog X-Machine – with arguably ‘super-Turing’ computational power. Subsequent work has been reported, both at a seminar held at Sussex University, and via the *Connectionists* electronic bulletin board. Following enquiries from some twenty sites world-wide, it is clear that, although much interest exists in this general area, few researchers are aware of existing results. Indeed, some have expressed surprise that ‘non-Turing computation’ could be a meaningful concept in the first place. In this paper, accordingly, we set out those few results which we have managed to trace in the literature, together with our own recent thoughts, and identify possible areas of interest over the next few years.

We have tried throughout to write with the technical non-specialist in mind; where appropriate, the more theoretically minded reader should consult the references for detailed proofs. A few technical details have been embedded in the text in the form of *notes*.

1. INTRODUCTION

There is a long standing claim within the computer science and mathematical logic communities which asserts that we can characterise, mathematically, all those functions which can possibly be

‘calculated’. In his forthcoming survey concerning the nature and status of this claim, the *Church-Turing Thesis* [CT], Antony Galton [Gal90] explains the problem succinctly:

[CT] asserts the identity of two classes of functions, the effectively computable functions on the one hand, and the recursive functions on the other ... [CT] is not a theorem, and thus cannot be proved, because one of the terms it contains, namely ‘effective computability’, cannot be defined precisely, but rather refers to our vague, intuitive idea of what constitutes computability.

This renders our task somewhat difficult; we are trying to demonstrate the existence of certain modes of behaviour that should, under any reasonable interpretation, be considered ‘computation’. but which are, nonetheless, non-recursive. Our objective, accordingly, it to convince the reader that, at the very least, the models we survey below are ‘reasonable’. While this may seem a somewhat ‘non-rigorous’ approach to take in a supposedly ‘formal’ paper, it parallels the approach taken in justifying [CT] itself.

We shall divide extant computational models into four classes. On the one hand, a number of results from the 1970s and early 1980s are concerned with operations that could, in principle, be carried out using analog machinery then available. Together with the Turing model itself, these form what we shall call the *Newtonian* models. On the other hand, more recent models, principally Deutsch’s [Deu85] *Universal Quantum Computer* (UQC), have arisen through the observation (which we uphold) that computer science should be seen not as a branch of mathematics, but of physics. We argue that one should, therefore, expect that to each model of physics there corresponds a model of computation. Those models of computation which correspond to ‘post-Newtonian’ models of physics constitute our second class, the *post-Newtonian* models.

The models in each of these classes are, in general, directly constructed in an attempt to describe ‘computation’. Consequently, they are very much ‘implementation-centred’. Our third class of model is more specification-centred. It is easily seen, for example, that many specification languages are capable of describing systems which are patently *not* recursive. It is of both practical and theoretical interest, therefore, to know in what ways we can *restrict* these languages, so as to guarantee that our specifications are implementable. Of course, the extent to which any given restriction succeeds in this goal will depend on our understanding of ‘computation’. so that categorising those restrictions which are in some sense ‘reasonable’ offers another means to categorising computational models. Those models which arise in this way constitute our third class, the *restrictional* models.

Finally, we introduce a fourth class of models, the *algebraic field* models. The motivation here comes from the following observation. One can argue that one’s concept of ‘computability’ changes as society’s rules for ‘correct construction’ advance. Thus, as the field of all reals constructible using ruler and compasses extends the field of rationals, so it is in turn extended by the field of all algebraic reals. The algebraic field is, likewise, extended by the field of Turing-computable reals. There are very real sense in which each of these fields could, at different historical epochs, be deemed ‘the computable numbers’. The question arises, therefore, whether there is some *algebraic* characterisation, applicable to subfields of the real line, which corresponds in some natural way to our evolving notions of ‘computability’. If so, it is reasonable

to argue that to *every* field possessing this characterisation there corresponds a model of computation.

Having described the four classes of computational mode in more detail, we ask how models in these classes may be related to one another. For instance, we shall argue that *CCS* [Mil89] is related both to a potentially realisable extension of Deutsch's UQC, which we call the *Full Quantum Computer* (FQC), and also, through restriction, to more standard models.

2. NEWTONIAN MODELS OF COMPUTATION

We begin our survey with a brief summary of the main result in our earlier paper; this survey starts where [Sta90] left off.

2.1 Turing machines and the AXM

The model of computation upon which we originally based our work was Eilenberg's *X-machine* [Eil74]. This is a conceptually simple model, equivalent in computational power to the Turing machine; it cleanly separates the control and processing aspects of those computational systems which it can simulate. The control structure of a system is represented by a labelled transition system, the arcs of which are labelled by relations defined on some underlying datatype X . Traversing an arc corresponds to implementation of the relation which labels it.

The representation of system control as a labelled transition system makes it simple to extend the model to analog systems. We simply insist that traversal of an arc is to be regarded as a time-evolving *process*, rather than as a discrete action.

The model so obtained is the *analog X-machine* (AXM). Its original conceptualisation suffers several shortcomings, which we now address in more detail. The reader is strongly advised to consult [Sta90] for clarification of the AXM model, although we give a very brief explanation here.

2.1.1 X-machines, Turing machines, and process calculus

One of the more familiar constructs in theoretical computer science is the *labelled transition system* (LTS). Such a system comprises three parts,

- a collection S of states;
- a collection L of labels; and
- for each label $l \in L$, a relation \rightarrow^l on S .

Intuitively, an LTS is best explained by drawing a picture. We construct a labelled graph with nodes S . An arc is drawn between two nodes s_1 and s_2 , and labelled with l , precisely when $\langle s_1, s_2 \rangle \in \rightarrow^l$. For example, *Figure 2.1.1(a)* represents the LTS defined by taking $S = \{a, b, c\}$ and $L = \{\alpha, \beta\}$, where $\rightarrow^\alpha = \{\langle a, b \rangle, \langle b, c \rangle\}$ and $\rightarrow^\beta = \{\langle a, c \rangle, \langle c, b \rangle\}$.

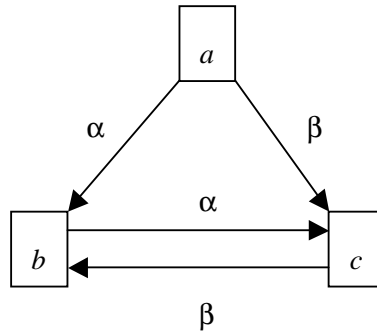


Figure 2.1.1(a)

An *X-machine* is an LTS on which extra structure is imposed. First we assume there exists some space X on which the machine is to operate (this is the ‘X’ in ‘X-machine’). Each label is taken from the space of all relations on X (technically, $L \subseteq [X \rightarrow 2^X]$).

Note. We can obtain an equivalent model if we restrict labels to represent *functions* on X , rather than relations, since every non-deterministic X -machine is equivalent to a deterministic 2^X -machine. However, we have not yet investigated the validity of this equivalence for the AXM, so retain the more general setting.

In order to ‘run’ an X-machine, we need to identify an initial state, and one or more terminal states. Suppose we can find some path through the LTS, which originates at the initial state and terminates at the terminal state. This path determines a relation on X , generated by composing the labels encountered along the path in the order in which they occur. The relation computed by an X-machine is then the union of all such path-computations within the LTS.

Those familiar with Milner’s process calculus *CCS* will recognise immediately that the X-machine model is very similar to the LTS structures used to give the calculus an operational semantics [Mil88]. This similarity, and the problems it highlights, are discussed in greater depth below.

The AXM is generated in much the same way as the X-machine. However, while we draw diagrams on *paper* (such as *Figure 2.1.1(a)*) to represent the control structures of X-machines, we imagine the diagram to be drawn in a *function space*, in the case of the AXM. To demonstrate that this is actually reasonable (*i.e.*, that we might produce an intuitively acceptable computational model by doing this), we consider a simple example, and then discuss the problems inherent in formalising our intuitions.

Consider the schematic diagram of a water-tank (*Figure 2.1.1(b)*). If the tap at the base of the tank is opened, water will flow from the tank. At what rate will this water be flowing a time t after the

tap is opened? Obviously, the answer will depend on how much water was in the tank at the time the tap was opened.

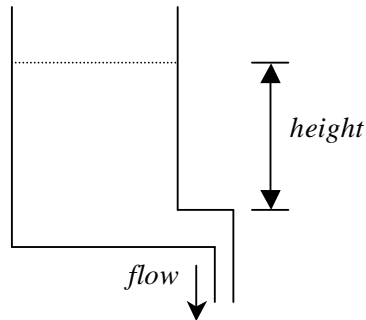


Figure 2.1.1(b)

Accordingly, we can argue that to each time t there corresponds a function

$$flow_t: \text{height of water} \rightarrow \text{flow rate}$$

Consequently, we can regard the variable t as parametrising a *family* of functions, or equivalently (in this particular case) as a path in the space of functions $[0, \infty) \rightarrow [0, \infty)$.

2.1.2 Reformulation of the AXM model: outstanding problems

Readers of [Sta90] will readily observe that the *semantics* of an AXM are given only sketchily. Formalising these semantics is non-trivial. We have stated above our current view that the transition from X-machine to AXM is tantamount to introducing a concept of arc traversal as a time-evolving process. Intuitively, we assumed that ‘time’ here meant ‘real-time’, but of course there is no guarantee that this is the best choice. In particular, if we are to compare the AXM with *post-Newtonian* models, we need to ask what models of time are best suited to these models. For example, it is a standard result of non-classical physics (*i.e.*, physical models which incorporate the uncertainty principle) that no temporal measurement can be made exactly. Now, we could develop a new semantics for each model of time we wished to investigate, but this seems wasteful. Instead, we should attempt to find some class of semantic model which could be effectively parameterised by ones temporal model.

Doing so, while preserving computational realism, is likely to be difficult. For example, if one upholds the view that the various aspects of the world that are amenable to observation are inextricably linked to one another, then it is unlikely that an underlying computational ‘context’ can be found, into which *any* temporal model could be ‘slotted’. Indeed, since the AXM model implicitly involves asynchronous realtime concurrent computation, one has to resolve one of the model difficult questions in the theory of concurrent semantics, namely the correct relationship between time and causality. We are aware of *no* models which address this problem in sufficient detail for our purposes, although we incline towards Murphy’s view [Mur90a, Mur90b] that time

should not be ‘thrust upon’ extant non-temporal concurrent models (see also [MoT89, Tof89, Tof90] for opposing views, however), but that a new model is needed which treats these aspects equably. Accordingly, we envisage the need for a complete reformulation of the AXM, if these more difficult tasks are to be undertaken.

Another unresolved problem arising from the need to compare the AXM with post-Newtonian models of computation arises from quantum mechanical interpretations of the nature of *observation*. According to quantum theorists [FrT90], we can think of an unobserved system as existing in a superposition of different states; the effect of observing the system is to force this superposition to *collapse* into a single state, namely the state observed. Let us consider how this would be represented within the AXM model. According to this model, we represent the evolution of the system in terms of movement through a function space, where this movement is parameterised by position within the control graph. The possibility that a system can *discontinuously* move into a new state requires us to allow movement within the associated function space to be discontinuous as well. Consequently, we need to decide precisely what function space topologies we are prepared to accept, and which ‘paths’ represent feasible evolutions through these spaces. We can expect no help from the topology itself, since we have shown elsewhere [Sta86, Sta88] that function space topologies can be rather general – in fact *every* topological space \mathfrak{S} can be embedded naturally within a corresponding function space.

Topological note. Let \mathfrak{S} denote some fixed but arbitrary topological space. Let p be any topological property satisfied by all singleton subsets of \mathfrak{S} , and such that whenever two subsets of \mathfrak{S} satisfy p , so does their union. For each such property p there corresponds a topology on the group $G(\mathfrak{S})$ of all autohomeomorphisms of \mathfrak{S} . We denote by $G_p(\mathfrak{S})$ this topological group. Then, for each such property p , we can embed \mathfrak{S} as a subspace of the function space $C(G_p(\mathfrak{S}), \mathfrak{S})$. Moreover, under this embedding, $\beta\mathfrak{S}$ becomes a retract of $\beta C(G_p(\mathfrak{S}), \mathfrak{S})$. See [Sta86, Sta88] for detailed proofs.

Finally, we should note that the title of [Sta90] is misleading. Although we have claimed that the AXM can solve the Turing Halting Problem, we have *not* shown that every Turing computable function is AXM-computable. Until we do so, we are not justified in referring to it as a *super-Turing* model, merely as a *non-Turing* model.

2.2 Traditional analog models

Our main sources concerning the computational power of standard analog machines are taken from the papers of Pour-El and Richards [Pou71, Pou74, PoR81], together with that of Myhill [Myh71], and other sources referenced therein. As can be seen from this list, many of these results are very old, going back two decades in some cases.

On the assumption that a certain standard collection of analog components are available, Pour-El demonstrated as early as 1970 that a real-valued function of the reals is *analog generable* precisely when it can be expressed in terms of certain non-linear differential functions. It is then shown that there are necessarily recursive functions which are *not* analog generable.

Consequently, we may conclude that the class of (standard) computable functions is *not* wholly included within the class of analog generable functions.¹

On the other hand, Pour-El and Richards' 1981 work demonstrates that the operation of recursively constructed physical systems can (although this interpretation necessarily assumes a Newtonian view of the world) result in the production of *non-recursive* numbers, after a recursive amount of time. This answers a question we touched on in [Sta90] – we were unaware of these results at that time – namely that a physical system can, in principle, be *designed* using a Turing machine, which is nonetheless not *simulable upon* a Turing machine.

This can be seen as providing direct evidence *against* [CT]. For certainly, we can regard analog generable functions as being 'effectively computable', and moreover, we would expect that the class of 'effectively computable' functions should be closed under composition (intuitively, it should be possible to use the output of one computable function as the input to the next). Nonetheless, the composition of Turing design-systems with analog evolution can yield a *nonrecursive* function.

Overall, the conclusion to be drawn from these two examples is that

neither of the classes 'computable functions' and 'analog generable functions' is larger than the other – they are merely different.

Further evidence against [CT] can be derived from Myhill's 1970 work (published in 1971), which demonstrates a recursive function, whose (continuous) derivative is *not* recursive.

3. POST-NEWTONIAN MODELS

Our starting point here is Deutsch's 1985 *Universal Quantum Computer* (UQC) [Deu85] (Deutsch's paper gives details of earlier quantum models). The UQC is, as Deutsch readily admits, incapable of performing operations that a Turing machine cannot. Nonetheless, it is arguably capable of 'outperforming' the Turing machine, in the sense that the UQC-complexity of a problem may be strictly lower than its Turing-complexity.

Deutsch's model is, however, more limited than need be the case. Since all computing devices are physical systems, he seeks to determine when such a system may be perfectly simulated by a computing machine. He develops the following argument (*pp.* 99-100)

'Every finitely realizable physical system can be perfectly simulated by a universal model computing machine operating by finite means' ... The 'finitely realizable physical systems' referred to ... must include any physical object upon which experimentation is

¹ *Note added July 2000* – We initially took this to mean that analog computation can't simulate digital computation, but this deduction is logically unfounded. Pour-El's result says only that the analog systems *under discussion* aren't capable of generating all recursive functions. It doesn't necessarily rule out the possibility that other analog models, *e.g.* analog recursive neural nets, might be powerful enough.

possible. The ‘universal computing machine’, on the other hand, need only be an idealized (but theoretically permitted) finitely specifiable model. The labellings implicitly referred to ... must also be finitely specifiable.

However, the UQC is more restricted than this argument requires, *viz.* (pp. 102-3)

Like a Turing machine, a model quantum computer Q consists of two components, a finite *processor* and an infinite *memory*, of which only a finite portion is ever used. The computation proceeds in steps of fixed duration T , and during each step only the processor and a finite part of the memory interact, the rest of the memory remaining static.

The fact remains that a finitely specifiable model *need not* yield a finite processor. For example, although a differential equation (and in particular Schrödinger’s equation [for the neutral Hydrogen atom]) can be regarded as a finite specification, it may actually specify *infinitely* many solutions. If we regard each solution as representing a ‘state’ within a solution space, then it is possible that no finite basis can be found with respect to which all solutions can be expressed. In such a situation, we would effectively have defined an infinite-state machine. Deutsch attempts to circumvent this problem when he observes (p. 100)

If we think of a computing machine as proceeding in a sequence of steps whose duration has a non-zero lower bound, then it operates ‘by finite means’ if (i) only a finite subsystem (though not always the same one) is in motion during any one step, and (ii) the motion depends only on the state of a finite subsystem, and (iii) the rule that specifies the motion can be given finitely in the mathematical sense (for example as an integer).

We have already identified our dissatisfaction with his point (iii). But points (i) and (ii) also cause us difficulties. For, in order to identify the relevant ‘finite subsystems’, one must first state what it means for such a subsystem to exist. In particular, one must explain how a boundary can be drawn around part of a physical system, in such a way that there is no coupling between entities within, and outside, this boundary. Indeed, Deutsch himself states that this is impossible (p. 109)

... every realizable physical system interacts with other systems, in certain states. But the effect of its dynamical coupling to systems outside itself cannot be reduced to zero by a finite process because the temperature of the correlation degrees of freedom would then have been reduced to zero. Therefore there can be no realizable way of placing the system in states on which the components of the time evolution operator which mix internal and external degrees of freedom have no effect.

Accordingly, there is no *a priori* justification for placing finiteness restrictions on the spatial extent of a system.² While we may be observing effects locally, we cannot know that they do not depend, in part, on causes at arbitrarily distant locations.

Finally, we note that it comes as no surprise that the UQC is only as powerful as the Turing machine, since their designs are so closely related. Indeed, if we concentrate only on the temporal features of the two models (where we assume the non-deterministic version of the Turing

² Of course, this depends very much on whether you believe the Universe to be bounded or unbounded; see *e.g.* [Haw88, Hal87] for discussions of cosmology. Chris Tofts (Edinburgh) also points out [private communication] the significance of the $\Delta E\text{-}\Delta T$ uncertainty principle as a major difficulty to be surmounted if the FQC is ever to be regarded as a realisable device.

machine), they are identical, since in both cases the observer is led to perceive the processor evolving through finitely branching discrete time.

It is the case, as we shall show below, that *any* (finitely specified) machine whose control structure can be represented within a finitely branching discrete temporal model is output-simulable upon a finite-state Turing machine.³ It follows that, if ones computational model is to be non-Turing in power, the underlying temporal structure must be either non-discrete, or else not finitely branching.

3.1 The Full Quantum Computer

We consider the computational power that Deutsch's UQC would embody, were it endowed with an infinite state set; we call this model the *Full Quantum Computer* (FQC).. Since we are interested only in the gross power of such a machine at this stage, and not with any questions of complexity, we can simplify proceedings by considering the analogous standard model, namely the infinite-state Turing machine. Of what is such a machine capable?

Again, the answer depends on the causal/temporal structure which we assume. If we assume that computation takes place in discrete steps, and that only finitely many of the available 'next-states' are available to the machine at any time, then we are again restricted to no more than Turing computational power. We have so far examined the effect of relaxing the two aspects of this condition independently, and even then our analysis is severely restricted. Since we have already shown that using non-discrete time leads to interesting effects, we concentrate instead on the effect of allowing processes to be infinitely branching.

The discrete time, infinitely branching FQC actually comprises *two* models, depending on whether computations are *allowed* to be infinitely branching at each step (but need not be), or else are *constrained* to be infinitely branching. Effectively, ones choice depends upon whether your model of quantum theory allows the existence of arbitrary finite as well as infinite non-determinism. We consider only the first possibility here (we have not yet examined the second possibility sufficiently).

Note first that we can use labelled transition systems to describe an operational semantics for the FQC, in exactly the same way as for CCS [Mil88].⁴ First we consider any particular instance of the computation's evolution as a path within a derivation tree, then for each arrow " $s \rightarrow t$ " within

³ This result has been noted by David Murphy (Glasgow), who commented in late 1990 that a process was almost certainly Turing-simulable precisely when its timing could be embedded topologically within Baire space. The proof given here is, however, original, since Murphy seems not to have published his observation.

⁴ The parallels between CCS and quantum theory are striking (at the observational level). Indeed, one can regard the CCS communication event as a direct analog of observation within quantum theory.

the tree we label it with the (formal) symbol st . (In general, of course, we would not be able to *construct* this derivation tree, even though we can – we claim reasonably – assume its existence.)

If we allow the FQC to be infinitely branching (but also allow arbitrary finite branching), then we have a computational model precisely equal in power to Milner’s CCS with infinite summation (with process identity defined to be observational equivalence), since we can use the LTS-descriptions of both systems to translate (non-constructively) between them.

3.2 Relativistic models

Another physical theory to have emerged during the twentieth century, General Relativity, also has a direct influence upon the study of computational systems, because of the way it undermines simultaneity.

Recall Einstein’s observer-on-a-train *Gedankenexperiment* [Ein20], in which he considered the movement of a train past an embankment.

Are two events ... which are simultaneous *with reference to the railway embankment* also simultaneous *relatively to the train*? We shall show directly that the answer must be in the negative.

We can extend this *Gedankenexperiment* to generate an apparent ‘paradox’ in concurrency theory. We envisage a box, equipped with a bell, a light, and two heat-sensitive panels, and suppose that a pulse of heat has just been generated at the midpoint of the box (as measured by an observer moving inertially with the box).

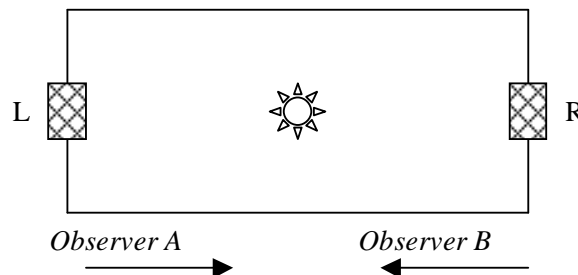


Figure 3.2(a)

According to an observer moving with the train, the heatwave is sensed simultaneously at the left (L) and right (R) sensors. *Observer A*, moving inertially to the right, would argue – since she can perceive herself to be stationary, with the box moving to the left – that sensor R is activated before sensor L. Similarly, *Observer B* argues that sensor L is triggered before sensor R.

So far, this is merely a restatement of Einstein’s original scenario. But let us now suppose that the box is a computer, known to be running the ‘program’ listed in *Figure 3.2(b)* below. According to *Observer A*, the bell should ring, while *Observer B* would predict that the light should flash. Now, the question of whether it is a bell which rings or a light that flashes is not open to debate; our

point is that the two observers *must* agree on the actual outcome of the experiment – the principles of Relativity do not allow for ringing bells to be somehow ‘transformed’ into flashing lights, simply because an observer happens to be moving.⁵

```
IF triggered(L) THEN BEGIN
    de-activate(R);
    ring(bell)
END;
IF triggered(R) THEN BEGIN
    de-activate(L);
    flash(light)
END.
```

Figure 3.2(b)

If we consider the outcome of this experiment, there are clearly three possibilities

- neither event occurs
- precisely one of the events occurs
- both events occur

In the first case, both observers will be forced to agree that something has ‘gone wrong’, since they are both expecting an outcome. In the second case, one of the observers will claim that the program is working correctly, but the other will be forced to conclude that it is *not* running correctly, since it produces the wrong output. In the final case, both observers will once again agree that the program has ‘gone wrong’, since they each observe an outcome which makes no sense from their point of view.

We are forced to conclude, therefore, that the concept of *correctness* (in its traditional pre/post-condition form) is a relative one. Moreover, we have the worst possible result: no matter what outcome actually arises in this experiment, it is *always* the case that an observer can be found who can assert that the program has failed.

⁵ *Note added July 2000:* This relativistic ‘paradox’ can be resolved by careful analysis of the physics involved, but the point of including it lay in the fact that it cannot be resolved within the computational frameworks available for modelling it – one has to step outside computer science and invoke physics instead. This situation is, we suggest, forced on us by computer science’s standard assumption that *process* can be considered independently of *substrate* (e.g. a process is a process whether it is instantiated as a silicon chip or in living tissue), which results in computational models that are largely divorced from physics. The desire to bring this supposed ‘paradox’ within the scope of computational argument has been an important factor in our ongoing re-conceptualisation of generalised X-machine computation.

It is not entirely clear how best to approach this problem, although one possible way forward would be the development of a ‘logic of relativity’, *viz.* a non-absolute logic, the semantics of which took into account not only standard logical requirements (*e.g.*, soundness), but which also allowed transformations of logical formulae, corresponding to transformations between inertial frames. A ‘relativistic Hoare logic’ (or Hennessy-Milner logic, or whatever) could then be used in the specification of systems, such as that described above, and the ‘paradox’ would evaporate.

Another approach to the problem arises through the study of non-interleaving concurrent semantics [KST90], which tackle much the same issues as are raised by the introduction of relativistic effects, namely the relationships between timing and causality [Mur90a, Mur90b]. We are currently attempting to devise a framework for such an attack.

4. RESTRICTIONAL MODELS

It is well known that standard logic can be used to specify systems that cannot be simulated on a Turing machine. Consequently, it is worthwhile asking in what ways different logics can be constructed, and to what extent they can be related to realisable computational models.

It is, perhaps, not entirely clear what we mean by this statement, and so we illustrate it by considering the (simple) relationship between CCS and specifiable sets.

4.1 Example: CCS and finitely specified sets

We observe that given *any* set that can be defined in any language using a ‘sensible’ keyboard, it can be generated by a CCS agent. We assume throughout that

- any well-formed specification (in whatever language) must be of finite length, and capable of being typed on a standard keyboard (comprising only a finite character set);
- the language *generated* by a CCS agent is the set of observable behaviour traces which it can exhibit.

4.1.1 Proposition

Let S be some set specifiable in the language L , where both the syntax and semantics of L can be expressed by a finite sequence of characters using the standard keyboard (*e.g.*, in a text book). (We shall call such a set *finitely specified*.) Then there exists a CCS agent which generates S .

Proof.⁶ Let $\langle L \rangle$ be the finite description of the language L , and $\langle S \rangle$ the finite description of the set S in the language L . Let *Agent* be the CCS agent $\Sigma \{ s. \mathbf{0} \mid s \in S, \text{ where } \langle S \rangle \text{ and } \langle L \rangle \}$. Then *Agent* generates (a copy of) S . ■

This proposition shows that the gross behaviour of CCS agents is generally not Turing-simulable. In fact, it is also easy to give specific examples of CCS agents which, although they generate Turing-computable sets, are not themselves Turing-simulable. This has important consequences, since it reflects the fact that considerations of, for example, the languages generated by machines, are not in themselves sufficiently powerful to express the *equivalence* of models.

4.1.2 Example: a random number generator

Consider the CCS agent $Random = \Sigma \{ n. \mathbf{0} \mid n \in \mathbf{N} \}$. The language generated by this agent is simply \mathbf{N} , the set of natural numbers, and this set is certainly generable by a Turing machine. However, the underlying process generating the set *cannot* be simulated. To see this, notice that *Random* is guaranteed to produce an output, but can do so in infinitely many different ways. On the other hand, Turing machines are boundedly non-deterministic, and so any proposed Turing-simulation would also be capable of failing to terminate.

This raises an interesting question, which we have not yet investigated. We have already seen that every finitely specified set is CCS-generable. This justifies our claim that every finitely specified machine whose control structure can be represented within a finitely branching discrete temporal model is output-simulable upon a finite-state Turing machine, since the output of any such machine is inherently generable by a CCS agent using finite summation. However, as we have just observed, it does not automatically follow that every finitely specified *process* is CCS-simulable.

Do there exist finitely specified processes which are not CCS-simulable?

The fact that CCS is so powerful is rather disturbing, given its use as a specification language for distributed systems. Unless one is certain that the concepts ‘CCS-simulable’ and ‘effectively computable’ coincide, one is led to the conclusion that designers may easily define systems which are not implementable. Of course, this problem is not restricted to the use of CCS only; many other extant languages also allow the specification of non-Turing processes.

⁶ *Note added July 2000:* The original definition invoked an (undefined!) family of observable output symbols $\{out_s \mid s \in S\}$, where out_s generates the output string s . This complication was entirely unnecessary – as long as we don’t allow s to be the silent action, we may as well write s as out_s .

It is important to determine, therefore, ways in which accepted specification languages and logics need to be restricted, in order to guarantee that all specifiable systems can be implemented. For example, we observe that a CCS agent is Turing-simulable precisely when it can be expressed without the use of infinite summation.

4.1.3 Proposition

Let *Agent* be a CCS agent. The following are equivalent:

- a) *Agent* can be specified without infinite summation;
- b) *Agent* is Turing-simulable.

Proof. (a \Rightarrow b) There exist software tools, *e.g.*, the Concurrency Workbench [Wal89] capable, in principle, of simulating such agents. (b \Rightarrow a) Let *Prog* be a Turing simulation of *Agent*. Then, since Turing machines are boundedly non-deterministic, it is sufficient to show that a well-defined procedure exists for representing programs as CCS agents, which preserves finiteness of branching structures. But such a procedure is easily established (see *e.g.*, [Wal89], where six mutual exclusion algorithms are converted into CCS format). ■

It would be interesting to know whether computational models exist with gross power strictly greater than that of the Turing machine, and strictly less than that of CCS. For example, does such a model result if we restrict CCS by requiring all recursive constructs to involve only finite summation, while allowing infinite summation elsewhere within the agent specification?

5. ALGEBRAIC FIELD MODELS

Students of the history of mathematics will be familiar with the progression of different views as to what precisely constitutes a ‘constructible’ number. To the early logicians the answer was simple: a number was constructible precisely if it was rational. If one allows the use of ‘ruler-and-compass’ constructions, the collection of constructible numbers is somewhat larger, since one is able to construct numbers which satisfy certain polynomials of even order. For example, given any rational r , and any positive integer m , we can construct the positive $(2^m)^{\text{th}}$ root of r . However, some theorists might like to go further, and suggest that any algebraic number should be regarded as constructible,⁷ since we can use standard iteration techniques to approximate it arbitrarily closely. Moreover, there are certain mathematical constants that might be considered constructible even though they are not algebraic, such as e and π .

⁷ A number is *algebraic* over a field F provided it satisfies a polynomial with coefficients in F . When used without further qualification, the term *algebraic* usually means *algebraic over \mathbf{Q}* .

According to these different viewpoints, there are several historical candidates for the title ‘the set of constructible real numbers’. We have mentioned four so far, namely

- \mathbf{Q} , the field of rationals
- *Ruler-and-compass*, the field of numbers constructible from \mathbf{Q} using these tools
- \mathbf{A} , the field of algebraic reals
- $\mathbf{A}[\pi, e, \text{etc}]$, the subfield of \mathbf{R} generated by augmenting \mathbf{A} with various standard non-algebraic mathematical constants.

In general, the feature which each of these sets have in common is that they all form countable subfields of \mathbf{R} . Finally, of course, we can construct the collection of *Turing-generable* reals. This comprises all numbers x , such that a program can be written whose only output is the decimal expansion of x (this may be a non-terminating program, of course). Once again, this is a countable subfield of \mathbf{R} , containing each of the four fields described above.

Indeed, we would argue that any candidate for this title must intuitively form a subfield – not just a subset – of the reals, since it is unreasonable that we should be able to construct two numbers, but not their sum, difference, product and quotient (assuming that the divisor is non-zero). It is not so clear whether this subfield should necessarily be countable.

So far, we have considered the relationship between constructibility and the subfield of \mathbf{R} in one direction only. That is, we have given a finite specification of a set of numbers which we consider to be constructible, and have observed that this set forms a countable subfield of \mathbf{R} . But one can ask certain questions in reverse. If we regard ‘constructible’ as synonymous with ‘effectively computable’, then it is sensible to determine precisely which subfields of \mathbf{R} correspond to intuitively sensible definitions of ‘constructibility’. Once again, our investigations here are only just beginning, and we would welcome further discussion; initial questions might include:

- is every countable of \mathbf{R} finitely specifiable?
- do there exist uncountable proper subfields of \mathbf{R} which are finitely specifiable?
- is every real number CCS-generable?
- for which subfields F of \mathbf{R} do there correspond models of computation M , for which F is precisely the field of M -computable numbers?
- assuming that some characterisation can be found which identifies precisely those fields which correspond to computational models, can this characterisation be given entirely in algebraic terms (so that [CT] can be re-expressed in terms of other mathematical axioms)?

6. SUMMARY AND FURTHER QUESTIONS

It should be clear to the reader that this work is very much in its early stages, and that only a few of the terms used above have been rigorously defined. Accordingly, such proofs as have been included rely heavily on the reader's intuition. As we pointed out at the beginning of the paper, however, our main tenet – that there are many effectively computable processes which are not Turing-computable – cannot be supported by anything other than an intuitive argument. Nonetheless, it is to be hoped that standard definitions will be agreed upon by those working in the field, so that outstanding questions can be tackled sooner rather than later. In addition to the questions embedded within the text, we note that any successful refutation of [CT] brings with it a number of related problems. Of particular immediate interest to the author are:

- complexity theory is based on the notion of Turing-machine computations, and the length of input tapes. For computational models which act on non-discrete input domains, or which use non-discrete temporal control systems, these notions will need extensive revision.
- how does one express the notion of a language generated by a non-Turing machine so as to take into account the manner in which it is produced (*cf.* the discussion of the CCS agent *Random* above)?

Topological/Concurrency question. Consider a continuous-time process which is always able to communicate a symbol a . In terms of trace language theory, we can make transfinite observations of this process, and since all ordinals embeddable in \mathbf{R} are countable, we know that the underlying ordinal base of any such trace is itself countable. Suppose we could define a metric on the space of such traces. Then by 'forgetting' the action a , we would have defined a metric on the first uncountable ordinal – a space which is well-known not to be metrizable. Consequently, no such metric can be defined.

Does it follow that no metric can be defined which gives sensible information regarding the 'distance' between two continuous-time processes?

7. REFERENCES AND RELATED READING

KST \equiv [KST90] below

- [Deu85] Deutsch D. (1985) Quantum theory, the Church-Turing Principle and the Universal Quantum Computer. *Proc. Royal Soc.*, **A400**, 87-117.
- [Eil74] Eilenberg S. (1974) *Automata, Languages, and Machines (vol. A)*. Academic Press.
- [Ein20] Einstein A. (1920) *Relativity – the special and the general theory*. Methuen.
- [FrT90] French A.P., and Taylor E.F. (1990) *An introduction to quantum physics*. MIT Introductory Physics Series, Chapman and Hall.

- [Gal90] Galton A. (1990) The Church-Turing Thesis: its nature and status. *AISBQ* **74**, 9-19. (Also to appear in the *Proceedings of the 1990 Turing Conference*, OUP.)
- [GSW87] Green M.B., Schwarz J.H., and Witten E. (1987) *Superstring Theory* (2 vols). Cambridge University Press (reprinted 1988 with corrections).
- [HaH70] Hartley B., and Hawkes T.O. (1970) *Rings, modules and linear algebra*. Chapman and Hall.
- [HaI87] Hawking S., and Israel W. (eds) (1987) *Three hundred years of gravitation*. Cambridge University Press (reprinted 1989 with corrections).
- [Her64] Herstein I.N. (1964) *Topics in Algebra*. Wiley.
- [Hob23] Hobson E.W. (1923) *The domain of natural science*. Cambridge University Press (also available from Dover Publications, 1968).
- [HoY61] Hocking J.G., and Young G.S. (1961) *Topology*. Addison-Wesley (also available in corrected form from Dover Publications, 1988).
- [Kil75] Killingbeck J.P. (1975) *Techniques of applied quantum mechanics*. Butterworths.
- [KST90] Kwiatkowska M.Z., Shields M.W., and Thomas R.M. (1990) *Semantics for concurrency, Leicester 1990*. Springer Workshops in Computing.
- [Kwi89] Kwiatkowska M.Z. (1989) *Fairness for non-interleaving concurrency (PhD Thesis)*. Technical Report **22**, Dept of Computing Studies, University of Leicester, Leicester LE1 7RH, United Kingdom.
- [Mil88] Milner R. (1988) *Operational and algebraic semantics of concurrent processes*. Technical Report **ECS-LFCS-88-46**, LFCS, Department of Computer Science, University of Edinburgh, The King's Buildings, Edinburgh EH9 3JZ, United Kingdom.
- [Mil89] Milner R. (1989) *Communication and Concurrency*. Prentice-Hall.
- [MoT89] Moller F., and Tofts C. (1989) *A Temporal Calculus of Communicating Systems*. Technical Report **ECS-LFCS-89-104**, LFCS, Department of Computer Science, University of Edinburgh, The King's Buildings, Edinburgh EH9 3JZ, United Kingdom.
- [Mur90a] Murphy D.V.J. (1990) Approaching a Real-Timed Concurrency Theory. *KST* 295-310.
- [Mur90b] Murphy D.V.J. (1990) *Time, Causality and Concurrency (PhD Thesis, University of Surrey)*. Technical Report **CSC 90/R32**, Department of Computing Science, University of Glasgow, Glasgow G12 8QQ, United Kingdom.

- [MyH71] Myhill J. (1971) A recursive function, defined on a compact interval and having a continuous derivative that is not recursive. *Michigan Math. J.* **18**, 97-8.
- [PoR81] Pour-El M.B., and Richards I. (1981) The wave equation with computable initial data such that its unique solution is not computable. *Advances in Mathematics* **39**, 215-39.
- [Pou71] Pour-El M.B. (1971) Abstract computability versus analog-computability (a survey). *Cambridge Summer School in Mathematical Logic*, Springer Lecture Notes in Mathematics **337**, 345-60.
- [Pou74] Pour-El M.B. Abstract computability and its relation to the general purpose analog computer (some connections between logic, differential equations and analog computers). *Trans. AMS* **199**, 1-28.
- [Rin82] Rindler W. (1982) *Introduction to special relativity*. Clarendon Press.
- [Sta86] Stannett M (1986) *Internal Topology (PhD Thesis)*. Department of Pure Mathematics, University of Sheffield, Hicks Building, Sheffield S3 7RH, United Kingdom.
- [Sta88] Stannett M (1988) Representations of spaces as function spaces. *Glasgow Math. J.* **30**, 189-93.
- [Sta90] Stannett M (1990) X-Machines and the Halting Problem: Building a super-Turing machine. *Formal Aspects of Computing* **2**, 331-41.
- [Tof89] Tofts C (1989) *Timing Concurrent Processes*. Technical Report **ECS-LFCS-89-103**, LFCS, Department of Computer Science, University of Edinburgh, The King's Buildings, Edinburgh EH9 3JZ, United Kingdom.
- [Tof90] Tofts C. (1990) Timed Concurrent Processes. *KST* 281-94.
- [Vic89] Vickers S. (1989) *Topology via Logic (Cambridge Tracts in Theoretical Computer Science 5)*. Cambridge University Press.
- [Wal89] Walker D.J. (1989) Automated analysis of mutual exclusion algorithms using CCS. *Formal Aspects of Computing* **1**, 273-92.
- [Wey50] Weyl H. (1950) *Space Time Matter*. Methuen (also available from Dover Publications, 1952).
- [Wil70] Willard S. (1970) *General Topology*. Addison-Wesley.