The Church-Turing Thesis and Timed Computations (Draft)

Martin Berger*

September 22, 2004

Consider the following program.

```
every 31 seconds
{
        print "looking at my Gucci, it's about that time!"
}
```

Our contentions are simple.

1. This program is mechanically computable, hence it must be expressible in every model of mechanical computability.
2. One cannot verify that a given model of computation allows to express programs such as the above unless elementary steps of computations are assigned durations in the model. In particular, formalisms such as π-calculi, Turing Machines or μ-recursive functions do not allow this verification and can thus not express the program above.
3. Assigning durations to computations strictly extends conventional models of computation.

But let's begin at the beginning ...

The Church-Turing Thesis [2, 5] is one of the most famous and fundamental conjectures of the whole of computing theory. Its precise meaning and epistemological status are controversial but its validity is rarely doubted. To simplify matters, we shall avoid the problem of Church's and Turing's original intentions. We are interested in computation, or rather the limits of computation, not history. In our reading, the Church-Turing Thesis asserts that computation is a phenomenon with sharp conceptual boundaries, sharp enough in fact, to allow convincing mathematisation, for example by way of Turing Machines or π-calculi. Our contention above, then, may be summarised as claiming that the conventional demarcation of computation given by the Church-Turing Thesis, is too restrictive and ought to be liberalised.

When we speak of the Church-Turing Thesis, we have statements like the following in mind.

*M.Berger@dcs.qmul.ac.uk. Thanks to Alexis Richardson for various discussions about this paper.

> *The process of interaction with an environment, exchanging finite data in each interaction, by a mechanical process with a finite program of instructions, in accord with the laws of physics, apart from resource constraints, can be precisely simulated by a π-calculus process.*

Several issues are worth noting. Firstly, the relation "simulating computation" is one between a mathematical formalism and something informal, intangible, such as physical processes. Hence it seems in principle impossible to mathematically decide if a given formalism models computation. Secondly, although there are good reasons to prefer interaction-based formalisms, the reference to the π-calculus could be replaced by other models such as Turing Machines or μ-recursive functions. Finally, it is left open what it means for a formalism to be "precisely simulated".

The first two points, while raising interesting issues, do not concern us here. It is the last point that is found wanting once one thinks seriously about *timed* computation. The problem is not that incorporating some notion of time into a model of computation would allow to solve the halting problem, although we have not seen a proof that such a drastic extension of computational power is impossible. The problem is that the received form of the Church-Turing Thesis excludes timed algorithms, such as our example above, tout-court from the realm of mechanical computability. Conventional models of computation simply do not have temporal properties. We propose to ameliorate this shortcoming by integrating the passing of physical time into models of computation. Extending the work of this thesis, a *real-timed π-calculus* would be the result, where each reduction step is assigned its duration, a real number greater than 0. The Church-Turing Thesis could then be rephrased.

> *The process of interaction with an environment, exchanging finite data in each interaction, by a mechanical process with a finite program of instructions, in accord with the laws of physics, apart from resource constraints, can be precisely simulated by a real-timed π-calculus process.*

Before delving into the details of our criticism, we would like to emphasise the speculative nature of this undertaking: we are neither fully convinced of its correctness nor can we provide the details of how an appropriate real-timed π-calculus would look like. In addition, one must distinguish two things:

- The problem under discussion here, whether models of computation ought to be augmented so as to enable them to deal with timed computation.
- The question of the truth or falsehood of the temporally extended Church-Turing Thesis. After all, the existence of universal models of computation may be peculiar to the world of untimed computation. Maybe timed computation is a fundamentally more vague concept.

The following points are also worthwhile to be borne in mind.

- We do not address the issue of just what kinds of duration assignments are legitimate. Would it be a good idea to allow the assignment of 7 seconds to every computational step in a terminating computation and 10 for all others? Wouldn't that allow a timed observer to solve the halting problem? It may be necessary to permit only computable (in the conventional sense) assignments. Another problematic duration assignment was proposed in the context of *Accelerating Turing Machines* [3]: the n^{th} step in the computation takes $\frac{1}{2^n}$ seconds. Accelerating Turing Machines can decide the halting problem in finite time, although taking an infinite number of steps in the process. Since all the available evidence suggests that Accelerating Turing Machines are not in accordance

with the laws of physics, such *Xeno*-assignments should probably also not be admissible.

- Is it sufficient to extend conventional models of computation with a notion of duration to capture all of timed computation? This might not be the case. It could be necessary to integrate the passing of time more tightly with the computational process, for example by adding timers. We will discuss this problem only very briefly.

After this little excursion we will now defend and discuss the three claims above. Regarding (1), the strongest argument in favour of our little program's being mechanically computable is that it can easily be written with any modern programming environment on run-of-the-mill hardware. All that's required is a conventional computer and (maybe) a clock. Both are prototypical mechanical devices [6] and there does not seem to be a reason to believe that this is no longer true for their combination. If the Church-Turing Thesis wants to be taken seriously, it better allowed to express trivial and ubiquitous programs such as the initial example.

Our argument for (2) has two parts. First, if we assign durations to all computational steps, we can clearly decide (ignoring for simplicity the problem of the decidability of the assignment) if the model generates what we interpret to be appropriate temporal behaviour. Secondly, we could not think of a way of doing this verification without an assignment of durations.

In defence of (3), it is clear that there is no such assignment in conventional models of computations. Hence assignments do extend these models, at least set-theoretically.

We expect the following types of (related) reactions from defenders of the Church-Turing Thesis.

The "Old-School Recursion Theorist's" Reply: *Of course Turing Machines can simulate this algorithm. Its timing is an inessential detail that can and should be ignored. All that matters is the function being computed, not the timing of the computational process. After all, even normal untimed computation proceeds in physical time and space but we don't care about this when pondering the essence of computation.*

This objection may be the easiest to deal with. Timed algorithms abound in the very fabric of computation: for example network flow-control algorithms such as TCP [7] or various operating systems' schedulers [1] are implemented using timers and their temporal properties are vital to these algorithms. Of course the Old-School Recursion Theorist could object that such claims misunderstand what it means for an algorithm to function: the point of models of computation is to abstract away from pesky little details like actual execution time. Well, maybe, and in case of the computation of, say, the factorial function it might be reasonable to ignore how long its computation takes. Why? Because we get something else: a number as a hard and fast result. We can "measure" the correctness of the algorithm by looking at this result. In the case of network flow-control algorithms or OS schedulers, this is very different. The correctness of any implementation seems directly and indissolubly connected with temporal properties. Crudely put, a scheduler is something that guarantees my processes access to the CPU at least every 250 milliseconds. TCP ensures certain rather involved ratios between network capacity and transmission speed. It makes little sense to talk about the correctness of a scheduler or TCP without mentioning time. To appreciate the significance of the temporal aspects involved, consider the following transformation of sequential composition.

$$[\![\mathsf{P}\ ;\ \mathsf{Q}]\!] = [\![\mathsf{P}]\!]\ ;\ \texttt{sleep(}\ \texttt{1 year}\ \texttt{)}\ ;\ [\![\mathsf{Q}]\!]$$

If timing was an inessential detail to TCP or OS schedulers, we could apply this transformation without changing their semantic essence. Unfortunately, applying $[\![\cdot]\!]$ to all the world's C programs would render the entire planet's computational infrastructure humanly unusable. While that is an irrelevant detail sub speciae aeternis, it appears flippant to say that the intended semantics of such programs remains unaffected, at least for humans with an average lifetime of 54 years.

The Old-School Recursion Theorist might now reply that one can say the same thing about factorial functions: we could not humanly evaluate $!n$ for most n, if we'd apply the transform. But does that not suggest that timing is relevant even for factorials?

The "Separate Physics and Computation!" Reply: *It does not matter if Turing Machines can simulate timed computation or not because timed computation is not about pure algorithms. They are interactions of pure algorithms with physical devices, in this case clocks. The Church-Turing Thesis is only concerned with pure computation. The addition of clocks is no more relevant than the possibility of having computers extended with loudspeakers to produce sound or with wheels to allow physical movement.*

We could just say: "fair enough, but if timed computation is not computation, it is nevertheless something closely related and at least as interesting. It is also worth of mathematisation and hence of a timed equivalent of the Church-Turing Thesis."

But this is too conciliatory. Let's look at the defender's argument in more detail. It is a variant of the first objection and based on a distinction between (models) of pure algorithms and physical devices, the implication being that the latter have no role to play in the description of the former. This is problematic for two reasons. Firstly, ultimately every actual computation is a physical process and many models of computation idealise these physical processes and devices to some degree. An example would be the tape and head of a Turing Machine. So it cannot be the inclusion of models of physical entities in formalisms for computing that is deemed problematic, it is the inclusion of a specific kind of physical entity: the clock. But, and secondly, every model of computation (that we can think of) uses some form of sequentialisation: "this must happen, then that", clearly a temporal property. Even domain-theoretic models use fixpoint iteration [4] that is usually imagined as a discrete, temporal process. It seems fair to say that time-stepping or discrete sequencing is intrinsic in all models of computation already. What is missing in most models is an explicit *duration* of the computational step. Once durations are specified, questions such as: "does this program implement that algorithm which has these temporal properties?" seem natural. If these temporal properties never mattered, we would be justified to just have a generic notion of discrete sequencing in the models, as, for example, given by the reduction steps in λ-calculi or Turing Machines. But, as we have argued above, the temporal properties do matter a lot in many situations, so it seems appropriate to model them explicitly because they are not induced by interaction with some arbitrary physical device, they come from the physical behaviour of something essential to computation itself: the flow of time.

Another problem with "Separate Physics from Computation!" is that our initial example does not actually require interaction with a clock, although in practise, implementations will. Conventional CPUs are constructed such that their computational steps are executed within tight time bounds. It may be possible to use these bounds to achieve the required temporal behaviour by judicious choice of translation into machine code alone. Of course these time bounds are almost always achieved with a clocking mechanism that "drives" the CPU. But this mechanism is not *explicit* in

the code being executed. It is also possible, although currently unusual, for CPUs to lack an explicit clocking mechanism. That does not mean however, that computations executed on such devices have no temporal properties. It is just that these properties emerge from the physical properties of the executing hardware in a way that observers describe as clockless. Whether this is useful terminology touches on interesting problems (summarised by the question "what is a clock?"), that we do not wish to discuss further here.

The "That's what I'm Saying!" Reply: *Of course Turing Machines can simulate these algorithms. Just assume one step of a Turing Machines corresponds to one unit of time of the algorithm. In other words, conventional models of computation implicitly assign unit time, normalised to 1, to each computational step.*

Yes, but this assignment needs to be done. Why pretend it is not part of the computational model? This answer assumes an assignment, it is just not honest about it.

It also suffers from a more serious defect. It is not enough to simply fix an assignment once and for all, because every such assignment induces a minimal granularity of time and hence excludes certain more finely timed algorithms. If, for example, we'd assign 47 seconds as the duration of every computational step, our initial example would not be computable. Unless there is convincing empirical evidence of a physical limits to time-divisibility relevant to computation, none should be enforced by our models of computation.

OK, Now We Have Durations, Are We Done Yet?

The arguments adumbrated above compellingly suggest to include a notion of duration into models of computation. But is that all we need to capture timed computation? Consider the following variant of our initial example.

```
on 14:Jan:1986 at 14:01:0221 GMT
{
        print "looking at my Gucci, it's about that time!"
}
```

Some, but not all of our arguments can be adapted to support the additional inclusion of an absolute notion of time into models of computation to accommodate this example.

Even ignoring the problem of absolute time and its relation to computation, is our augmented Church-Turing Thesis expressive enough? As already alluded to, many timed algorithms implement their temporal behaviour with the help of timers. Is it always possible, just armed with conventional models of computation augmented with durations, to simulate behaviour induced by the presence of timers? We are not sure. It might be possible to use busy-waiting to simulate timer driven interrupts, especially when there are no lower limits on the granularity of time steps and if the criteria that would warrant speaking of busy-waiting allowing to simulate timers, allow for some imprecision ...

This raises another issue: what does it mean to observe a timed computation? Should there be limits to observer's time-measuring abilities? Relatedly, is it appropriate to assign *exact* durations to computational steps? Wouldn't intervals or probabilities be better? Questions, questions ...

A Modest Proposal for a Research Programme

As pointed out, the Church-Turing Thesis is an empirico-philosophical assertion, not a mathematical one and cannot be verified or contradicted by purely formal means. Nevertheless, mathematical theorems can serve as evidence for or against its acceptance, the various mutual embeddability results of classical recursion theory being a prime example.

What kind of theorem could be significant evidence for or against the temporally extended Church-Turing Thesis? How about the following conjecture?

CONJECTURE 1 Let π_a be the asynchronous π-calculus and $\approx_a$ one of its reasonable equivalences, such as reduction-congruence or weak trace-equivalence. Let π_a^d be the asynchronous π-calculus extended with a reasonable notion of duration and $\approx_a^d$ one of its reasonable equivalences. Then, in general, there is no mapping $[\![\cdot]\!]$ from π_a^d to π_a such that

$$[\![\mathsf{P}]\!] \approx_a^d [\![\mathsf{Q}]\!] \qquad \text{iff} \qquad \mathsf{P} \approx_a \mathsf{Q}.$$

This conjecture is attractive not only because despite its simplicity, establishing its truth value appears difficult, but also because it assumes little apart from duration assignment, essentially only the availability of appropriate notions of equivalence. This should not be a surprise given, on one hand, our preceding discussion of the Church-Turing Thesis with its subtext "what does it mean to observe a timed computation?", and the tight connection between notions of observation and equivalence on the other. Indeed, it does not appear to be an exaggeration to say that our a solution of our problem of timed computation would be but a special case towards answers for the following two questions. What does it mean to observe computation? When are two computations equal?

References

[1] Maurice Bach. *The Design of the Unix Operating System*. Prentice-Hall, 1986.

[2] B. Jack Copeland. The Church-Turing Thesis. Entry in the Stanford Encyclopedia of Philosophy, http://plato.stanford.edu/entries/church-turing/.

[3] B. Jack Copeland and R. Sylvan. Beyond the universal turing machine. *Australasian Journal of Philosophy*, 1998.

[4] Carl A. Gunter. *Semantics of Programming Languages*. MIT Press, 1995.

[5] Rolf Herken, editor. *The Universal Turing Machine: a Half-Century Survey*. Springer, 1995.

[6] Giuseppe Longo. The difference between clocks and turing machines. In Arturo Carsetti, editor, *Functional Models of Cognition*. Kluwer, 1999.

[7] Andrew S. Tannenbaum. *Computer Networks*. Prentice Hall, 1996.