THE UNIVERSITY of EDINBURGH

Edinburgh Research Explorer

Computing is Interaction.

Citation for published version:
Milner, R 1994, 'Computing is Interaction.'. in Technology and Foundations - Information Processing '94, Volume 1, Proceedings of the IFIP 13th World Computer Congress. pp. 232-233.

Link:
Link to publication record in Edinburgh Research Explorer

Document Version:
Preprint (usually an early version)

Published In:
Technology and Foundations - Information Processing '94, Volume 1, Proceedings of the IFIP 13th World Computer Congress

Publisher Rights Statement:
Copyright © IFIP 1994. Reproduced with permission.

General rights
Copyright for the publications made accessible via the Edinburgh Research Explorer is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
The University of Edinburgh has made every reasonable effort to ensure that Edinburgh Research Explorer content complies with UK legislation. If you believe that the public display of this file breaches copyright please contact openaccess@ed.ac.uk providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 20. Feb. 2015

13th World Computer Congress 94, Volume 1
B. Pehrson and I. Simon (Editors)
Elsevier Science B.V. (North-Holland)
© 1994 IFIP. All rights reserved.

Computing is Interaction

Robin Milner[a]

[a] Laboratory for Foundations of Computer Science,
Computer Science Department, University of Edinburgh,
The King's Buildings, Edinburgh EH9 3JZ, UK.

Traditionally, computing has been overwhelmingly concerned with algorithms which find the answer to a well-defined question and deliver it. This was the case when the only computers were human; not surprisingly, the non-human variety followed the trend. The definite article in "*the* answer" is doubly definite; each time the computation takes place there will be one answer, and each time this answer will be the same.

So the words "algorithm" and "function" have dominated computer science. This has been true in at least two major branches of computation theory; one of these is the joint study of computational complexity and the analysis of algorithms, and the other is the semantics of computation using both logic and abstract mathematics (e.g. domain theory).

To present this world to the programmer, the inherent (unsavoury?) concurrency and interactivity among components of a computer was hidden below a (savoury?) platform consisting of the raw machine code; then higher languages only had to be concerned with abstractions which could well be explained – and implemented – in terms of this sequential basis. No doubt this was a good way to begin the serious study of computing; it avoided many awkward difficulties, and the result has been structures, concepts and methods which are natural and elegant both in theory and at the level of language.

In the early sixties, Carl-Adam Petri noticed that the automata theory which had made great strides since the fifties rested firmly upon the sequential platform, and was therefore inadequate to explain interactive systems. His invention of (Petri) net theory amounted to allowing a transition in a system A to depend upon, i.e. to interact with, a transition occurring in another system B running alongside A. Thus a theoretician was among the first to let the computer's concurrent, interactive subconscious break through the sequential barrier.

In practice, software appears to have begun to break through this barrier –naturally enough– with the advent of multiprocessors. It is not surprising that most of today's concurrent programming languages preserve the idioms of sequential programming (via such devices as monitors, remote procedure call etc) as much as they can, providing a top-dressing of concurrency and interaction. This preservation is salutary; the only discipline we really understand is (still!) the sequential one, and anarchy would reign without it.

However, this stratification (sequential below, interactive above) brings increasing discomfort. For one thing, as noted above, machines (at the lowest level) are not sequential! For another thing, it turns out to be impossible to give a semantic theory of interactive computation entirely in classical functional terms, if only because nondeterminism cannot

be avoided; therefore theories which match the stratified form of software tend to superimpose interactive notions on top of sequential ones, leading to redundancy. Thirdly, we have to admit that computer science –or informatics– is no longer just a matter of *prescibing* what will happen in the small world of a sequential computer; it must also *describe* what happens at large in parallel computers, in networks, in hybrid human-machine systems, and even in nature.

To develop a non-normative, descriptive science of computing and information we have to seek unifying low-level concepts, even if systems engineers will always need higher levels of modelling and explanation. The analogy with physics should be strong; atomic and subatomic notions are vital in physics, even if civil engineers don't commonly use them. There are strong trends in software design –notably the still obscure notion of *object*– which lead away from the functional, sequential model and towards the interactive model; there have also been independent theoretical explorations in that direction by Petri, Hoare and others, often employing the term "process" rather than "object".

The thesis of this lecture is that while we *cannot* base the whole of computation – both prescriptive and descriptive– upon the functional model, there are are encouraging signs that we *can* base it upon the notion of interaction. In this interactive model, a computational entity –a process or object– is nothing more than its ability to interact with others; moreover, the effect of such interaction is itself to change the availability of further interactions. Concepts like *link*, *reaction* and *neighbour* play a basic part in this model, in place of basic notions of the sequential model such as assignment or function call.

If this view of computation is correct then it will lead, not to anarchy, but to a rich repertoire of higher levels of explanation (of which the functional model is just one, but a highly important one), resting upon the interactive model at the lowest level.