Computational flux

Robin Milner

University of Cambridge, Computer Laboratory

Lecture summary

The lecture will be about a simple graphical model for mobile computing.

Graphical or geometric models of computing are probably as old as the stored-program computer, possibly older. I do not know when the first flowchart was drawn. Though undeniably useful, flowcharts were denigrated because vital notions like parametric computing –the *procedure*, in Algol terms– found no place in them. But a graphical reduction model was devised by Wadsworth [15] for the lambda calculus, the essence of parametric (functional) computing. Meanwhile, Petri nets [13] made a breakthrough in understanding synchronization and concurrent control flow. Later, the chemical abstract machine (Cham) [2] –employing chemical analogy but clearly a spatial concept– clarified and generalised many features of process calculi.

Before designing CCS, I defined *flowgraphs* [9] as a graphical presentation of *flow algebra*, an early form of what is now called structural congruence; it represented the *static* geometry of interactive processes. The pi calculus and related calculi are all concerned with a form of mobility; they all use some form of structural congruence, but are also informed by a kind of *dynamic* geometrical intuition, even if not expressed formally in those terms.

There are now many such calculi and associated languages. Examples are the pi calculus [11], the fusion calculus [12], the join calculus [5], the spi calculus [1], the ambient calculus [3], Pict [14], nomadic Pict [16], explicit fusions [6]. While these calculi were evolving, in the action calculus project [10] we tried to distill their shared mobile geometry into the notion of *action graph*. This centred around a notion of *molecule*, a node in which further graphs may nest. All action calculi share this kind of geometry, and are distinguished only by a *signature* (a set of molecule types) and a set of *reaction rules*. The latter determine what configurations of molecules can react, and the contexts in which these reactions can take place.

Such a framework does not necessarily help in designing and analysing a calculus for a particular purpose. It becomes useful when it supplies non-trivial theory relevant to all, or a specific class of, calculi. Most process calculi are equipped with a behavioural theory – often a labelled transition system (LTS), or a reaction (= reduction) relation, together with a trace-based or (bi)simulation-based behavioural preorder or equivalence. Developing this theory is often hard work, especially proving that the behavioural relation is preserved by (some or all) contexts. Recently [8] we have defined a simple categorical notion of *reactive system*, and shown that under certain conditions an LTS may be uniformly *derived* for it, in such a way that various behavioural relations –including the failures preorder and bisimilarity– will automatically be congruential (i.e. preserved by contexts). We have also shown [4] that a substantial class of action calculi satisfy the required conditions. Thus we approach a non-trivial general theory for those calculi which fit the framework, as many do.

This work has encouraged us to base the theory on a simpler notion: a *linear action graph*, one in which edges may not fork or merge; an example is shown in Fig. 1. These graphs are a generalisation of Lafont's interaction nets [7]. They consist just of nodes (with many ports) and edges, but with a *locality* –i.e. a forest structure– imposed upon nodes quite independently of the edge wiring. This notion has grown out of action calculi but is also inspired by the Cham of Berry and Boudol [2], the ambient calculus of Cardelli and Gordon [3], the language Nomadic Pict of Sewell and Wojciechowski [16], and the fusion concept of Parrow and Victor [12] further developed by Gardner and Wischik [6]. The intuition is that *nodes* have locality, *wires* (per se) don't. A node and its (nodal) contents can be an ambient, a physical location, a λ-abstraction, a program script, an administrative region, ... A node without contents can be a date constructor, a cryptographic key, a merge or copy node, a message envelope, ...

In the lecture I shall outline the basic behavioural theory of *shallow* linear graphs –those without nesting. I hope to indicate how this behavioural theory extends smoothly to nested graphs, and how the reaction rules in these can conveniently represent action-at-a-distance – a fiction which is essential in a higher level model of (say) the worldwide web. I shall also discuss how the non-linear theory may be recovered, by a form of quotient. This is work in progress.

Acknowledgement I would like to thank my colleagues Luca Cattani, Philippa Gardner, Jamey Leifer and Peter Sewell for their co-operation, inspiration and patience.

POPL '01 1/01 London, UK
 ISBN 1-58113-336-7/01/0001...$5.00

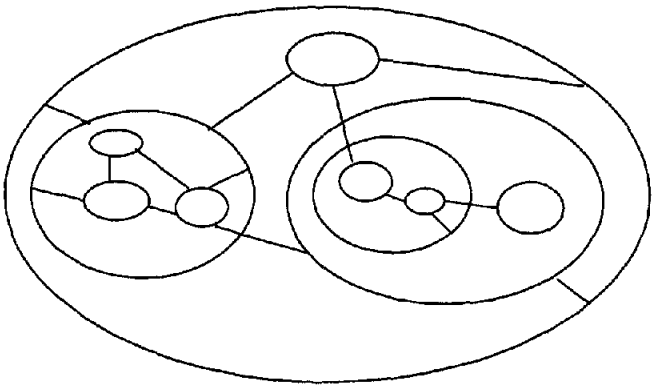

Fig. 1: A linear action graph

References

[1] Abadi, M. and Gordon, A.D. (1997), A calculus for cryptographic protocols: the spi calculus. Proc. 4th ACM Conference on Computer and Communications Security, ACM Press, 36-47.

[2] Berry, G. and Boudol, G. (1992), The chemical abstract machine. Journal of Theoretical Computer Science, Vol 96, pp217–248.

[3] Cardelli, L. and Gordon, A.D. (2000), Mobile ambients. Foundations of System Specification and Computational Structures, LNCS 1378, 140–155.

[4] Cattani, G.L., Leifer, J.J. and Milner, R. (2000),. Contexts and Embeddings for closed shallow action graphs. University of Cambridge Computer Laboratory, Technical Report 496. [Submitted for publication. Available at `http://www.cam.cl.ac.uk/users/jjl21`.]

[5] Fournet, C. and Gonthier, G. (1996), The reflexive Cham and the join calculus. Proc. 23rd Annual ACM Symposium on Principles of Programming Languages, Florida, pp372–385.

[6] Gardner, P.A. and Wischik, L.G. (2000), Explicit fusions. Proc. MFCS 2000. LNCS 1893.

[7] Lafont, Y. (1990), Interaction nets. Proc. 17th ACM Symposium on Principles of Programming Languages (POPL 90), pp95–108.

[8] Leifer, J.J. and Milner, R. (2000), Deriving bisimulation congruences for reactive systems. Proc. CONCUR2000. [Available at `http://www.cam.cl.ac.uk/users/jjl21`.]

[9] Milner, R. (1979), Flowgraphs and Flow Algebras. Journal of ACM, 26,4,1979, pp794–818.

[10] Milner, R. (1996), Calculi for interaction. Acta Informatica 33, 707–737.

[11] Milner, R., Parrow, J. and Walker D. (1992), A calculus of mobile processes, Parts I and II. Journal of Information and Computation, Vol 100, pp1–40 and pp41–77.

[12] Parrow, J. and Victor, B. (1998), The fusion calculus: expressiveness and symmetry in mobile processes. Proc. LICS'98, IEEE Computer Society Press.

[13] Petri, C.A. (1962), Fundamentals of a theory of asynchronous information flow. Proc. IFIP Congress '62, North Holland, pp386–390.

[14] Pierce, B.C. and Turner, D.N. (2000), Pict: A programming language based on the pi-calculus. In *Proof, Language and Interaction: Essays in Honour of Robin Milner*, ed. G.D.Plotkin, C.P.Stirling and M.Tofte, MIT Press, pp455–494.

[15] Wadsworth, C.P. (1971), Semantics and pragmatics of the lambda-calculus. Dissertation, Oxford University.

[16] Wojciechowski, P.T. and Sewell, P. (1999), Nomadic Pict: Language and infrastructure design for mobile agents. Proc. ASA/MA '99, Palm Springs, California.