Modelling logic programming: an LFCS adventure

John Power, University of Bath

I was introduced to logic programming by Leon Sterling in 1988–89, the year before I came to Edinburgh. That was already informed by LFCS: Leon had done a PhD in Pure Mathematics in Australia, then joined Alan Bundy's group in Edinburgh, learning logic programming there, before I met him; Alan in turn was closely associated with LFCS, and in fact was a member of the ESPRIT LF programme that I joined as Gordon Plotkin's research assistant within LFCS in 1989.

Upon coming to Edinburgh, my research on logic programming quietly percolated, informed decisively by a multitude of LFCS researchers and visitors. The bulk of my research on the topic has been joint with Ekaterina Komendantskaya, a regular visitor to LFCS.

Gordon Plotkin and his research associates and PhD students, such as Bartek Klin and Daniele Turi, inspired by Robin Milner's development of CCS, were experts in coalgebra. Inspired by that, we observed that a propositional logic program gives rise to, and is essentially equivalent to, a $P_f P_f$-coalgebra, where P_f is the endofunctor on Set that sends a set X to the set of finite subsets of X. Specifically, given a logic program P, any atomic formula is the head of finitely many clauses in P, and each such clause contains finitely many atomic formulae in its tail, yielding a coalgebra structure on the set At of atomic formulae.

A strong feature of their work was the study of the cofree comonad on an endofunctor. We followed suit, yielding an elegant characterisation of coinductive trees, a standard construction, albeit under a different name, in logic programming. That construction has been central to the whole body of our work.

Gordon Plotkin with his LF group, Rod Burstall with his LEGO group, Don Sannella with his Algebraic Specification group, and Mike Fourman with his categorical logic group, all contributed to the next idea: in order to extend from propositional logic programs to arbitrary ones, one attempts to extend from Set to the presheaf category $[L^{op}, \mathrm{Set}]$, where L is the Lawvere theory freely generated by the function symbols in the logic program.

In fact, the situation is more complex than that, as the naturality condition of a map in $[L^{op}, \mathrm{Set}]$ fails. At that point, Tony Hoare, a frequent visitor to LFCS, in particular to Robin Milner's group, springs to mind. Tony had modelled data refinement in terms of "upward simulations" and "downward simulations". These

are exactly lax and oplax natural transformations. Inspired by that, we refined $[L^{op}, \mathrm{Set}]$ to $Lax(L^{op}, \mathrm{Poset})$, resolving the naturality problem.

That allowed us to model a substantial class of logic programs, but not ones with existential variables, i.e., variables in the tail of a clause that do not appear in the head. In stepped Peter O'Hearn, another welcome visitor to LFCS, this time primarily to Plotkin's group. That problem, albeit in a somewhat different context, was central to Peter's modelling of local state. So, inspired by his account of local state, we further refined our constructions, allowing us to model existential variables.

That reaches the present day. For the future, we seek to model the recursion of logic programming by taking coinductive trees and studying the graphs generated by identifying nodes with the same labels: that yet again is inspired by LFCS practice, that technique having been used by Colin Stirling in his use of the tableau method in studying CCS.

I should mention that, in the noble tradition of LFCS, this is not purely theoretical research. Rather it has inspired a variant of SLD-resolution, one built upon maximal use of pattern-matching before using arbitrary unification. This typically adds efficiency and is currently being developed by Komendantskaya's group at Heriot-Watt University, with particular application to functional programming.