Programming in Lygon: A Brief Overview

James Harland
RMIT
Australia

David Pym
Queen Mary & Westfield College
University of London, UK

Michael Winikoff
University of Melbourne
Australia

Recently, there has been much interest in the application of linear logic, a logic of resource-consumption, to computer science. In particular, the present authors (and others) have considered how logic programming languages can be derived by purely proof-theoretic analyses of linear logic. Such languages provide a notion of resource-oriented programming, often leading to programs that are more elegant and concise than their equivalents in languages, such as Prolog, based on classical logics. We give a brief overview of the linear logic programming language *Lygon*.[1]

In common with other linear logic programming languages, Lygon allows clauses to be used exactly once in a computation, thereby avoiding the need for the explicit resource-counting often necessary in Prolog-like languages. However, just as linear logic is a strict extension of classical logic, Lygon is a strict extension of (pure) Prolog: all (pure) Prolog programs can be executed by the Lygon system. Hence all the features of classical pure logic programs are available in Lygon, together with new ones based on linear logic. These include *global variables*, a theoretically transparent notion of *state*, *mutual exclusion operators* and various constructs for manipulating clauses. All of these follow from the basis of Lygon in linear logic and *do not require* extra-logical features for their definition.

One example of an elegant use of Lygon is in the problem of finding paths in cyclic graphs. The well-known transitive closure program, a simple and elegant Prolog program, will find an infinite number of paths. However, by treating the specification of each edge as a linear predicate, a transliteration of this program into Lygon can be used. By stipulating that each edge can be used (at most) once, just a finite number of paths will be found between any two points in the graph. Moreover, experience with this and other examples suggests that many of the programming constructs, such as the multiplicative (or parallel) disjunction $\parr$ of goals, arising from the basis of Lygon in linear logic facilitate a programming style that is elegant and natural for a variety of applications.

The most important characteristic of the implementation of Lygon is the *lazy splitting* of multiplicative branches of potential proofs.[2] This proof-search strategy is supported by a deterministic resource-management technique which can be exploited in many well-known state-and-action problems, such as the Yale shooting problem, the blocks world, counting programs and bin-packing problems.

The implementation of Lygon that is currently available (Version 0.4) is an interpreter written in BinProlog.[3]

---

[1] For the theoretical basis of Lygon, see: D.J. Pym and J.A. Harland, A Uniform Proof-theoretic Investigation of Linear Logic Programming, *J. Logic Computat.* 4:2:175–207, 1994.

[2] M. Winikoff and J. Harland, Implementing the Linear Logic Programming Language Lygon, *Proc. ILPS'95*, Portland, Oregon, December, 1995. (This volume.)

[3] Implementation available via `http://www.cs.mu.oz.au/~winikoff/lygon/lygon.html` or from the authors. Harland: `jah@cs.rmit.edu.au`, `http://www.cs.rmit.edu.au/~jah`. Pym: `pym@dcs.qmw.ac.uk`, `http://www.dcs.qmw.ac.uk/~pym`. Winikoff: `winikoff@cs.mu.oz.au`, `http://www.cs.mu.oz.au/~winikoff`.