# Coalgebraic Derivations in Logic Programming

Ekaterina Komendantskaya*
*School of Computing
University of Dundee
katya@computing.dundee.ac.uk

John Power†
†Department of Computer Science
University of Bath
A.J.Power@bath.ac.uk

**Abstract**

Coalgebra may be used to provide semantics for SLD-derivations, both finite and infinite. We first give such semantics to classical SLD-derivations, proving results such as adequacy, soundness and completeness. Then, based upon coalgebraic semantics, we propose a new sound and complete algorithm for parallel derivations. We analyse this new algorithm in terms of the Theory of Observables, and we prove soundness, completeness, correctness and full abstraction results.

## 1 Motivation

There are two trends in logic programming that are both desirable and problematic: coinductive definitions and concurrent computations. We illustrate both using the following example.

**Example 1** *The program* `Stream` *defines the infinite stream of binary bits:*

$$\begin{aligned} \texttt{bit}(0) &\leftarrow \\ \texttt{bit}(1) &\leftarrow \\ \texttt{stream(scons (x,y))} &\leftarrow \texttt{bit(x)}, \texttt{stream(y)} \end{aligned}$$

`Stream` is a coinductive definition, with proof search for the goal `stream(x)` resulting in an infinite derivation. Programs like `Stream` can be given declarative semantics via the *greatest* fixed point of the semantic operator $T_P$. But fixed point semantics is incomplete (Lloyd (1987)) as it fails for some infinite derivations.

**Example 2** *The program $R(x) \leftarrow R(f(x))$ is characterised by the greatest fixed point of the $T_P$ operator, which contains $R(f^\omega(a))$, although no infinite term is computed by SLD-resolution.*

There have been numerous attempts to resolve the mismatch between infinite derivations and greatest fixed point semantics, e.g., Gupta et al. (2007); Jaume (2002); Lloyd (1987); Paulson and Smith (1989); Simon et al. (2007). But infinite SLD derivations of both finite and infinite objects have not yet received a uniform semantics.

Another distinguishing feature of logic programming languages is that they allow implicit parallel execution of programs. The three main types of parallelism used in implementations are *and-parallelism*, *or-parallelism*, and their combination: see Gupta and Costa (1994); Pontelli and Gupta (1995).

*Or-parallelism* arises when more than one clause unifies with the goal: the corresponding bodies can be executed in or-parallel fashion. *And-parallelism* arises when more than one atom is present in the goal. That is, given a goal $G = \leftarrow B_1, \ldots B_n$, an *and-parallel algorithm* of resolution looks for derivations for each $B_i$ simultaneously. *And-or parallelism* features both kinds of parallelism. However, many first-order algorithms are P-complete and hence inherently sequential (Dwork et al. (1984); Kanellakis (1988)). This especially concerns first-order unification and variable substitution in the presence of variable dependencies.

**Example 3** *The goal* `stream(cons(x, cons(y,x)))`*, if processed sequentially, will lead to a failed derivation in three derivation steps. But, if the goal is processed in and-or parallel fashion, the derivation algorithm will not be able to determine inconsistency between substitutions for $x$ in parallel branches of the derivation tree.*

## 2 Summary of results

We apply the coalgebraic semantics proposed in Komendantskaya et al. (2010); Komendantskaya and Power (2011) to address the above two problems. Coalgebraic semantics can be used instead of greatest fixed point semantics. Unlike

greatest fixed point semantics, coalgebraic semantics yields soundness and completeness results for both finite and infinite SLD-derivations.

We test our coalgebraic semantics using the *theory of observables*, Comini et al. (2001). According to this theory, the traditional characterisation of logic programs in terms of their input/output behavior, successful derivations and their corresponding fixed point semantics, is not sufficient for program analysis and optimisation. One requires more complete information about SLD-derivations, such as sequences of goals, most general unifiers and variants of clauses.

The idea of observational semantics for logic programs is to observe equal behaviour of logic programs and to distinguish logic programs with different computational behaviour. So the choice of observables and semantic models is closely related to the choice of equivalence relation defined on logic programs. In the theory of observables, programs $P_1$ and $P_2$ are said to be observationally equivalent if, for any goal $G$, they yield the same call patterns in the SLD derivations. Given a suitable semantics, one can prove that if $P_1$ and $P_2$ have equal semantic models, then $P_1$ and $P_2$ are observationally equivalent (*correctness result*). The converse is a *full abstraction* result.

Our coalgebraic semantics yields the correctness result relative to the sequential algorithm of SLD resolution but not the full abstraction result. The failure of the latter does not mean that coalgebraic semantics is not suitable for characterizing observational behavior of logic programs. Rather, it indicates the mismatch between the sequential algorithm of SLD resolution and the concurrent nature of coalgebraic semantics.

We therefore propose a new *coinductive derivation algorithm* inspired by coalgebraic semantics. The algorithm provides an elegant solution to the problem of implementing both corecursion and concurrency in logic programming. We prove soundness, completeness, correctness and full abstraction results for the new coinductive derivations relative to the coalgebraic semantics of Komendantskaya and Power (2011).

# References


M. Comini, G. Levi, and M. C. Meo. A theory of observables for logic programs. *Inf. Comput.*, 169(1):23–80, 2001.

C. Dwork, P.C. Kanellakis, and J.C. Mitchell. On the sequential nature of unification. *Journal of Logic Programming*, 1: 35–50, 1984.

G. Gupta and V.S. Costa. Optimal implementation of and-or parallel prolog. In *Conference proceedings on PARLE'92*, pages 71–92, New York, NY, USA, 1994. Elsevier North-Holland, Inc.

G. Gupta, A. Bansal, R. Min, L. Simon, and A. Mallya. Coinductive logic programming and its applications. In *ICLP 2007*, volume 4670 of *LNCS*, pages 27–44. Springer, 2007.

M. Jaume. On greatest fixpoint semantics of logic programming. *J. Log. Comput.*, 12(2):321–342, 2002.

P. C. Kanellakis. Logic programming and parallel complexity. In *Foundations of Deductive Databases and Logic Programming.*, pages 547–585. M. Kaufmann, 1988. ISBN 0-934613-40-0.

E. Komendantskaya and J. Power. Coalgebraic semantics for derivations in logic programming. In *Submitted to CALCO'11*, 2011.

E. Komendantskaya, G. McCusker, and J. Power. Coalgebraic semantics for parallel derivation strategies in logic programming. In *Proc. of AMAST'2010 - 13th Int. Conf. on Algebraic Methodology and Software Technology*, volume 6486 of *LNCS*, 2010.

J.W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, 2nd edition, 1987.

L.C. Paulson and A.W. Smith. Logic programming, functional programming, and inductive definitions. In *ELP*, pages 283–309, 1989.

E. Pontelli and G. Gupta. On the duality between or-parallelism and and-parallelism in logic programming. In *Euro-Par*, pages 43–54, 1995.

L. Simon, A. Bansal, A. Mallya, and G. Gupta. Co-logic programming: Extending logic programming with coinduction. In *ICALP*, volume 4596 of *LNCS*, pages 472–483. Springer, 2007.