Using a Generalisation Critic to Find Bisimulations for Coinductive Proofs*

Louise Dennis Alan Bundy Ian Green
University of Edinburgh**

Abstract. Coinduction is a method of growing importance in reasoning about functional languages, due to the increasing prominence of lazy data structures. Through the use of bisimulations and proofs that bisimilarity is a congruence in various domains it can be used to prove the congruence of two processes.

A coinductive proof requires a relation to be chosen which can be proved to be a bisimulation. We use proof planning to develop a heuristic method which automatically constructs a candidate relation. If this relation does not allow the proof to go through a proof critic analyses the reasons why it failed and modifies the relation accordingly.

Several proof tools have been developed to aid coinductive proofs but all require user interaction. Crucially they require the user to supply an appropriate relation which the system can then prove to be a bisimulation.

1 Introduction

Recursive data structures and functions are of central importance in computer science. As a result, inductive definitions and proofs form a major research area in the semantics of programming languages and in the field of program verification.

Inductive definitions specify the least set generated by some recursive function. The dual notion is of the greatest set. The least and greatest closed sets can be expressed as the least and greatest fixpoints of some function. Least fixpoints give inductive definitions; greatest fixpoints give “coinductive” definitions. The greatest closed set will contain infinite as well as finite datatypes. Hence coinduction, the associated proof method, allows reasoning about such datatypes.

Coinduction was first seen as an important proof method in the theory of concurrency. Milner’s bisimulation proof method [18] is a form of coinduction. There is now a great deal of interest in using coinduction to reason about lazy functional languages. Abramsky first motivated this with the lazy lambda calculus [1] which defined applicative bisimilarity and showed that it was a congruence within the calculus. Milner and Tofte used coinduction to show the consistency of the dynamic and static semantics of a small functional language [19]. Abramsky’s congruence result was taken by Howe [15] and used to devise a general

* Research funded by EPSRC grant GR/L/11724, Postgraduate Studentship 94314689.
** Department of AI, University of Edinburgh, 80 South Bridge, Edinburgh EH1 1HN, Scotland. Email louised@dai.ed.ac.uk, {A.Bundy, I.Green}@ed.ac.uk
procedure for proving congruence. Work has been done by Gordon [12] proving congruences, setting down the syntax and semantics for a number of lazy functional languages. Paulson has also done work providing a theory for coinduction within HOL [21].

Other work has been done applying coinduction to input/output effects [13], Object-oriented languages [14] and generally to recursively defined domains [23] and over recursive datatypes [10].

Several theorem provers have capabilities for coinductive proof although they all require user interaction. Perhaps the most work has been done in Isabelle for which a special package has been developed for coinductive definitions [22] and in which Milner and Tofte’s work has been reproduced [11]. However work has also been done in Coq [20] and HOL [8].

This paper discusses the use of Clam [7], a proof-planning system, to develop a series of methods for guiding the tactics used in systems like Isabelle in the hope of more fully automating coinductive proofs. In particular it focuses on providing a relation, a hard task which all other systems have had to leave to the user.

2 Coinduction

We have adopted Paulson’s formalization of coinduction, as described in [21]. This, in turn, is based on the work of Tarski [24] who showed that the fixpoints of monotone functions form a lattice.

The greatest fixpoint operator is defined by

\[ \text{gfp}(\mathcal{F}) \equiv \bigcup \{S \mid S \subseteq \mathcal{F}(S)\}, \] 

and can be used to derive the coinduction rule

\[
\begin{align*}
\text{a} \in X \\
X \subseteq \mathcal{F}(X)
\end{align*}
\]

\[ \Rightarrow \text{a} \in \text{gfp}(\mathcal{F}). \] 

The coinduction rule is used to show that something is a member of the greatest fixpoint of some function.

2.1 Bisimilarity

Coinduction is useful when we can show that all members of some greatest fixpoint have some property of interest. For example the property examined in this paper is bisimilarity. Two lists, \( l_1 \) and \( l_2 \), can be said to be bisimilar, \( l_1 \equiv l_2 \) if for all finite \( k \) the first \( k \) elements of each list are the same. This is based on work by Bird and Wadler [3]. In other domains, e.g., CCS, the relevant property (observation equivalence, applicative bisimulation, etc.) may be defined differently, but it always hinges on the idea that the behaviour of both objects to any observer watching for a finite time is the same. Much work has been done showing that bisimilarity is a congruence for various domains. However this is not always the case, for instance in CCS [18].