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

$$\mathtt{gfp}(\mathcal{F}) \equiv \bigcup \{\mathcal{S} \mid \mathcal{S} \subseteq \mathcal{F}(\mathcal{S})\}, \tag{1}$$

and can be used to derive the coinduction rule

$$\frac{a \in X \qquad X \subseteq \mathcal{F}(X)}{a \in \mathtt{gfp}(\mathcal{F})}. \tag{2}$$

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].

$\equiv$ is a property of the greatest fixpoint of the function[1]

$$LlistD\_fun(\mathcal{R}) \stackrel{\text{def}}{=} \{\langle h :: t_1, h :: t_2\rangle \mid \langle t_1,\ t_2\rangle \in \mathcal{R}\} \cup \{\langle nil, nil\rangle\}. \tag{3}$$

In this case the coinduction rule (2) can be specialised to

$$\frac{\langle a, b\rangle \in \mathcal{R} \qquad \mathcal{R} \subseteq LlistD\_fun(\mathcal{R})}{a \equiv b}. \tag{4}$$

A binary relation is called a *bisimulation* if it contains only pairs $\langle a, b\rangle$ with $a \equiv b$. The second condition of (4) demands that $\mathcal{R}$ is a bisimulation.

In this paper we shall only consider proofs of the bisimilarity of two lists so we shall be using (4) as the coinduction rule. However the techniques described can be extended to other datatypes.

Notice that $\mathcal{R}$ is not determined by the conclusion of the rule. It is introduced in the conditions, the subgoals that will need to be formed by any attempt at a proof. Hence, an important step in a coinductive proof is the choice of a suitable $\mathcal{R}$. Once this is done, a number of theorem provers, using standard theorem proving techniques, can successfully produce a fully automated proof from that point.

However, the choice of $\mathcal{R}$ is a hard problem. The technique outlined in this paper allows a totally automated system to discover an appropriate $\mathcal{R}$ and has proved very successful.

3 An Example of a Simple Coinductive Proof

Here is an example where the choice of $\mathcal{R}$ is fairly simple

Example 1. The conjecture is:

$$\forall f, m.\ map(f, iter(f, m)) = iter(f, f(m)),$$

map and *iter* are both functions over lazy lists and are defined by the following rewrite rules[2]:

$$\begin{aligned} map(F, nil) &\Rightarrow nil, \\ map(F, H :: T) &\Rightarrow F(H) :: map(F, T), \\ iter(F, M) &\Rightarrow M :: iter(F, F(M)). \end{aligned}$$

As mentioned above equality over lazy lists is a property of $\texttt{gfp}(LlistD\_fun)$.

Let $\mathcal{R} \stackrel{\text{def}}{=} \{\langle map(f, iter(f, m)), iter(f, f(m))\rangle \mid m : \tau,\ f : \tau \to \tau\}$. Following the presentation of coinduction within Isabelle/HOL we shall represent $\mathcal{R}$ as the range of a function, i.e., $\mathcal{R} \stackrel{\text{def}}{=} rng(\langle \lambda f, m.\ map(f, iter(f, m)), iter(f, f(m))\rangle)$. The second pre-condition of (4) gives the goal:

$$\begin{aligned} &rng(\langle \lambda f, m.\ map(f, iter(f, m)), iter(f, f(m))\rangle) \subseteq \\ &\quad LlistD\_fun(rng(\langle \lambda f, m.\ map(f, iter(f, m)), iter(f, f(m))\rangle)). \end{aligned} \tag{5}$$

[1] *LlistD_fun* is the name Paulson gives this function.

[2] In what follows $\Rightarrow$ is the rewriting relation.

To prove this, it is necessary to show that any two lists in $\mathcal{R}$ have equal heads and that their tails are related by $\mathcal{R}$. We shall start by showing that the tails of any two lists in $\mathcal{R}$ are also in $\mathcal{R}$. This is done by proving the implication

$$\forall \mathcal{T}, F, M.\ \langle map(F, iter(F, M)),\ iter(F, F(M))\rangle \in \mathcal{T} \rightarrow \tag{6}$$

$$\forall F', M'.\ \langle tl(map(F', iter(F', M'))),\ tl(iter(F', F'(M')))\rangle \in \mathcal{T} \tag{7}$$

This goal is a little bit like an inductive goal and for that reason I've called (6) the *coinduction hypothesis* and (7) the *coinduction conclusion*. The coinduction conclusion will be manipulated by rewriting to find the tails which will then match the coinduction hypothesis:

$$\begin{aligned}&\langle map(F, iter(F, M)),\ iter(F, F(M))\rangle \in \mathcal{T} \rightarrow\\&\quad \langle tl(map(F', iter(F', M'))),\ tl(iter(F', F'(M')))\rangle \in \mathcal{T},\end{aligned}$$

rewrites to

$$\begin{aligned}&\langle map(F, iter(F, M)),\ iter(F, F(M))\rangle \in \mathcal{T} \rightarrow\\&\quad \langle map(F', iter(F',\ F'(M'))),\ iter(F',\ F'(F'(M')))\rangle \in \mathcal{T}.\end{aligned}$$

Which is trivial.

The proof is completed by following a similar rewriting process to show that the heads are equal.

$$\begin{aligned}hd(map(F, iter(F, M))) &= hd(iter(F, F(M))),\\F(M) &= F(M).\end{aligned}$$

4 Proof Planning and Clam

Proof plans were first proposed by Bundy [5] and have been successfully applied to inductive theorem proving and other domains.

The idea is to make a plan of the tactics needed to conduct a given proof in advance of applying those tactics. A plan consists of a series of methods each of which is linked to a tactic and contains pre-conditions and effects of applying the tactic.

When a method fails to apply, a proof critic looks at the reasons for failure and may try to modify or patch the proof plan to allow it to continue.

Once a critic's pre-conditions are satisfied it proposes a patch for the proof plan, e.g., proposing a different induction scheme. At this point it can jump back to a previous node of the proof tree, e.g., where the induction scheme was first proposed, and restart the attempt to build a proof plan from that point, alternatively (e.g., in the case of missing lemmas) it can continue from the point where the proof failed.

Critics have been successfully used in inductive proof plans to speculate missing lemmas, revise the induction scheme and generalise theorems [16, 17].

A completed proof plan can be executed by giving it to a tactic based theorem prover which will provide a formal verification.

Proof planing is implemented in Clam [7]. The following techniques have been implemented in Clam, but the methods have not yet been linked to any object-level theorem prover.

5 Rule-of-Thumb Coinduction

There are two main stages to a coinductive proof and these are represented by two proof methods in Clam, `coinduction` and `gfp_membership`.

The first stage, the `coinduction` method, involves recognising that a greatest fixpoint is involved and reformulating the goal in terms of this[3].

A bisimilarity problem is generally of the form:

$$\forall \bar{x}:\bar{\tau}.\ f(\bar{x}) \equiv g(\bar{x}), \tag{8}$$

where f and g are functions on variables $\bar{x}$ of types $\bar{\tau}$ to lazy lists of type σ.

Given a problem of this form the `coinduction` method applies the coinduction rule (4). This involves deciding upon a candidate for $\mathcal{R}$. The `coinduction` method uses the obvious heuristic for this, choosing $\mathcal{R}$ to be the set determined by the pair under consideration. This has the added advantage of automatically discharging the rule's first premiss $\langle a,\ b\rangle \in \mathcal{R}$. This choice of $\mathcal{R}$ is defeasible in that its definition can be changed in the event of the proof failing to go through.

Hence the `coinduction` method produces the goal

$$rng(\lambda\bar{x}.\langle f(\bar{x}), g(\bar{x})\rangle) \subseteq \mathit{LlistD\_fun}(rng(\lambda\bar{x}.\langle f(\bar{x}), g(\bar{x})\rangle)), \tag{9}$$

$rng(\lambda\bar{x}.\langle f(\bar{x}), g(\bar{x})\rangle)$[4] is the relation $\mathcal{R}$. This "first guess" heuristic for choosing an appropriate relation for $\mathcal{R}$ is, in fact, remarkably successful at finding proofs without the need for any critics. We have called this *rule-of-thumb* coinduction. The definition of $\mathcal{R}$ at any one time is referred to as the *trial bisimulation.*

The second stage involves proving that the relation $\mathcal{R}$ is a bisimulation. This is performed through the `gfp_membership` method and rewriting.

Should the `gfp_membership` method, rewriting and fertilization fail to find a proof, Clam will use a proof critic to attempt to find a suitable revision of $\mathcal{R}$.

The `gfp_membership` method transforms (9) into the two goals[5]

$$\forall \mathcal{T}, \forall \bar{x}.\ \langle (f(\bar{x}), g(\bar{x})\rangle \in \mathcal{T} \rightarrow \forall \bar{y}.\ \langle tl(f(\bar{y})), tl(g(\bar{y}))\rangle \in \mathcal{T}, \tag{10}$$

$$\forall \bar{y}.\ hd(f(\bar{y})) = hd(g(\bar{y})). \tag{11}$$

At this point we want to use *fertilization* to prove (10). There are two sorts of fertilization of which only *strong* fertilization is of interest here. In strong fertilization a direct appeal is made to the coinduction hypothesis matching it to the (now rewritten) coinduction conclusion. In order for fertilzation to take place we are going to have to rewrite the coinduction conclusion. If fertilzation fails then a critic will be called upon to patch the proof. The second goal (11) does not require fertilization.

We need a method for rewriting which

---

[3] The method works fairly simply at present, consulting a database of known greatest fixpoints to see if any are relevant.

[4] For presentational reasons we omit λ abstractions from the relation.

[5] The theorem in the appendix justifies this step. Actually (10) and (11) are disjoined with the additional case $rng(\langle f(\bar{x}), g(\bar{x})\rangle) \in \{\langle nil, nil\rangle\}$. This case is not needed for any of the proofs in this paper and discussion of it has been omitted in order to simplify the presentation [9].

1. Terminates in a finite time. If the rewriting can carry on infinitely then there is no way to determine the failure of fertilization.
2. If fertilization is possible the rewriting always places the coinduction conclusion in a form to allow this

Currently we use a rewriting method which satisfies the first of these requirements, but sometimes fails to satisfy the second. This is clearly an area for further research.

The `coinduction` method and `gfp_membership` method only deal with the second precondition for coinduction, $\mathcal{S} \subseteq \mathcal{F}(\mathcal{S})$. The first precondition, $a \in \mathcal{S}$ is presumed to follow from the heuristic used to form $\mathcal{S}$.

6 Critics for Coinduction

The rule-of-thumb does not always work, as illustrated by the following example. We will use this to explain the use of the basic form of the `revise_bisimulation` critic. Most examples where the critic is needed involve a more advanced version discussed in §7.

Example 2.

$$\forall a, b.\, lswap(a, b) = merge(lconst(a), lconst(b)),$$

$lswap$, $merge$ and $lconst$ are defined by the following rewrite rules

$$\begin{gathered} lswap(A, B) \Rightarrow A :: B :: lswap(A, B), \\ lconst(A) \Rightarrow A :: lconst(A), \\ merge(nil, L) \Rightarrow L, \qquad merge(L, nil) \Rightarrow L, \\ merge(H_1 :: T_1, H_2 :: T_2) \Rightarrow H_1 :: H_2 :: merge(T_1, T_2). \end{gathered}$$

The `coinduction` method uses the rule-of-thumb and chooses

$$\mathcal{R} \stackrel{\text{def}}{=} rng(\langle lswap(a, b), merge(lconst(a), lconst(b))\rangle,$$

to produce the subgoal

$$\begin{aligned} &rng(\langle lswap(a, b), merge(lconst(a), lconst(b))\rangle) \subseteq \\ &\quad LlistD\_fun(rng(\langle lswap(a, b), merge(lconst(a), lconst(b))\rangle)). \end{aligned} \tag{12}$$

The `gfp_membership` method then produces subgoals to check that the heads of each list are equal and the tails are in the bisimulation. The discussion will centre around proving that the tails are members of the bisimulation so we shall only consider the first subgoal (10). In the proofs below, we shall ignore subgoals dealing with the equality of the heads of both lists. The first subgoal is

$$\begin{aligned} &\langle lswap(A, B),\ merge(lconst(A), lconst(B))\rangle \in \mathcal{T} \rightarrow \\ &\quad \langle tl(A' :: B' :: lswap(A', B')),\ tl(merge(A' :: lconst(A'), B' :: lconst(B')))\rangle \in \mathcal{T}, \end{aligned} \tag{13}$$

which rewrites to

$$\begin{aligned}&\langle lswap(A,B),\ merge(lconst(A), lconst(B))\rangle \in \mathcal{T} \rightarrow \\ &\quad \langle B' :: lswap(A',B'),\ B' :: merge(lconst(A'), lconst(B'))\rangle \in \mathcal{T}.\end{aligned} \tag{14}$$

Clam fails to prove this because none of its proof methods will apply.[6]

What the method actually needed was a different definition of $\mathcal{R}$:

$$\begin{aligned}\mathcal{R} = {} & rng(\langle lswap(a,b), merge(lconst(a), lconst(b))\rangle) \cup \\ & rng(\langle b :: lswap(a,b), b :: merge(lconst(a), lconst(b))\rangle).\end{aligned} \tag{15}$$

The aim of the critic is to modify the choice of $\mathcal{R}$ in the light of failure analysis.

The critic is called the `revise_bisimulation` critic (see figure 1) and it comes into play if fertilization has failed. This could be caused by a number of things (for instance some lemma required by the rewriting might not be available to the prover), however in general it happens because the choice of $\mathcal{R}$ was inappropriate and needs to be revised.

Critics have a number of preconditions. The `revise_bisimulation` critic checks that a coinductive proof is being attempted. It then patches the proof by altering the trial bisimulation and returning to that point in the proof search where it was introduced. Figure 1 presents the critic's pre-conditions in a natural language format for ease of understanding. The actual critic is written in Prolog.

If all of

1. The coinduction method has been used in this branch of the proof.
2. The current goal is $\langle f_1, g_1\rangle \in \mathcal{T} \wedge \cdots \wedge \langle f_m, g_m\rangle \in \mathcal{T} \rightarrow \langle k, l\rangle \in \mathcal{T}$. Where neither k nor l are of the form $tl(\cdots)$.
3. Fertilization is not applicable

Then

Change the trial bisimulation by adding the set $rng(\langle k, l\rangle)$ to it and start the proof again from the most recent call of the coinduction method, supplying the revised relation as the new trial bisimulation.

Fig. 1. The Basic Revise Bisimulation Critic

Using the critic, Clam revises the trial bisimulation to

$$\begin{aligned}\mathcal{R}' = {} & rng(\langle lswap(a,b), merge(lconst(a), lconst(b))\rangle) \ \cup \\ & \langle b :: lswap(a,b), b :: merge(lconst(a), lconst(b))\rangle),\end{aligned} \tag{16}$$

[6] N.B. B' cannot simply be cancelled since the specific goal here is to show that $\langle B' :: l_1, B' :: l_2\rangle \in \{\langle l_1, l_2\rangle\}$ this is not necessarily the case (for instance if B' were the number 1 and l_1 and l_2 lists containing only 0, $\langle [1,0],[1,0]\rangle \notin \{\langle [0],[0]\rangle\}$)

and starts out again with the goal $\mathcal{R}' \subseteq LlistD\_fun(\mathcal{R}')$.

The `gfp_membership` method splits this into two new goals, one for each function range in the trial bisimulation:

$$\begin{aligned}&\langle lswap(A,B),\ merge(lconst(A), lconst(B))\rangle \in \mathcal{T} \wedge \\ &\langle B' :: lswap(A',B'),\ B' :: merge(lconst(A'), lconst(B'))\rangle \in \mathcal{T} \rightarrow \\ &\quad \langle tl(A'' :: B'' :: lswap(A'',B'')),\ tl(merge(A'' :: lconst(A''), B'' :: lconst(B'')))\rangle \in \mathcal{T},\end{aligned} \tag{17}$$

and

$$\begin{aligned}&\langle lswap(A,B),\ merge(lconst(A), lconst(B))\rangle \in \mathcal{T} \wedge \\ &\langle B' :: lswap(A',B'),\ B' :: merge(lconst(A'), lconst(B'))\rangle \in \mathcal{T} \rightarrow \\ &\quad \langle tl(B'' :: lswap(A'',B'')),\ tl(B'' :: merge(lconst(A''), lconst(B'')))\rangle \in \mathcal{T}.\end{aligned} \tag{18}$$

The proof of (17) proceeds identically as in the first proof attempt, however, the extra hypothesis allows fertilization to occur. (18) resolves similarly.

7 The Use of Generalisation in the Critic

This approach described above breaks down when a series of relations are progressively added to the trial bisimulation, but in such a way that the bisimulation never becomes strong enough for the proof to be completed. As a result, the bisimulation is extended indefinitely, and the proof search diverges. In such cases a larger bisimulation must be found—in this paper we use a *generalisation* process to find a bisimulation that encompasses *all* these extensions.

The following is an example in which the repeated extension of the bisimulation leads to fruitless proof search.

Example 3.

$$\forall x, f.\ h_f(x) = x :: map(f, h_f(x)) \rightarrow \forall x, f.\ h_f(x) = iter(f, x).$$

The `coinduction` method produces the goal

$$rng(\langle h_f(x), iter(f,x)\rangle) \subseteq LlistD\_fun(rng(\langle h_f(x), iter(f,x)\rangle)). \tag{19}$$

The `gfp_membership` method then gives the goal:

$$\begin{aligned}&\langle h_F(x),\ iter(F,x)\rangle \in \mathcal{T} \rightarrow \\ &\quad \langle tl(X' :: map(F', h_{F'}(X'))),\ tl(X' :: iter(F'), F'(X'))\rangle \in \mathcal{T},\end{aligned} \tag{20}$$

which rewrites to

$$\begin{aligned}&\langle h_F(x),\ iter(F,x)\rangle \in \mathcal{T} \rightarrow \\ &\quad \langle map(F', h_{F'}(X')),\ iter(F', F'(X'))\rangle \in \mathcal{T}.\end{aligned}$$

At this point the proof attempt fails because fertilization is not possible. This failure causes the `revise_bisimulation` critic to intervene, adding

$$rng(\langle map(f, h_f(x)), iter(f, f(x))\rangle)$$

to $\mathcal{R}$ and (with the `gfp_membership` method) producing two subgoals. The first subgoal is similar to (20). The second new subgoal is:

$$\begin{aligned}&\langle h_F(X),\ iter(F,X)\rangle \in \mathcal{T} \wedge \langle map(F', h_{F'}(X')),\ iter(F', F'(X'))\rangle \in \mathcal{T} \rightarrow \\ &\quad \langle tl(map(F'', X'' :: map(F'', h_{F''}(X'')))),\ tl(F''(X'') :: iter(F'', F''(F''(X''))))\rangle \in \mathcal{T}.\end{aligned} \tag{21}$$

Once again the `revise_bisimulation` critic will intervene and suggest adding $rng(\langle map(f, map(f, h_f(x))), iter(f, f(f(x)))\rangle)$ to $\mathcal{R}$. Clearly we have embarked upon a divergent process.

To deal with this divergence caused by the repeated extension of $\mathcal{R}$, the `revise_bisimulation` critic has been revised. It recognises this situation and proposes a suitable generalisation. This revision, a divergence check, is based on Walsh's divergence critic [26] for proofs by implicit induction.

7.1 The Divergence Check

A divergence check (figure 2) is added to the `revise_bisimulation` critic to spot divergence and provide information about its cause. The check attempts to find some term structure introduced by the revisions which is accumulating in the sequence of equations and which was preventing fertilization solving some of the goals. The critic identifies this structure using difference matching [2].

For instance take the sequence produced in Example 3

$$\begin{aligned} s_0 &= \langle h_F(X),\ iter(F,X)\rangle \in \mathcal{T}, \\ s_1 &= \langle map(F, h_F(X)),\ iter(F, F(X))\rangle \in \mathcal{T}, \\ s_2 &= \langle map(F, map(F, h_F(X))),\ iter(F, F(F(X)))\rangle \in \mathcal{T}. \end{aligned}$$

Difference matching successive equations adds the annotations

$$\begin{aligned} s'_0 &= \langle h_F(X),\ iter(F,X)\rangle \in \mathcal{T}, \\ s'_1 &= \langle \boxed{map(F, \underline{h_F(X)})},\ iter(F, \boxed{F(\underline{X})})\rangle \in \mathcal{T}, \\ s'_2 &= \langle \boxed{map(F, \underline{map(F, h_F(X))})},\ iter(F, \boxed{F(\underline{F(X)})})\rangle \in \mathcal{T}. \end{aligned}$$

An annotation consists of a *wave-front*, a box with a *wave-hole*, an underlined term. The *skeleton* is formed by deleting everything that appears in the wavefront but not in the wave-hole. The annotations above are determined on the conditions that the skeleton of each term in the annotated sequence is the same as the previous term in the unannotated sequence. Difference matching is a process which annotates a term s with respect to a term, t. The *erasure* of an annotated term is that term with all the annotations removed. Hence, formally, s' is a *difference match* of s and t iff $skeleton(s') = t$ and $erase(s') = s$.

It should be clear from viewing the above sequence that the accumulating term structure in the sequence is being marked out by the wave fronts. This should not be surprising since the difference matching singles out differences between two equations and it is precisely these differences which are preventing fertilization occurring.

1. There is a sequence of sets within the trial bisimulation, $rng(\langle s_i, t_i\rangle)$ which have been generated by the `revise_bisimulation` critic.
2. There exist G, L, H, M (at least one of H, M is non trivial) such that for each j difference matching gives $rng(\langle s_j, t_j\rangle) = rng(\langle G(U_j), L(W_j)\rangle)$, and $rng(\langle s_{j+1}, t_{j+1}\rangle) = rng(\langle G(\boxed{H(\underline{U_j})}), L(\boxed{M(\underline{W_j})})\rangle)$

Fig. 2. Conditions for the Divergence Check

Once divergence is spotted it is necessary to find an appropriate patch. Walsh's divergence critic patched the proofs he was attempting by speculating and proving additional lemmas. What is needed for coinductive proofs is some generalisation to replace the trial bisimulation.

The divergence is being caused by the repeated addition of H and M (as defined by the divergence check preconditions) every time the tail of the latest addition to the trial bisimulation is examined. This suggests using the function $(\cdots)^n$

$$F^0(X) = X, \qquad F^{s(N)}(X) = F(F^N(X))$$

to produce the generalisation $rng(\langle G(H^N(U_0)), L(M^N(W_0))\rangle)$ and put it in place of the previous sequence of sets in the trial bisimulation.[7]

This produces the full `revise_bisimulation` critic described in figure 3.

7.2 Back to the Example

In the proof the critic assigns G, L, H, M and U_0 and W_0 from figure 3 as id (the identity function), $iter(F)$, $map(F)$, F, $h_F(X)$ and X respectively so giving the goal

$$\begin{array}{l} rng(\langle map(f)^n(h_f(x)), iter(f, f^n(x))\rangle) \subseteq \\ \quad LlistD\_fun(rng(\langle map(f)^n(h_f(x)), iter(f, f^n(x))\rangle)). \end{array} \tag{22}$$

The `gfp_membership` method produces the goal

$$\begin{array}{l} \langle map(F)^N(h_F(X)),\ iter(F, F^N(X))\rangle \in \mathcal{T} \rightarrow \\ \quad \langle tl(map(F')^{N'}(X') :: map(F', h_{F'}(X'))),\ tl(F'^{N'}(X') :: iter(F', F'(F'^{N'}(X'))))\rangle \in \mathcal{T}, \end{array}$$

which rewrites to

$$\begin{array}{l} \langle map(F)^N(h_F(X)),\ iter(F, F^N(X))\rangle \in \mathcal{T} \rightarrow \\ \quad \langle map(F')^{s(N')}(h_{F'}(X')),\ iter(F', F'^{s(N')}(X'))\rangle \in \mathcal{T}, \end{array}$$

which can be solved by fertilization.

[7] Although this function is written using a superscript, there is no need for it to be regarded as a "meta function" in any way. It could for instance be described as, say $apply\_n(s(N), F, X) = F(apply\_n(N, F, X))$, the superscript is simply adopted since this is the common and intuitive presentation of this function

If

1. The coinduction method has been used in this branch of the proof.
2. The current goal is $\langle f_1, g_1\rangle \in \mathcal{T} \wedge \cdots \wedge \langle f_m, g_m\rangle \in \mathcal{T} \rightarrow \langle k, l\rangle \in \mathcal{T}$, where neither k nor l are of the form $tl(\cdots)$.
3. Rewriting has terminated fertilization is not possible.

Then if

The Divergence Check has succeeded identifying G, L, H, M and U_0 and W_0 for some sequence of sets within the trial bisimulation.

Then

Change the trial bisimulation by replacing the sequence with the set $rng(\langle G(H^n(U_0)), L(M^n(W_0))\rangle)$ and start the proof again from the most recent call of the coinduction method, supplying the revised relation as the new trial bisimulation.

Else

Change the trial bisimulation by adding the set $rng(\langle k, l\rangle)$ to it and start the proof again from the most recent call of the `coinduction` method, supplying the revised relation as the new trial bisimulation.

Fig. 3. The Full Revise Bisimulation Critic

8 Results

The `coinduction` and `gfp_membership` methods and the `revise_bisimulation` critic have been implemented in Clam.v3.2.

It has been tried on 18 example problems (shown below with definitions) involving the bisimilarity of lists of which it was able to solve 14. These were taken from a variety of sources and included standard problems from the literature as well as problems adapted from a textbook on ML [27] and problems devised by ourselves.

The methods and critic have been tried on theorems involving other coinductive datatypes (e.g., infinite binary trees) with similarly encouraging results.

Some Definitions
$nil@X \Rightarrow X \qquad X :: Y@Z \Rightarrow X :: (Y@Z)$
$inflist(N, H, T) \Rightarrow H(N) :: inflist(T(N), H, T)$
$explode(nil) \Rightarrow nil \qquad explode(H :: T) \Rightarrow (H :: nil) :: explode(T)$
$jump(N, M) \Rightarrow N :: jump(N + M, M)$
$del(X, nil) \Rightarrow nil \qquad del(X, X :: T) \Rightarrow del(X, T)$ $X \neq H \rightarrow del(X, H :: T) \Rightarrow H :: del(X, T)$

Theorems Proved by Rule-of-Thumb Coinduction Alone
$l@nil \equiv l$
$l@(m@n) \equiv (l@m)@n$
$map(f, iter(f, m)) \equiv iter(f, f(m))$
$map(f, x@y :: z) \equiv map(f, x)@map(f, y :: nil@z)$
$map(f \circ g, l) \equiv map(f, map(g, l))$
$lconst(m) \equiv map(id, lconst(m))$
$map(h, iter(f, a)) \equiv inflist(a, h, t)$
$flatten(map2(f, l)) \equiv map(f, flatten(l))$
$flatten(explode(l)) \equiv l$

Theorems Proved Using the Revise Bisimulation Critic
$lswap(a, b) \equiv merge(lconst(a), lconst(b))$
$h_f(x) \equiv iter(f, x)$
$iter(s, 2) \equiv jump(2, 1)$
$lconst(m) \equiv iter(id, m)$
$inflist(m, id, id) \equiv lconst(m)$

Theorems Clam failed to Prove
$jump(0, 1) \equiv merge(jump(0, 2), jump(1, 2))$
$del(0, lconst(1)) \equiv del(0, lswap(1, 0))$
$A \subseteq B \to A \cup B \equiv B$
$inflist(a, id, s) \equiv jump(a, 1)$

Of the four examples which failed, two were due to a weakness in Clam's reasoning in the hypothesis. The other two failures are of interest however, caused by an over-generalisation in one form or another. For instance the theorem $jump(0, 1) \equiv merge(jump(0, 2), jump(1, 2))$. When the naturals are presented in terms of s and 0, Clam will produce the goal

$$\begin{aligned}&\langle jump(0, s(0)),\ merge(jump(0, s(s(0))), jump(s(0), s(s(0))))\rangle \in \mathcal{T} \to \\ &\quad \langle jump(s(0), s(0),\ merge(jump(s(0), s(s(0))), jump(s(s(0)), s(s(0)))\rangle \in \mathcal{T}.\end{aligned} \tag{23}$$

At this point Clam attempts to generalise all the terms $s(0)$ to $s^n(0)$ — which is equivalent to the number n giving the new trial bisimulation

$$\mathcal{R} \stackrel{\text{def}}{=} rng(\langle jump(n, n), merge(jump(n, s(n)), jump(s(n), s(n))\rangle, \tag{24}$$

which is not a bisimulation. The desired generalisation is

$$\mathcal{R} \stackrel{\text{def}}{=} rng(\langle jump(n, 1), merge(jump(n, 2), jump(s(n), 2)\rangle. \tag{25}$$

Clearly a longer process of additions to the trial bisimulation would have produced a better indication in the divergence check of the generalisation required.

9 Related Work

Collins has created a system to support reasoning about lazy functional languages within HOL. The coinduction rule has been derived and support for coinductive definitions provided as well as tactics for coinduction. The first of these tactics, when supplied with a relation $\mathcal{R}$, proves the first pre-condition of the coinduction rule (4) and forms goals equivalent to those formed by the `gfp_membership` method, (10) and (11). It then uses a series of simplification and evaluation tactics to prove those goals. Collins reports [8] that the level of interaction required by these tools is similar to a proof on paper. That is the sort of level of guidance which the above proof plan could be expected to provide.

Similar work has been done in Isabelle to provide support for coinductive definitions [22] allowing the coinduction rule to be derived and used. No specific tactics for coinduction have been provided, but Isabelle's simplification tactics are capable of handling much of the proof in an automated fashion, once the bisimulation has been provided.

As far as we are aware no work has been done on the automatic generation of bisimulations for these proofs, both the above systems relying on these relations being provided by the user.

Using divergence to motivate generalizations was first reported by Thomas and Jantke [25]. Walsh's divergence critic [26], on which the divergence check is based, was designed to work with an implicit induction theorem prover called SPIKE [4]. Induction is performed in SPIKE by means of test sets (finite descriptions of the initial model). SPIKE attempts to instantiate induction variables in the conjecture to be proved with members of the test set and then to use rewriting to simplify the resulting expressions. The process of generate and simplify often produces a divergent set of equations if an appropriate generalisation or lemma was not present. The preconditions for the divergence check used for coinduction are translated more-or-less directly across. This gives us confidence that they represent a general process for spotting divergence, and are not tailored to the specific examples we have examined.

10 Conclusion and Further Work

This paper has discussed the application of proof planning to coinductive proofs. In particular it has focused on how the choice of a trial bisimulation may be determined via the use of a simple heuristic in a proof method which can be patched using a proof critic, if necessary. It is this choice of trial bisimulation which interactive theorem provers always leave to the user. The results obtained so far are very pleasing and suggest that the proposed methods and critics will provide proof plans for a number of coinductive proofs.

Further work needs to be done to refine the rewriting process as described in §5 and to try and prevent over-generalisation occurring in the critics already developed (cf. §8).

Whilst we have followed Paulson's formulation of lazy lists, most work in developing proof tools and assistants for coinduction has used labelled transition systems. Since the tactics for coinduction have yet to be formally defined, we hope to be able to adapt the proof plans to use the labelled transition system style representation and bring this work more in line with the rest of the field. For example, Collins' system provides such a framework.

There is also a further interesting subset of coinductive problems (not considered here) where $hd(l)$ may be undefined, for instance $del(m, lconst(m))$. Any attempt to find $hd(del(m, lconst(m)))$ will result in non-termination. This class of processes can be treated with a more sophisticated analysis of l and the inclusion of $\Uparrow$, divergence, into the theory of bisimilarity. We have not attempted to extend our heuristics in this direction.

Appendix The following result justifies `gfp_membership`. A proof is given in a longer version of this paper [9].

Theorem 1. *If* Φ *is* $\{\langle h :: l_1, h :: l_2\rangle \mid \langle l_1, l_2\rangle \in rng(\langle\langle f(\bar{x}), g(\bar{x})\rangle\rangle)\} \cup \{\langle nil, nil\rangle\}$ *then* $rng(\langle f(\bar{x}), g(\bar{x})\rangle) \subseteq \Phi$ *iff*

$$\begin{aligned}&\forall \mathcal{T}, \bar{x}.\, \langle f(\bar{x}), g(\bar{x})\rangle \in \mathcal{T} \rightarrow \forall \bar{y}.\, \langle tl(f(\bar{y})), tl(g(\bar{y}))\rangle \in \mathcal{T} \wedge hd(f(\bar{y})) = hd(g(\bar{y}))\\ &\qquad \vee\, rng(\langle f(\bar{x}), g(\bar{x})\rangle) \in \{\langle nil, nil\rangle\}\end{aligned}$$

References

1. S. Abramsky. The lazy lambda calculus. In D. Turner, editor, *Research Topics in Functional Programming*, pages 65–117. Addison Wesley, 1990.
2. D. Basin and T. Walsh. Difference matching. In Deepak Kapur, editor, *11th Conference on Automated Deduction*, pages 295–309, Saratoga Springs, NY, USA, June 1992. Published as Springer Lecture Notes in Artificial Intelligence, No 607.
3. Richard S. Bird and Philip Wadler. *Introduction to Functional Programming.* Prentice-Hall, 1988.
4. A. Bouhoula and M. Rusinowitch. Automatic case analysis in proof by induction. In *Proceedings of the 13th IJCAI.* International Joint Conference on Artificial Intelligence, 1993.
5. Alan Bundy. The use of explicit plans to guide inductive proofs. Research Paper 349, Dept. of Artificial Intelligence, University of Edinburgh, 1988. Short version published in the proceedings of CADE-9.
6. Alan Bundy, editor. *12th Conference on Automated Deduction*, Lecture Notes in Artificial Intelligence, Vol. 814, Nancy, France, 1994. Springer-Verlag.
7. Alan Bundy, F. van Harmelen, C. Horn, and A. Smaill. The Oyster-Clam system. In M. E. Stickel, editor, *10th International Conference on Automated Deduction*, pages 647–648. Springer-Verlag, 1990. Lecture Notes in Artificial Intelligence No. 449. Also available from Edinburgh as DAI Research Paper 507.
8. G. Collins. A proof tool for reasoning about functional programs. In J. von Wright, J. Grundy, and J Harrison, editors, *9th International Conference of Theorem Proving in Higher Order Logics*, volume 1125 of *Lecture Notes in Computer Science*, pages 109–124. Springer, 1996.
9. L. Dennis, A. Bundy, and I. Green. Using a generalisation critic to find bisimulations for coinductive proof. Research paper 834, Dept. of Artificial Intelligence, University of Edinburgh, December 1996.
10. M. Fiore. A coinduction principle for recursive data types based on bisimulation. In *Proceedings of the Eight IEEE Symposium on Logic in Computer Science*, pages 110–119, 1993.

11. J. Frost. A case study of co-induction in isabelle. Forthcoming technical report, University of Cambridge, Computer Laboratory, 1995.
12. A. D. Gordon. Bisimilarity as a theory of functional programming. In *Proceedings of 11th Conference on the Mathematical Foundations of Programming Semantics*, To appear as Electronic Notes in Computer Science, v.1, New Orleans, 1995. Elsevier.
13. A. D. Gordon. A tutorial on co–induction and functional programming. In *Proceedings of the 1994 Glasgow Workshop on Functional Programming*, Springer Workshops in Computing, 1995.
14. A. D. Gordon. Bisimilarity for a first-order calculus of objects with subtyping. In *Proceedings, 23rd Symposium on Principles of Programming Languages*, pages 386–395, St. Petersburg Beach, Florida, USA, 21–24 January 1996. ACM SIGPLAN-SIGACT.
15. Douglas J. Howe. Equality in lazy computation systems. In *Proceedings, Fourth Annual Symposium on Logic in Computer Science*, pages 198–203, Asilomar Conference Center, Pacific Grove, California, 5–8 June 1989. IEEE Computer Society Press.
16. A. Ireland. The Use of Planning Critics in Mechanizing Inductive Proofs. In A. Voronkov, editor, *International Conference on Logic Programming and Automated Reasoning – LPAR 92, St. Petersburg*, Lecture Notes in Artificial Intelligence No. 624, pages 178–189. Springer-Verlag, 1992. Also available from Edinburgh as DAI Research Paper 592.
17. A. Ireland and A. Bundy. Productive use of failure in inductive proof. *Journal of Automated Reasoning*, 16(1–2):79–111, 1996. Also available as DAI Research Paper No 716, Dept. of Artificial Intelligence, Edinburgh.
18. R. Milner. *Communication and Concurrency*. Prentice Hall, London, 1989.
19. Robin Milner and Mads Tofte. Co-induction in relational semantics. *Theoretical Computer Science*, 87:209–220, 1991.
20. Christine Paulin-Mohring. Circuits as streams in Coq verification of a sequential multiplier. Technical Report RR95–16, Laboratoire de l'Informatique du Parallelisme, 1995.
21. L. C. Paulson. Co-induction and Co-recursion in Higher-order Logic. Technical Report 304, University of Cambridge, Computer Laboratory, 1993.
22. Lawrence C. Paulson. A fixedpoint approach to implementing (co)inductive definitions. In Bundy [6], pages 148–161.
23. A. M. Pitts. A Co-Induction Principle for Recursively Defined Domains. Technical Report 252, University of Cambridge, Computer Laboratory, April 1992.
24. A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–309, 1955.
25. M. Thomas and K. P. Jantke. Inductive inference for solving divergance in Knuth-Bendix. In K. P. Jantke, editor, *Analogical and Inductive Inference, Procs. of AII'89*, pages 288–303. Springer-Verlag, 1989.
26. T. Walsh. A divergence critic. In Bundy [6], pages 14–28.
27. Å. Wikström. *Functional Programming Using Standard ML*. Prentice-Hall, 1987.