Using a Generalisation Critic to Find Bisimulations for Coinductive Proofs

Louise Dennis, Alan Bundy and Ian Green

DAI Research Paper No. 834

January 10, 1997

Submitted to Cade 14

Department of Artificial Intelligence
University of Edinburgh
80 South Bridge
Edinburgh EH1 1HN
Scotland

© Louise Dennis, Alan Bundy and Ian Green

Using a Generalisation Critic to Find Bisimulations for Coinductive Proofs *

Louise Dennis, Alan Bundy and Ian Green

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 observational equivalence is a congruence in various domains it can be used to proof the congruence of two processes. 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.

A method is proposed which uses the idea of proof plans to make a heuristic guess at a suitable relation. If the proof fails for that relation the reasons for failure are analysed using a proof critic and a new relation is proposed to allow the proof to go through.

Key words and phrases. coinduction, proof planning, proof critics

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 a an important proof method in the theory of concurrency. Milner's bisimulation proof method [19] 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 bisimulations and showed that observational equivalence 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 [20]. Abramsky's congruence result was taken by Howe [13] and used to devise a general procedure for proving congruence. Work has been done by Andrew Gordon [15] 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 [22].

Other work has been done applying coinduction to Input/Output Effects [14], Object–oriented Languages [16] and generally to recursively defined domains [25] and over recursive datatypes [11].

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 [24] and in which Milner and Tofte's work has been reproduced [12]. However work has also been done in Coq [21] and HOL [10].

*The research reported in this paper was supported by EPSRC grant GR/L/11724

This paper discusses the use of $CLAM$[6], a proof–planning system, to develop a series of methods for guiding the tactics used in systems like Isabelle in the hopes of more fully automating coinductive proofs.

2 Least and Greatest Fixpoints

I have adopted Paulson's formalization of coinduction, as described in [22]. This, in turn, is based on the work of Tarski [26] who showed that the fixpoints of monotone functions form a lattice.

The least fixpoint operator is defined by

$$\mathtt{lfp}(\mathcal{F}) \equiv \bigcap \{\mathcal{S} | \mathcal{F}(\mathcal{S}) \subseteq \mathcal{S}\} \tag{1}$$

This can be used to derive a form of the induction rule

$$\frac{a \in \mathtt{lfp}(\mathcal{F}) \quad mono(\mathcal{F}) \quad \begin{array}{c}[x \in \mathcal{F}(\mathtt{lfp}(\mathcal{F}) \cap \{x.\Psi(x)\})]_x \\ \vdots \\ \Psi(x)\end{array}}{\Psi(a)} \tag{2}$$

The greatest fixpoint operator is defined by

$$\mathtt{gfp}(\mathcal{F}) \equiv \bigcup \{\mathcal{S} | \mathcal{S} \subseteq \mathcal{F}(\mathcal{S})\} \tag{3}$$

and can be used to derive the coinduction rule

$$\frac{a \in X \quad X \subseteq \mathcal{F}(X)}{a \in \mathtt{gfp}(\mathcal{F})} \tag{4}$$

It is important at this point to note the difference between the two rules (2) and (4). The induction rule is used to show that all members of the least fixpoint of some function have a property, Ψ. The coinduction rule is used to show that something is a member of the greatest fixpoint of some function.

2.1 Observational Equivalence

Coinduction is useful when we can show that all members of some greatest fixpoint have some property of interest. In most cases this property is *observational equivalence* over some relation. In the case of lazy lists, observational equivalence is generally defined in terms of $take(k, l)$. $take(k, l)$ is the first k elements of l presented as a finite list. Two lists, l_1 and l_2, are observationally equivalent if $take(k, l_1) = take(k, l_2)$ for all finite k, this is based on work by Bird and Wadler [2]. In other domains, e.g. CCS, observational equivalence is 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. A lot of work has been done showing that observational equivalence is a congruence for various domains. However this is not always the case, for instance in CCS observational equivalence and observation congruence are not the same thing [19].

Observational equivalence over lists is a property of the greatest fixpoint of the function

$$LlistD_fun(\mathcal{R}) \stackrel{\text{def}}{=} \{\langle h :: t_1,\, h :: t_2\rangle | \langle t_1,\, t_2\rangle \in \mathcal{R}\} \cup \{\langle nil,\, nil\rangle\}^1 \tag{5}$$

In this case the coinduction rule (4) can be specialised to

$$\frac{\langle a,\, b\rangle \in \mathcal{R} \quad \mathcal{R} \subseteq LlistD_fun(\mathcal{R})}{a \equiv b} \tag{6}$$

[1] $LlistD_fun$ is so named to follow Paulson, who also uses these functions.

where $\equiv$ is observational equivalence.

In this paper I shall only consider proofs of the observational equivalence of two lists, involving coinduction. So I shall be using (6) as my coinduction rule. However the techniques described can be extended to other datatypes.

Choosing $\mathcal{R}$

Relations, $\mathcal{R}$, which contain observationally equivalent pairs are called bisimulations.

$\mathcal{R}$ is not determined by the conclusion of (6) it is introduced in the pre–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 $\mathcal{R}$. This is what is often termed a "eureka step", where an intelligent guess is made as to an object that will allow the proof to go through. It is this step that has held up fuller automation of coinduction. Once $\mathcal{R}$ is chosen, a number of theorem provers, e.g. Isabelle [23] can successfully produce a fully automated proof from that point.

The technique outlined in this paper allows an automated system to discover an appropriate $\mathcal{R}$.

3 An Example of a Simple Coinductive Proof

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

Example1. The conjecture is:

$$map(f, iterates(f, m)) = iterates(f, f(m))$$

map and *iterates* are both functions over lazy lists. As mentioned above equality over lazy lists is a property of $\mathtt{gfp}(LlistD_fun)$. *map* and *iterates* are defined by the following rewrite rules.

$$\begin{array}{l} map(F, nil) \Rightarrow nil \\ map(F, H :: T) \Rightarrow F(H) :: map(F, T) \\ iterates(F, M) \Rightarrow M :: iterates(F, F(M)) \end{array}$$

A commonly used heuristic is to pick a fairly simple relation, e.g. $\mathcal{R} \stackrel{\text{def}}{=} \{\langle map(f, iterates(f, m)),\ iterates(f, f(m))\rangle\}$ and attempt to show that this lies within $\mathtt{gfp}(LlistD_fun)$. In fact, it is easier to represent $\mathcal{R}$ as the range of a function, i.e. $\mathcal{R} \stackrel{\text{def}}{=} range(\langle map(f, iterates(f, m)),\ iterates(f, f(m))\rangle)$. The second pre–condition of (6) gives the goal:

$$\begin{array}{c} range(\ \langle map(f, iterates(f, m)),\ iterates(f, f(m))\rangle) \subseteq \\ LlistD_fun(range(\langle map(f, iterates(f, m)),\ iterates(f, f(m))\rangle)) \end{array} \tag{7}$$

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 $\mathcal{R}$. I 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 goal

$$\forall \mathcal{T}.\ \langle map(f, iterates(f, m)),\ iterates(f, f(m))\rangle \in \mathcal{T} \quad \rightarrow \tag{8}$$

$$\langle tl(map(f, iterates(f, m))),\ tl(iterates(f, f(m)))\rangle \in \mathcal{T} \tag{9}$$

This goal is a little bit like an inductive goal and for that reason I've called (8) the *coinduction hypothesis* and (9) the *coinduction conclusion*. The coinduction conclusion will be manipulated by rewriting to find the tails which will then match the coinduction hypothesis.

$$\begin{array}{lll} & \langle map(f, iterates(f, m)),\ iterates(f, f(m))\rangle \in \mathcal{T} \rightarrow & \\ & \qquad \langle tl(map(f, iterates(f, m))),\ tl(iterates(f, f(m)))\rangle \in \mathcal{T} & \\ \ldots & \rightarrow \quad \langle tl(map(f, m :: iterates(f, f(m)))),\ tl(f(m) :: iterates(f, f(f(m))))\rangle \in \mathcal{T} & \\ \ldots & \rightarrow \quad \langle tl(f(m) :: map(f, iterates(f, f(m)))),\ iterates(f, f(f(m)))\rangle \in \mathcal{T} & \\ \ldots & \rightarrow \quad \langle map(f, iterates(f,\ f(m))),\ iterates(f,\ f(f(m)))\rangle \in \mathcal{T} & \end{array}$$

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

$$\begin{aligned} hd(map(f, iterates(f, m))) &= hd(iterates(f, f(m))) \\ hd(map(f, m :: iterates(f, f(m)))) &= hd(f(m) :: iterates(f, f(f(m)))) \\ hd(f(m) :: map(f, iterates(f, f(m)))) &= f(m) \\ f(m) &= f(m) \end{aligned}$$

The simple heuristic shown here for choosing an appropriate $\mathcal{R}$ often fails. Examples of this will be shown later in this paper.

4 Proof Planning and CL^AM

Proof plans were first proposed by Alan Bundy [8] and have been successfully applied to inductive theorem proving [5] 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. A completed proof plan is executed by executing the tactic part of the plan by giving it to a tactic based theorem prover which will provide a formal verification of the theorem.

Proof planing is implemented in *Oyster–*CL^AM [7][6]. *Oyster* is a tactic based theorem prover for Martin–Löf Type theory. The following techniques have been implemented in CL^AM but the appropriate *Oyster* tactics and theory have not yet been built.

4.1 Critics

Critics are an extension of the proof planning paradigm. When a method fails to apply, a critic looks at the reasons for failure and may try to modify or patch the proof plan to allow it to continue.

Typically a critic does this by looking at the pre–conditions for the method and seeing which ones failed. CL^AM's critics facilities also allow the critic to examine the current branch of the proof tree to see which methods have previously been applied, this is needed since some critics are only appropriate when certain types of proof are being attempted (e.g. the `revise_bisimulation` critic presented in this paper is only appropriate in coinductive proofs, though it is initiated when the rewriting not the `coinduction` method fails.)

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 will usually 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.

Critics have been successfully used in inductive proof plans to speculate missing lemmas, revise the induction scheme and generalise theorems [17][18].

5 Rule–of–Thumb Coinduction

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

The first, the `coinduction` method, involves recognising that a greatest fixpoint is involved and reformulating the goal in terms of this greatest fixpoint[1].

An observational equivalence problem is generally of the form:

$$\forall \bar{x} : \tau_1 \times \ldots \times \tau_n . f(\bar{x}) \equiv g(\bar{x}) \tag{10}$$

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

where f and g are functions from $\tau_1 \times \ldots \times \tau_n$ to lazy lists of type σ.

Given a problem of this form the `coinduction` method produces the goal

$$range(\langle f,\ g\rangle) \subseteq LlistD_fun(range(\langle f,\ g\rangle)) \tag{11}$$

$range(\langle f,\ g\rangle)$ is the relation $\mathcal{R}$, the choice of which was described as a eureka step in §2.1. It is a first guess at an appropriate relation for $\mathcal{R}$ and is, in fact, remarkably successful at finding proofs without the need for any critics. We have called this method of guessing $\mathcal{R}$ *Rule–of–Thumb* coinduction.

Rule–of–thumb coinduction fails when the chosen relation isn't general enough for the problem. In effect it picks out the smallest possible candidate for $\mathcal{R}$, given the problem under investigation. It fails when the tails of the lists are not also related by $\mathcal{R}$ which suggests that, if the theorem is true, the relation will have to be extended to allow a proof.

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

The `gfp_membership` method transforms (11) into the two goals

$$\forall \mathcal{T}, \forall \bar{x}.\langle(f(\bar{x}),\ g(\bar{x})\rangle \in \mathcal{T} \rightarrow \forall \bar{y}.\langle tl(f(\bar{y})),\ tl(g(\bar{y}))\rangle \in \mathcal{T} \tag{12}$$

$$\forall \bar{y}.hd(f(\bar{y})) = hd(g(\bar{y})) \tag{13}$$

The following theorem justifies this step. Its proof included as an appendix.

Theorem1. *Let* $\Phi \stackrel{\text{def}}{=} \{\langle h :: l_1,\ h :: l_2\rangle : \langle l_1,\ l_2\rangle \in range(\langle(f(\bar{x}),\ g(\bar{x}))\rangle)\} \cup \{\langle nil,\ nil\rangle\}$

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

It will be noticed that there is an extra "case" here where $f(x) = g(x) = nil$. This isn't always needed in the proof, and isn't used in any of the examples discussed in this paper. It is somewhat equivalent to the base case in an inductive proof.

At this point rippling, a method for guiding rewriting developed by the MRG group in Edinburgh [9] is used to attempt to complete the proof. Rippling is a terminating rewrite method and is used in conjunction with the `fertilize` method. Inductive proofs use two sorts of fertilization: *weak* fertilization where the induction hypothesis is used as a rewrite rule within the conclusion and *strong* fertilization where a direct appeal is made to the hypothesis since it is identical to the (now rewritten) conclusion. *CLAM* uses strong fertilization in coinductive proofs to prove that the tails of two lists are related by the trial bisimulation when the coinduction hypothesis and conclusion are identical. *CLAM* always attempts fertilization before it attempts rippling.

We believe the exact rewriting method used in coinductive proofs to be relatively unimportant so long as it is terminating since termination is required to determine failure and hence motivate the use of critics.

Should the `gfp_membership` method and rewriting fail to find a proof, *CLAM* will use a proof critic to attempt to find a suitable revision of $\mathcal{R}$. The current relation, $\mathcal{R}$, under investigation at any one time is referred to as the *trial bisimulation*.

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 $\mathcal{S}$. Clearly once these methods are linked to *Oyster* then this precondition will also have to be proved in order for a formal proof to be developed. This isn't necessary in the proof planning stage since knowledge about the heuristic used by the `coinduction` method is sufficient.

6 Critics for Coinduction

The rule–of–thumb coinduction heuristic doesn't always work, as illustrated by the following example. We will use this as a worked example to explain the use of the basic form of the `revise_bisimulation` critic. Most of the examples involve the use of a more advanced form of the critic.

Example2.

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

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

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

The `coinduction` method recognises that the theorem can be proved using coinduction with the function $LlistD_fun$ and uses the rule–of–thumb method, choosing $\mathcal{R} \stackrel{\text{def}}{=} range(\langle lswap(a, b),\ merge(lconst(a), lconst(b))\rangle$ to produce the subgoal

$$\begin{array}{c} range(\langle lswap(a, b),\ merge(lconst(a), lconst(b))\rangle\ ) \subseteq \\ LlistD_fun(range(\langle lswap(a, b),\ merge(lconst(a), lconst(b))\rangle\ )) \end{array} \tag{14}$$

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. In all the proofs in what remains of this paper, we shall ignore subgoals dealing with the equality of the heads of both lists and base cases. The first subgoal is

$$\begin{array}{c} \langle lswap(a, b),\ merge(lconst(a), lconst(b))\rangle \in \mathcal{T} \rightarrow \\ \langle tl(a :: b :: lswap(a, b)),\ tl(merge(a :: lconst(a), b :: lconst(b)))\rangle \in \mathcal{T} \end{array} \tag{15}$$

which rewrites to

$$\begin{aligned} \ldots &\rightarrow \langle b :: lswap(a, b),\ tl(a :: b :: merge(lconst(a), lconst(b)))\rangle \in \mathcal{T} \\ \ldots &\rightarrow \langle b :: lswap(a, b),\ b :: merge(lconst(a), lconst(b))\rangle \in \mathcal{T} \end{aligned}$$

CLAM fails to prove this because none of its proof methods will apply.

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

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

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 the process of rippling has terminated without fertilization occurring.

This means that at the point the proof of example 2 has reached *CLAM* has identified that no amount of rewriting is going to show that $\langle b :: lswap(a, b),\ b :: merge(lconst(a), lconst(b))\rangle$ is in $range(\langle lswap(a, b),\ merge(lconst(a), lconst(b))\rangle)$. This is why it is necessary to have a terminating rewriting method for the critic to be called.

As discussed above a critic will have a number of preconditions which must be satisfied before it comes into play. *CLAM* includes facilities for examining the proof plan already generated for occurrences of certain methods and for restarting proofs from previous nodes in a plan. The

If

1. The coinduction method has been used in this branch of the proof.
2. The current goal is $\langle f_1,\, g_1\rangle \in \mathcal{T} \wedge \ldots \wedge \langle f_n,\, g_n\rangle \in \mathcal{T} \rightarrow \langle k,\, l\rangle \in \mathcal{T}$. Where k and l are not of the form $tl(\ldots)$
3. Rewriting has terminated but the goal is not a theorem.

Then

Change the trial bisimulation by adding the set $range(\langle k,\, l\rangle)$ 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.

Figure 1: The Basic Revise Bisimulation Critic

`revise_bisimulation` critic checks that a coinduction proof is being attempted. It then patches the proof by altering the trial bisimulation provided by the `coinduction` method, and returning to that point in the proof search. Figure 1 presents the critic's pre–conditions in a natural language format for ease of understanding. The actual critic is written in Prolog.

Using the critic outlined in figure 1, CL^AM revises the trial bisimulation to

$$\begin{aligned}\mathcal{R}' \;=\;& range(\lambda a, b.\, \langle lswap(a,b),\, merge(lconst(a), lconst(b))\rangle) \;\cup \\ & \langle b :: lswap(a,b),\, b :: merge(lconst(a), lconst(b))\rangle)\end{aligned} \tag{17}$$

and starts out again with the goal.

$$\mathcal{R}' \subseteq LlistD_fun(\tau,\, \mathcal{R}') \tag{18}$$

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

$$\begin{gathered}\langle lswap(a,b),\, merge(lconst(a), lconst(b))\rangle \in \mathcal{T} \wedge \\ \langle b :: lswap(a,b),\, b :: merge(lconst(a), lconst(b))\rangle \in \mathcal{T} \rightarrow \\ \langle tl(a :: b :: lswap(a,b)),\, tl(merge(a :: lconst(a), b :: lconst(b)))\rangle \in \mathcal{T}\end{gathered} \tag{19}$$

and

$$\begin{gathered}\langle lswap(a,b),\, merge(lconst(a), lconst(b))\rangle \in \mathcal{T} \wedge \\ \langle b :: lswap(a,b),\, b :: merge(lconst(a), lconst(b))\rangle \in \mathcal{T} \rightarrow \\ \langle tl(b :: lswap(a,b)),\, tl(b :: merge(lconst(a), lconst(b)))\rangle \in \mathcal{T}\end{gathered} \tag{20}$$

The proof of (19) proceeds identically as in the first proof attempt, however, the extra hypothesis allows fertilization to occur. (20) resolves similarly.

7 The Use of Generalisation in the Critic

What commonly happens when this approach is taken to patching a proof is that a series of relations are progressively added to the trial bisimulation in a divergent process, as illustrated below. In this case a generalisation that encompasses all the revisions is required.

Example3.

$$\forall X : \tau,\ f : \tau \to \tau.\ h_f(X) = X :: map(f, h_f(X)) \longrightarrow$$
$$\forall x : \tau,\ f : \tau \to \tau.\ h_f(x) = iterates(f, x)$$

The `coinduction` method produces the goal

$$range(\langle h_f(x),\ iterates(f,x)\rangle) \subseteq LlistD_fun(range(\langle h_f(x),\ iterates(f,x)\rangle\ )) \tag{21}$$

The `gfp_membership` method then gives the goal:

$$\langle h_f(x),\ iterates(f,x)\rangle \in \mathcal{T} \to \langle tl(x :: map(f, h_f(x))),\ tl(x :: iterates(f), f(x))\rangle \in \mathcal{T} \tag{22}$$

which rewrites to:

$$\langle h_f(x),\ iterates(f,x)\rangle \in \mathcal{T} \to \langle map(f, h_f(x)),\ iterates(f, f(x))\rangle \in \mathcal{T} \tag{23}$$

At this point the proof attempt fails.

The `revise bisimulation` critic will then intervene, adding $range(\langle map(f, h_f(x)),\ iterates(f, f(x))\rangle)$ to $\mathcal{R}$ and (with the `gfp_membership` method) producing two subgoals. The first subgoal is similar to (22), only this time the addition of the extra elements to $\mathcal{R}$ means it is provable. The second new subgoal is:

$$\begin{array}{c}\langle h_f(x),\ iterates(f,x)\rangle \in \mathcal{T} \wedge \\ \langle map(f, h_f(x)),\ iterates(f, f(x))\rangle \in \mathcal{T} \to \\ \langle tl(map(f, x :: map(f, h_f(x)))),\ tl(f(x) :: iterates(f, f(f(x))))\rangle \in \mathcal{T}\end{array} \tag{24}$$

Once again the `revise bisimulation` critic will intervene and suggest adding $range(\langle map(f, map(f, h_f(x))),\ iterates(f, f(f(x)))\rangle)$ to $\mathcal{R}$. Clearly this is going to get the prover nowhere; we have embarked upon a divergent process. We need an extension of the `revise bisimulation` critic, to recognise when a divergent set of revisions has been embarked upon, and propose a suitable generalisation. This extension, a divergence check, is based on Walsh's divergence critic [27].

7.1 The Divergence Check

A divergence check is added to the `revise_bisimulation` critic to spot when the sort of divergence described above is occurring and to 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 the accumulating structure using difference matching [3].

For instance take the sequence of sets added to the trial bisimulation in Example 3

$$\begin{array}{c}s_0 = \langle h_f(x),\ iterates(f,x)\rangle \in \mathcal{T} \\ s_1 = \langle map(f, h_f(x)),\ iterates(f, f(x))\rangle \in \mathcal{T} \\ s_2 = \langle map(f, map(f, h_f(x))),\ iterates(f, f(f(x)))\rangle \in \mathcal{T}\end{array}$$

Difference matching successive equations adds the annotations

$$\begin{array}{c}s'_0 = \langle h_f(x),\ iterates(f,x)\rangle \in \mathcal{T} \\ s'_1 = \langle \boxed{map(f, \underline{h_f(x)})},\ iterates(f, \boxed{f(\underline{x})})\rangle \in \mathcal{T} \\ s'_2 = \langle \boxed{map(f, \underline{map(f, h_f(x))})},\ iterates(f, \boxed{f(\underline{f(x)})})\rangle \in \mathcal{T}\end{array}$$

An annotation consists of a *wavefront*, a box with a *wavehole*, an underlined term. The *skeleton* is formed by deleting everything that appears in the wavefront but not in the wavehole. The annotations above are determined on the conditions that the skeleton of each term in the

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

Figure 2: Conditions for the Divergence Check

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 with a substitution σ. 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 with substitution σ iff $\sigma(skeleton(s')) = t$ 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 shouldn't be surprising since the difference matching singles out differences between two equations and it is precisely these differences which are presenting fertilization occurring between them.

The conditions for the divergence check to succeed appear in figure 2 and are adapted closely from those described for Walsh's divergence critic[27].

Of course, identifying that divergence is taking place is only half the battle, it is also necessary to find an appropriate generalisation to replace the trial bisimulation. Walsh's divergence critic which so far has been followed very closely, patched the proofs he was attempting by speculating and proving additional lemmas. What is needed for coinductive proofs is some generalisation of the trial bisimulation.

The divergence is being caused by the repeated addition of H^i (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 $(\ldots)^n$

$$\begin{aligned} F^0(X) &\Rightarrow X \\ F^{s(N)}(X) &\Rightarrow F(F^N(X)) \end{aligned}$$

to produce the generalisation $range(\langle G^s((H^s)^N(U^s_0)),\, G^t((H^t)^N(U^t_0))\rangle)$ and put it in place of the previous sequence of sets in the trial bisimulation.

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

7.2 Back to the Example

In the proof we are attempting the critic assigns G^s, G^t, H^s, H^t and U^s_0 and U^t_0 as id (the identity function), $iterates(f)$, $map(f)$, f, $h_f(x)$ and x respectively so giving the goal

$$\begin{array}{c} range(\langle map(f)^n(h_f(x)),\, iterates(f, f^n(x))\rangle) \subseteq \\ LlistD_fun(range(\langle map(f)^n(h_f(x)),\, iterates(f, f^n(x))\rangle\ )) \end{array} \tag{25}$$

The `gfp_membership` method produces the goal

$$\begin{aligned} & \langle map(f)^n(h_f(x)),\, iterates(f, f^n(x))\rangle \in \mathcal{T} \rightarrow \\ & \qquad \langle tl(map(f)^n(x :: (map(f, h_f(x))))),\, tl(f^n(x) :: iterates(f, f(f^n(x))))\rangle \in \mathcal{T} \\ \ldots \;\rightarrow\; & \langle tl(f^n(x) :: map(f)^n(map(f, h_f(x)))),\, iterates(f, f(f^n(x)))\rangle \in \mathcal{T} \\ \ldots \;\rightarrow\; & \langle map(f)^{s(n)}(h_f(x)),\, iterates(f, f^{s(n)}(x))\rangle \in \mathcal{T} \end{aligned}$$

which can be solved by fertilization.

If

1. The coinduction method has been used in this branch of the proof.
2. The current goal is $\langle f_1,\ g_1\rangle \in \mathcal{T} \wedge \ldots \wedge \langle f_n,\ g_n\rangle \in \mathcal{T} \rightarrow \langle k,\ l\rangle \in \mathcal{T}$. Where k and l don't both equal $tl(\ldots)$.
3. Rewriting has terminated but the goal is not a theorem.

Then if

The Divergence Check has succeeded identifying G^s, G^t, H^s, H^t and U_0^s and U_0^t for some sequence of sets within the trial bisimulation.

Then

Change the trial bisimulation by replacing the sequence with the set $range(\langle G^s((H^s)^n(U_0^s),\ G^t((H^t)^n(U_0^t)\rangle)$ 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 $range(\langle k,\ l\rangle)$ 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.

Figure 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, using Sicstus Prolog.

It has been tried on 19 example problems involving the observational equivalence 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 [28] and problems devised by ourselves.

The rule-of-thumb method has also been tried on theorems involving other coinductive datatypes with similarly encouraging results. At the time of writing, however, the `revise_bisimulation` critic had not been extended to deal with such datatypes.

Of the five examples *CLAM* failed to prove two failed because, as yet, *CLAM* isn't really equipped to cope with additional hypotheses in the theorem or rewrite rules e.g. $A \subseteq B \rightarrow A \cup B \equiv B$.

The rest failed because of over–generalisation in one form or another. For instance the theorem $jump(0,1) \equiv merge(jump(0,2), jump(1,2))$ where

$$jump(N, M) \Rightarrow N :: jump(N + M, M) \tag{26}$$

When the integers 1 and 2 are presented in terms of s and 0 *CLAM* will eventually produce the goal

$$\begin{array}{c}\langle jump(0, s(0)),\ merge(jump(0, s(s(0))), jump(s(0), s(s(0))))\rangle \in \mathcal{T} \rightarrow \\ \langle jump(s(0), s(0),\ merge(jump(s(0), s(s(0))), jump(s(s(0)), s(s(0))))\rangle \in \mathcal{T}\end{array} \tag{27}$$

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

$$\mathcal{R} \stackrel{\text{def}}{=} range(\langle jump(n, n),\ merge(jump(n, s(n)), jump(s(n), s(n))\rangle \tag{28}$$

which is not a bisimulation. The desired generalistion was

$$\mathcal{R} \stackrel{\text{def}}{=} range(\langle jump(n, 1),\ merge(jump(n, 2), jump(s(n), 2)\rangle \tag{29}$$

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

Theorems Proved by Rule–of–Thumb Coinduction Alone
$nil <> l = l$
$l <> (m <> n) = (l <> m) <> n$
$map(f, iterates(f, m)) = iterates(f, f(m))$
$map(f, x <> y :: z) = map(f, x) <> map(f, y :: nil <> z)$
$map(f \circ g, l) = map(f, map(g, l))$
$lconst(m) = map(id, lconst(m))$
$map(h, iterates(f, a)) = inflist(a, h, t)$
$flatten(map2(f, l)) = map(f, flatten(l))$
$flatten(explode(l)) = l$

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

Theorems *CLAM* failed to Prove
$jump(0, 1) = merge(jump(0, 2), jump(1, 2))$
$del(0, lconst(1)) = del(0, lswap(1, 0))$
$parity(T, T) = numparity(0, T)$
$A \subseteq B \rightarrow A \cup B = B$
$inflist(a, id, s) = jump(a, 1)$

9 Related Work

Work has been done in both Isabelle and HOL to provide support for inductive definitions.

Graham 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 (6) and forms goals equivalent to those formed by the `gfp_membership` method, (12) and (13). It then uses a series of simplification and evaluation tactics to prove those goals. Collins reports [10] 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 [24] allowing the coinduction rule to be derived and used. No specific tactics for coinductive proofs have been provided, but Isabelle's own very powerful simplification tactics are more than capable of handling much of the proof in an automated fashion.

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

Toby Walsh's [27] divergence critic, 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 wasn't present. The preconditions for the divergence check used for coinduction are translated more–or–less directly across from those used for SPIKE. However Walsh's critic didn't hypothesize generalisations, instead it sought to speculate and prove lemmas needed to complete the proof.

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 which offer support for coinduction 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 try and prevent over–generalisation occurring in the critics already developed.

Whilst I have followed Paulson's formulation of lazy lists here, 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, I hope to be able to adapt the proof plans to use the lts style representation and bring this work more in line with the rest of the field.

There are 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. These can be coped with through more sophisticated analysis of l and the inclusion of $\Uparrow$, divergence, into the theory of observational equivalence. I haven't attempt to extend the methods and critics described above to this kind of theorem.

References

[1] Abramsky, S. The Lazy Lambda Calculus. In Turner, D. editor, Research Topics in Functional Programming, pages 65–116. Addison–Wesley, 1977.

[2] Bird, R., Wadler, P., Introduction to Functional Programming. Prentice–Hall, 1988.

[3] Basin, D. and Walsh, T. Difference Matching. In Kapur, D. editor, 11th Conference on Automated Deduction, pages 295–309. Springer–Verlag, 1992. LNCS No. 607

[4] Bouhoula, A., and Rusinowitch, M. Automatic Case Analysis in Proof by Induction. In Proceedings of the 13th IJCAI. International Joint Conference on Artificial Intelligence, Chambery, France, 1993.

[5] Bundy, A. The Use of Explicit Plans to Guide Inductive Proofs. In Lusk, R. and Overbeek, R. editors, 9th Conference on Automated Deduction, pages 111–120, Springer–Verlag, 1988. Longer version availabel from Edinburgh as DAI Research Paper No. 349.

[6] Bundy, A., van Harmelen, F., Horn, C., Smaill, A. The Oyster–Clam system. In Stickel, M.E. editor, 10th International Conference on Automated Deduction, pages 647–648. Springer–Verlag, 1990. LNAI No.449. Also availabel from Edinburgh as DAI Research Paper 507.

[7] Bundy, A., van Harmelen, F., Hesketh, J., Smaill, A. Experiments with proof plans for induction. Journal of Automated Reasoning, 7:303–324, 1991. Earlier version availabel from Edinburgh as DAI Research Paper No, 413.

[8] Bundy, A. A Science of Reasoning. In Lassez, J–L. and Plotkin, G. editors, Computational Logic: Essays in Honour of Alan Robinson, pages 178–198. MIT Press, 1991. Also availabel from Edinburgh as DAI Research Paper 445.

[9] Bundy, A., Stevens, A., van Harmelen, F., Ireland, A., Smaill, A. Rippling: A heuristic for guiding inductive proofs. Research Paper 567. Dept. of Artificial Intelligence. Edinburgh, 1991. In the Journal of Artificial Intelligence.

[10] Collins, G. A Proof Tool for Reasoning About Functional Programs. In von Wright, J., Grundy, J. and Harrison, J. editors. 9th International Conference of Theorem Proving in Higher Order Logics, Turku, Finland, 1996, pp. 109–124.

[11] Fiore, M, P. A Coinduction Principle for Recursive Data Types Based on Bisimulation. In Proc 8th Annual Symposium on Logic in Computer Science, Montreal, pages 110–119, IEEE Computer Society Press, Washington.

[12] Frost, J. A Case Study of Co–induction in Isabelle HOL. Technical Report 308, University of Cambridge, Computer Laboratory. 1993.

[13] Howe, D.J., Equality in lazy computation systems. In Proceedings of the 4th IEEE Symposium on Logic in Computer Science, pp. 198–203

[14] Gordon, A. D. Functional Programming and Input/Output. Cambridge University Press, 1994.

[15] Gordon, A. D. Bisimilarity as a theory of functional programming. In 11th Annual Conference on Mathematical Foundations of Programming Semantics, Volume 1 of Electronic Notes in Theoretical Computer Science. Elsevier Science Publishers B.V. To appear. Extended version availabel as BRICS Note NS–95–3, Aarhus University, 1995.

[16] Gordon, A. D., Rees, G. D. Bisimilarity for a First–Order Calculus of Objects with Subtyping. To appear in Proceedings of the 23rd Annual ACM Symposium on Principles of Programming Languages, St. Petersburg Beach, Florida, 1996.

[17] Ireland, A. 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 availabel from Edinburgh as DAI Research Paper 592.

[18] Ireland, A., Bundy, A., Productive use of Failure in Inductive Proof. Journal of Automated Reasoning, 16(1–2):79–111, 1996. Also availabel as DAI Research Paper No. 716, Dept. of Artificial Intelligence, Edinburgh.

[19] Milner, R. Communication and Concurrency. Prentice–Hall, 1989.

[20] Milner, R. and Tofte, M. Co–Induction in Relational Semantics. Theoretical Computer Science, 87:209–220, 1991.

[21] Paulin–Mohring, C. Circuits as streams in Coq Verification of a sequential multiplier. Research Report RR95–16, Laboratoire de l'Informatique du Parallelisme, 1995.

[22] Paulson, L.C. Co-induction and Co-recursion in Higer-Order Logic. Technical Report 304, University of Cambridge, Computer Laboratory, 1993.

[23] Paulson, L.C. Isabelle: A Generic Theorem Prover. Springer-Verlag LNCS 828, 1994.

[24] Paulson, L.C. A Fixpoint Approach to Implementing (Co)Inductive Definitions. In Alan Bundy, editor, Proc 12th Int. Conf. Automated Deduction, volume 814 of LNCS, pages 148–161. Springer–Verlag, 1994.

[25] Pitts, A. M. A Co-induction Principle for recursively Defined Domains. Theoretical Computer Science **124**:195–219, 1994.

[26] Tarski, A., A Lattice-Theoretical Fixpoint Theorem and its Applications. Pacific Journal of Mathematics, 5:285–309, 1955.

[27] Walsh, T., A Divergence Critic. In Alan Bundy, editor, 12th Conference on Automated Deduction, Lecture Notes in Artificial Intelligence, Vol. 814, pages 14-28, Nancy, France. Springer-Verlag

[28] Wikström, Å. Functional Programming Using Standard ML. Prentice–Hal, 1987.

A Proof of Theorem 1

Let $\Phi \stackrel{\text{def}}{=} \{\langle h :: l_1,\, h :: l_2\rangle : \langle l_1,\, l_2\rangle \in range(\langle (f(\bar{x}),\, g(\bar{x}))\rangle)\} \cup \{\langle nil,\, nil\rangle\}$

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

Proof. $\Rightarrow$

$$range(\langle f,\, g\rangle) \subseteq \{\langle h :: l_1,\, h :: l_2\rangle : \langle l_1,\, l_2\rangle \in range(\langle f,\, g\rangle)\} \cup \{\langle nil,\, nil\rangle\} \tag{30}$$

This means that for any set of variables $\bar{x}$

$$\exists \bar{z}.\ \langle f(\bar{x}),\, g(\bar{x})\rangle = \langle h :: f(\bar{z}),\, h :: g(\bar{z})\rangle \ \vee\ \langle f(\bar{x}),\, g(\bar{x})\rangle \in \{\langle nil,\, nil\rangle\} \tag{31}$$

This implies that for all $\bar{x}$ either $hd(f(\bar{x})), hd(g(\bar{x}))$, etc. are all defined or $f(\bar{x}) = g(\bar{x}) = nil$. Hence we shall split the domain of $\langle f,\, g\rangle$ into two sets Ψ and Ξ. Ψ is defined to contain only those $\bar{x}$ for which $hd(f(\bar{x})), hd(g(\bar{x}))$, etc. are defined and Ξ contains only those $\bar{x}$ for which $f(\bar{x}) = g(\bar{x}) = nil$. It should be noted that $\Phi \cap \Xi \equiv \{\}$ and $\Phi \cup \Xi \equiv domain(\langle f,\, g\rangle)$.

Rewriting the first disjunct of (31) by splitting off the heads and tails of $f(\bar{x})$ and $g(\bar{x})$ gives

$$hd(f(\bar{x})) = hd(g(\bar{x})) \tag{32}$$

$$\langle tl(f(\bar{x})),\, tl(g(\bar{x}))\rangle = \langle f(\bar{z}),\, g(\bar{z})\rangle \tag{33}$$

Let $\mathcal{A}$ be the set $\{\langle tl(l_1),\, tl(l_2)\rangle : \langle l_1,\, l_2\rangle \in range(\langle f,\, g\rangle|\Psi) \wedge hd(l_1) = hd(l_2)\}$. Where $range(f|\mathcal{S})$ is the range of the function f when its domain is restricted to the set $\mathcal{S}$.

Then

$$\mathcal{A} \subseteq range(\langle f,\, g\rangle) \tag{34}$$

Suppose $\langle l_1,\, l_2\rangle \in range(\langle f,\, g\rangle|\Psi)$. By (32) this means that $hd(tl(l_1)) = hd(tl(l_2))$. In which case (from the definition of $\mathcal{A}$) $\langle tl(l_1),\, tl(l_2)\rangle$ is in $\mathcal{A}$. Hence by (34) $\langle tl(l_1),\, tl(l_2)\rangle$ is in $range(\langle f,\, g\rangle)$.

So

$$\begin{aligned} \langle l_1,\, l_2\rangle \in range(\langle f,\, g\rangle|\Psi) \quad \rightarrow \quad & (\langle tl(l_1),\, tl(l_2)\rangle \in range(\langle f,\, g\rangle) \\ & \wedge\, hd(l_1) = hd(l_2)) \end{aligned}$$

The second disjunct of (31) refers to those elements of the domain of $\langle f,\, g\rangle$ in Ξ.

Since $\Psi \cup \Xi = domain(\langle f,\, g\rangle)$

$$\begin{aligned} \langle l_1,\, l_2\rangle \in range(\langle f,\, g\rangle) \quad \rightarrow \quad & (\langle tl(l_1),\, tl(l_2)\rangle \in range(\langle f,\, g\rangle) \\ & \wedge\, hd(l_1) = hd(l_2)) \\ & \vee\, l_1 = l_2 = nil \end{aligned}$$

Rewriting this slightly gives

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

$\Leftarrow$

$$\forall \mathcal{T}.(\forall \bar{x}.\ \langle f(\bar{x}),\ g(\bar{x})\rangle \in \mathcal{T} \quad \rightarrow \quad \forall \bar{x}.(\ \langle tl(f(\bar{x})),\ tl(g(\bar{x}))\rangle \in \mathcal{T} \wedge hd(f(\bar{x})) = hd(g(\bar{x}))) \\ \vee \langle f(\bar{x}),\ g(\bar{x})\rangle \in \{\langle nil,\ nil\rangle\}))$$

Then this is true even if $\mathcal{T}$ is the set $range(\langle f,\ g\rangle)$ so

$$\forall \bar{x}.\ \langle f(\bar{x}),\ g(\bar{x})\rangle \in range(\langle f,\ g\rangle) \quad \rightarrow \quad \forall \bar{x}.(\langle tl(f(\bar{x})),\ tl(g(\bar{x}))\rangle \in range(\langle f,\ g\rangle) \\ \wedge\ hd(f(\bar{x})) = hd(g(\bar{x}))) \\ \vee \langle f(\bar{x}),\ g(\bar{x})\rangle \in \{\langle nil,\ nil\rangle\}$$

$\langle f(\bar{x}),\ g(\bar{x})\rangle \in range(\langle f,\ g\rangle)$ is trivially true and once again we will leave consideration of the second disjunct $\langle f(\bar{x}),\ g(\bar{x})\rangle \in \{\langle nil,\ nil\rangle\}$ to one side for the moment.

Hence, dealing only with the first disjunct.

$$\forall \bar{x}.\ (\langle tl(f(\bar{x})),\ tl(g(\bar{x}))\rangle \in range(\langle f,\ g\rangle)\ \wedge\ hd(f(\bar{x})) = hd(g(\bar{x}))) \tag{35}$$

Stripping off the set notation

$$\forall \bar{x}.\exists \bar{y}.\ \langle tl(f(\bar{x})),\ tl(g(\bar{x}))\rangle = \langle f(\bar{y}),\ g(\bar{y})\rangle\ \wedge\ hd(f(\bar{x})) = hd(g(\bar{x}))) \tag{36}$$

and moving the heads and tails around.

$$\forall \bar{x}.\exists \bar{y}.\ \langle f(\bar{x}),\ g(\bar{x})\rangle = \langle hd(f(\bar{x})) :: (f(\bar{y})),\ hd(g(\bar{x})) :: g(\bar{y})\rangle\ \wedge\ hd(f(\bar{x})) = hd(g(\bar{x}))) \tag{37}$$

Adding the second disjunct back in and rewriting as sets once more gives.

$$range(\langle f,\ g\rangle) \subseteq \{\langle h :: l_1,\ h :: l_2\rangle : \langle l_1,\ l_2\rangle \in range(\langle f,\ g\rangle)\} \cup \{\langle nil,\ nil\rangle\}$$

□