Circular Coinduction with Special Contexts

Dorel Lucanu[1] and Grigore Roşu[2]

[1] Faculty of Computer Science
Alexandru Ioan Cuza University, Iaşi, Romania, dlucanu@info.uaic.ro
[2] Department of Computer Science
University of Illinois at Urbana-Champaign, USA, grosu@cs.uiuc.edu

Abstract. Coinductive proofs of behavioral equivalence often require human ingenuity, in that one is expected to provide a "good" relation extending one's goal with additional lemmas, making automation of coinduction a challenging problem. Since behavioral satisfaction is a Π^0_2-hard problem, one can only expect techniques and methods that approximate the behavioral equivalence. Circular coinduction is an automated technique to prove behavioral equivalence by systematically exploring the behaviors of the property to prove: if all behaviors are circular then the property holds. Empirical evidence shows that one of the major reasons for which circular coinduction does not terminate in practice is that the circular behaviors may be guarded by a context. However, not all contexts are safe. This paper proposes a large class of contexts which are safe guards for circular behaviors, called special contexts, and extends circular coinduction appropriately. The resulting technique has been implemented in the CIRC prover and experiments show that the new technique can prove many interesting behavioral properties fully automatically.

1 Introduction

Coinduction allows us to prove properties about infinite objects, such as, for example, streams of numbers. Since many system specifications manifest infinite behaviors, coinduction is increasingly gaining interest among computer scientists. There are many efforts to mechanize proofs by coinduction, e.g., [6, 14, 18, 10, 13] among many others. Circular coinduction [18] is an automated technique to prove behavioral equivalence by systematically exploring the behaviors of the property to prove. More specifically, it derives the behavioral task until one obtains, on every derived path, either a truth or a cycle. Variants of circular coinduction have been implemented in at least three systems so far: in a behavioral extension of OBJ called BOBJ [18] (not maintained anymore), in Isabelle/HOL for CoCasl [10], and in CIRC [11].

Circular coinduction can be formalized as a three-rule proof system deriving pairs of the form $\mathcal{H} \Vdash^{\circlearrowleft} \mathcal{G}$, where $\mathcal{H}$ (*hypotheses*) and $\mathcal{G}$ (*goals*) are sets of equations [17]. Some hypotheses are *frozen* and written in a box (e.g., $\boxed{e}$), with the intuition that those cannot be used in contextual reasoning. This can be easily achieved by defining the box as a wrapper operator of a fresh sort result, *Frozen*. The hypotheses $\mathcal{H}$ can contain both frozen and normal (or "unfrozen") equations. The goals $\mathcal{G}$ are frozen. The circular coinduction proof system in [17] is the particular case of that in Fig. 1, when the set Γ of special contexts used in the third rule is always empty. The first rule says that the derivation is finished

when there are no goals left. The second rule discards those goals which can be proved using the base entailment system extended over frozen equations. The third rule is the distinguished rule of circular coinduction that enables circular reasoning and it essentially says that in order to prove behavioral property e, assume it and prove its derivatives $\Delta[e]$ (Δ is a distinguished set of operations called *derivatives*). Without freezing, the above would hold vacuously by the congruence rule; freezing inhibits the applications of congruence.

In this paper we extend the basic circular coinduction proof system with the ability to use hypotheses defined by *special contexts*. The result is a more powerful proof system, presented in Figure 1, able to automatically prove a larger class of behavioral properties. The techniques presented in this paper have been implemented and extensively evaluated in CIRC [11], a behavioral extension of Full Maude [5] tuned and optimized for automated and combined inductive and coinductive proving.

$$\frac{\cdot}{\mathcal{H} \Vdash^{\circlearrowleft} \emptyset} \qquad \frac{\mathcal{H} \Vdash^{\circlearrowleft} \mathcal{G}, \quad \mathcal{H} \vdash \boxed{e}}{\mathcal{H} \Vdash^{\circlearrowleft} \mathcal{G} \cup \{\boxed{e}\}}$$

$$\frac{\mathcal{H} \cup \boxed{\Gamma[e]} \cup \{\boxed{e}\} \Vdash^{\circlearrowleft} \mathcal{G} \cup \boxed{\Delta[e]}}{\mathcal{H} \Vdash^{\circlearrowleft} \mathcal{G} \cup \{\boxed{e}\}, \quad \begin{array}{l} e \text{ hidden equation} \\ \Gamma \text{ special contexts} \end{array}}$$

Fig. 1. Three-rule proof system for circular coinduction with special contexts

We exemplify the notions and techniques in this paper using the following behavioral specification of infinite streams. The elements of streams belong to an algebraic structure consisting of a commutative ring enriched with an additional operation *not*, defined for example using the following equations: $D + 0 = D$, $D \cdot 1 = D$, $D + (-D) = 0$, $D \cdot 0 = 0$, $(-1) \cdot D = -D$, $D \cdot (D_1 + D_2) = D \cdot D_1 + D \cdot D_2$, $not(0) = 1$, $not(1) = 0$, $not(not(D)) = D$, We consider streams defined in terms of head (*hd*) and tail (*tl*), as derivatives, together with several operations behaviorally defined in Fig. 2. The operation $[\cdot]$ expresses the embedding of data into streams, and the operations *zero*, $+$ and $\times$ extend the corresponding operations over data to streams. The stream *morse* is the famous Thue-Morse sequence (see, e.g., [2]). An alternative definition for the Thue-Morse sequence is given by *altMorse*, using an auxiliary function f. The Thue-Morse sequence is a fixed point for f; this can be proved by circular coinduction [17].

$zip(odd(S), even(S)) = S$	
$f(not(S)) = not(f(S))$	
$f(S) = zip(S, not(S))$	
$f(morse) = morse$	
$morse = altMorse$	*
$S \times zero = zero$	*
$S_1 \times (S_2 + S_3) = S_1 \times S_2 + S_1 \times S_3$	*
$(S_1 + S_2) \times S_3 = S_1 \times S_3 + S_2 \times S_3$	*

Having given the definition in Fig. 2, we can now prove automatically several properties about streams using the new CIRC system which implements the circular coinductive technique proposed in this paper. The table to the right includes some of these properties, tagging with a "*" those that need the special contexts extension proposed in this paper (the tagged properties cannot be proved automatically using the previous circular coinductive technique in [17]). Let us discuss two of these properties, pointing out only the high level proof steps using the circular coinduction proof system and not how CIRC works.

Opn. name	Equations	
$zero$	$hd(zero) = 0$	$tl(zero) = zero$
$[\cdot]$	$hd[S] = hd(S)$	$tl[S] = zero$
$+$	$hd(S_1 + S_2) = hd(S_1) + hd(S_2)$	$tl(S_1 + S_2) = tl(S_1) + hd(S_2)$
$\times$	$hd(S_1 \times S_2) = hd(S_1) \cdot hd(S_2)$	
	$tl(S_1 \times S_2) = tl(S_1) \times S_2 + [hd(S_1)] \times tl(S_2)$	
odd	$hd(odd(S)) = hd(S)$	$tl(odd(S)) = even(tl(S))$
$even$	$even(S) = odd(tl(S))$	
zip	$hd(zip(S_1, S_2)) = hd(S_1)$	$tl(zip(S_1, S_2)) = zip(S_2, tl(s_1))$
$morse$	$hd(morse) = 0$	$hd(tl(morse)) = 1$
	$tl^2(morse) = zip(tl(morse), not(tl(morse)))$	
f	$hd(f(S)) = hd(S)$	$hd(tl(f(S))) = not(hd(S))$
	$tl^2(f(S)) = f(tl(S))$	
$altMorse$	$hd(altMorse) = 0$	$hd(tl(altMorse)) = 1$
	$tl^2(altMorse) = f(tl(altMorse))$	

Fig. 2. `STREAM`: A behavioral specification of streams and operations on them.

Consider property $zip(odd(S), even(S)) = S$. Start with $\mathcal{H}_0 = \texttt{STREAM}$ and $\mathcal{G}_0 = \{\boxed{zip(odd(S), even(S))} = \boxed{S}\}$. Apply third derivation rule with e_0 the equation $zip(odd(S), even(S)) = S$, and obtain $\mathcal{H}_1 = \mathcal{H}_0 \cup \{\boxed{e_0}\}$ and $\mathcal{G}_1 = \{\boxed{e_1}, \boxed{e_2}\}$, where e_1 and e_2 are the derivatives of e_0, respectively: $hd(zip(odd(S), even(S))) = hd(S)$ and $tl(zip(odd(S), even(S))) = tl(S)$. Apply second rule for $\boxed{e_1}$ using the stream equations, and obtain $\mathcal{H}_2 = \mathcal{H}_1$ and $\mathcal{G}_2 = \{\boxed{e_2}\}$. Apply second rule for $\boxed{e_2}$, by first reducing it to $\boxed{zip(odd(tl(S)), even(tl(S)))} = \boxed{tl(S)}$ and then using the frozen hypothesis $\boxed{e_0}$ at its top with substitution $S \mapsto tl(S)$, we obtain $\mathcal{H}_3 = \mathcal{H}_2$ and $\mathcal{G}_3 = \emptyset$. It is easy to see now that the sequence $\mathcal{H}_i \Vdash^{\circlearrowleft} \mathcal{G}_i$, $i = 3, 2, 1, 0$, is a derivation of the proof system. Note that freezing is necessary, otherwise the derivatives would always follow from the equational congruence rule, no matter whether the task holds or not.

Consider now the property $S \times zero = zero$. By the third rule, add the frozen form of this goal $\boxed{S \times zero} = \boxed{zero}$ as hypothesis and thus generate two new goals: $\boxed{hd(S \times zero)} = \boxed{hd(zero)}$ and $\boxed{tl(S \times zero)} = \boxed{tl(zero)}$. The former is eliminated by the second rule and the latter is reduced, using the stream equations, to $\boxed{tl(S) \times zero + [hd(S)] \times zero} = \boxed{zero}$. Even if the subterms $tl(S) \times zero$ and $[hd(S)] \times zero$ are instances of the added hypothesis, this cannot be applied because the hypothesis is frozen. However, for this case it is sound to apply the frozen hypothesis under the $+$ operator, i.e., by adding "on the fly" the special hypotheses $\boxed{S \times zero + S'} = \boxed{zero + S'}$ and $\boxed{S' + S \times zero} = \boxed{S' + zero}$. These hypotheses can be automatically generated by using the "special contexts" $\gamma_1 = \ * : Stream + S'$ and $\gamma_2 = S' + * : Stream$. The new special hypotheses added are $\boxed{\gamma_1[e]}$ and $\boxed{\gamma_2[e]}$, respectively, and are added to the hypotheses using the third proof rule of the system presented in Figure 1. Circular coinduction and the two hypotheses succeeds to successfully finish the proof.

Note that some contexts cannot be special, for example, $odd(* : Stream)$. The following example is inspired from [7]. Let a and b be specified by $hd(a) = hd(b)$, $tl(a) = odd(a)$ and $tl^2(b) = odd(b)$, and let $odd(b) = a$ be the goal we

want to prove. Applying the third rule, this goal is added as frozen hypothesis $\boxed{odd(b)} = \boxed{a}$ and the following two new goals are generated: $hd(odd(b)) = hd(a)$ and $tl(odd(b)) = tl(a)$. The former is eliminated by the second rule, and the latter is reduced to $odd(odd(b)) = odd(a)$. If we assume that $odd(*{:}Stream)$ is special, and hence the hypothesis $\boxed{odd(odd(b))} = \boxed{odd(a)}$ is automatically added, then we would wrongly deduce that $odd(b) = a$. A counter-example is given by $a = 0 : 0 : 1 : 2^\infty$ and $b = 0 : 1 : 0^\infty$.

The rest of the paper is structured as follows. Section 2 introduces basic notions and notations. Section 3 and Section 4 recall from [17] the proof theoretical approach for the behavioral satisfaction and coinduction and the circular coinduction proof system, focusing on the role of the freezing operator. Section 5 introduces the concept of special hypotheses as a closure operator and extends the coinductive circularity principle to the case when the special hypotheses are used. Then the concept of special context is introduced and it is shown how these yield a particular class of special hypotheses. Section 6 presents how the CIRC theorem prover implements both the circular coinduction and the special contexts. An algorithm for automatically computing of the special contexts is briefly presented. The section ends with a result showing how sometimes the infinite execution of the circular coinduction are in fact proofs.

2 Preliminaries

We assume the reader familiar with basics of many sorted algebraic specifications and only briefly recall our notation. An algebraic specification, or simply a *specification*, is a triple (S, Σ, E), where S is a set of *sorts*, Σ is a $(S^* \times S)$-*signature* and E is a set of Σ-*equations* of the form $(\forall X)\, t = t'$ `if` $\wedge_{i \in I}\, u_i = v_i$ with t, t', u_i, and v_i Σ-terms with variables in X, $i = 0, \ldots, n$; the two terms appearing in any equality in an equation, that is the terms t, t' and each pair u_i, v_i for each $i \in I$, have, respectively, the same sort. If the sort of t and t' is s we may say that the sort of the equation is also s. When $i = 0$ we call the equation unconditional and omit the condition (i.e., write it $(\forall X)\, t = t'$). When $X = \emptyset$ we drop the quantifier and call the equation *ground*.

If Σ is a many sorted signature, then a Σ-*context* C is a Σ-term which has one occurrence of a distinguished variable $*{:}s$ of sort s; to make this precise, we may write $C[*{:}s]$ instead of just C. When Σ is understood, a Σ-context may be called just a *context*. When the sort s of $*$ is important, we may call $C[*{:}s]$ a *context for sort* s; also, when the sort of a context C (regarded as a term), say s', is important, C may be called a *context of sort* s'. If $C[*{:}s]$ is a context for sort s of sort s' and t is a term of sort s, then $C[t]$ is the term of sort s' obtained by replacing t for $*{:}s$ in C. If C is a context, $|C|$ is the *depth* of C: $|* :s| = 0$ and $|C[\sigma[*{:}s]]| = |C| + 1$ when $\sigma \in \Sigma$. A Σ-context $C[*{:}s]$ induces a partially defined *equation transformer* $e \mapsto C[e]$: if e is an equation $(\forall X)\, t = t'$ `if` c of sort s, then $C[e]$ is the equation $(\forall X \cup Y)\, C[t] = C[t']$ `if` c, where Y is the set of non-star variables occurring in $C[*{:}s]$. Moreover, if $\mathbf{C}$ is a set of contexts and E a set of equations, then $\mathbf{C}[e] = \{C[e] \mid C \in \mathbf{C}\}$, $C[E] = \{C[e] \mid e \in E\}$ and $\mathbf{C}[E] = \bigcup_{e \in E} \mathbf{C}[e]$. We extend notation $C[e]$ to the cases when C is not a context (it does not include $*$); in such cases, $C[e]$ is the identity equation $(\forall X) C = C$.

The theoretical results in this paper will be parametric in a given entailment relation $\vdash$ on many sorted equational specifications, which may, but is not enforced to, be the usual equational deduction relation [9]. For instance, it can also be the "rewriting" entailment relation ($E \vdash t = t'$ iff t and t' rewrite to the same term using E as a rewrite system), or some behavioral entailment system, etc. We need though some properties of $\vdash$, which we axiomatize here by adapting to our context the general definition of entailment system as given in [12]. Fix a signature Σ (in a broader setting, $\vdash$ could be regarded as a set $\{\vdash_\Sigma | \ \Sigma \text{ signature}\}$ indexed by the signature; however, since we work with only one signature in this paper and that signature is understood, for simplicity we fix the signature).

Definition 1. *If Δ is a set of Σ-contexts, then a* ***Δ-contextual entailment system*** *is an (infix) relation $\vdash$ between sets of equations and equations, with:* ***(reflexivity)*** *$\{e\} \vdash e$;* ***(monotonicity)*** *If $E_1 \supseteq E_2$ and $E_2 \vdash e$ then $E_1 \vdash e$;* ***(transitivity)*** *If $E_1 \vdash E_2$ and $E_2 \vdash e$ then $E_1 \vdash e$;* ***(Δ-congruence)*** *If $E \vdash e$ then $E \vdash \Delta[e]$. In the above, E, E_1, E_2 range over sets of equations and e over equations; also, we tacitly extended $\vdash$ to relate two sets of equations: $E_1 \vdash E_2$ iff $E_1 \vdash e$ for any $e \in E_2$. We let $E^\bullet$ denote the set of equations $\{e \mid E \vdash e\}$.*

One can use the above to prove many properties of $\vdash$ on sets of equations. Here are some of them used later in the paper (their proofs are simple exercises): $E \vdash \emptyset$, $E \vdash E$, if $E_1 \vdash E_2$ and $E_2 \vdash E_3$ then $E_1 \vdash E_3$, if $E_1 \vdash E_2$ then $E \cup E_1 \vdash E \cup E_2$, if $E_1 \vdash E_2$ then $E_1 \vdash \Delta[E_2]$, if $E_1 \vdash E_2$ then $E_1 \vdash E_1 \cup E_2$, if $E_1 \supseteq E_2$ then $E_1 \vdash E_2$, if $E \vdash E_1$ and $E \vdash E_2$ then $E \vdash E_1 \cup E_2$.

We take the liberty to slightly abuse the syntax of entailment and allow one to write a specification instead of a set of equations, with the obvious meaning: if $\mathcal{B} = (S, \Sigma, E)$ is a specification and e is a Σ-equation, then $\mathcal{B} \vdash e$ iff $E \vdash e$. Also, if $\mathcal{B} = (S, \Sigma, E)$ then we may write $\mathcal{B}^\bullet$ instead of $E^\bullet$.

3 Behavioral Specifications and Coinduction

A *behavioral specification* is a pair $(\mathcal{B}, \Delta)$, where $\mathcal{B} = (S, \Sigma, E)$ is a many sorted algebraic specification and Δ is a set of Σ-contexts, called *derivatives*. We let Δ_s denote all the derivatives of sort s in Δ. If $\delta[*{:}h] \in \Delta$ is a derivative, then the sort h is called a *hidden sort*; we let $H \subseteq S$ denote the set of all hidden sorts of $\mathcal{B}$. Remaining sorts are called *data, or visible, sorts* and we let $V = S - H$ denote their set. A *data operator* is an operator in Σ taking and returning only visible sorts; a *data term* is a term built with only data operators and variables of data sorts; a *data equation* is an equation built with only data terms.

Sorts are therefore split into hidden and visible, so that one can derive terms of hidden sort until they possibly become visible. Formally, a *Δ-experiment* is a Δ-context of visible sort, that is: (1) each $\delta[*{:}h] \in \Delta_v$ with $v \in V$ is an experiment, and (2) if $C[*{:}h']$ is an experiment and $\delta[*{:}h] \in \Delta_{h'}$, then so is $C[\delta[*{:}h]]$. An equation $(\forall X)\, t = t'$ `if` c is called a *hidden equation* iff the common sort of t and t' is hidden, and it is called a *data, or visible, equation* iff the common sort of t and t' is visible. In this paper we consider only equations whose conditions are conjunctions of visible equalities. If Δ is understood, then

we may write experiment for Δ-experiment and context for Δ-context. If $\mathcal{G}$ is a set of Σ-equations, we let $\mathit{visible}(\mathcal{G})$ and $\mathit{hidden}(\mathcal{G})$ denote the sets of $\mathcal{G}$'s visible and hidden equations, respectively.

Example 1. Here is a brief description of the behavioral specification of the streams: S includes a hidden sort *Stream*, for streams, and a visible sort *Data*, for data elements; Σ includes the signatures of the operations over data and streams; E include the equations over the data and streams; and Δ includes the derivatives $hd(*{:}\mathit{Stream})$ and $tl(*{:}\mathit{Stream})$.

Definition 2. *$\mathcal{B}$ **behaviorally satisfies** equation e, written $\mathcal{B} \Vdash e$, iff: $\mathcal{B} \vdash e$ if e is visible, and $\mathcal{B} \vdash C[e]$ for each appropriate experiment C if e is hidden. Let $\equiv$ be the set of equations $\{e \mid \mathcal{B} \Vdash e\}$, called the **behavioral equivalence** of $\mathcal{B}$. A set of equations $\mathcal{G}$ is **behaviorally closed** iff $\mathcal{B} \vdash \mathit{visible}(\mathcal{G})$ and $\Delta[\mathcal{G} - \mathcal{B}^\bullet] \subseteq \mathcal{G}$.*

It can be shown that $\Vdash$ is a Δ-contextual entailment system extending $\vdash$ (see [17]); in other words, $\Vdash$ satisfies the axioms in Definition 1 and if $\mathcal{B} \vdash e$ then $\mathcal{B} \Vdash e$. Our approach in this paper is proof-theoretical rather than model-theoretical, so our notion of behavioral equivalence is defined proof-theoretically rather than using models like in [8, 3, 1]; also, our $\equiv$ can also contain conditional equations (of visible conditions). A behaviorally closed set $\mathcal{G}$ of equations is one whose visible equations are provable from $\mathcal{B}$ using the base entailment system and whose equations not provable from $\mathcal{B}$ using the base system remain in $\mathcal{G}$ when derived. Hence, the only way an equation can "escape" the derivation process in a behaviorally closed set is to be proved using the base entailment system $\vdash$. The following result plays a central role in hidden logic:

Theorem 1. ***(Coinduction)**[17] For any behavioral specification, the behavioral equivalence $\equiv$ is the largest behaviorally closed set of equations.*

Theorem 1 is the foundation for the coinduction proving technique. An entailment-based coinductive proving technique is presented in [17]. The main idea is to find a set of equations G such that $\Delta(G) \subseteq \overline{G \cup \mathcal{B}^\bullet}$, where $\overline{\mathcal{E}}$ denotes the closure of $\mathcal{E}$ under substitution, symmetry, and transitivity.

Example 2. A proof by coinduction of the property $S \times zero = zero$ is given by $G = \{S \times zero = zero, S \times zero + S' = S'\}$ and the following equations which shows that $\Delta(G) \subseteq \mathcal{G} = \overline{G \cup \mathtt{STREAM}^\bullet}$:

$hd(S \times zero) = hd(zero)$	(in $\mathtt{STREAM}^\bullet$)
$hd(S \times zero + S') = hd(S')$	(in $\mathtt{STREAM}^\bullet$)
$tl(S \times zero) = tl(S) \times zero + [hd(S)] \times zero$	(in $\mathtt{STREAM}^\bullet$)
$tl(S) \times zero + [hd(S)] \times zero = [hd(S)] \times zero$	(substit.)
$[hd(S)] \times zero = zero$	(substitution)
$tl(S) \times zero + [hd(S)] \times zero = zero$	(transitivity)
$tl(S \times zero) = zero$	(transitivity)
$tl(S \times zero + S') = tl(S) \times zero + [hd(S)] \times zero + tl(S')$	(in $\mathtt{STREAM}^\bullet$)
$tl(S) \times zero + [hd(S)] \times zero + tl(S') = [hd(S)] \times zero + tl(S')$	(substitution)
$[hd(S)] \times zero + tl(S') = tl(S')$	(substitution)
$tl(S \times zero + S') = tl(S')$	(transitivity)

As seen in the examples above, coinductive proofs of behavioral equivalence require human intervention, to provide an appropriate behaviorally closed set of equations $\mathcal{G}$, which can be thought of as "approximations" of $\equiv$. It is worth noting that it is virtually impossible to compute $\equiv$ precisely, because, as shown in [15], the problem of behavioral satisfaction is a Π_2^0 hard problem even for the particular specification of streams discussed in this paper. Therefore, approximating $\equiv$ is perhaps the best we can do. Circular coinduction [16, 18, 17] automates coinductive proving by dynamically (i.e., during the proving process) inferring a suitable behaviorally closed set $\mathcal{G}$ including the property(ies) to prove.

4 Circular Coinduction

A key notion in our formalization and even implementation of circular coinduction is that of a "frozen" equation. The motivation underlying frozen equations is that they structurally inhibit their use underneath proper contexts; because of that, they will allow us to capture the above-mentioned informal notion of "circular behavior" elegantly, rigorously, and generally (modulo a restricted form of equational reasoning). Formally, let $(\mathcal{B}, \Delta)$ be a behavioral specification and let us extend its signature Σ with a new sort *Frozen* and a new operation $\boxed{\text{-}} : s \to \mathit{Frozen}$ for each sort s. If t is a term, then we call $\boxed{t}$ the *frozen (form of)* t. Note that freezing only acts on the original sorts in Σ, so double freezing, e.g., $\boxed{\boxed{t}}$, is not allowed. If e is an equation $(\forall X)\, t = t'$ `if` c, then we let $\boxed{e}$ be the *frozen equation* $(\forall X)\, \boxed{t} = \boxed{t'}$ `if` c; note that the condition c stays unfrozen, but recall that we only assume visible conditions. By analogy, we may call the equations over the original signature Σ *unfrozen equations*. If e is an (unfrozen) visible equation then $\boxed{e}$ is called a *frozen visible equation*; similarly when e is hidden. If C is a context for e, then we take the freedom to write $C[\boxed{e}]$ as a shortcut for $\boxed{C[e]}$. It is important to note here is that if $E \cup \mathcal{F} \vdash \mathcal{G}$ for some unfrozen equation set E and frozen equation sets $\mathcal{F}$ and $\mathcal{G}$, it is not necessarily the case that $E \cup \mathcal{F} \vdash C[\mathcal{G}]$ for a context C. Freezing therefore inhibits the free application of the congruence deduction rule of equational reasoning.

Recall that, for generality, we work with an arbitrary entailment system in this paper, which may or may not necessarily be the entailment relation of equational deduction. We next add two more axioms:

Definition 3. *A* ***Δ-contextual entailment system with freezing*** *is a Δ-contextual entailment system extended as above such that:*

(A1) $E \cup \mathcal{F} \vdash \boxed{e}$ *iff* $E \vdash e$;

(A2) $E \cup \mathcal{F} \vdash \mathcal{G}$ *implies* $E \cup \delta[\mathcal{F}] \vdash \delta[\mathcal{G}]$ *for each* $\delta \in \Delta$, *equivalent to saying that for any Δ-context C,* $E \cup \mathcal{F} \vdash \mathcal{G}$ *implies* $E \cup C[\mathcal{F}] \vdash C[\mathcal{G}]$.

Above, E ranges over unfrozen equations, e over visible unfrozen equations, and $\mathcal{F}$ and $\mathcal{G}$ over frozen hidden equations.

Our working entailment system $\vdash$ is now defined over both unfrozen and frozen equations. It is easy to check these additional axioms for concrete entailment relations and to see that they are conservative, that is, one cannot infer any new entailment of unfrozen equations that were not possible before [17].

Figure 3 defines circular coinduction as a proof system for deriving pairs of the form $\mathcal{H} \Vdash^{\circlearrowleft} \mathcal{G}$, where $\mathcal{H}$, the *hypotheses*, can contain both frozen and unfrozen equations, and where $\mathcal{G}$, the *goals*, contains only frozen equations. Initially, $\mathcal{H}$ is the original behavioral specification $\mathcal{B}$ and $\mathcal{G}$ is the frozen version $\boxed{G}$ of the original goals G to prove. Circular coinduction iteratively attempts to complete the original set of goals G to a behaviorally closed set of equations; freezing is necessary to inhibit the application of the congruence rule of equational deduction because, otherwise, the hypothesis of [Derive] would hold superfluously whenever $\mathcal{H} \Vdash^{\circlearrowleft} \mathcal{G}$ derivable, so the proof system would be unsound.

$$\frac{}{\mathcal{H} \Vdash^{\circlearrowleft} \emptyset} \quad \text{[Done]}$$

$$\frac{\mathcal{H} \Vdash^{\circlearrowleft} \mathcal{G}, \quad \mathcal{H} \vdash \boxed{e}}{\mathcal{H} \Vdash^{\circlearrowleft} \mathcal{G} \cup \{\boxed{e}\}}, \text{ if } e \text{ visible or hidden} \quad \text{[Reduce]}$$

$$\frac{\mathcal{H} \cup \{\boxed{e}\} \Vdash^{\circlearrowleft} \mathcal{G} \cup \boxed{\Delta[e]}}{\mathcal{H} \Vdash^{\circlearrowleft} \mathcal{G} \cup \{\boxed{e}\}}, \text{ if } e \text{ hidden} \quad \text{[Derive]}$$

Fig. 3. Circular coinduction as a proof system: If $\mathcal{B} \Vdash^{\circlearrowleft} \boxed{G}$ is derivable then $\mathcal{B} \Vdash G$

Theorem 2. ***(soundness of circular coinduction)****[17] If $\mathcal{B}$ is a behavioral specification and G is a set of equations such that $\mathcal{B} \Vdash^{\circlearrowleft} \boxed{G}$ is derivable using the proof system in Figure 3, then $\mathcal{B} \Vdash G$.*

The interested reader can find the proof of Theorem 2 in [17], or consult the next section where a more general result is proved.

Example 3. The table below summarizes the derivation steps followed to prove $zip(odd(S), even(S)) = S$. Each row comprises a derived pair $\mathcal{H}_i \Vdash^{\circlearrowleft} \mathcal{G}_i$, where $\mathcal{H}_i = \{\boxed{e} \mid e \in \mathtt{STREAM} \cup \mathcal{F}_i^{\circ}\}$ and $\mathcal{G}_i = \{\boxed{e} \mid e \in \mathcal{G}_i^{\circ}\}$.

	Rule	$\mathcal{G}_i^{\circ}$	$\mathcal{F}_i^{\circ}$
0		$\mathtt{zip}(\mathtt{odd}(\mathtt{S}), \mathtt{even}(\mathtt{S})) = \mathtt{S}$	
1	[Derive]	$\mathtt{hd}(\mathtt{zip}(\mathtt{odd}(\mathtt{S}), \mathtt{even}(\mathtt{S}))) = \mathtt{hd}(\mathtt{S})$ $\mathtt{tl}(\mathtt{zip}(\mathtt{odd}(\mathtt{S}), \mathtt{even}(\mathtt{S}))) = \mathtt{tl}(\mathtt{S})$	$\mathtt{zip}(\mathtt{odd}(\mathtt{S}), \mathtt{even}(\mathtt{S})) = \mathtt{S}$
2	[Reduce]	$\mathtt{tl}(\mathtt{zip}(\mathtt{odd}(\mathtt{S}), \mathtt{even}(\mathtt{S}))) = \mathtt{tl}(\mathtt{S})$	$\mathtt{zip}(\mathtt{odd}(\mathtt{S}), \mathtt{even}(\mathtt{S})) = \mathtt{S}$
3	[Reduce]		$\mathtt{zip}(\mathtt{odd}(\mathtt{S}), \mathtt{even}(\mathtt{S})) = \mathtt{S}$

The reader is invited to see [17] for more examples of proofs by circular coinduction.

5 Special Hypotheses and Special Contexts

In this section we show that the circular coinduction can be extended by adding "on the fly" new hypothesis which are sound provided the derivation process successfully terminates. If that is the case, then the result is a better approximation of the behavioral equivalence.

Definition 4. ***(special hypotheses)*** *Let $(\mathcal{B}, \Delta)$ be a behavioral specification and F a set of hidden equations. Hidden equation e is a **special hypothesis** for F iff $\mathcal{B} \vdash C[e]$ whenever $\mathcal{B} \vdash C^{\leq}[F]$, where $C^{\leq}$ is the set of Δ-experiments D*

with $|D| \leq |C|$. The set of special hypotheses for F is written $F^{\leq}$ and is called the ***special-hypothesis closure*** *of F.*

Therefore, a special hypothesis for a set of hidden equations F is a hidden equation e which holds under experiment C whenever the equations in F hold under all the experiments smaller than or equal to C (in depth, not in size). The intuition for special hypotheses, formalized in Theorem 3, is that they can be soundly used in reasoning when checking the closure of F under derivatives.

It is easy to check that $\cdot^{\leq}$ is a closure operator on sets of equations, that is, it is extensive ($F \subseteq F^{\leq}$), increasing ($F_1 \subseteq F_2$ implies $F_1^{\leq} \subseteq F_2^{\leq}$), and idempotent ($(F^{\leq})^{\leq} = F^{\leq}$). Also, $(\equiv\restriction_H)^{\leq} = \equiv\restriction_H$ and $\equiv\restriction_H \subseteq F^{\leq}$ for any hidden equation set F, where recall from Section 3 that H is the set of hidden sorts, so $\equiv\restriction_H$ is the set of hidden equations e such that $\mathcal{B} \Vdash e$; in particular, if $F \subseteq \equiv\restriction_H$ then $F^{\leq} = \equiv\restriction_H$. We next discuss some examples.

Example 4. If F consists of an equality of two streams $\boxed{a} = \boxed{b}$, then the following equations are in $F^{\leq}$: $\boxed{S + a} = \boxed{S + b}$, $\boxed{a + S} = \boxed{b + S}$, $\boxed{S \times a} = \boxed{S \times b}$, $\boxed{a \times S} = \boxed{b \times S}$, $\boxed{not(a)} = \boxed{not(b)}$, $\boxed{zip(S, a)} = \boxed{zip(S, b)}$, $\boxed{zip(a, S)} = \boxed{zip(b, S)}$, $\boxed{f(a)} = \boxed{f(b)}$, where S is a variable over streams. The equations $\boxed{odd(a)} = \boxed{odd(b)}$ and $\boxed{even(a)} = \boxed{even(b)}$ are not in $F^{\leq}$. For instance, we cannot deduce $hd(tl(odd(a))) = hd(tl(odd(b)))$ knowing only $hd(a) = hd(b)$ and $hd(tl(a)) = hd(tl(b))$ since $hd(tl(odd(a))) = hd(tl(tl(a)))$ and $hd(tl(odd(b))) = hd(tl(tl(b)))$.

The coinductive circularity principle (Theorem 2 in [17]) states that if F is a set of hidden equations such that $\mathcal{B} \cup \boxed{F} \vdash \boxed{\Delta(F)}$ then $\mathcal{B} \Vdash F$. This coinductive principle is the fundamental result underlying the soundness of circular coinduction. We next extend it by allowing $F^{\leq}$ instead of F as hypotheses:

Theorem 3. ***(extended coinductive circularity principle)*** *If F is a set of hidden equations such that $\mathcal{B} \cup \boxed{F^{\leq}} \vdash \boxed{\Delta[F]}$, then $\mathcal{B} \Vdash F^{\leq}$ (in fact, $F^{\leq} = \equiv\restriction_H$).*

Proof. We prove property $(\forall \Delta\text{-experiment } C)\ \mathcal{B} \vdash C[F^{\leq}]$ by well-founded induction on experiments C. For the base case, note that hypothesis $\mathcal{B} \cup \boxed{F^{\leq}} \vdash \boxed{\Delta[F]}$ and Definition 3 *(A1)* imply $\mathcal{B} \vdash D[F]$ for any visible $D \in \Delta$, so Definition 4 implies that $\mathcal{B} \vdash C[F^{\leq}]$ for any visible $C \in \Delta$. For the inductive step, suppose that $C = C'[\delta]$ for some Δ-experiment C' and $\delta \in \Delta$, and suppose that $\mathcal{B} \vdash D'[F^{\leq}]$ for all Δ-experiments D' with $|D'| < |C|$; further, Definition 3 *(A2)* implies $\mathcal{B} \cup \boxed{D'[F^{\leq}]} \vdash \boxed{D'[\Delta[F]]}$, so $\mathcal{B} \vdash \boxed{D'[\Delta[F]]}$. Hence, we showed that $\mathcal{B} \vdash D[F]$ for all Δ-experiments D with $|D| \leq |C|$, so $\mathcal{B} \vdash C[F^{\leq}]$ by Definition 4.

In practice, one needs not add all the special hypotheses in $F^{\leq}$, but only those that help to derive $\boxed{\Delta[F]}$. Indeed, if SH is a subset of special hypotheses such that one can derive $\mathcal{B} \cup \boxed{SH} \cup \boxed{F} \vdash \boxed{\Delta[F]}$, then Theorem 3 implies $\mathcal{B} \Vdash SH \cup F$. In particular, if $SH = \emptyset$ then we obtain the coinductive circularity principle (Theorem 2 in [17]) as a special case. It is worthwhile noticing that no proof obligation is generated for the added special hypotheses; however, checking the

condition in Definition 4 may not be trivial. In what follows we give a more effective approach to define useful special hypotheses, based on *special contexts*.

A first variant of special context was first introduced in [7]: a context $\gamma[*{:}h]$ was called "special" in [7] iff for any experiment C for γ there is some experiment D with $|D| \leq |C|$ and $\mathcal{B} \vdash C[\gamma[*{:}h]] = D[*{:}h]$. The intuition for special contexts is therefore that whenever they appear at the bottom of an experiment they can be eliminated yielding a strictly smaller experiment. This way, again intuitively, their application on top of goals to prove does not change the behavioral validity status of those goals, so with our terminology above, their application on goals to prove can be added as special hypotheses. In what follows we formalize and prove this claim. Before we do so, motivated by practical needs, we first extend the definition of a special context by allowing the right hand side of the equation above, $D[*{:}h]$, to be replaced by any Σ-term whose occurrences of $*{:}h$ appear only in subterms of the form $D[*{:}h]$, where D is an experiment with $|D| \leq |C|$.

Definition 5. ***(special contexts)*** *Context $\gamma[*{:}h]$ is* ***special*** *iff for any experiment C for γ there is some term t such that $\mathcal{B} \vdash C[\gamma[*{:}h]] = t$ and each occurrence of $*{:}h$ in t appears only in a subterm in $C^{\leq}$ (see Definition 4).*

Example 5. For the streams specified in Section 1, the following are special contexts: $*{:}\mathit{Stream}{+}S{:}\mathit{Stream}$, $S{:}\mathit{Stream}{+}*{:}\mathit{Stream}$, $*{:}\mathit{Stream}{\times}S{:}\mathit{Stream}$, $S{:}\mathit{Stream}{\times}*{:}\mathit{Stream}$, $\mathit{not}(*{:}\mathit{Stream})$, $\mathit{zip}(*{:}\mathit{Stream}, S{:}\mathit{Stream})$, $\mathit{zip}(S{:}\mathit{Stream}, *{:}\mathit{Stream})$ and $f(*{:}\mathit{Stream})$. Moreover, any combination of these contexts (e.g., $(*{:}\mathit{Stream} \times S{:}\mathit{Stream}){+}S'{:}\mathit{Stream}$) is special, as well. In contrast, the contexts $\mathit{odd}(*{:}\mathit{Stream})$ and $\mathit{even}(*{:}\mathit{Stream})$ are not special (e.g., $\texttt{STREAM} \vdash \mathit{hd}(\mathit{tl}(\mathit{odd}(*{:}\mathit{Stream}))) = \mathit{hd}(\mathit{tl}(\mathit{tl}(*{:}\mathit{Stream})))$ and $|\mathit{hd}(\mathit{tl}(\mathit{tl}(*{:}\mathit{Stream})))| > |\mathit{hd}(\mathit{tl}(*{:}))|$.

From now on, we assume that $\vdash$ is also closed under substitution, that is, if $E \vdash e$ and θ is a substitution, then $E \vdash \theta(e)$. This requirement is reasonable and satisfied by any entailment system that we aware of.

Theorem 4. *If F is a hidden equation set and γ a special context, $\gamma[F] \subseteq F^{\leq}$.*

Proof. We have to show $\mathcal{B} \vdash C[\gamma[F]]$ for any Δ-experiment C with $\mathcal{B} \vdash C^{\leq}[F]$. Fix such a C. Since γ is special, by Definition 5 there is some term t such that $\mathcal{B} \vdash C[\gamma[*{:}h]] = t$ and each occurrence of $*{:}h$ in t appears only in a subterm D that is an experiment in $C^{\leq}$. Since $\mathcal{B} \vdash C^{\leq}[F]$, we deduce that $\mathcal{B} \vdash D[F]$ for any such experiment D, so the congruence of $\vdash$ implies $\mathcal{B} \vdash t[F]$. Finally, the closure of $\vdash$ under substitution applied on equation $C[\gamma[*{:}h]] = t$ yields $\mathcal{B} \vdash C[\gamma[F]]$.

Therefore, special contexts automatically yield a distinguished set of special hypotheses for any set of hidden equations. We empirically found that these distinguished special hypotheses are sufficient to prove all the behavioral properties that we and other colleagues considered so far, so we next extend our circular coinductive proof system with special contexts and then prove its soundness.

Let us replace the rule [Derive] in Figure 3 with the more general one below:

$$\frac{\mathcal{H} \cup \boxed{\Gamma[e]} \cup \{\boxed{e}\} \Vdash^{\circlearrowleft} \mathcal{G} \cup \boxed{\Delta[e]}}{\mathcal{H} \Vdash^{\circlearrowleft} \mathcal{G} \cup \{\boxed{e}\}}, \quad \begin{array}{l}\text{when } e \text{ is hidden and}\\ \Gamma \text{ set of special contexts}\end{array} \qquad [\mathsf{Derive}^{\mathsf{scx}}]$$

Theorem 5. ***(soundness of circular coinduction with special contexts)*** *If $\mathcal{B}$ is a behavioral specification and G is a set of equations such that $\mathcal{B} \Vdash^{\circlearrowleft} \boxed{G}$ is derivable using the extended proof system with rule* [Derive$^{\mathsf{scx}}$]*, then $\mathcal{B} \Vdash G$.*

Proof. Any derivation of $\mathcal{H} \Vdash^{\circlearrowleft} \mathcal{G}$ using the extended proof system yields a sequence of pairs $\mathcal{H}_0 \Vdash^{\circlearrowleft} \mathcal{G}_0, \mathcal{H}_1 \Vdash^{\circlearrowleft} \mathcal{G}_2, \ldots \mathcal{H}_n \Vdash^{\circlearrowleft} \mathcal{G}_n$, where $\mathcal{H}_0 = \mathcal{B}$, $\mathcal{G}_0 = \boxed{G}$, $\mathcal{G}_n = \emptyset$, and for every $0 \leq i < n$, there is some $\boxed{e} \in \mathcal{G}_i$ such that one of the following holds: $\mathcal{H}_i \vdash \boxed{e}$ and $\mathcal{G}_{i+1} = \mathcal{G}_i - \{\boxed{e}\}$ and $\mathcal{H}_{i+1} = \mathcal{H}_i$; or e is hidden and $\mathcal{G}_{i+1} = (\mathcal{G}_i - \{\boxed{e}\}) \cup \boxed{\Delta[e]}$ and $\mathcal{H}_{i+1} = \mathcal{H}_i \cup \{\boxed{e}\} \cup \boxed{\Gamma_i[e]}$ for some set of special contexts Γ_i. Let $\mathcal{G} = \bigcup_{i=0}^{n} \mathcal{G}_i$, let $\mathcal{G}^\circ = \{e \mid \boxed{e} \in \mathcal{G}\}$, let $F = \mathit{hidden}(\mathcal{G}^\circ)$ and let Γ be the union of all the special contexts used in the derivation.

For each $0 \leq i < n$, $\mathcal{H}_i = \mathcal{B} \cup \mathcal{F}_i \cup \mathcal{K}_i$ for some set of frozen hidden equations $\mathcal{F}_i$ and $\mathcal{K}_i$ with $\mathcal{F}_i \cup \Delta[\mathcal{F}_i] \subseteq \mathcal{G}$ and $\mathcal{K}_i \subseteq \boxed{\Gamma[F]}$. If $\mathcal{H}_i \vdash \boxed{e}$ for some $\boxed{e} \in \mathcal{G}$ (applying [Reduce]), then either $\mathcal{B} \vdash e$ by *(A1)* in Definition 3 when e is visible, or $\mathcal{B} \cup \Delta[\mathcal{K}_i] \cup \Delta[\mathcal{F}_i] \vdash \boxed{\Delta[e]}$ by *(A2)* in Definition 3 when $e \in F$.

If $e \in \mathcal{G}^\circ$ visible, then there must be some $0 \leq i < n$ such that $\mathcal{H}_i \vdash \boxed{e}$, so $\mathcal{B} \vdash e$. Its is easy to see that if $\mathcal{B} \cup \boxed{\Gamma[F]} \cup \boxed{F} \vdash \mathcal{G}_{i+1}$, then $\mathcal{B} \cup \boxed{\Gamma[F]} \cup \boxed{F} \vdash \mathcal{G}_i$: indeed, for the only equation $\boxed{e}$ that is in $\mathcal{G}_i$ and not in $\mathcal{G}_{i+1}$ we have either $\mathcal{H}_i \vdash \boxed{e}$ or $\boxed{e} \in \mathcal{F}_i$. Since $\mathcal{G}_n = \emptyset$, we deduce $\mathcal{B} \cup \boxed{\Gamma[F]} \cup \boxed{F} \vdash \mathcal{G}$. For each $\boxed{e} \in \mathcal{G}_i$ with $i < n$, either $\mathcal{B} \cup \boxed{\Gamma[F]} \cup \boxed{F} \vdash \boxed{\Delta[e]}$ (if $\mathcal{H}_i \vdash \boxed{e}$) or $\boxed{\Delta[e]} \subseteq \mathcal{G}_{i+1}$. In either case, $\mathcal{B} \cup \boxed{\Gamma[F]} \cup \boxed{F} \vdash \boxed{\Delta[F]}$. The equations in $\Gamma[F]$ are special by Theorem 4. We apply now Theorem 3 and we obtain $\mathcal{B} \Vdash F$. Since F contains all the hidden equations of $\mathcal{G}^\circ$ and since we already proved that $\mathcal{B} \vdash e$, i.e., $\mathcal{B} \Vdash e$, for all $e \in \mathcal{G}^\circ$ visible, we conclude that $\mathcal{B} \Vdash \mathcal{G}^\circ$. Since $G \subseteq \mathcal{G}^\circ$, it follows that $\mathcal{B} \Vdash G$.

Example 6. We next discuss the automated proving of $S \times \mathit{zero} = \mathit{zero}$ using the extended proof system. The first use of [Derive]$^{\mathsf{scx}}$ is at step 1: e is $S \times \mathit{zero} = \mathit{zero}$ and Γ is $\{(*{:}\mathit{Stream} + S')[e], (S' + *{:}\mathit{Stream})[e]\}$. [Derive]$^{\mathsf{scx}}$ is used at step 3 with $\Gamma = \emptyset$. The new goal in step 2 is the effect of the use of the special hypotheses:

$tl(S \times \mathit{zero}) = tl(\mathit{zero})$
$tl(S) \times \mathit{zero} + [hd(S)] \times \mathit{zero} = \mathit{zero}$ (using stream equations)
$\mathit{zero} + [hd(S)] \times \mathit{zero} = \mathit{zero}$ (using the first special hypothesis)
$\mathit{zero} + \mathit{zero} = \mathit{zero}$ (using the second special hypothesis)

	Rule	$\mathcal{G}_i^\circ$	$\mathcal{F}_i^\circ$
0		$S \times \mathit{zero} = \mathit{zero}$	
1	[Derive]$^{\mathsf{scx}}$	$hd(S \times \mathit{zero}) = hd(\mathit{zero})$ $tl(S \times \mathit{zero}) = tl(\mathit{zero})$	$S \times \mathit{zero} = \mathit{zero}$, $S \times \mathit{zero} + S' = \mathit{zero} + S'$ $S' + S \times \mathit{zero} = S' + \mathit{zero}$
2	[Reduce]	$\mathit{zero} + \mathit{zero} = \mathit{zero}$	$S \times \mathit{zero} = \mathit{zero}$, $S \times \mathit{zero} + S' = \mathit{zero} + S'$ $S' + S \times \mathit{zero} = S' + \mathit{zero}$
3	[Derive]$^{\mathsf{scx}}$	$hd(\mathit{zero} + \mathit{zero}) = hd(\mathit{zero})$ $tl(\mathit{zero} + \mathit{zero}) = tl(\mathit{zero})$	$S \times \mathit{zero} = \mathit{zero}$, $S \times \mathit{zero} + S' = \mathit{zero} + S'$ $S' + S \times \mathit{zero} = S' + \mathit{zero}$
4	[Reduce]	$tl(\mathit{zero} + \mathit{zero}) = tl(\mathit{zero})$	$S \times \mathit{zero} = \mathit{zero}$, $S \times \mathit{zero} + S' = \mathit{zero} + S'$ $S' + S \times \mathit{zero} = S' + \mathit{zero}$
5	[Reduce]		$S \times \mathit{zero} = \mathit{zero}$, $S \times \mathit{zero} + S' = \mathit{zero} + S'$ $S' + S \times \mathit{zero} = S' + \mathit{zero}$

[EqRed]	:	$\langle \mathcal{B},\ \mathcal{G} \cup \{\boxed{e}\} \rangle \Rightarrow \langle \mathcal{B}, \mathcal{G} \rangle$	if $\mathcal{E} \vdash_{\mathrm{nf}} \boxed{e}$
[Fail]	:	$\langle \mathcal{B},\ \mathcal{G} \cup \{\boxed{e}\} \rangle \Rightarrow \mathit{fail}$	if $\mathcal{B} \not\vdash_{\mathrm{nf}} \boxed{e}$ and e is visible
[CCStep]	:	$\langle \mathcal{B},\ \mathcal{G} \cup \{\boxed{e}\} \rangle \Rightarrow \langle \mathcal{B} \cup \{\boxed{e}\},\ \mathcal{G} \cup \Delta[\boxed{e}] \rangle$	if $\mathcal{B} \not\vdash_{\mathrm{nf}} \boxed{e}$ and e is hidden.

Fig. 4. CIRC's reduction rules implementing the circular coinduction

6 Implementation in CIRC

CIRC implements the circular coinduction proof system by the reduction rules given in Fig. 4. The entailment relation used in CIRC is $\mathcal{E} \vdash_{\mathrm{nf}} (\forall X) t = t'$ `if` $\wedge_i u_i = v_i$ if and only if $\mathrm{nf}(t) = \mathrm{nf}(t')$, where $\mathrm{nf}(t)$ is computed as follows:

– the variables of the equations are turned into fresh constants;
– the condition equalities are added as equations to the specification;
– the equations in the specification are oriented and used as rewrite rules.

[EqRed] implements the proof rule [Reduce]: if a goal is a $\vdash_{\mathrm{nf}}$-consequence of the current specification, then it is removed from the set of goals. [Fail] has no correspondent in the proof system. A failure does not necessarily means that the answer is false. The failure relation $\mathcal{E} \not\vdash_{\mathrm{nf}} \boxed{e}$ says that the corresponding normal forms are different. Since we do not impose any confluence conditions on the specification, it is possible as the normal forms to be different even if the equation is a $\vdash$-consequence of the current specification. So, a failing ending of the algorithm needs (human) analysis in order to know the source of the failure. However, since CIRC includes an implementation of the circular induction (it will be presented in a different paper), a technique similar to that in [4] can be used to check if the failed visible equation or the disequality of the two normal forms is an inductive theorem. [CCStep] implements the proof rule [Derive]. If the current goal is hidden and it is not a $\vdash_{\mathrm{nf}}$-consequence, then it is added to the current specification and its derivatives are added as new goals.

The above discussion is summarized by the following result:

Theorem 6. ***(soundness of CIRC)*** *Let $(\mathcal{B}, \Delta)$ be a behavioral specification and let G be a set of frozen equations. If $(\mathcal{B}, G) \Rightarrow^* (\mathcal{B}', \emptyset)$ applying the reduction rules of the circular coinduction system in Figure 4, then $\mathcal{B} \Vdash_{\mathrm{nf}} G$.*

In order to make the rule [Derive$^{\mathsf{scx}}$] effective, we have to know which contexts are special. A less efficient way is to manually prove that some contexts are special and then include them in the behavioral specification. Though CIRC include such a facility, it is more challenging and elegant to automatically detect the special contexts. If the composition $\gamma_1[\gamma_2]$ of two special contexts γ_1 and γ_2 is a special context as well, then it is enough to find a maximal subset Γ of contexts γ of minimal depth. An example of such Γ is given in Example 5. Knowing Γ, there is a very simple and efficient way to implement [Derive$^{\mathsf{scx}}$]: for each special context $\gamma[*{:}h]$ in Γ, the following equation is added to the specification:

$$\boxed{\gamma[x]} = \boxed{\gamma[y]} \quad \texttt{if} \quad \boxed{y} := \boxed{x}$$

where the execution of the matching equation $\boxed{y} := \boxed{x}$ instantiate y by matching $\boxed{y}$ against the normal form of $\boxed{x}$. However, we have to notice that this nice implementation uses the matching mechanisms specific to the Maude system. Let us explain how this mechanism works for $\gamma = *{:}Stream + S'$ and $\boxed{S \times zero} = \boxed{zero}$. If $x \mapsto S \times zero$, then $\boxed{y} := \boxed{x}$ returns $y \mapsto zero$. Replacing γ, x and y $\boxed{\gamma[x]} = \boxed{\gamma[y]}$ we obtain the desired result: $S \times zero + S' = zero + S'$.

The algorithm used by CIRC for computing a set Γ of special contexts of minimal depth is quite complex. Here we briefly describe the intuition behind this algorithm. The condition in the definition of the special contexts is soundly replaced with a computable predicate $Comp(C, t)$, defined for all Δ-contexts C and which satisfies the following two conditions: 1) from $Comp(C_1, t)$ and $Comp(C_2, t)$ we can deduce $Comp(C_1[C_2], t)$, and 2) from $Comp(C, t_1)$ and $Comp(C, t_2)$ we can deduce $Comp(C, t_1[t_2])$, provided the sort-compatibility conditions hold. The algorithm has two steps: 1) compute the set Υ of all contexts of depth one which can be generated from the non-derivative hidden operations which have at least one hidden argument; 2) compute a maximal subset $\Gamma \subseteq \Upsilon$ such that $(\forall \delta \in \Delta)(\forall \gamma \in \Gamma)\, Comp(\delta, \gamma)$. The algorithm can be specialized for the cases when the contexts are defined with other contexts than derivatives, as it is the case of the function f defined over streams. The special contexts in Example 5 were computed with this algorithm.

Playing with the special contexts, we learned a very interesting lesson: the fact that CIRC system does not terminate is not necessarily a bad news. We have seen that the basic circular coinduction does not terminate for $S \times zero = zero$. However, we proved that this property holds using the special contexts. Hence we may deduce that the infinite execution is in fact a proof. In the following we identify a subclass of infinite executions which can be used as proofs. We first give a behavioral version for the coinductive circularity principle.

Theorem 7. *If $(\mathcal{B}, \Delta)$ is a behavioral specification and F is a set of hidden equations with $\mathcal{B} \cup \boxed{F} \Vdash \boxed{\Delta[F]}$ then $\mathcal{B} \Vdash F$.*

The proof of Theorem 7 is similar to that given in [17] for Theorem 2. Comparing with Theorem 2 in [17] (coinductive circularity principle), Theorem 7 uses the behavioral entailment relation to check that F is a fixed point. Since the initial entailment relation is sound for the behavioral one, it follows that the coinductive circularity principle is a particular case of Theorem 7. However, the latter one is not very useful in practice because it express the behavioral entailment by means of ... the behavioral entailment. Even if this is a very nice circular feature, recall that the efficiency of the circular coinduction system comes from the fact it proves behaviorally satisfied properties using the basic entailment relation.

Definition 6. *An **infinite fair execution** of* CIRC *system is an infinite execution $(\mathcal{B}_0, \mathcal{G}_0) \Rightarrow (\mathcal{B}_1, \mathcal{G}_1) \Rightarrow \cdots$ such that for all $\boxed{e} \in \mathcal{G}_i$, $i \geq 0$, there is $j \geq i$ such that $\boxed{e}$ is processed at step j.*

Since the execution never fails, the equation e is processed at step j applying either [EqRed] or [CCStep].

Theorem 8. *Let $(\mathcal{B}, \Delta)$ a behavioral specification and G is a set of hidden equations. If $(\mathcal{B}, G) = (\mathcal{B}_0, \mathcal{G}_0) \Rightarrow (\mathcal{B}_1, \mathcal{G}_1) \Rightarrow \cdots$ is an infinite fair execution of* CIRC *system, then $\mathcal{B} \Vdash_{\mathrm{nf}} G$.*

Proof. Let $\mathcal{G} = \bigcup_{i \geq 0} \mathcal{G}_i$, let $\mathcal{G}^\circ = \{e \mid \boxed{e} \in \mathcal{G}\}$, and let $F = \mathit{hidden}(\mathcal{G}^\circ)$. Note that for each $i \geq 0$, $\mathcal{B}_i = \mathcal{B} \cup \mathcal{F}_i$ for some set of frozen hidden equations $\mathcal{F}_i \subseteq \boxed{F}$. Let $\mathcal{F} = \bigcup_{i \geq 0} \mathcal{F}_i$. If $\boxed{e} \in \mathcal{F}$, then $\Delta[e] \subseteq \mathcal{G}^\circ$ and, because the execution is fair, each equation in $\boxed{\Delta[e]}$ is processed at some step i (normally, i is depending on the equation, but we abusively omit to explicitly write this dependence). We first show by induction on experiments C that $\mathcal{B} \cup \boxed{F} \vdash_{\mathrm{nf}} \boxed{C[e]}$, for all $\boxed{e} \in \mathcal{F}$. If $C \in \Delta$, then $(\mathcal{B} \cup \mathcal{F}) \vdash_{\mathrm{nf}} \boxed{C[e]}$ because the execution does not fail; hence $(\mathcal{B} \cup \boxed{F}) \vdash_{\mathrm{nf}} \boxed{C[e]}$ by the monotonicity of $\vdash_{\mathrm{nf}}$. We assume that $C = C'[\delta]$ and the property holds for $C'[e']$, for all $e' \in \mathcal{F}$. If $(\mathcal{B} \cup \mathcal{F}_i) \vdash_{\mathrm{nf}} \boxed{\delta[e]}$ ([EqRed] is applied at the step i), then $(\mathcal{B} \cup \boxed{F}) \vdash_{\mathrm{nf}} \boxed{\delta[e]}$ and hence $(\mathcal{B} \cup \boxed{F}) \Vdash_{\mathrm{nf}} \boxed{\delta[e]}$; in particular $(\mathcal{B} \cup \boxed{F}) \vdash_{\mathrm{nf}} \boxed{C'[\delta[e]]}$. If [CCStep] is applied at the step i, then $\boxed{\delta[e]} \in \mathcal{F}_{i+1}$ and we have $(\mathcal{B} \cup \boxed{F}) \vdash_{\mathrm{nf}} \boxed{C'[\delta[e]]}$ by the induction hypothesis.
We show now that $(\mathcal{B} \cup \boxed{F}) \Vdash_{\mathrm{nf}} \boxed{F}$. If $e \in F$, then $\boxed{e} \in \mathcal{G}_i$ for some $i \geq 0$. Since the execution is fair, there is $j \geq i$ such that $\boxed{e}$ is processed at the step j. If [EqRed] is applied at the step j, then $(\mathcal{B} \cup \mathcal{F}_j) \vdash_{\mathrm{nf}} \boxed{e}$, which implies $(\mathcal{B} \cup \mathcal{F}_j) \Vdash_{\mathrm{nf}} \boxed{e}$ and hence $(\mathcal{B} \cup \boxed{F}) \Vdash_{\mathrm{nf}} \boxed{e}$ by the monotonicity of $\Vdash_{\mathrm{nf}}$. If [CCStep] is applied at the step j, then $\boxed{e} \in \mathcal{F}$ and $(\mathcal{B} \cup \boxed{F}) \Vdash_{\mathrm{nf}} \boxed{e}$ by the property proved by induction above. Since e is an arbitrary equation in F, it follows $(\mathcal{B} \cup \boxed{F}) \Vdash_{\mathrm{nf}} \boxed{F}$, which implies $(\mathcal{B} \cup \boxed{F}) \Vdash_{\mathrm{nf}} \boxed{\Delta[F]}$. The rest follows applying Theorem 7.

Theorem 8 is not practical, but it encourages us to search for new capabilities which allow to give finite presentations for certain fixed points given by Theorem 8. Special contexts are such a capability.

7 Conclusion

The main contributions of this paper are: introduces special hypotheses as a closure operator and uses it to extend the coinductive circularity principle; uses special contexts as a means to obtain a distinguished class of special hypotheses, and extends the circular coinductive proof system with special contexts; explains how the circular coinductive proof system and its extension with special contexts are implemented in CIRC; briefly describes an algorithm which automatically finds special contexts in a given specification (the algorithm is already implemented in CIRC); shows that fair infinite executions of CIRC are in fact proofs.
Acknowledgments. We are grateful to Hans Zantema for the fruitful discussion regarding the special contexts and to Georgiana Caltais and Eugen Goriac for implementing in a short time the algorithm computing the special contexts.

References

1. J. Adámek. Introduction to coalgebra. *Theory and Applications of Categories*, 14(8):157–199, 2005.
2. J.-P. Allouche and J. Shallit. The ubiquitous prouhet-thue-morse sequence. In C. Ding, T. Helleseth, and H. Niederreiter, editors, *Sequences and Their applications (Proc. SETA'98)*, pages 1–16. Springer-Verlag, 1999.
3. M. Bidoit, R. Hennicker, and A. Kurz. Observational logic, constructor-based logic, and their duality. *Theoretical Computer Science*, 3(298):471–510, 2003.
4. A. Bouhoula and M. Rusinowitch. Observational proofs by rewriting. *Theor. Comput. Sci.*, 275(1-2):675–698, 2002.
5. M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. L. Talcott, editors. *All About Maude - A High-Performance Logical Framework, How to Specify, Program and Verify Systems in Rewriting Logic*, volume 4350 of *Lecture Notes in Computer Science*. Springer, 2007.
6. L. Dennis, A. Bundy, and I. Green. Using a generalisation critic to find bisimulations for coinductive proofs. In W. McCune, editor, *Proceedings of the 14th Conference on Automated Deduction*, volume 1249 of *Lecture Notes in Artificial Inteligence*, pages 276–290. Springer, 1996.
7. J. Goguen, K. Lin, and G. Rosu. Conditional circular coinductive rewriting with case analysis. In M. Wirsing, D. Pattinson, and R. Hennicker, editors, *WADT*, volume 2755 of *Lecture Notes in Computer Science*, pages 216–232. Springer, 2002.
8. J. Goguen and G. Malcolm. A hidden agenda. *J. of TCS*, 245(1):55–101, 2000.
9. J. Goguen and J. Meseguer. Completeness of Many-Sorted Equational Logic. *Houston Journal of Mathematics*, 11(3):307–334, 1985.
10. D. Hausmann, T. Mossakowski, and L. Schröder. Iterative circular coinduction for cocasl in isabelle/hol. In M. Cerioli, editor, *FASE*, volume 3442 of *Lecture Notes in Computer Science*, pages 341–356. Springer, 2005.
11. D. Lucanu and G. Rosu. Circ : A circular coinductive prover. In T. Mossakowski, U. Montanari, and M. Haveraaen, editors, *CALCO*, volume 4624 of *Lecture Notes in Computer Science*, pages 372–378. Springer, 2007.
12. J. Meseguer. General logics. In H.-D. E. et al., editor, *Logic Colloquium '87*, pages 275–329, North Holland, Amsterdam, 1989.
13. M. Niqui. Coinductive formal reasoning in exact real arithmetic. *Logical Methods in Computer Science*, 4(3:6):1–40, Sept. 2008.
14. L. C. Paulson. Mechanizing coinduction and corecursion in higher-order logic. *J. Logic and Computation*, 7:175204, 1997.
15. G. Roşu. Equality of streams is a pi_2^0-complete problem. In *Proceedings of the 11th ACM SIGPLAN International Conference on Functional Programming (ICFP'06)*. ACM, 2006.
16. G. Roşu and J. Goguen. Circular coinduction. 2001. Short paper at the *International Joint Conference on Automated Reasoning (IJCAR'01)*.
17. G. Roşu and D. Lucanu. Circular Coinduction –A Proof Theoretical Foundation. Technical Report UIUCDCS-R-2009-3037, University of Illinois at Urbana-Champaign, 2009. Submitted.
18. G. Rosu. *Hidden Logic*. PhD thesis, University of California at San Diego, 2000.

This appendix is for reviewers' convenience, to see how the circular coinductive proof system and its extension with special contexts presented in the paper works in CIRC. If this paper is accepted then the appendix will be removed and the reader will be referred to CIRC's website at `http://fsl.cs.uiuc.edu/circ`*, where these examples and many others can be executed online.*

A CIRC Proofs

Here we show the behavioral specification of streams discussed in Section 1 formalized in CIRC, the proof scripts of all the properties mentioned in the paper, as well as CIRC's output. The specification of the streams is included into a file called `stream.maude` and consists of three theories: for data, the equational specification of the streams and the behavioral specification of the streams. The proof script is included into a file called `stream.cmd` and includes commands for loading the specification, setting the parameters of the tool and for proving the desired properties. The following CIRC commands are used:

`loop init .` – put CIRC in the initial state
`set show details on .` – instructs CIRC to output all the proof details
`set auto contexts on/off` – instructs CIRC either to compute (`on`) or to not compute (`off`) the special contexts
`coinduction .` – CIRC attempts to prove all the existing goals
`in` ⟨*file-name*⟩ – loads the parameter file
`load goal` ⟨*property*⟩ – loads the the property to be proved
`coinduction .` – CIRC attempts to prove all the existing goals
`quit proof .` – resets the state before the last proofs; the previously proved properties are discarded
`set max no steps` ⟨*number*⟩ – sets the maximum number of steps to the *number*. The number of steps made by the CIRC tool is bigger than the number of circularities (how many times the rule [Derive] or [Derive^{scx}] is applied), because the implementation includes some additional rules. In the paper the maximum number is 50, in order to have a reasonable size for the output.

The output file includes the output messages of CIRC together with some info supplied by the Maude system. Note that the real run time is that written in parentheses and NOT the CPU time.

The properties checked are split into three categories. The first two properties show how the circular coinduction works. The next four properties show how the circular coinduction extended with the special contexts is used. Finally, we show that without the computed special contexts, the circular coinduction does not terminate for these properties (this is exhibited only for two properties but it is true for all four, in this case).

A.1 Input

```
(theory DATA is
  sort Data .
  ops 0 1 : -> Data .
  op _+_ : Data Data -> Data [prec 33 assoc comm] .
  op -_ : Data -> Data .
  op _*_ : Data Data -> Data [prec 31 assoc comm] .
  op not : Data -> Data .

  vars D D1 D2 : Data .

  eq not(0) = 1 .          eq not(1) = 0 .         eq not(not(D)) = D .
  eq D + 0 = D .           eq D * 1 = D .
  eq D + (- D) = 0 .       eq D * 0 = 0 .
  eq (- 1) * D = - D .     eq D * (D1 + D2) = D * D1 + D * D2 .
endtheory)


(theory EQ-STREAM is
  including  DATA .
  sort Stream .
  vars S S1 S2 : Stream .            var M : Data .

  --- these will be derivatives; here they are just ordinary operations
  op hd : Stream -> Data  .          op tl : Stream -> Stream .

  --- zero stream                     --- complements a stream
  op zero : -> Stream .               op not : Stream -> Stream .
  eq hd(zero) = 0 .                   eq hd(not(S)) =  not(hd(S)) .
  eq tl(zero) = zero .                eq tl(not(S)) = not(tl(S)) .

  --- summing operator                --- inclusion operation
  op _+_ : Stream Stream -> Stream
           [assoc comm] .             op `[_`] : Data -> Stream .
  eq hd(S1 + S2) = hd(S1) + hd(S2) .  eq hd([M]) = M .
  eq tl(S1 + S2) = tl(S1) + tl(S2) .  eq tl([M]) = zero .

  --- convolution product
  op _x_ : Stream Stream -> Stream [prec 31] .
  eq hd(S1 x S2) = hd(S1) * hd(S2) .
  eq tl(S1 x S2) = tl(S1) x S2 + [hd(S1)] x tl(S2) .

  --- odd and even streams            --- zip of streams
  ops odd even : Stream -> Stream .   op zip : Stream Stream -> Stream .
  eq hd(odd(S)) = hd(S) .             eq hd(zip(S1, S2)) = hd(S1) .
  eq tl(odd(S)) = even(tl(S)) .       eq tl(zip(S1, S2)) = zip(S2, tl(S1)) .
  eq even(S) = odd(tl(S)) .

  --- alternative function            --- Thue-Morse sequence M = f(0:tail(M))
  op f : Stream -> Stream .           op altMorse : -> Stream .
  eq hd(f(S)) = hd(S) .               eq hd(altMorse) = 0 .
  eq hd(tl(f(S))) = not(hd(S)) .      eq hd(tl(altMorse)) = 1 .
  eq tl(tl(f(S))) = f(tl(S)) .        eq tl(tl(altMorse)) =  f(tl(altMorse)) .

  --- Thue-Morse sequence M = 0:zip(inv(M),tail(M))
  op morse : -> Stream .
  eq hd(morse) = 0 .
  eq hd(tl(morse)) = 1 .
  eq tl(tl(morse)) = zip(tl(morse), not(tl(morse))) .
endtheory)
```

```
(ctheory STREAM is
  inc EQ-STREAM .
  derivative hd(*:Stream) .
  derivative tl(*:Stream) .
endctheory)
```

A.2 Proof Script

```
 loop init .

(set show details on .)

(set auto contexts off .)

in stream .

--->  STREAM |||- zip(odd(S), even(S)) = S

(add goal zip(odd(S:Stream), even(S:Stream)) = S:Stream .)

(coinduction .)

--->  STREAM |||- { f(S) = zip(S, not(S)), f(morse) = morse }

(add goal f(S:Stream) = zip(S:Stream, not(S:Stream)) .)

(add goal f(morse) = morse .)

(coinduction .)

---> The proving of the following properties needs special contexts

(set auto contexts on .)

in stream .

--->  STREAM |||- S x zero = zero

(add goal S:Stream x zero = zero .)

(coinduction .)

--->  STREAM |||- S1 x (S2 + S3) = S1 x S2 + S1 x S3

(add goal S1:Stream x (S2:Stream + S3:Stream) =
          S1:Stream x S2:Stream + S1:Stream x S3:Stream .)
(coinduction .)

--->  STREAM |||- (S1 + S2) x S3 = S1 x S3 + S2 x S3

(add goal (S1:Stream + S2:Stream) x S3:Stream =
          S1:Stream x S3:Stream + S2:Stream x S3:Stream .)
(coinduction .)

--->  STREAM |||- morse = altMorse

---> needs the proved above lemma: f(S) = zip(S, not(S))
---> ... or we may prove the two properties together:
```

```
(add goal f(S:Stream) = zip(S:Stream, not(S:Stream)) .)

(add goal morse = altMorse .)

(coinduction .)

---> The proving of the following properties DOES NOT terminate
--->     without special contexts

(set auto contexts off .)

(set max no steps 50 .)

in stream .

--->  STREAM |||- S x zero = zero DOES NOT terminate
--->     without special contexts

(add goal S:Stream x zero = zero .)

(coinduction .)

(quit proof .)

(set max no steps 100 .)

--->  STREAM |||- morse = altMorse DOES NOT terminate
--->     without special contexts

(add goal f(S:Stream) = zip(S:Stream, not(S:Stream)) .)

(add goal morse = altMorse .)

(coinduction .)
```

A.3 Output

```
> in stream-cmd.maude
rewrites: 6 in 783425647ms cpu (3ms real) (0 rewrites/second)

                CIRC 1.4 (May 19th, 2008)

rewrites: 20 in 6094666579ms cpu (40ms real) (0 rewrites/second)

Details will be shown.

rewrites: 20 in 6094666579ms cpu (4ms real) (0 rewrites/second)

Contexts will not be automatically computed.

==========================================
 .
Reading in file: "stream.maude"
rewrites: 1955 in 6094666579ms cpu (36ms real) (0 rewrites/second)
Introduced theory DATA

rewrites: 5853 in 6094666579ms cpu (70ms real) (0 rewrites/second)
Introduced theory EQ-STREAM

rewrites: 921 in 6094666579ms cpu (12ms real) (0 rewrites/second)
```

```
.......................................
---------------------------------------

Introduced ctheory STREAM


Done reading in file: "stream.maude"
=======================================
--->  STREAM |||- zip(odd(S), even(S)) = S
rewrites: 391 in 6094666579ms cpu (12ms real) (0 rewrites/second)

Goal added: zip(odd(S:Stream),even(S:Stream)) = S:Stream


rewrites: 2693 in 6094666579ms cpu (95ms real) (0 rewrites/second)
Goal zip(odd(S:Stream),even(S:Stream)) = S:Stream reduced to
     zip(odd(S:Stream),odd(tl(S:Stream))) = S:Stream

Hypo zip(odd(S:Stream),odd(tl(S:Stream))) = S:Stream added and coexpanded to
1.  hd(zip(odd(S:Stream),odd(tl(S:Stream)))) = hd(S:Stream)
2.  tl(zip(odd(S:Stream),odd(tl(S:Stream)))) = tl(S:Stream)
Goal hd(zip(odd(S:Stream),odd(tl(S:Stream)))) = hd(S:Stream) reduced to
     hd(S:Stream) = hd(S:Stream)
Goal hd(S:Stream) = hd(S:Stream) proved by reduction.
Goal tl(zip(odd(S:Stream),odd(tl(S:Stream)))) = tl(S:Stream) reduced to
     tl(S:Stream) = tl(S:Stream)
Goal tl(S:Stream) = tl(S:Stream) proved by reduction.

Proof succeeded.

=======================================
--->  STREAM |||- { f(S) = zip(S, not(S)), f(morse) = morse }
rewrites: 397 in 6094666579ms cpu (11ms real) (0 rewrites/second)

Goal added: f(S:Stream) = zip(S:Stream,not(S:Stream))


rewrites: 293 in 6094666579ms cpu (9ms real) (0 rewrites/second)

Goal added: f(morse) = morse


rewrites: 7470 in 6094666579ms cpu (364ms real) (0 rewrites/second)

Hypo f(morse) = morse added and coexpanded to
1.  hd(f(morse)) = hd(morse)
2.  tl(f(morse)) = tl(morse)

Hypo f(S:Stream) = zip(S:Stream,not(S:Stream)) added and coexpanded to
1.  hd(f(S:Stream)) = hd(zip(S:Stream,not(S:Stream)))
2.  tl(f(S:Stream)) = tl(zip(S:Stream,not(S:Stream)))
Goal hd(f(morse)) = hd(morse) reduced to
     0 = 0
Goal 0 = 0 proved by reduction.

Hypo tl(f(morse)) = tl(morse) added and coexpanded to
1.  hd(tl(f(morse))) = hd(tl(morse))
2.  tl(tl(f(morse))) = tl(tl(morse))
Goal hd(f(S:Stream)) = hd(zip(S:Stream,not(S:Stream))) reduced to
```

```
     hd(S:Stream) = hd(S:Stream)
Goal hd(S:Stream) = hd(S:Stream) proved by reduction.
Goal tl(f(S:Stream)) = tl(zip(S:Stream,not(S:Stream))) reduced to
     tl(f(S:Stream)) = zip(not(S:Stream),tl(S:Stream))

Hypo tl(f(S:Stream)) = zip(not(S:Stream),tl(S:Stream)) added and coexpanded to
1.  hd(tl(f(S:Stream))) = hd(zip(not(S:Stream),tl(S:Stream)))
2.  tl(tl(f(S:Stream))) = tl(zip(not(S:Stream),tl(S:Stream)))
Goal hd(tl(f(morse))) = hd(tl(morse)) reduced to
     1 = 1
Goal 1 = 1 proved by reduction.
Goal tl(tl(f(morse))) = tl(tl(morse)) reduced to
     f(tl(morse)) = f(tl(morse))
Goal f(tl(morse)) = f(tl(morse)) proved by reduction.
Goal hd(tl(f(S:Stream))) = hd(zip(not(S:Stream),tl(S:Stream))) reduced to
     not(hd(S:Stream)) = not(hd(S:Stream))
Goal not(hd(S:Stream)) = not(hd(S:Stream)) proved by reduction.
Goal tl(tl(f(S:Stream))) = tl(zip(not(S:Stream),tl(S:Stream))) reduced to
     f(tl(S:Stream)) = f(tl(S:Stream))
Goal f(tl(S:Stream)) = f(tl(S:Stream)) proved by reduction.

Proof succeeded.

=========================================
---> The proving of the following properties needs special contexts
rewrites: 20 in 6094666579ms cpu (21ms real) (0 rewrites/second)

Contexts will be automatically computed.

=========================================
 .
Reading in file: "stream.maude"
rewrites: 1970 in 6094666579ms cpu (20ms real) (0 rewrites/second)
Introduced theory DATA
Advisory: Module DATA redefined.

rewrites: 5864 in 6094666579ms cpu (54ms real) (0 rewrites/second)
Introduced theory EQ-STREAM
Advisory: Module EQ-STREAM redefined.

rewrites: 16563 in 6094666579ms cpu (218ms real) (0 rewrites/second)

.......................................
---------------------------------------

Introduced ctheory STREAM

The special contexts are:
*:Stream + V#2:Stream
V#1:Stream + *:Stream
*:Stream x V#2:Stream
V#1:Stream x *:Stream
not(*:Stream)
zip(*:Stream,V#2:Stream)
zip(V#1:Stream,*:Stream)

Advisory: Module STREAM redefined.

Done reading in file: "stream.maude"
=========================================
```

```
--->  STREAM |||- S x zero = zero
rewrites: 349 in 6094666579ms cpu (37ms real) (0 rewrites/second)

Goal added: S:Stream x zero = zero


rewrites: 3507 in 6094666579ms cpu (160ms real) (0 rewrites/second)

Hypo S:Stream x zero = zero added and coexpanded to
1.  hd(S:Stream x zero) = hd(zero)
2.  tl(S:Stream x zero) = tl(zero)
Goal hd(S:Stream x zero) = hd(zero) reduced to
     0 = 0
Goal 0 = 0 proved by reduction.
Goal tl(S:Stream x zero) = tl(zero) reduced to
     zero + zero = zero

Hypo zero + zero = zero added and coexpanded to
1.  hd(zero + zero) = hd(zero)
2.  tl(zero + zero) = tl(zero)
Goal hd(zero + zero) = hd(zero) reduced to
     0 = 0
Goal 0 = 0 proved by reduction.
Goal tl(zero + zero) = tl(zero) reduced to
     zero = zero
Goal zero = zero proved by reduction.

Proof succeeded.

========================================
--->  STREAM |||- S1 x (S2 + S3) = S1 x S2 + S1 x S3
rewrites: 582 in 6094666579ms cpu (11ms real) (0 rewrites/second)

Goal added: S1:Stream x(S2:Stream + S3:Stream) =
            S1:Stream x S2:Stream + S1:Stream x S3:Stream


rewrites: 5608 in 6094666579ms cpu (134ms real) (0 rewrites/second)

Hypo S1:Stream x(S2:Stream + S3:Stream) =
     S1:Stream x S2:Stream + S1:Stream x S3:Stream
added and coexpanded to
1.  hd(S1:Stream x(S2:Stream + S3:Stream)) =
    hd(S1:Stream x S2:Stream + S1:Stream x S3:Stream)
2.  tl(S1:Stream x(S2:Stream + S3:Stream)) =
    tl(S1:Stream x S2:Stream + S1:Stream x S3:Stream)
Goal hd(S1:Stream x(S2:Stream + S3:Stream)) =
     hd(S1:Stream x S2:Stream + S1:Stream x S3:Stream)
reduced to
     hd(S1:Stream)* hd(S2:Stream)+ hd(S1:Stream)* hd(S3:Stream) =
     hd(S1:Stream)* hd(S2:Stream)+ hd(S1:Stream)* hd(S3:Stream)
Goal hd(S1:Stream)* hd(S2:Stream)+ hd(S1:Stream)* hd(S3:Stream) =
     hd(S1:Stream)* hd(S2:Stream)+ hd(S1:Stream)* hd(S3:Stream)
proved by reduction.
Goal tl(S1:Stream x(S2:Stream + S3:Stream)) =
     tl(S1:Stream x S2:Stream + S1:Stream x S3:Stream)
reduced to
     [hd(S1:Stream)]x(tl(S2:Stream)+ tl(S3:Stream))+
     tl(S1:Stream)x(S2:Stream + S3:Stream) =
     [hd(S1:Stream)]x(tl(S2:Stream)+ tl(S3:Stream))+
```

```
     tl(S1:Stream)x(S2:Stream + S3:Stream)
Goal [hd(S1:Stream)]x(tl(S2:Stream)+ tl(S3:Stream))+
     tl(S1:Stream)x(S2:Stream + S3:Stream) =
     [hd(S1:Stream)]x(tl(S2:Stream)+ tl(S3:Stream))+
     tl(S1:Stream)x(S2:Stream + S3:Stream)
proved by reduction.

Proof succeeded.

=========================================
--->  STREAM |||- (S1 + S2) x S3 = S1 x S3 + S2 x S3
rewrites: 597 in 6094666579ms cpu (11ms real) (0 rewrites/second)

Goal added: (S1:Stream + S2:Stream)x S3:Stream =
            S1:Stream x S3:Stream + S2:Stream x S3:Stream


rewrites: 14547 in 6094666579ms cpu (345ms real) (0 rewrites/second)

Hypo (S1:Stream + S2:Stream)x S3:Stream =
     S1:Stream x S3:Stream + S2:Stream x S3:Stream
added and coexpanded to
1.  hd((S1:Stream + S2:Stream)x S3:Stream) =
    hd(S1:Stream x S3:Stream + S2:Stream x S3:Stream)
2.  tl((S1:Stream + S2:Stream)x S3:Stream) =
    tl(S1:Stream x S3:Stream + S2:Stream x S3:Stream)
Goal hd((S1:Stream + S2:Stream)x S3:Stream) =
     hd(S1:Stream x S3:Stream + S2:Stream x S3:Stream)
reduced to
     hd(S1:Stream)* hd(S3:Stream)+ hd(S2:Stream)* hd(S3:Stream) =
     hd(S1:Stream)* hd(S3:Stream)+ hd(S2:Stream)* hd(S3:Stream)
Goal hd(S1:Stream)* hd(S3:Stream)+ hd(S2:Stream)* hd(S3:Stream) =
     hd(S1:Stream)* hd(S3:Stream)+ hd(S2:Stream)* hd(S3:Stream)
proved by reduction.
Goal tl((S1:Stream + S2:Stream)x S3:Stream) =
     tl(S1:Stream x S3:Stream + S2:Stream x S3:Stream)
reduced to
     [hd(S1:Stream)+ hd(S2:Stream)]x tl(S3:Stream)+
     (tl(S1:Stream)+ tl(S2:Stream))x S3:Stream =
     ([hd(S1:Stream)]+[hd(S2:Stream)])x tl(S3:Stream)+
     (tl(S1:Stream)+ tl(S2:Stream))x S3:Stream

Hypo [hd(S1:Stream)+ hd(S2:Stream)]x tl(S3:Stream)+
     (tl(S1:Stream)+ tl(S2:Stream))x S3:Stream =
     ([hd(S1:Stream)]+[hd(S2:Stream)])x tl(S3:Stream)+
     (tl(S1:Stream)+ tl(S2:Stream))x S3:Stream
added and coexpanded to
1.  hd([hd(S1:Stream)+ hd(S2:Stream)]x tl(S3:Stream)+
    (tl(S1:Stream)+ tl(S2:Stream))x S3:Stream) =
    hd(([hd(S1:Stream)]+[hd(S2:Stream)])x tl(S3:Stream)+
    (tl(S1:Stream)+ tl(S2:Stream))x S3:Stream)
2.  tl([hd(S1:Stream)+ hd(S2:Stream)]x tl(S3:Stream)+
    (tl(S1:Stream)+ tl(S2:Stream))x S3:Stream) =
    tl(([hd(S1:Stream)]+[hd(S2:Stream)])x tl(S3:Stream)+
    (tl(S1:Stream)+ tl(S2:Stream))x S3:Stream)
Goal hd([hd(S1:Stream)+ hd(S2:Stream)]x tl(S3:Stream)+
     (tl(S1:Stream)+ tl(S2:Stream))x S3:Stream) =
     hd(([hd(S1:Stream)]+[hd(S2:Stream)])x tl(S3:Stream)+
     (tl(S1:Stream)+ tl(S2:Stream))x S3:Stream)
reduced to
```

```
     hd(S1:Stream)* hd(tl(S3:Stream))+ hd(S2:Stream)*hd(tl(S3:Stream))+
     hd(S3:Stream)* hd(tl(S1:Stream))+ hd(S3:Stream)*hd(tl(S2:Stream)) =
     hd(S1:Stream)* hd(tl(S3:Stream))+ hd(S2:Stream)*hd(tl(S3:Stream))+
     hd(S3:Stream)* hd(tl(S1:Stream))+ hd(S3:Stream)* hd(tl(S2:Stream))
Goal hd(S1:Stream)* hd(tl(S3:Stream))+ hd(S2:Stream)*hd(tl(S3:Stream))+
     hd(S3:Stream)* hd(tl(S1:Stream))+ hd(S3:Stream)* hd(tl(S2:Stream)) =
     hd(S1:Stream)* hd(tl(S3:Stream))+ hd(S2:Stream)* hd(tl(S3:Stream))+
     hd(S3:Stream)* hd(tl(S1:Stream))+ hd(S3:Stream)* hd(tl(S2:Stream))
proved by reduction.
Goal tl([hd(S1:Stream)+ hd(S2:Stream)]x tl(S3:Stream)+(tl(S1:Stream)+
     tl(S2:Stream))x S3:Stream) =
     tl(([hd(S1:Stream)]+[hd(S2:Stream)])x tl(S3:Stream)+
     (tl(S1:Stream)+ tl(S2:Stream))x S3:Stream)
reduced to
    [hd(S1:Stream)+ hd(S2:Stream)]x tl(tl(S3:Stream))+
    (zero +[hd(tl(S1:Stream))+ hd(tl(S2:Stream))])x tl(S3:Stream)+
    (tl(tl(S1:Stream))+ tl(tl(S2:Stream)))x S3:Stream =
    [hd(S1:Stream)+ hd(S2:Stream)]x tl(tl(S3:Stream))+
    (zero +[hd(tl(S1:Stream))+ hd(tl(S2:Stream))])x tl(S3:Stream)+
    (tl(tl(S1:Stream))+ tl(tl(S2:Stream)))x S3:Stream
Goal [hd(S1:Stream)+ hd(S2:Stream)]x tl(tl(S3:Stream))+
     (zero +[hd(tl(S1:Stream))+ hd(tl(S2:Stream))])x tl(S3:Stream)+
     (tl(tl(S1:Stream))+ tl(tl(S2:Stream)))x S3:Stream =
     [hd(S1:Stream)+ hd(S2:Stream)]x tl(tl(S3:Stream))+
     (zero +[hd(tl(S1:Stream))+ hd(tl(S2:Stream))])x tl(S3:Stream)+
     (tl(tl(S1:Stream))+ tl(tl(S2:Stream)))x S3:Stream
proved by reduction.

Proof succeeded.

========================================
--->  STREAM |||- morse = altMorse
========================================
---> needs the following lemma proved above: f(S) = zip(S, not(S))
========================================
---> ... or we may prove the two properties together:
rewrites: 409 in 6094666579ms cpu (12ms real) (0 rewrites/second)

Goal added: f(S:Stream) = zip(S:Stream,not(S:Stream))


rewrites: 301 in 6094666579ms cpu (11ms real) (0 rewrites/second)

Goal added: morse = altMorse


rewrites: 7759 in 6094666579ms cpu (453ms real) (0 rewrites/second)

Hypo morse = altMorse added and coexpanded to
1.  hd(morse) = hd(altMorse)
2.  tl(morse) = tl(altMorse)

Hypo f(S:Stream) = zip(S:Stream,not(S:Stream)) added and coexpanded to
1.  hd(f(S:Stream)) = hd(zip(S:Stream,not(S:Stream)))
2.  tl(f(S:Stream)) = tl(zip(S:Stream,not(S:Stream)))
Goal hd(morse) = hd(altMorse) reduced to
     0 = 0
Goal 0 = 0 proved by reduction.

Hypo tl(morse) = tl(altMorse) added and coexpanded to
```

```
1.  hd(tl(morse)) = hd(tl(altMorse))
2.  tl(tl(morse)) = tl(tl(altMorse))
Goal hd(f(S:Stream)) = hd(zip(S:Stream,not(S:Stream))) reduced to
     hd(S:Stream) = hd(S:Stream)
Goal hd(S:Stream) = hd(S:Stream) proved by reduction.
Goal tl(f(S:Stream)) = tl(zip(S:Stream,not(S:Stream))) reduced to
     tl(f(S:Stream)) = zip(not(S:Stream),tl(S:Stream))

Hypo tl(f(S:Stream)) = zip(not(S:Stream),tl(S:Stream))
added and coexpanded to
1.  hd(tl(f(S:Stream))) = hd(zip(not(S:Stream),tl(S:Stream)))
2.  tl(tl(f(S:Stream))) = tl(zip(not(S:Stream),tl(S:Stream)))
Goal hd(tl(morse)) = hd(tl(altMorse)) reduced to
     1 = 1
Goal 1 = 1 proved by reduction.
Goal tl(tl(morse)) = tl(tl(altMorse)) reduced to
     f(tl(altMorse)) = f(tl(altMorse))
Goal f(tl(altMorse)) = f(tl(altMorse)) proved by reduction.
Goal hd(tl(f(S:Stream))) = hd(zip(not(S:Stream),tl(S:Stream)))
reduced to
     not(hd(S:Stream)) = not(hd(S:Stream))
Goal not(hd(S:Stream)) = not(hd(S:Stream)) proved by reduction.
Goal tl(tl(f(S:Stream))) = tl(zip(not(S:Stream),tl(S:Stream)))
reduced to
     f(tl(S:Stream)) = f(tl(S:Stream))
Goal f(tl(S:Stream)) = f(tl(S:Stream)) proved by reduction.

Proof succeeded.

=========================================
---> The proving of the following properties DOES NOT terminate
=========================================
--->     without special contexts
rewrites: 20 in 6094666579ms cpu (21ms real) (0 rewrites/second)

Contexts will not be automatically computed.

rewrites: 42 in 6094666579ms cpu (8ms real) (0 rewrites/second)

The maximum number of proving steps was set to 50 .

=========================================
 .
Reading in file: "stream.maude"
rewrites: 1970 in 6094666579ms cpu (19ms real) (0 rewrites/second)
Introduced theory DATA
Advisory: Module DATA redefined.

rewrites: 5864 in 6094666579ms cpu (54ms real) (0 rewrites/second)
Introduced theory EQ-STREAM
Advisory: Module EQ-STREAM redefined.

rewrites: 934 in 6094666579ms cpu (14ms real) (0 rewrites/second)

.......................................
---------------------------------------

Introduced ctheory STREAM

Advisory: Module STREAM redefined.
```

```
Done reading in file: "stream.maude"
==========================================
--->  STREAM |||- S x zero = zero DOES NOT terminate
==========================================
--->     without special contexts
rewrites: 349 in 6094666579ms cpu (11ms real) (0 rewrites/second)

Goal added: S:Stream x zero = zero


rewrites: 10172 in 6094666579ms cpu (318ms real) (0 rewrites/second)

Hypo S:Stream x zero = zero added and coexpanded to
1.  hd(S:Stream x zero) = hd(zero)
2.  tl(S:Stream x zero) = tl(zero)
Goal hd(S:Stream x zero) = hd(zero) reduced to
     0 = 0
Goal 0 = 0 proved by reduction.
Goal tl(S:Stream x zero) = tl(zero) reduced to
     [hd(S:Stream)]x zero + tl(S:Stream)x zero = zero

Hypo [hd(S:Stream)]x zero + tl(S:Stream)x zero = zero
added and coexpanded to
1.  hd([hd(S:Stream)]x zero + tl(S:Stream)x zero) = hd(zero)
2.  tl([hd(S:Stream)]x zero + tl(S:Stream)x zero) = tl(zero)
Goal hd([hd(S:Stream)]x zero + tl(S:Stream)x zero) = hd(zero) reduced to
     0 = 0
Goal 0 = 0 proved by reduction.
Goal tl([hd(S:Stream)]x zero + tl(S:Stream)x zero) = tl(zero) reduced to
     zero x zero +[hd(S:Stream)]x zero +[hd(tl(S:Stream))]x zero +
     tl(tl(S:Stream))x zero =
     zero

Hypo zero x zero +[hd(S:Stream)]x zero +[hd(tl(S:Stream))]x zero +
     tl(tl(S:Stream))x zero = zero
added and coexpanded to
1.  hd(zero x zero +[hd(S:Stream)]x zero +[hd(tl(S:Stream))]x zero +
       tl(tl(S:Stream))x zero) = hd(zero)
2.  tl(zero x zero +[hd(S:Stream)]x zero +[hd(tl(S:Stream))]x zero +
       tl(tl(S:Stream))x zero) = tl(zero)
Goal hd(zero x zero +[hd(S:Stream)]x zero +[hd(tl(S:Stream))]x zero +
        tl(tl(S:Stream))x zero) = hd(zero)
reduced to
     0 = 0
Goal 0 = 0 proved by reduction.
Goal tl(zero x zero +[hd(S:Stream)]x zero +[hd(tl(S:Stream))]x zero +
        tl(tl(S:Stream))x zero) = tl(zero)
reduced to
    zero x zero + zero x zero + zero x zero +[0]x zero +
    [hd(S:Stream)]x zero + [hd(tl(S:Stream))]x zero +
    [hd(tl(tl(S:Stream)))]x zero + tl(tl(tl(S:Stream)))x zero = zero

Hypo zero x zero + zero x zero + zero x zero +[0]x zero +
     [hd(S:Stream)]x zero + [hd(tl(S:Stream))]x zero +
     [hd(tl(tl(S:Stream)))]x zero + tl(tl(tl(S:Stream)))x zero = zero
added and coexpanded to
1.  hd(zero x zero + zero x zero + zero x zero +[0]x zero +
       [hd(S:Stream)]x zero + [hd(tl(S:Stream))]x zero +
       [hd(tl(tl(S:Stream)))]x zero + tl(tl(tl(S:Stream)))x zero) =
```

```
    hd(zero)
2.  tl(zero x zero + zero x zero + zero x zero +[0]x zero +
       [hd(S:Stream)]x zero + [hd(tl(S:Stream))]x zero +
       [hd(tl(tl(S:Stream)))]x zero + tl(tl(tl(S:Stream)))x zero) =
    tl(zero)
Goal hd(zero x zero + zero x zero + zero x zero +[0]x zero +
        [hd(S:Stream)]xzero + [hd(tl(S:Stream))]x zero +
        [hd(tl(tl(S:Stream)))]x zero + tl(tl(tl(S:Stream)))x zero) =
     hd(zero)
reduced to
     0 = 0
Goal 0 = 0 proved by reduction.
Stopped: the number of prover steps was exceeded.
rewrites: 227 in 6094666579ms cpu (23ms real) (0 rewrites/second)
All hypotheses and lemmas gathered during previous proofs have been removed.

rewrites: 42 in 6094666579ms cpu (10ms real) (0 rewrites/second)

The maximum number of proving steps was set to 100 .

========================================
--->  STREAM |||- morse = altMorse DOES NOT terminate
========================================
--->     without special contexts
rewrites: 401 in 6094666579ms cpu (13ms real) (0 rewrites/second)

Goal added: f(S:Stream) = zip(S:Stream,not(S:Stream))


rewrites: 293 in 6094666579ms cpu (6ms real) (0 rewrites/second)

Goal added: morse = altMorse


rewrites: 12925 in 6094666579ms cpu (596ms real) (0 rewrites/second)

Hypo morse = altMorse added and coexpanded to
1.  hd(morse) = hd(altMorse)
2.  tl(morse) = tl(altMorse)

Hypo f(S:Stream) = zip(S:Stream,not(S:Stream)) added and coexpanded to
1.  hd(f(S:Stream)) = hd(zip(S:Stream,not(S:Stream)))
2.  tl(f(S:Stream)) = tl(zip(S:Stream,not(S:Stream)))
Goal hd(morse) = hd(altMorse) reduced to
     0 = 0
Goal 0 = 0 proved by reduction.

Hypo tl(morse) = tl(altMorse) added and coexpanded to
1.  hd(tl(morse)) = hd(tl(altMorse))
2.  tl(tl(morse)) = tl(tl(altMorse))
Goal hd(f(S:Stream)) = hd(zip(S:Stream,not(S:Stream))) reduced to
     hd(S:Stream) = hd(S:Stream)
Goal hd(S:Stream) = hd(S:Stream) proved by reduction.
Goal tl(f(S:Stream)) = tl(zip(S:Stream,not(S:Stream))) reduced to
     tl(f(S:Stream)) = zip(not(S:Stream),tl(S:Stream))

Hypo tl(f(S:Stream)) = zip(not(S:Stream),tl(S:Stream))
added and coexpanded to
1.  hd(tl(f(S:Stream))) = hd(zip(not(S:Stream),tl(S:Stream)))
2.  tl(tl(f(S:Stream))) = tl(zip(not(S:Stream),tl(S:Stream)))
```

```
Goal hd(tl(morse)) = hd(tl(altMorse)) reduced to
     1 = 1
Goal 1 = 1 proved by reduction.
Goal tl(tl(morse)) = tl(tl(altMorse)) reduced to
     f(tl(morse)) = f(tl(altMorse))

Hypo f(tl(morse)) = f(tl(altMorse)) added and coexpanded to
1.  hd(f(tl(morse))) = hd(f(tl(altMorse)))
2.  tl(f(tl(morse))) = tl(f(tl(altMorse)))
Goal hd(tl(f(S:Stream))) = hd(zip(not(S:Stream),tl(S:Stream))) reduced to
     not(hd(S:Stream)) = not(hd(S:Stream))
Goal not(hd(S:Stream)) = not(hd(S:Stream)) proved by reduction.
Goal tl(tl(f(S:Stream))) = tl(zip(not(S:Stream),tl(S:Stream))) reduced to
     f(tl(S:Stream)) = f(tl(S:Stream))
Goal f(tl(S:Stream)) = f(tl(S:Stream)) proved by reduction.
Goal hd(f(tl(morse))) = hd(f(tl(altMorse))) reduced to
     1 = 1
Goal 1 = 1 proved by reduction.

Hypo tl(f(tl(morse))) = tl(f(tl(altMorse))) added and coexpanded to
1.  hd(tl(f(tl(morse)))) = hd(tl(f(tl(altMorse))))
2.  tl(tl(f(tl(morse)))) = tl(tl(f(tl(altMorse))))
Goal hd(tl(f(tl(morse)))) = hd(tl(f(tl(altMorse)))) reduced to
     0 = 0
Goal 0 = 0 proved by reduction.
Goal tl(tl(f(tl(morse)))) = tl(tl(f(tl(altMorse)))) reduced to
     f(zip(tl(morse),not(tl(morse)))) = f(f(tl(altMorse)))

Hypo f(zip(tl(morse),not(tl(morse)))) = f(f(tl(altMorse)))
added and coexpanded to
1.  hd(f(zip(tl(morse),not(tl(morse))))) = hd(f(f(tl(altMorse))))
2.  tl(f(zip(tl(morse),not(tl(morse))))) = tl(f(f(tl(altMorse))))
Goal hd(f(zip(tl(morse),not(tl(morse))))) = hd(f(f(tl(altMorse))))
reduced to
     1 = 1
Goal 1 = 1 proved by reduction.

Hypo tl(f(zip(tl(morse),not(tl(morse))))) = tl(f(f(tl(altMorse))))
added and coexpanded to
1.  hd(tl(f(zip(tl(morse),not(tl(morse)))))) = hd(tl(f(f(tl(altMorse)))))
2.  tl(tl(f(zip(tl(morse),not(tl(morse)))))) = tl(tl(f(f(tl(altMorse)))))
Goal hd(tl(f(zip(tl(morse),not(tl(morse)))))) = hd(tl(f(f(tl(altMorse)))))
    reduced to
     0 = 0
Goal 0 = 0 proved by reduction.
Stopped: the number of prover steps was exceeded.
```