ICFEM'09, LNCS 5885, pp 639-659. 2009

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@illinois.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 or infinite behaviors of systems. 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., [8, 18, 6, 19, 13, 17] among many others. Circular coinduction [19] 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 [19] (not maintained anymore), in Isabelle/HOL for CoCasl [13], and in CIRC [15].

Circular coinduction can be formalized as a three-rule proof system deriving pairs of the form $\mathcal{B} \cup \mathcal{F} \Vdash^{\circlearrowleft} \mathcal{G}$, where $\mathcal{B}$ is the *(initial) specification* (or *initial hypotheses*), $\mathcal{F}$ is the set of *frozen hypotheses*, and $\mathcal{G}$ is the set of *goals* [22] (see also Figure 2). Both the hypotheses and goals are sets of equations. The frozen hypotheses are written in a box (e.g., $\boxed{e}$), with the intuition that those cannot be used in contextual reasoning. The freezing operation is essential in

proving the soundness of the circular coinduction. We illustrate the circular coinduction proof system using an intuitive behavioral specification of infinite streams. We do not assume the reader is familiar with behavioral specifications and/or coinduction, so our notions are explained in detail. We make use of conventional algebraic specification notation, briefly explained in Appendix A. Intuitively, a stream is an infinite sequence $x_1 : x_2 : x_3 : \ldots$. The derivatives Δ for streams are given by the operations head, hd, and tail, tl, defined by $hd(x : S) = x$ and $tl(x : S) = S$. The intuition for the derivatives is that they can be used to completely "derive" any stream, in that they can eventually reach, or observe, any of the stream's elements; derivatives are dual to constructors in inductive data types. Let `STREAM` be the specification of streams including besides data axioms the following operations defined in terms of head and tail:

$odd,\ even : Stream \to Stream$ $\qquad$ $zip : Stream \times Stream \to Stream$
$hd(odd(S)) = hd(S)$ $\qquad$ $hd(zip(S, S')) = hd(S)$
$tl(odd(S)) = even(tl(S))$ $\qquad$ $tl(zip(S, S')) = zip(S', tl(S))$
$even(S) = odd(tl(S))$

Here are the intuitive definitions for the three stream operations above:

$zip(x_1{:}x_2{:}\ldots, y_1{:}y_2{:}\ldots) = x_1{:}y_1{:}x_2{:}y_2{:}\ldots,$
$odd(x_1{:}x_2{:}x_3{:}x_4\ldots) = x_1{:}x_3{:}\ldots,$ and $even(x_1{:}x_2{:}x_3{:}x_4{:}\ldots) = x_2{:}x_4{:}\ldots.$

Streams S, S' are *behaviorally equivalent* in `STREAM`, written $\texttt{STREAM} \Vdash S = S'$, iff $\texttt{STREAM} \vdash hd(tl^i(S)) = hd(tl^i(S'))$ for $i = 0, 1, 2, \ldots$; this corresponds to the intuition that `S` and `S'` are indistinguishable under experiments using the derivatives. The behavioral equivalence $\Vdash$ over streams is a Π^0_2-hard problem [20], i.e., it is strictly harder than equational satisfaction; thus, there is no complete procedure to enumerate all the behavioral truths. Thus, the best we can do is to approximate behavioral equivalence, which is what circular coinduction does.

Since the circular coinductive deduction $\Vdash^{\circlearrowleft}$ is sound for the behavioral equivalence $\Vdash$ (see, e.g., Theorem 2), it follows that the following proof tree shows that $\texttt{STREAM} \Vdash e$, where e is the property $zip(odd(S),even(S)) = S$:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\texttt{STREAM} \cup \left\{\boxed{zip(odd(S), even(S))} = \boxed{S}\right\} \Vdash^{\circlearrowleft} \emptyset} \quad \texttt{STREAM} \cup \left\{\boxed{zip(odd(S), even(S))} = \boxed{S}\right\} \vdash \boxed{hd(zip(odd(S), even(S)))} = \boxed{hd(S)}}{\texttt{STREAM} \cup \left\{\boxed{zip(odd(S), even(S))} = \boxed{S}\right\} \Vdash^{\circlearrowleft} \left\{\boxed{hd(zip(odd(S), even(S)))} = \boxed{hd(S)}\right\}} \quad \texttt{STREAM} \cup \left\{\boxed{zip(odd(S), even(S))} = \boxed{S}\right\} \vdash \boxed{tl(zip(odd(S), even(S)))} = \boxed{tl(S)}}{\texttt{STREAM} \cup \left\{\boxed{zip(odd(S), even(S))} = \boxed{S}\right\} \Vdash^{\circlearrowleft} \left\{\begin{array}{l}\boxed{hd(zip(odd(S), even(S)))} = \boxed{hd(S)},\\ \boxed{tl(zip(odd(S), even(S)))} = \boxed{tl(S)}\end{array}\right\}}}{\texttt{STREAM} \Vdash^{\circlearrowleft} \left\{\boxed{zip(odd(S), even(S))} = \boxed{S}\right\}}$$

We give a "bottom-up" description of the above proof tree, as it is built by CIRC prover [15, 14]. In the first step, $\texttt{STREAM} \Vdash e$ is reduced to showing that $\texttt{STREAM} \cup \{\boxed{e}\} \Vdash^{\circlearrowleft} \boxed{\Delta[e]}$, where (when e is an equation $t = t'$, $\boxed{e}$ is $\boxed{t} = \boxed{t'}$)

$\Delta[e] = \{hd(zip(odd(S), even(S))) = hd(S), tl(zip(odd(S), even(S))) = tl(S)\}$
$\quad = \{hd(zip(odd(S),odd(tl(S)))) = hd(S), tl(zip(odd(S),odd(tl(S)))) = tl(S)\},$

where the definition of *even* was applied. The first equation in $\Delta[e]$ is discarded

because it is a consequence of the STREAM axioms, $hd(zip(odd(S), odd(tl(S)))) = hd(odd(S)) = hd(S)$. The derivation of the second equation in $\Delta[e]$ is trickier: first, $\texttt{STREAM} \vdash \boxed{tl(zip(odd(S),odd(tl(S))))} = \boxed{zip(odd(tl(S)),odd(tl(tl(S))))}$ by the STREAM axioms, then $\texttt{STREAM} \cup \{\boxed{e}\} \vdash \boxed{zip(odd(tl(S)),odd(tl(tl(S))))} = \boxed{tl(S)}$ using the frozen hypothesis $\boxed{e}$ with the substitution $\theta(S) = tl(S)$, and finally by equational transitivity it follows that $\texttt{STREAM} \cup \{\boxed{e}\} \vdash \boxed{tl(zip(odd(S), odd(tl(S))))} = \boxed{tl(S)}$, so the second equation in $\Delta[e]$ is also discarded. Note that freezing is necessary, otherwise the derivatives in $\Delta[e]$ would follow by the congruence rule from e, no matter whether the property holds or not.

Many interesting properties like the above can be proved by simple circular coinduction. However, its success strictly depends upon the existence of a finite set of frozen equations $\mathcal{F}$ extending the original set of proof goals that would allow for the derivation of a circular coinductive proof (in the above example $\mathcal{F}$ is $\{\boxed{zip(odd(S), even(S))} = \boxed{S}\}$). Unfortunately, the Π^2_0-hardness result in [20] tells us that for some goals there is no such finite set of frozen equations.

Let us next discuss an example where the simple circular coinduction system fails to build a finite proof tree. A technique based on behavioral equivalence for checking well-definedness of stream operations is proposed in [27]. For instance, the well-definedness of *zip* follows by defining streams g and h by $hd(g) = hd(h) = 1$ (or any other constant), $tl(g) = zip(g, g)$, $tl(h) = zip(h, h)$, and then showing that $\texttt{STREAM} \Vdash g = h$.[3] Circular coinduction fails to find a proof for this property because the building process of the proof tree does not terminate. We show that a finite proof tree can be quickly obtained if the additional hypotheses defined by the special contexts are used (a context is a term with a hole, written "$*$" in this paper; special contexts are defined in Section 4). Two special contexts are needed, namely $\Gamma = \{zip(*{:}Stream, S{:}Stream), zip(S{:}Stream, *{:}Stream)\}$. These contexts together with the frozen hypothesis (corresponding to the initial goal) yield the following two special hypotheses:
$\Gamma[g = h] = \{zip(g, S{:}Stream) = zip(h, S{:}Stream), zip(S{:}Stream, g) = zip(S{:}Stream, h)\}$.
Here is the proof tree generated by the extended proof system:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\texttt{STREAM} \cup \{\boxed{g} = \boxed{h}\} \cup \boxed{\Gamma[g = h]} \Vdash^{\circlearrowleft} \emptyset}}{\texttt{STREAM} \cup \{\boxed{g} = \boxed{h}\} \cup \boxed{\Gamma[g = h]} \vdash \boxed{hd(g)} = \boxed{hd(h)}}}{\texttt{STREAM} \cup \{\boxed{g} = \boxed{h}\} \cup \boxed{\Gamma[g = h]} \Vdash^{\circlearrowleft} \left\{\boxed{hd(g)} = \boxed{hd(h)}\right\}}}{\texttt{STREAM} \cup \{\boxed{g} = \boxed{h}\} \cup \boxed{\Gamma[g = h]} \vdash \boxed{tl(g)} = \boxed{tl(h)}}}{\texttt{STREAM} \cup \{\boxed{g} = \boxed{h}\} \cup \boxed{\Gamma[g = h]} \Vdash^{\circlearrowleft} \left\{\begin{array}{l}\boxed{hd(g)} = \boxed{hd(h)}, \\ \boxed{tl(g)} = \boxed{tl(h)}\end{array}\right\}}$$

$$\dfrac{}{\texttt{STREAM} \Vdash^{\circlearrowleft} \boxed{g} = \boxed{h}}$$

The special hypotheses are used in the deduction of $\boxed{tl(g)} = \boxed{tl(h)}$ (fourth line of the proof tree above) as follows: we have $tl(g) = zip(g, g)$ and $tl(h) = zip(h, h)$ as defining axioms, and $\boxed{zip(g, g)} = \boxed{zip(h, g)} = \boxed{zip(h, h)}$ follow from $\boxed{\Gamma[g = h]}$.

[3] The authors warmly thank Hans Zantema for supplying this example.

Special contexts must satisfy a certain well-foundedness condition w.r.t. derivatives (see Definition 5). Not all contexts are special. For example, if $odd(*{:}Stream)$ were special then our proof system with special contexts would be unsound, as shown by the following scenario inspired from [10]. 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 $b = 0:1:0^\infty$.

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 able to automatically prove a larger class of behavioral properties than that of [22]. The soundness of the new proof system is proved. The new system is effective for a given behavioral specification only if the special contexts are known. An algorithm that computes the special contexts is presented. Since the correctness proof and complexity of the algorithm needs more space, it will be presented in an extended version of this paper.

The techniques presented in this paper have been implemented and extensively evaluated in CIRC [15, 14], a behavioral extension of Full Maude [5] tuned and optimized for automated and combined inductive and coinductive proving. CIRC implements the proof rules as reduction rules such that for an input $(\mathcal{B}, \mathcal{G})$, it incrementally computes $\mathcal{F}$ such that $\mathcal{B} \cup \mathcal{F} \Vdash^{\circlearrowleft} \mathcal{G}$. CIRC implements a criterion for automatic detection of special contexts and it can automatically prove both properties discussed above among many others *requiring special contexts*; for example, if the stream elements come from a commutative ring, CIRC can automatically prove that $zeros = 0{:}0{:}0{:}...$ and $[1] = 1{:}zeros$ are zero and unit elements for $\times$ (the convolution product), the distributivity of $\times$ over $+$ of streams, the equivalence of the two definitions for the Thue-Morse stream (see Example 9), the well-definedness of stream operations, etc. Similar properties are proved for the shuffle product of streams [24] and for infinite binary trees [25]. The special contexts are also useful for proving equivalences of basic process algebra (BPA) processes (see Example 8). All these and many other examples can be found on and executed using the online version of CIRC (`http://fsl.cs.uiuc.edu/CIRC`).

The rest of the paper is structured as follows. Sections 2 and 3 recall from [22] our proof theoretical approach for behavioral satisfaction and circular coinduction, focusing on the role of the freezing operator. Section 4 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 it yields a particular class of special hypotheses. Section 5 presents how the CIRC theorem prover implements both the circular coinduction and the special contexts. An algorithm for automatically computing special contexts is briefly presented.

2 Behavioral Specifications and Coinduction

We assume the reader familiar with the basics of many sorted algebraic specifications. A list of terms and notations used in this paper is given in Appendix A.

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*. If $\delta[*{:}h] \in \Delta$ then the sort h is called a *hidden sort*. Let $H \subseteq S$ be the set of all hidden sorts of $\mathcal{B}$. Remaining sorts are called *data, or visible, sorts*; let $V = S - H$ be 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 or visible, equation* is an equation built with only data terms. Equation "$(\forall X)\, t = t'$ `if` *cond*" is called a *hidden equation* iff the common sort of t and t' is hidden. We consider only equations whose conditions are conjunctions of visible equalities. If $\mathcal{G}$ is a set of Σ-equations, let *visible*($\mathcal{G}$) and *hidden*($\mathcal{G}$) be the sets of $\mathcal{G}$'s visible and hidden equations, respectively.

Sorts are thus 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 $\delta[*{:}h] \in \Delta_{h'}$ and $C[*{:}h']$ is an experiment, then so is $C[\delta[*{:}h]]$. Note that we only consider unary contexts, that is, contexts with only one hole "$*$". If Δ is clear, we may write experiment for Δ-experiment and context for Δ-context.

Example 1. (*Streams*) A stream over D is an infinite sequence $a_1 : a_2 : a_3 : \ldots$ whose elements a_i belong to D. In this paper we assume that $(D, +, \cdot, not)$ is a boolean ring (with 0 the unit for $+$, 1 the unit for $\cdot$, $not(0) = 1$, $not(1) = 0$, etc.). The operations over D can be extended to streams using corecursive equations:

$a : s + a' : s' = (a + a') : (s + s')$
$a : s \times a' : s' = (a \cdot a') : (s \times a' : s' + [a] \times s')$, where $[a] = a : 0^\infty$
$not(a : s) = not(a) : not(s)$

If $s = a_1{:}a_2{:}a_3{:}\ldots$ and $s' = a'_1{:}a'_2{:}a'_3{:}\ldots$ are streams, then $s \times s'$ is the stream $(a_1 \cdot a'_1) : (a_1 \cdot a'_2 + a_2 \cdot a'_1) : (a_1 \cdot a'_3 + a_2 \cdot a'_2 + a_3 \cdot a'_1) : \ldots$ and is called the *convolution product* of s and s'. Other stream operations, e.g., *zeros*, *ones*, *odd*, *even*, and *zip* can be defined like in Section 1. For more details on streams and their properties see, e.g., [23, 24, 27].

The equational part $\mathcal{B}$ of the behavioral specification of streams includes a hidden sort *Stream*, for streams, a visible sort *Data*, for elements of the streams, the operations over data and streams, and the equations over data (e.g., the axioms of the boolean ring) and streams. The set of the derivatives Δ includes two derivatives: $hd(*{:}Stream)$ and $tl(*{:}Stream)$. The experiments are of the form $hd(tl^i(*{:}Stream))$, where $i \geq 0$. Any specification of streams can be expressed in terms of the derivatives. For instance, the constant stream *zeros* and the convolution product are specified as follows:

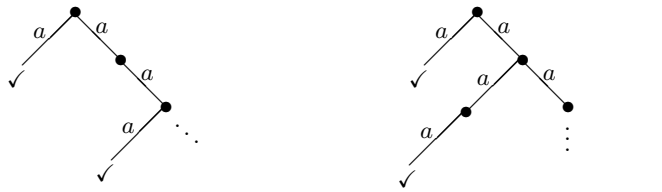
$hd(zeros) = 0$ $\qquad$ $hd(S \times S') = hd(S) \cdot hd(S')$
$tl(zeros) = zeros$ $\qquad$ $tl(S \times S') = tl(S) \times S' + [hd(S)] \times tl(S')$

The other operations are specified in a similar way. Let `STREAM` denote the behavioral specification of streams expressed in terms of the derivatives *hd* and *tl*.

Example 2. (*Processes*) We next consider the particular class of processes defined by basic process algebra (BPA) terms and guarded recursive specifications [9].

The equational specification of the processes is defined by the following items:

- a sort $Alph$ for the atomic actions (the alphabet),
- a sort Pid for the process variables,
- a sort $Pexp$ for the process terms (expressions),
- the constructors for the process terms (we regard subsorting as constructor):
  $$Alph < Pexp \qquad Pid < Pexp$$
  $$\_+\_ : Pexp\ Pexp \to Pexp \qquad \_;\_ : Pexp\ Pexp \to Pexp$$
  which describe the grammar $p ::= a \mid X \mid p + p \mid p\,;p$, where p ranges over $Pexp$, a over $Alph$, and X over Pid,
- a sort Peq together with the constructor $\_=_{def}\_ : Pid\ Pexp \to Peq$ for the possibly recursive process equations,
- a sort $Set\{Peq\}$ together with the constructors
  $$Peq < Set\{Peq\} \qquad \_,\_ : Set\{Peq\}\ Set\{Peq\} \to Set\{Peq\}$$
  and associativity/commutativity/idempotence axioms for sets of process equations, plus axioms ensuring that each Pid is defined at most once.



a) $X =_{def} a + a\,;a\,;X$ b) $A =_{def} a + a\,;A\,;a$

Fig. 1. Two infinitary completed trace equivalent processes

Fig. 1 shows two processes specified by guarded recursive equations. Recall that guarded recursive specifications allow a unique solution modulo bisimulation equivalence [9]. A specification like $X =_{def} a + X$ is not guarded.

The behavioral aspect we consider here is intended to capture the *infinitary completed trace equivalence* (see, e.g., [26]) and therefore it is given by two derivatives: $\_\{\_\} : Pexp \to Pexp$, with the intuitive meaning that $p\{a\}$ is the process $+\{q \mid p \xrightarrow{a} q\}$, and $\checkmark? : Pexp \to Bool$, with the intuitive meaning that an experiment $\checkmark?(p\{a_1\} \ldots \{a_n\})$ returns true if and only if $a_1 \ldots a_n$ is a completed trace for p, i.e., $p \xrightarrow{a_1} p_1 \ldots \xrightarrow{a_n} \checkmark$.

The problem is that the transition relation is partial and we work with derivatives which are totally defined. For instance, only an a-transition is defined for $a\,;p$. Since the derivatives are applied on all processes, we have to evaluate expressions like $a\,;p\{b\}$ with $b \neq a$. Therefore we consider two new process constants: $\perp$ with the meaning $p \xrightarrow{a} \perp$ iff there is no $q \neq \perp$ such that $p \xrightarrow{a} q$, and $\checkmark$ to denote the result obtained when we evaluate $a\{a\}$ (this corresponds to a successful termination).

We therefore consider only one hidden sort $Pexp$ and the derivatives $\Delta = \{*{:}Pexp\{A{:}Alph\}, \checkmark?(*{:}Pexp)\}$. The definitions of the process operations in terms

of the derivatives are given by the following equations:

$(p+q)\{a\} = p\{a\} + q\{a\}$	$a\{a\} = \checkmark$
$\checkmark?(p+q) = \checkmark?(p) \vee \checkmark?(q)$	$b\{a\} = \bot \;\; \texttt{if} \;\; b \neq a$
$(p\,;q)\{a\} = p\{a\}\,;q \;\; \texttt{if} \;\; p \neq \checkmark \wedge p \neq \bot$	$\checkmark?(a) = \mathit{false} \qquad \checkmark?(\checkmark) = \mathit{true}$
$\checkmark?(p\,;q) = \checkmark?(p) \wedge \checkmark?(q)$	$\checkmark\,;p = p$
$X\{a\} = p\{a\}$ for each equation $X =_{def} p$	$\checkmark\{a\} = \bot$
$\checkmark?(X) = \mathit{false}$	$\bot\,;p = \bot \qquad \bot + p = p$

Let BPA denote the above behavioral specification of the basic process algebra.

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 [12]. 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), 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 [16]. Fix a signature Σ.

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 extend $\vdash$ 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 $E^\bullet$.

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 $\mathcal{B} \vdash$ visible$(\mathcal{G})$ and $\Delta[\mathcal{G} - \mathcal{B}^\bullet] \subseteq \mathcal{G}$.*

For instance, if $(\forall X)\, t = t'$ is a stream equation, then $\texttt{STREAM} \Vdash (\forall X)\, t = t'$ if and only if $(\forall n \geq 0)\, \texttt{STREAM} \vdash (\forall X)\, hd(tl^n(t)) = hd(tl^n(t'))$. Similarly, if $(\forall X)\, t = t'$ expresses a property over processes, then $\texttt{BPA} \Vdash (\forall X)\, t = t'$ if and only if $(\forall a_1 \ldots a_n)\, \texttt{BPA} \vdash (\forall X)\, \checkmark?((t\{a_1\} \ldots \{a_n\})) = \checkmark?((t'\{a_1\} \ldots \{a_n\}))$. In both cases we assume that $\vdash$ is the equational deduction relation.

It can be shown that $\Vdash$ extends $\vdash$, i.e., if $\mathcal{B} \vdash e$ then $\mathcal{B} \Vdash e$ (see [22]). 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 [11, 3, 1]; also, our $\equiv$ may 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 $\vdash$.

Theorem 1. ***(Coinduction)****[22] 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 [22]. 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 3. A proof by coinduction of the stream property $S \times zeros = zeros$ is given by $G = \{S \times zeros = zeros, S \times zeros + S' = S'\}$ and the following equations showing that $\Delta(G) \subseteq \mathcal{G} = \overline{G \cup \mathtt{STREAM}^\bullet}$:

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

Example 4. Here we consider a proof by coinduction of a property over processes. Having the guarded recursive specification $U =_{def} a\,;V$, $V =_{def} b\,;U$, $Y =_{def} a\,;b\,;Y$, then $\{V = b\,;Y,\ U = Y\}$ together with the equations given below is a proof by coinduction of $U = Y$:

$\checkmark?(U) = \checkmark?(Y)$	(in $\mathtt{BPA}^\bullet$)	$U\{b\} = Y\{b\}$	(in $\mathtt{BPA}^\bullet$)
$V\{a\} = b\,;Y\{a\}$	(in $\mathtt{BPA}^\bullet$)	$\checkmark?(V) = \checkmark?(b\,;Y)$	(in $\mathtt{BPA}^\bullet$)
$V\{b\} = U$	(in $\mathtt{BPA}^\bullet$)	$b\,;Y\{b\} = Y$	(in $\mathtt{BPA}^\bullet$)
$V\{b\} = b\,;Y\{b\}$	(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 an "approximation" of $\equiv$. It is worth noting that it is virtually impossible to compute $\equiv$ precisely, because, as shown in [20], the problem of behavioral satisfaction is a Π_2^0 hard problem even for the particular specification of streams discussed in this paper. Circular coinduction [21, 19, 22] automates coinductive proving by dynamically inferring a suitable behaviorally closed set $\mathcal{G}$ including the property(ies) to prove.

3 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 $\boxed{-} : s \to \mathit{Frozen}$ 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. It is important to note here 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 axiomatically defined entailment system in this paper. 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) If e is a visible unfrozen equation then $E \cup \mathcal{F} \vdash \boxed{e}$ 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 $E \cup C[\mathcal{F}] \vdash C[\mathcal{G}]$.

(E ranges over 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 [22].

Figure 2 defines circular coinduction as a proof system for deriving pairs of the form $\mathcal{B} \cup \mathcal{F} \Vdash^{\circlearrowleft} \mathcal{G}$, where $\mathcal{B}$ is the original behavioral specification, $\mathcal{F}$ is a set of *frozen hypotheses*, and $\mathcal{G}$ is a set of (frozen) *goals*. Initially, $\mathcal{F}$ is empty and $\mathcal{G}$ is the frozen version $\boxed{G}$ of the original goals G to prove. Circular coinduction iteratively at-

$$\frac{\cdot}{\mathcal{B} \cup \mathcal{F} \Vdash^{\circlearrowleft} \emptyset} \quad \text{[Done]}$$

$$\frac{\mathcal{B} \cup \mathcal{F} \Vdash^{\circlearrowleft} \mathcal{G}, \quad \mathcal{B} \cup \mathcal{F} \vdash \boxed{e}}{\mathcal{B} \cup \mathcal{F} \Vdash^{\circlearrowleft} \mathcal{G} \cup \{\boxed{e}\}} \quad \text{[Reduce]}$$

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

Fig. 2. *Circular coinduction as a proof system: If $\mathcal{B} \Vdash^{\circlearrowleft} \boxed{G}$ is derivable then $\mathcal{B} \Vdash G$*

tempts to complete $\boxed{G}$ 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 $\mathcal{B} \cup \mathcal{F} \Vdash^{\circlearrowleft} \mathcal{G}$ is derivable, so the proof system would be unsound. An example circular coinduction proof tree is presented in Section 1.

Theorem 2. ***(soundness of circular coinduction)*** *[22] 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 2, then $\mathcal{B} \Vdash G$.*

4 Special Hypotheses and Special Contexts

We now show that circular coinduction can be extended by adding "on the fly" new hypothesis which are sound provided that the derivation process successfully terminates. 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 $(\forall C)\, \mathcal{B} \vdash C[e]$ whenever $\mathcal{B} \vdash C^{\leq}[F]$, where $C^{\leq}$ is the set of Δ-experiments D with $|D| \leq |C|$ ($|C|$ is the depth of C; see Appendix A). The set of special hypotheses for F, written $F^{\leq}$, 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 2 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 5. If F consists of an equality of two streams $a = b$, then the following equations are in $F^{\leq}$: $S + a = S + b$, $a + S = b + S$, $S \times a = S \times b$, $a \times S = b \times S$, $not(a) = not(b)$, $zip(S, a) = zip(S, b)$, $zip(a, S) = zip(b, S)$, where S is a variable over streams. The equations $odd(a) = odd(b)$ and $even(a) = even(b)$ 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 [22]) 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, $F^{\leq} = \equiv\restriction_H$).*

Example 6. The additional equation $S \times zeros + S' = S'$ in Example 3 is special for $F = \{S \times zeros = zeros\}$, so, by Theorem 3, its frozen form can be used as hypothesis without including it into the initial set of goals.

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 [22]) 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 introduced in [10]: a context $\gamma[*{:}h]$ was called "special" in [10] 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 7. For the streams specified in Example 1, the following are special contexts: $*{:}Stream + S{:}Stream$, $S{:}Stream + *{:}Stream$, $*{:}Stream \times S{:}Stream$, $S{:}Stream \times *{:}Stream$, $not(*{:}Stream)$, $zip(*{:}Stream, S{:}Stream)$, and $zip(S{:}Stream, *{:}Stream)$. Moreover, any combination of these contexts (e.g., $(*{:}Stream \times S{:}Stream) + S'{:}Stream$) is special, as well. In contrast, $odd(*{:}Stream)$ and $even(*{:}Stream)$ are not special contexts: e.g., $\texttt{STREAM} \vdash hd(tl(odd(*{:}Stream))) = hd(tl(tl(*{:}Stream)))$ and $|hd(tl(tl(*{:}Stream)))| > |hd(tl(*{:}Stream))|$.

The problem of detecting special contexts appears to be very hard in its full generality. However, in practice it turns out that a small number of special contexts and compositions of them, like described in Section 5, are sufficient. More precisely, it tends to suffice to search for special contexts among the operations already defined by the specification, focusing on particular arguments of them. For example, one may ask oneself if the operation *zip* on streams, with focus on its first argument, is a special context. To check that, all one

needs to do is to show that for any experiment applied to the focused operation, in our case to $zip(*{:}Stream, S{:}Stream)$, one can, via equational deduction, reduce the overall depth to the $*$ variable. In our case, it is easy to see that a generic stream experiment $hd(tl^n(*{:}Stream))$ applied to $zip(*{:}Stream, S{:}Stream)$ will eventually reduce the depth to $*$: if $n = 0$ one gets $hd(*{:}Stream)$ of depth 1, so the depth is reduced by 1; if $n = 1$ one gets $hd(S{:}Steam)$ so there is no $*$ anymore; if $n \geq 2$ then one can apply the two depth-most tl operations on the $zip(*{:}Stream, S{:}Stream)$ context and obtain, via equational deduction, a new context $zip(tl(*{:}Stream), tl(S{:}Stream))$, so two tl operations have been removed and only one has been inserted above $*$. One can do the same for the other argument of zip. The above certify that zip is indeed special. While the above is not a fully general technique to find all the special contexts, it works so well in practice that we implemented it as integral part of the CIRC prover (Section 5).

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 are aware of.

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

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 2 with the more general one below:

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

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 proof system extended with rule* [$\mathsf{Derive}^{\mathsf{scx}}$]*, then $\mathcal{B} \Vdash G$.*

Example 8. Figure 3 shows the proof tree for $\mathtt{BPA} \Vdash^{\circlearrowleft} A = X$, where A and X are defined in Fig. 1. The following notations are used:

$F_1 = \{A = X\} \quad F_2 = F_1 \cup \{A\{a\} = X\{a\}\} \quad F_3 = F_2 \cup \{A\{a\}\{a\} = X\{a\}\{a\}\}$

$\Gamma_i = \Gamma(F_i) \qquad$ for $i = 1, 2, 3$

where Γ is the closure under context composition of

$$\{p + *{:}Pexp,\ *{:}Pexp + p,\ p\,;*{:}Pexp,\ *{:}Pexp\,;p\}.$$

The role of the special hypotheses is explained by showing how $A\{a\}\{a\}\{a\} = X\{a\}\{a\}\{a\}$ is deduced:

$A\{a\} = \checkmark + A\,;a$	(by the axioms in BPA)
$\checkmark + A\,;a = \checkmark + X\,;a$	(a special hypothesis in $\Gamma_1 \subset \Gamma_3$)
$(\checkmark + X\,;a)\{a\}\{a\} = \checkmark + X\,;a$	(by the axioms in BPA)
$A\{a\}\{a\}\{a\} = \checkmark + X\,;a$	(congruence & transitivity)
$X\{a\} = \checkmark + a\,;X$	(by the axioms in BPA)

$$\frac{\begin{array}{l}\texttt{BPA} \cup \boxed{F_3} \cup \boxed{\Gamma_3} \Vdash^{\circlearrowleft} \emptyset \\ \texttt{BPA} \cup \boxed{F_3} \cup \boxed{\Gamma_3} \vdash \boxed{A\{a\}\{a\}\{a\}} = \boxed{X\{a\}\{a\}\{a\}}\end{array}}{\begin{array}{l}\texttt{BPA} \cup \boxed{F_3} \cup \boxed{\Gamma_3} \Vdash^{\circlearrowleft} \boxed{A\{a\}\{a\}\{a\}} = \boxed{X\{a\}\{a\}\{a\}} \\ \texttt{BPA} \cup \boxed{F_3} \cup \boxed{\Gamma_3} \vdash \boxed{\checkmark?(A\{a\}\{a\})} = \boxed{\checkmark?(X\{a\}\{a\})}\end{array}}{\texttt{BPA} \cup \boxed{F_3} \cup \boxed{\Gamma_3} \Vdash^{\circlearrowleft} \left\{\begin{array}{l}\boxed{A\{a\}\{a\}\{a\}} = \boxed{X\{a\}\{a\}\{a\}} \\ \boxed{\checkmark?(A\{a\}\{a\})} = \boxed{\checkmark?(X\{a\}\{a\})}\end{array}\right\}}$$

$$\frac{\begin{array}{l}\texttt{BPA} \cup \boxed{F_2} \cup \boxed{\Gamma_2} \Vdash^{\circlearrowleft} \boxed{A\{a\}\{a\}} = \boxed{X\{a\}\{a\}} \\ \texttt{BPA} \cup \boxed{F_3} \cup \boxed{\Gamma_3} \vdash \boxed{\checkmark?(A\{a\})} = \boxed{\checkmark?(X\{a\})}\end{array}}{\texttt{BPA} \cup \boxed{F_2} \cup \boxed{\Gamma_2} \Vdash^{\circlearrowleft} \left\{\begin{array}{l}\boxed{A\{a\}\{a\}} = \boxed{X\{a\}\{a\}} \\ \boxed{\checkmark?(A\{a\})} = \boxed{\checkmark?(X\{a\})}\end{array}\right\}}$$

$$\frac{\begin{array}{l}\texttt{BPA} \cup \boxed{F_1} \cup \boxed{\Gamma_1} \Vdash^{\circlearrowleft} \boxed{A\{a\}} = \boxed{X\{a\}} \\ \texttt{BPA} \cup \boxed{F_3} \cup \boxed{\Gamma_3} \vdash \boxed{\checkmark?(A)} = \boxed{\checkmark?(X)}\end{array}}{\texttt{BPA} \cup \boxed{F_1} \cup \boxed{\Gamma_1} \Vdash^{\circlearrowleft} \left\{\begin{array}{l}\boxed{A\{a\}} = \boxed{X\{a\}} \\ \boxed{\checkmark?(A)} = \boxed{\checkmark?(X)}\end{array}\right\}}$$

$$\frac{}{\texttt{BPA} \Vdash^{\circlearrowleft} \boxed{A} = \boxed{X}}$$

Fig. 3. The proof tree for $\texttt{BPA} \Vdash^{\circlearrowleft} A = X$

$X\{a\}\{a\}\{a\} = \checkmark + a\,;X$ (by the axioms in BPA)
$A\{a\}\{a\}\{a\} = X\{a\}\{a\}\{a\}$ (by the axioms in $F_2 \cup \Gamma_1 \subset F_3 \cup \Gamma_3$)

The special hypothesis used above is obtained as follows: we have $A = X$ in F_1, $A\,;a = X\,;a$ is in Γ_1 by using the special context $*{:}Pexp\,;p$, and $\checkmark + A\,;a = \checkmark + X\,;a$ is in Γ_1 by using the special context $p + *{:}Pexp$. The behavioral closure of $F_3 \cup \Gamma_3$ is an infinite set, hence this example exhibits the ability of the proof system with special contexts to handle infinite coinductive proofs.

Example 9. In this example we show that two (known) definitions for the famous Thue-Morse sequence (see, e.g., [2]) are equivalent. The Thue-Morse sequence is the stream of bits whose n-th bit is computed as follows: 1) write the number n in binary; if the number of ones in this binary expansion is odd then the n-th bit is 1; if even then the n-th bit is 0. The first definition we consider is given by

$$morse = 0 : zip(not(morse), tl(morse)).$$

The second definition uses an auxiliary function:

$$altMorse = f(0 : tl(altMorse)) \text{ where } f(a : s) = a : not(a) : f(s)$$

Figure 4 shows the (partial) proof tree for $\texttt{STREAM} \Vdash^{\circlearrowleft} morse = altMorse$, using the behavioral definitions expressed in terms of the derivatives head and tail. The proof of this property requires the following lemma: $f(S) = zip(S, not(S))$. Since the proof system is able to handle sets of goals, we prove the two properties simultaneously. Only the special hypotheses generated by the special context $f(*{:}Stream)$ are required. The following notations are used:

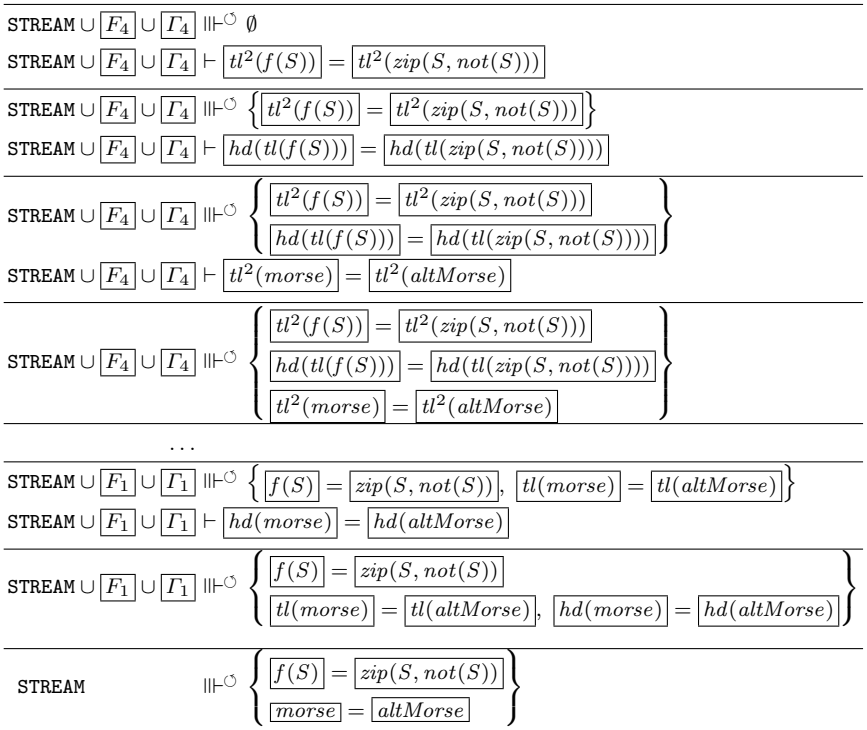


Fig. 4. The proof tree for $\texttt{STREAM} \Vdash^{\circlearrowleft} \{morse = altMorse, f(S) = zip(S, not(S))\}$

$F_1 = \{morse = altMorse\}$ $\quad F_2 = F_1 \cup \{f(S) = zip(S, not(S))\}$
$F_3 = F_2 \cup \{tl(morse)=tl(altMorse)\}$ $\quad F_4 = F_3 \cup \{tl(f(S))=tl(zip(S, not(S)))\}$
$\Gamma_i = \{f(t) = f(t') \mid t = t' \in F_i\}$, $i = 1, \ldots, 4$

In order to see the role of special hypotheses, we explain how $\boxed{tl^2(morse)} = \boxed{tl^2(altMorse)}$ is deduced. We get $tl^2(morse) = zip(tl(morse), not(tl(morse))) = f(tl(morse))$ and, similarly, $tl^2(altMorse) = f(tl(altMorse)$ by the definition of the operations. The conclusion follows now by applying the special hypothesis $f(tl(morse)) = f(tl(altMorse))$ from Γ_3. Again, the behavioral closure of $F_4 \cup \Gamma_4$ is an infinite set, hence this example exhibits the ability of the proof system with special contexts to handle infinite coinductive proofs.
It is worth noting that the above property can also be proved using the hypotheses generated by the special contexts in Example 7.

5 Implementation in CIRC

CIRC implements the circular coinduction proof system by the reduction rules in Fig. 5. The entailment relation used in CIRC is $\mathcal{E} \vdash_{\rightarrow,\leftarrow} (\forall X)t = t' \texttt{ if } \wedge_i u_i = v_i$ iff $\mathrm{nf}(t) = \mathrm{nf}(t')$, where $\mathrm{nf}(t)$, the *normal form* of 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;

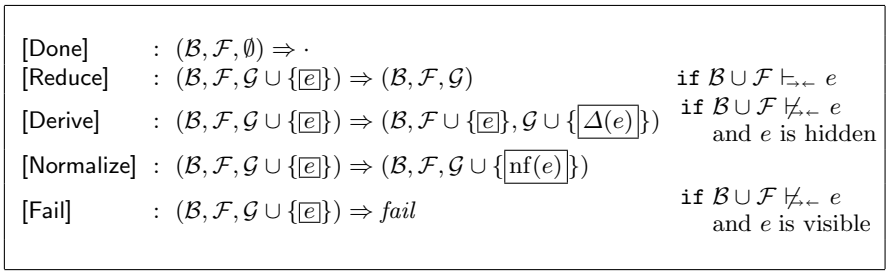


Fig. 5. CIRC's reduction rules implementing the circular coinduction

– the equations in the specification are oriented and used as rewrite rules.

It is easy to see that the reduction rules [Done], [Reduce], and [Derive] implement the proof rules with the same names given in Fig. 2. The reduction rules [Normalize] and [Fail] have no correspondent in the proof system and are used to ease the user interaction with the prover. A failure does not necessarily mean that the answer is false. The failure relation $\mathcal{E} \not\vdash_{\rightarrow\leftarrow} \boxed{e}$ says that the corresponding normal forms are different. Since we do not impose any confluence conditions on the specification, it is possible that the normal forms are 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 inequality of the two normal forms is an inductive theorem. More details about CIRC tool can be found in [14].

The implementation of the system extended with the special contexts is obtained by replacing the implementation of [Derive] with that of [Derive$^{\mathsf{scx}}$]. 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 includes 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 7. 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 $\boxed{y} := \boxed{x}$ instantiates y by matching $\boxed{y}$ against the normal form of $\boxed{x}$. However, note that this elegant implementation works in our case thanks to the matching-in-condition mechanism specific to Maude. Let us explain how this mechanism works for $\gamma = *{:}\mathit{Stream} + S'$ and $\boxed{S \times \mathit{zeros}} = \boxed{\mathit{zeros}}$. If $x \mapsto S \times \mathit{zeros}$, then $\boxed{y} := \boxed{x}$ returns $y \mapsto \mathit{zeros}$. Replacing γ, x and y in $\boxed{\gamma[x]} = \boxed{\gamma[y]}$ we obtain the desired result: $S \times \mathit{zeros} + S' = \mathit{zeros} + S'$.

The algorithm used by CIRC for computing a set Γ of special contexts of minimal depth is quite complex and it will be presented in detail, together with

its correctness and complexity, in an extended version. Here we briefly describe the intuition behind this algorithm.

We first introduce some notation. Let Ctx° be the set of contexts $f(x_1, \ldots, x_n)$ with $f \in \Sigma^{hidden}$, $x_i = *$ for exactly one hidden argument, say the ith, and in the rest x_j is a variable, where Σ^{hidden} is the set of operations with hidden result and at least one hidden argument. If C is a Δ-context, then the *hidden depth* $|C|^\cdot$ of C is defined by $|C|^\cdot = |C|$ if C is hidden, and $|C|^\cdot = |C| - 1$ if C is visible. Thus, the hidden depth of a context is the number of operations in Σ^{hidden} on the path to $*$. If $k \geq -1$ and $\Gamma \subseteq Ctx^\circ$, then a (k, Γ)-*composite* is defined as:

1. any non-star variable and any constant is a $(-1, \Gamma)$-composite;
2. any Δ-context C is a $(|C|^\cdot, \Gamma)$-composite;
3. if $f : v_1 \ldots v_n \to v$ is a data operator or a generalized constant and t_i is a (k_i, Γ)-composite for $i = 1, \ldots, n$, then $f(t_1, \ldots, t_n)$ is a (k, Γ)-composite, where $k = \max\{k_1, \ldots, k_n\}$;
4. if $\gamma \in \Gamma$ and t is a (k, Γ)-composite, then $\gamma[t]$ is a (k, Γ)-composite;
5. if C is a Δ-context, t a (k, Γ)-composite with $k = -1$ or t of the form $g(t_1, \ldots, t_n)$ with g a generalized constant, then $C[t]$ is a (k, Γ)-composite.

The goal is to find a predicate $Comp(C, t)$ and a set $\Gamma \subseteq Ctx^\circ$ such that the predicate $Special(\Gamma)$, given by

$$Special(\Gamma) \stackrel{\text{def}}{=} (\forall \gamma \in \Gamma)(\forall \delta \in \Delta)\ Comp(\delta, \gamma),$$

implies that each Γ-context is special. If we have an algorithm for computing such a predicate $Comp(C, t)$, then the searching for a suitable Γ requires the evaluation of the predicate for a small set of pairs (C, t). The algorithm implemented in CIRC uses the following definition for $Comp(C, t)$: if C is a Δ-context and t is a (k, Γ)-composite term, then

$$Comp(C, t) \stackrel{\text{def}}{=} \mathcal{B} \vdash C[t] = t' \wedge t' \text{ is } (k', \Gamma)\text{-composite} \wedge k' \leq k + |C|^\cdot$$

If the property $Comp(\delta, \gamma)$ can be algorithmically checked for $\delta \in \Delta$ and $\gamma \in \Gamma$, then the property $Special(\Gamma)$ can be checked as well. So, the only thing we have to do is to find a suitable set Γ. Our algorithm starts with $\Gamma = Ctx^\circ$. For each $\delta \in \Delta$ and $\gamma \in \Gamma$, it computes the normal form of $\delta[\gamma]$ and it checks if this normal form is $(|\delta|^\cdot, \Gamma)$-composite (this can be algorithmically checked). If the answer is yes in all the cases, then $Special(\Gamma)$ holds and the algorithm returns Γ. If the answer is no for some δ and γ, then γ is removed from Γ and the algorithm is applied again over the new set Γ. The algorithm stops when the current set of contexts becomes empty (no special contexts found) or, as we have seen above, we have $Special(\Gamma)$. The following algorithmic description summarizes the procedure described above:

```
check(Γ)
  if Γ ≠ ∅
  then for each γ ∈ Γ
            for each δ ∈ Δ
                if not Comp(δ, γ)
                then return check(Γ \ {γ})
return Γ
```

The special contexts basis Γ for a $\mathcal{B}$ is computed by the call `check`(Ctx°).

Here is an excerpt of the dialog with CIRC exhibiting how the special contexts in Example 9 are automatically computed:

```
Maude> (set auto contexts on .)
Maude> in stream
------------------------------------------

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

Setting on the switch `auto contexts`, CIRC automatically execute the algorithm computing the special contexts for each behavioral theory loaded after that. Recall that a behavioral theory includes, among other things, the declaration of the derivatives. We see that $f(*{:}Stream)$ is not found as a special context. This is because the definition of f is in terms of $hd(*{:}Stream)$, $hd(tl(*{:}Stream))$, and $tl(tl(*{:}Stream))$. The algorithm can be specialized for such cases and this specialization can be executed by introducing `check scx` command:

```
Maude> (check scx f(*:Stream) using hd(*:Stream) hd(tl(*:Stream))
                                    tl(tl(*:Stream)) .)
f(*:Stream) is a special context
```

We can use CIRC to see that the special contexts are essential in proving the properties described in Example 9. We first execute the circular coinduction without special contexts by setting the switch `auto contexts` off:

```
Maude> (set auto contexts off .)
Contexts will not be automatically computed.
Maude> in stream
Maude> (add goal f(S:Stream) = zip(S:Stream, not(S:Stream))
Maude> (add goal morse = altMorse .)
Maude> (coinduction .)
Stopped: the number of prover steps was exceeded.
```

The output message saying that the number of prover steps was exceeded is a clue that the algorithm does not terminate (we may check that by increasing the number of steps). Then we execute the algorithm with the special contexts:

```
Maude> (set auto contexts on .)
Maude> in stream
Maude> (add goal f(S:Stream) = zip(S:Stream, not(S:Stream)) .)
Maude> (add goal morse = altMorse .)
Maude> (coinduction .)
Proof succeeded.

  Number of derived goals: 8
  Number of proving steps performed: 41
  Maximum number of proving steps is set to: 256

Proved properties:
```

```
tl(f(S:Stream)) = zip(not(S:Stream),tl(S:Stream))
tl(morse) = tl(altMorse)
f(S:Stream) = zip(S:Stream,not(S:Stream))
morse = altMorse
```

We see that the algorithm successfully terminated this time. Moreover, it displays the set $\mathcal{F}$ of frozen hypotheses it discovered during the proving process. At this point, the user can display the proof tree like the one in Example 9 by introducing the command `(show proof .)`. We encourage the interested reader to try the examples in this paper, as well as many others, in CIRC using its online web interface at `http://fsl.cs.uiuc.edu/CIRC` (all the examples discussed in this paper are already provided there).

6 Conclusion, More Related Work, and Future Work

The main contributions of this paper are: we introduced *special hypotheses* as a closure operator and use it to extend the coinductive circularity principle; we used *special contexts* to obtain a distinguished class of special hypotheses, and extended the circular coinductive proof system with special contexts; we explained how the circular coinductive proof system and its extension with special contexts are implemented in CIRC; we described an algorithm which automatically finds special contexts in a given specification (the algorithm is already implemented in CIRC); we showed on non-trivial examples that the special contexts are useful.

The proof system presented in this paper and its implementation in CIRC has its roots in the circular coinductive rewriting algorithm given in [10] and early implemented in the BOBJ system [19]. An advantage of our proof theoretical approach discussed in this paper, which extends with special contexts the one we previously proposed in [22], is that it can be relatively easily combined with proof plans and proof critics similar to those defined in [7]. A part of them are already supported in the current version of CIRC [14].

An important aspect of behavioral specifications, that we did not discuss in this paper, is that of well-definedness. An ingenious method for checking well-definedness of behavioral specifications over streams that follow a common corecursive specification style is given in [27], using a reduction to ordinary termination, which can further be checked using off-the-shelf termination tools. It would be interesting to explore the relationship between behavioral well-definedness and behavioral equivalence, and to understand under what conditions one could reduce behavioral equivalence to termination; an incipient discussion on this topic can be found in [27].

A closely related topic is observational logic; see [3] for a recent reference. In particular, our proof theoretic definition of the behavioral entailment relation $\Vdash$ is reminiscent of the infinitary proof system defined in [3]. We believe that our circular coinductive proof system can be seamlessly adapted to observational logic and also that CIRC can be used as is to do sound observational logic proofs.

Our next goal is to incorporate case analysis into our automated behavioral prover, CIRC, both at the level of its underlying proof system and within its

algorithm that computes special contexts. Finally, even though CIRC provides support for combined inductive and coinductive proving, the proof theoretical foundations for circular induction still need to be elaborated in detail.

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 *SETA 1998*, 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. *Theoretical Computer Science*, 275(1-2):675–698, 2002.
5. M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. L. Talcott. All about Maude - a high-performance logical framework, how to specify, program and verify systems in rewriting logic. In *All About Maude*, volume 4350 of *LNCS*. Springer, 2007.
6. T. Coquand. Infinite objects in type theory. In *TYPES 1993*, volume 806 of *LNCS*, pages 62–78. Springer, 1994.
7. L. Dennis. *Proof Planning Coinduction*. PhD thesis, Edinburgh University, 1998.
8. L. Dennis, A. Bundy, and I. Green. Using a generalisation critic to find bisimulations for coinductive proofs. In *CADE 1996*, volume 1249 of *LNAI*, pages 276–290. Springer, 1996.
9. W. Fokkink. *Introduction to Process Algebra*. Springer-Verlag, Berlin, 2000.
10. J. Goguen, K. Lin, and G. Roşu. Conditional circular coinductive rewriting with case analysis. In *WADT 2002*, volume 2755 of *LNCS*, pages 216–232, 2003.
11. J. Goguen and G. Malcolm. A hidden agenda. *Theoretical Computer Science*, 245(1):55–101, 2000.
12. J. Goguen and J. Meseguer. Completeness of Many-Sorted Equational Logic. *Houston Journal of Mathematics*, 11(3):307–334, 1985.
13. D. Hausmann, T. Mossakowski, and L. Schröder. Iterative circular coinduction for CoCasl in Isabelle/HOL. In *FASE 2005*, volume 3442 of *LNCS*, pages 341–356. Springer, 2005.
14. D. Lucanu, E.-I. Goriac, G. Caltais, and G. Roşu. CIRC : A behavioral verification tool based on circular coinduction. In *CALCO 2009*, volume 5728 of *LNCS*, pages 433–442. Springer, 2009.
15. D. Lucanu and G. Roşu. Circ : A circular coinductive prover. In *CALCO 2007*, volume 4624 of *LNCS*, pages 372–378. Springer, 2007.
16. J. Meseguer. General logics. In H.-D. E. et al., editor, *Logic Colloquium '87*, pages 275–329, North Holland, Amsterdam, 1989.
17. M. Niqui. Coinductive formal reasoning in exact real arithmetic. *Logical Methods in Computer Science*, 4(3:6):1–40, Sept. 2008.
18. L. C. Paulson. Mechanizing coinduction and corecursion in higher-order logic. *Logic and Computation*, 7:175–204, 1997.
19. G. Roşu. *Hidden Logic*. PhD thesis, University of California at San Diego, 2000.

20. G. Roşu. Equality of streams is a Π^0_2-complete problem. In *ICFP'06*, pages 184–191. ACM, 2006.
21. G. Roşu and J. Goguen. Circular coinduction. 2001. Short paper *IJCAR'01*.
22. G. Roşu and D. Lucanu. Circular Coinduction – A Proof Theoretical Foundation. In *CALCO 2009*, volume 5728 of *LNCS*, pages 127–144. Springer, 2009.
23. J. Rutten. Behavioural Differential Equations: A Coinductive Calculus of Streams, Automata, and Power Series. *Theoretical Computer Science*, 308(1–3):1–53, 2003.
24. J. Rutten. A coinductive calculus of streams. *Mathematical Structures in Computer Science*, 15(1):93–147, 2005.
25. A. Silva and J. Rutten. Behavioural differential equations and coinduction for binary trees. In *WoLLIC 2007*, volume 4576 of *LNCS*, pages 322–336, 2007.
26. R. J. van Glabbeek. The linear time - branching time spectrum II. In *CONCUR 1993*, volume 715 of *LNCS*, pages 66–81. Springer, 1993.
27. H. Zantema. Well-definedness of streams by termination. In *RTA 2009*, volume 5595 of *LNCS*, pages 164–178. Springer, 2009.

A Glossary

Signature. A signature Σ over the set of *sorts* S is a $(S^* \times S)$-indexed set of operation names. If $\sigma \in \Sigma_{w,s}$ then $w \in S^*$ is the *arity* of σ and s is the *sort* of σ; we typically write $\sigma : w \to s$ instead of $\sigma \in \Sigma_{w,s}$ when Σ is understood;

Constant = an operation whose arity is empty and written as $\sigma : \to s$.

Variable = a symbol x having associated a sort s and often written as $x{:}s$.

Σ-term. A Σ-term with variables in X is inductively defined as follows: any variable $x{:}s$ of sort s is a term of sorts s; any constant $\sigma : \to s$ in Σ is a term of sorts s; if $\sigma : s_1 \dots s_n \to s$ is an operation name in Σ, t_i is a term of sort s_i for each $i \in \{1, \dots, n\}$, then $\sigma(t_1, \dots, t_n)$ is a term of sort s.

Ground term = a term without variables.

Σ-equation = an expression e of the form $(\forall X)\, t = t' \ \texttt{if} \bigwedge_{i \in \{1,\dots,n\}} u_i = v_i$ with t, t', u_i, and v_i Σ-terms with variables in X for all $i \in \{1, \dots, n\}$; 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 $i \in \{1, \dots, n\}$, have, respectively, the same sort. When $n = 0$ we call the equation unconditional and omit the condition (i.e., we write it $(\forall X)\, t = t'$). The *sort of* e is the common sort of t and t'.

Σ-context (for sort s) = a Σ-term C which has one occurrence of a distinguished variable $*{:}s$ of sort s; we may write $C[*{:}s]$ instead of just C.

$|C|$ = the *depth* of the context C: $|*{:}s| = 0$ and $|C[\sigma[*{:}s]]| = |C| + 1$, $\sigma \in \Sigma$.

$C[e]$. If $C[*{:}s]$ is a context and e is an equation $(\forall X)\, t = t' \ \texttt{if}\ c$ of sort s, then $C[e]$ is the equation $(\forall X \cup Y)\, C[t] = C[t'] \ \texttt{if}\ c$, where Y is the set of non-star variables occurring in $C[*{:}s]$. When C is not a context (it does not include the star variable $*$), $C[e]$ is the identity equation $(\forall X) C = C$. So, a Σ-context C induces a partially defined *equation transformer* $e \mapsto C[e]$.

$C[E]$. If C a context and e an equation, then $C[E] = \{C[e] \mid e \in E\}$.

$\mathbf{C}_s$. If $\mathbf{C}$ is a set of contexts, $\mathbf{C}_s$ denotes all the contexts of sort s in $\mathbf{C}$.

$\mathbf{C}[e]$. If $\mathbf{C}$ is a set of contexts and e an equation, then $\mathbf{C}[e] = \{C[e] \mid C \in \mathbf{C}\}$.

$\mathbf{C}[E]$. If $\mathbf{C}$ is a set of contexts and E a set of equations, then $\mathbf{C}[E] = \bigcup_{e \in E} \mathbf{C}[e]$.

Algebraic specification or simply a *specification*, is a triple (S, Σ, E), where S is a set of sorts, Σ is a signature over S, and E is a set of Σ-equations.