Iterative Circular Coinduction for CoCasl in Isabelle/HOL

Daniel Hausmann, Till Mossakowski, and Lutz Schröder

BISS, Dept. of Computer Science, University of Bremen

Abstract. Coalgebra has in recent years been recognized as the framework of choice for the treatment of reactive systems at an appropriate level of generality. Proofs about the reactive behavior of a coalgebraic system typically rely on the method of coinduction. In comparison to 'traditional' coinduction, which has the disadvantage of requiring the invention of a bisimulation relation, the method of *circular coinduction* allows a higher degree of automation. As part of an effort to provide proof support for the algebraic-coalgebraic specification language CoCasl, we develop a new coinductive proof strategy which iteratively constructs a bisimulation relation, thus arriving at a new variant of circular coinduction. Based on this result, we design and implement tactics for the theorem prover Isabelle which allow for both automatic and semiautomatic coinductive proofs. The flexibility of this approach is demonstrated by means of examples of (semi-)automatic proofs of consequences of CoCasl specifications, automatically translated into Isabelle theories by means of the Bremen heterogeneous Casl tool set Hets.

Introduction

Coalgebra is emerging as a standard unifying framework for the specification of reactive systems [11], complementing the use of universal algebra for the specification of the functional correctness of programs. Following this paradigm, several coalgebraic specification languages have recently been designed, e.g. the *Coalgebraic Class Specification Language* CCSL, which is geared towards object oriented programs, and the algebraic-coalgebraic specification language CoCasl [7], which extends the standard algebraic specification language Casl [1, 8] and thus allows not only the specification of both functional and reactive requirements, but also the intercombination of inductive datatypes and coinductive process types.

This work forms part of an effort to provide proof support for CoCasl. To this end, an existing embedding of Casl into the semiautomatic theorem prover Isabelle/HOL [9] has been extended to CoCasl, so that proofs about CoCasl specifications can now be conducted in a well-developed higher order logical environment. This embedding is the basis for the development of automatic tactics that serve to simplify the actual proof work.

In the same way as proofs about algebraic datatypes typically involve induction, the standard proof method for coalgebraic process types is coinduction. The

coinduction principle states that bisimilar, i.e. observationally indistinguishable states are actually equal. While inductive proofs of simple assertions are usually easy to mechanize, the automatization of coinduction is faced with the problem that standard coinduction requires the invention of a bisimulation relation. A variant of coinduction that lends itself more easily to mechanization is the method of circular coinduction [10], which works by 'reducing the claim to itself' adhering to certain restrictions in the permissible proof steps. Here, we introduce an implementation of a related proof method where the bisimulation is built up inductively from the proof goal. This process may be performed either automatically or, in cases where this fails, semiautomatically, with the inductive construction guided by the user by means of specialized tactics. The inductive completion process has the advantage that specifications are not limited to (conditional) equational logic, as with circular coinduction (as realized in BOBJ).

The use of this method is illustrated by means of example specifications in CoCasl. It turns out that many simple goals can indeed be solved automatically, and that more complicated goals require only a moderate amount of user interaction.

The material is organized as follows. Basic facts and notions concerning coalgebra and coinduction are reviewed in Sect. 1. Section 2 gives a brief introduction to CoCasl. The method of the iterative construction of a bisimulation, called iterative coinduction, is introduced in Sect. 3. Section 4 discusses the implementation of this method in Isabelle/HOL and the example proofs.

1   Coalgebra and Coinduction

We now briefly recall some basic notions from coalgebra.

Definition 1 (coalgebra). Let $\mathbf{C}$ be a category, and let $T : \mathbf{C} \to \mathbf{C}$ be a functor. A T-*coalgebra* (A, α) (or, somewhat imprecisely, just A) consists of an object A of $\mathbf{C}$ and a morphism $\alpha : A \to TA$. A *homomorphism* between two T-coalgebras (A, α) and (B, β) is a morphism $h : A \to B$ such that $\beta \circ h = (Th) \circ \alpha$. A T-coalgebra Z is called *final* if for each T-coalgebra A, there exists a unique homomorphism $A \to Z$.

Final coalgebras admit *corecursive definitions*: given an object A of $\mathbf{C}$, a function $f : A \to Z$ into the final T-coalgebra Z can be defined by exhibiting a T-coalgebra structure α on A. The function $f : A \to Z$ thus defined is then the unique homomorphism $(A, \alpha) \to Z$. Examples of corecursive definitions are given below.

There is also a principle of *coinductive proof* which relies on a coalgebraic notion of bisimulation and is particularly suitable for proving properties of corecursively defined functions.

Definition 2 (bisimulation and full abstraction). Let $\mathbf{C}$ be a category. A *relation* between two objects A and B of $\mathbf{C}$ is a subobject R of $A \times B$; equivalently, R is given by the two projection morphisms $\pi_1 : R \to A$ and $\pi_2 : R \to B$. If

A and B are coalgebras for a functor $T : \mathbf{C} \to \mathbf{C}$, then such a relation R is called a *bisimulation* if there exists a T-coalgebra structure on R that makes π_1 and π_2 into coalgebra homomorphisms. A coalgebra A is called *fully abstract* if every bisimulation on A is contained in the identity relation, i.e. the diagonal $\Delta : A \to A \times A$.

In the special case $\mathbf{C} = \mathbf{Set}$, the notion of relation as defined above coincides with the usual notion. In this case, elements of coalgebras are called *bisimilar* if there exists some bisimulation that relates them. Full abstractness of A means that we have the following *coinduction proof principle* on A:

If x and y are bisimilar elements of A, then $x = y$.

As indicated in the introduction, this proof principle, while indeed essentially dual to induction, carries the disadvantage that a bisimulation R relating x and y must actually be invented. Coinduction is always available on final coalgebras, and hence on their subcoalgebras:

Lemma 3. *Final coalgebras are fully abstract.*

Example 4. Let T be the set functor given by $TX = A \times X$ for a fixed set A. The final T-coalgebra $Z = (A^{\mathbb{N}}, \langle hd, tl \rangle : A^{\mathbb{N}} \to A \times A^{\mathbb{N}})$ has the set $A^{\mathbb{N}}$ of all infinite streams of elements from A as its carrier and the combined head and tail function as its coalgebra structure. We can define corecursive functions $odd, even : A^{\mathbb{N}} \to A^{\mathbb{N}}$ and $zip : A^{\mathbb{N}} \times A^{\mathbb{N}} \to A^{\mathbb{N}}$ by the equations shown in Fig. 1 below (where $A^{\mathbb{N}}$ corresponds to $Stream[Elem]$). In the case of odd, these equations correspond to requiring that odd is a homomorphism $(A^{\mathbb{N}}, \langle hd, tl \circ tl \rangle) \to Z$, i.e. to commutation of the diagrams

similarly for *even* and *zip*. By Lemma 3, the claim that $zip(odd(s), even(s)) = s$ for all $s \in A^{\mathbb{N}}$ can be proved by coinduction as follows. We have to define a bisimulation R which relates $zip(odd(s), even(s))$ and s for all $s \in A^{\mathbb{N}}$. To this end, we put $R = \{(zip(odd(s), even(s)), s) \mid s \in A^{\mathbb{N}}\}$. Showing that R is a bisimulation amounts to proving that $s \, R \, t$ implies $hd(s) = hd(t)$ and $tl(s) \, R \, tl(t)$. The former goal is solved trivially by just applying the definitions. The latter is shown as follows:

$$
\begin{aligned}
tl(zip(odd(s), even(s))) &= zip(even(s), tl(odd(s))) \\
&= zip(even(s), odd(tl(tl(s)))) \\
&= zip(odd(tl(s)), even(tl(s)))) \\
&R \; tl(s),
\end{aligned}
$$

where we have used the lemma

$$even = odd \circ tl.$$

This proof illustrates two difficulties w.r.t. mechanizability: not only did we have to invent the said lemma, we also had to apply this equation in two different directions during the calculation of $tl(zip(odd(s), even(s)))$. This point will be discussed in more detail below.

A further difficulty appears in the following example. Let $bzip : A^{\mathbb{N}} \times A^{\mathbb{N}} \times Bool \rightarrow A^{\mathbb{N}}$, where $Bool$ is the set $\{\top, \bot\}$ of truth values, be corecursively defined by

$$hd(bzip(s, t, b)) = \begin{cases} hd(s) & \text{if } b \\ hd(t) & \text{otherwise} \end{cases}$$

$$tl(bzip(s, t, b)) = \begin{cases} bzip(tl(s), t, \neg b) & \text{if } b \\ bzip(s, tl(t), \neg b) & \text{otherwise.} \end{cases}$$

Then the equation $zip(s, t) = bzip(s, t, \top)$ can be proved by coinduction. However, the initial guess at a bisimulation, $R = \{(zip(s, t), bzip(s, t, \top)) \mid s, t \in A^{\mathbb{N}}\}$, in fact fails to be a bisimulation. A bisimulation is obtained only by the improved guess $R' = R \cup \{(zip(t, s), bzip(s, t, \bot)) \mid s, t \in A^{\mathbb{N}}\}$.

Circular Coinduction

A coinduction proof principle similar to the one described above has also been introduced for behavioral specifications in the framework of hidden algebra. Roşu [10] has noted that coinduction based on behavioral rewriting loops for proof goals like $zip(odd(s), even(s)) = s$. He has therefore introduced *circular coinduction*, a proof rule that avoids looping by stopping whenever a subgoal is reached that is an instance of a proof goal that has already been decomposed using the observers. Circular coinduction has been implemented in the BOBJ system [10]. Our iterative coinduction method introduced below is very similar to circular coinduction, the essential difference being that it is tailored towards integration in a semiautomatic theorem prover like Isabelle (while a direct integration of the circular coinduction rule into Isabelle would actually lead to less automation because true narrowing instead of just rewriting would be needed).

2   CoCasl

The algebraic-coalgebraic specification language CoCasl has been introduced in [7] as an extension of the standard algebraic specification language Casl. For the basic Casl syntax, the reader is referred to [1, 8]. We briefly explain the CoCasl features relevant for the understanding of the present work using the example specification shown in Fig. 1.

Dually to Casl's datatype construct **type**, CoCasl offers a **cotype** construct which defines coalgebraic process types; it is formally proved in [7] that one can indeed define for each cotype signature a functor T such that models of

---

spec STREAM1 [**sort** *Elem*] =
  cofree cotype
      Stream ::= *cons*(*hd* : *Elem*; *tl* : *Stream*)
  ops *odd*, *even* : *Stream*[*Elem*] → *Stream*[*Elem*];
      zip : *Stream*[*Elem*] × *Stream*[*Elem*] → *Stream*[*Elem*];
  vars s, s_1, s_2 : *Stream*[*Elem*];
    • $hd(odd(s)) = hd(s)$
    • $tl(odd(s)) = odd(tl(tl(s)))$
    • $hd(even(s)) = hd(tl(s))$
    • $tl(even(s)) = even(tl(tl(s)))$
    • $hd(zip(s_1, s_2)) = hd(s_1)$
    • $tl(zip(s_1, s_2)) = zip(s_2, tl(s_1))$
then %**implies**
  var s : *Stream*[*Elem*]
    • $zip(odd(s), even(s)) = s$
end

Fig. 1. CoCasl specification of streams

the cotype correspond to T-coalgebras. A simple example is the cotype *Stream* defined in Fig. 1. Like a type declaration, a cotype declaration is just a short way of declaring operations; specifically, the declaration of *Stream* produces two operations $hd : Stream \rightarrow Elem$ and $tl : Stream \rightarrow Stream$. Models of the cotype *Stream* are essentially coalgebras for the functor $\lambda X. Elem \times X$.

Cotypes can be qualified by keywords expressing further constraints. In particular, the keyword **cofree** qualifying the cotype of streams in Fig. 1 has the effect of restricting the models of *Stream* to the final coalgebra (uniquely up to isomorphism), i.e. the set of streams. In particular, one thus has a coinduction principle for *Stream*, which we could also express by using the weaker constraint **cogenerated** instead of **cofree**. Moreover, the corecursive definitions of the functions *odd*, *even*, and *zip* indeed constitute a definitional extension, i.e. do not actually affect the model class.

We now recall some notions from the formal semantics of Casl and CoCasl:

A *many-sorted* Casl *signature* $\Sigma = (S, TF, PF, P)$ consists of a set S of sorts, two $S^* \times S$-indexed sets $TF = (TF_{w,s})$ and $PF = (PF_{w,s})$ of total and partial operation symbols, and an S^*-indexed set $P = (P_w)$ of predicate symbols. Function symbols in $TF_{w,s}$ are written $f : w \rightarrow s$.

Models are many-sorted partial first order structures, interpreting total (partial) function symbols as total (partial) functions and predicate symbols as relations. Homomorphisms between such models are so-called *weak homomorphisms*. That is, they are total as functions, and they preserve (but do not necessarily reflect) the definedness of partial functions and the satisfaction of predicates.

Definition 5 (Σ-cogeneration constraint). Given a signature $\Sigma = (S, TF, PF, P)$, a *cogeneration constraint* $\Theta = (\bar{S}, \bar{F})$ over Σ consists of a set of *observable* sorts $\bar{S} \subset S$ and a set of *observer* operation symbols $\bar{F} \subset TF \cup PF$.

Definition 6 (Observation functional). Let $\Theta = (\bar{S}, \bar{F})$ be a Σ-cogeneration constraint. The *observation functional* Obs_Θ computes the image of a relation under all observers with observable result. Formally, if M is a Σ-model and $R \subset |M| \times |M|$ is an S-sorted binary relation, $Obs_\Theta(R) = \{(f_M(x), f_M(y)) \mid (x, y) \in R, f \in \bar{F}_{w,s}, s \in \bar{S}\}$.

Definition 7 (Transition functional). Let $\Theta = (\bar{S}, \bar{F})$ be a Σ-cogeneration constraint. The *transition functional* $Trans_\Theta$ computes the image of a relation under all observers with non-observable result. Formally, for $R \subset |M| \times |M|$, $Trans_\Theta(R) = \{(f_M(x), f_M(y)) \mid (x, y) \in R, f \in \bar{F}_{w,s}, s \notin \bar{S}\}$.

Definition 8 (Θ-bisimulation). Let M be a Σ-model. A binary relation R on M is called a Θ-*bisimulation* if

$$Obs_\Theta(R) \subset \Delta \text{ and } Trans_\Theta(R) \subset R$$

for the Σ-cogeneration constraint Θ (Δ denotes the identity relation). Two elements of M are called Θ-*bisimilar* if they are in relation for some Θ-bisimulation. The constraint Θ is *satisfied* in a Σ-model M (written $M \vDash \Theta$) if each Θ-bisimulation on M is contained in the equality relation (this model M is then also called *cogenerated* by Θ).

If the cogeneration constraint Θ corresponds to the functor T (cf. [7]), then the notion of a bisimulation for T-coalgebras and the notion of a Θ-bisimulation coincide. The coinduction proof principle from Definition 2 thus takes the following form:

> Let Θ be a Σ-*cogeneration constraint and let* R *be a* Θ-*bisimulation.* Then $(x, y) \in R \Rightarrow x = y$ (*i.e.* $R \subset \Delta$).

Thus it suffices to exhibit a Θ-bisimulation R which relates two elements x and y of a cogenerated model of Σ in order to show that $x = y$. The difficulty is, again, in finding a suitable R.

Remark 9. The satisfaction of cogeneration constraints is defined in [7] in terms of co-congruences rather than in terms of bisimulation relations. For arbitrary functors, the arising coinduction principle is stronger than coinduction principles based on bisimulation, so that the method of coinductive proof described in Sect. 1 remains sound. For the more restricted functors considered here, the two notions are equivalent.

3   Iterative Construction of the Bisimulation

A first approach to the construction of a bisimulation R in coinductive proofs is as follows. Given a proof goal $\forall X. t_1 = t_2$,

1. Let $R = \{(x, y) \mid \exists X.\, x = t_1 \wedge y = t_2\}$ (following [2], we call R the current *trial bisimulation*).
2. Try to prove $Obs_\Theta(R) \subset \Delta$ and $Trans_\Theta(R) \subset R$ (i.e. try to show that R is a Θ-bisimulation).
3. If this succeeds, the proof is finished.

However, this approach will often fail:

Example 10. Consider again the example of infinite streams $A^{\mathbb{N}}$ of elements from A defined using the functor $TX = A \times X$. The corresponding signature is $\Sigma = (\{Elem, Stream\}, \{hd, tl\}, \emptyset, \emptyset)$. The observation functional for the Σ-cogeneration constraint $\Theta = (\{Elem\}, \{hd, tl\})$ is defined as $Obs_\Theta(R) = \{(hd(x), hd(y)) \mid (x, y) \in R\}$, the transition functional is defined as $Trans_\Theta(R) = \{(tl(x), tl(y)) \mid (x, y) \in R\}$. The attempt to prove $\forall s.\, zip(odd(s), even(s)) = s$ from Θ fails if the above algorithm is used:

1. Let $R = \{(x, y) \mid \exists s.\, x = zip(odd(s), even(s)) \wedge y = s\}$
2. Try to prove $Obs_\Theta(R) \subset \Delta$ and $Trans_\Theta(R) \subset R$. In order to prove the first inclusion, we have to prove $\forall s.\, hd(zip(odd(s), even(s))) = hd(s)$, which can be done by just rewriting the left term. For a proof of the second inclusion, one would have to prove $\forall s.\, tl(zip(odd(s), even(s)))\, R\, tl(s)$, which is indeed true. However, $tl(zip(odd(s), even(s))) = zip(even(s), odd(tl(tl(s))))$ and the latter term cannot be simplified any further, so that a proof attempt by mere rewriting fails.

It is hence necessary to use a 'larger' relation which *explicitly* contains $(zip(even(s), odd(tl(tl(s)))), tl(s))$ for all s (there is no harm in the fact that these pairs are indeed already contained in the original relation). A similar situation arises in the proof of the identity $zip(s, t) = bzip(s, t, \top)$ (cf. Example 4), where the original trial bisimulation actually fails to be a bisimulation and hence needs to be properly extended.

A more effective proof method is the iterative extension of the trial bisimulation: First one tries to prove that the current trial bisimulation is a bisimulation, and if this fails, one adds a new pair to the relation and again tries to show that the new relation is a bisimulation. This is done repeatedly until the proof of the fact that the current trial bisimulation is a bisimulation succeeds.

We now present an algorithm called *iterative coinduction* that uses this approach. Assuming the cogeneration constraint Θ, the proof goal $\forall X.\, t_1 = t_2$ is dealt with as follows.

1. Let $R = \{(x, y) \mid \exists X.\, x = t_1 \wedge y = t_2\}$, and let $n = 0$.
2. Let $R_n = R \cup Trans_\Theta(R_{n+1})$, with R_{n+1} a metavariable which can later be instantiated.
3. Try to prove $Obs_\Theta(R_0) \subset \Delta$ and $Trans_\Theta(R_0) \subset R_0$ by instantiating R_{n+1} with $\emptyset$ (i.e. try to prove that R_0 is bisimulation).
4. If this does not succeed, then set n to $n + 1$ and continue with 2.

5. Otherwise conclude by the coinduction proof principle that $\forall X. t_1 = t_2$, since $\forall X. t_1 R_0 t_2$.

Example 11. The proof attempt for $\forall s . zip(odd(s), even(s)) = s$ succeeds if the above algorithm is used:

1. Let $R = \{(x, y) \mid \exists s . x = zip(odd(s), even(s)) \wedge y = s\}$, let $n = 0$.
2. Let $R_0 = R \cup Trans_\Theta(R_1)$, with R_1 a metavariable.
3. Try to prove $Obs_\Theta(R_0) \subset \Delta$ and $Trans_\Theta(R_0) \subset R_0$ by instantiating R_1 with $\emptyset$. As discussed above, the first inclusion can be discharged by rewriting, while a rewriting proof of the second inclusion fails, although the inclusion does hold.
4. Thus set n to 1 and let $R_1 = R \cup Trans_\Theta(R_2)$ with R_2 a metavariable (now $R_0 = R \cup Trans_\Theta(R \cup Trans_\Theta(R_2)))$.
5. Try to prove $Obs_\Theta(R_0) \subset \Delta$ and $Trans_\Theta(R_0) \subset R_0$ by instantiating R_2 with $\emptyset$. Since Obs_Θ and $Trans_\Theta$ distribute over unions, the following proof goals arise:

$$Obs_\Theta(R) \subset \Delta \qquad\qquad Trans_\Theta(R) \subset R \cup Trans_\Theta(R)$$
$$Obs_\Theta(Trans_\Theta(R)) \subset \Delta \qquad Trans_\Theta(Trans_\Theta(R)) \subset R \cup Trans_\Theta(R)$$

The goal for $Obs_\Theta(R)$ was already discharged in step 3, and the goal for $Trans_\Theta(R)$ is trivial. The goal for $Obs_\Theta(Trans_\Theta(R))$ can be discharged by rewriting the left side. In order to establish the last inclusion, we have to prove that for all s, $tl(tl(zip(odd(s), even(s)))) \, R_0 \, tl(tl(s))$. Now

$$
\begin{aligned}
tl(tl(zip(odd(s), even(s)))) &= tl(zip(even(s), odd(tl(tl(s))))) \\
&= zip(odd(tl(tl(s))), even(tl(tl(s)))) \\
&R \, tl(tl(s)),
\end{aligned}
$$

which establishes the last goal.
6. We conclude by coinduction that $\forall s. zip(odd(s), even(s)) = s$.

The method of iterative coinduction is thus able to complete the proof. Furthermore, the method succeeds in proving the theorem $\forall s_1, s_2. zip(s_1, s_2) = bzip(s_1, s_2, \top)$ from Example 4. During the proof, the algorithm adds the pairs $(zip(s_2, tl(s_1)), bzip(tl(s_1), s_2, \bot))$ for all s_1, s_2 to the trial bisimulation $R = \{(zip(s_1, s_2), bzip(s_1, s_2, \top))\}$. These pairs constitute an actual extension of the trial bisimulation, i.e. they were (in contrast to the situation in the proof of $\forall s. zip(even(s), odd(s)) = s$) not previously contained in the relation.

Notice that the above coinduction method is not guaranteed to terminate, i.e. it is possible that the method just keeps adding new pairs to the trial bisimulation. Such looping may have different causes: firstly, of course, if the proof goal is not a consequence of the considered specification, the method will not be able to prove it and hence fail to terminate (however, if the inclusion $Obs_\Theta(R_0) \subset \Delta$ becomes false at some stage, then the method can actually be used to disprove

the incorrect goal). The algorithm may fail to terminate also on correct goals in cases where the iterative construction of a bisimulation requires infinitely many steps (see [2] for examples). Such goals can typically be solved by generalization: A more general proof goal is stated, which one may then, in turn, attempt to solve with the algorithm.

4   Iterative Coinduction in Isabelle/HOL

As part of the Bremen heterogeneous tool set Hets [6, 5], a translation of Co-Casl specifications into Isabelle/HOL theories has been implemented in order to allow for the interactive proving of properties of reactive systems (see e.g. Figure 2). This includes a translation of cogeneration constraints, so that coin-ductive proofs about CoCasl specifications in Isabelle/HOL are made possible in principle. Making coinductive proofs practically feasible requires a set of custom-tailored proof procedures, called *tactics* in Isabelle. Specifically, tactics have been implemented to support the method of iterative coinduction as introduced in the previous section. The *iterative-coinduction* tactic assembles the smaller tactics into one complex tactic which succeeds in proving a relatively large variety of different theorems over different cotypes automatically. In cases where this fails, it is usually possible to construct simple semi-manual proofs by means of the semiautomatic tactics provided by the implementation. The following automatic and semiautomatic tactics have been implemented:

- **The coinduction tactic:** Let the proof goal be $\forall X.\, t_1 = t_2$. This tactic then automatically chooses the basic relation R such that $(x, y) \in R \Leftrightarrow \exists X.\, x = t_1 \wedge y = t_2$. The appropriate cogeneration axiom is applied afterwards while instantiating R_0 with $R \cup Trans(R_1)$ where R_1 is an uninstantiated metavariable. The *coinduction* tactic generates two subgoals: the first subgoal states that R_0 is a bisimulation, and the second states that $(t_1, t_2) \in R_0$.
- **The init tactic:** The *init* tactic automatically proves $(t_1, t_2) \in R_0$ and thus solves the second of the two subgoals generated by the *coinduction* tactic.
- **The breakup tactic:** The *breakup* tactic splits a subgoal of the schematic form $(x, y) \in (R \cup Trans(R_n)) \Rightarrow C$ for some natural number n and formula C up into two subgoals $(x, y) \in R \Rightarrow C$ and $(x, y) \in Trans(R_n) \Rightarrow C$.
- **The close-or-step tactic:** This tactic tries to solve the current subgoal by simplification while speculating that R_n may be chosen as $\emptyset$ (this attempt is also called the *close*-part of the tactic). If this fails, the tactic instantiates R_n with $R \cup Trans(R_{n+1})$ (this is also called the *step*-part of the tactic).
- **The force-finish tactic:** The *force-finish* tactic instantiates R_n with the empty predicate and afterwards applies simplification steps in order to solve the last remaining subgoal.
- **The iterative-coinduction tactic:** This tactic combines the previous five tactics in order to allow for automatic proofs.

4.1   Examples

We will now demonstrate the use of the tactics described above by several example proofs.

Recall the CoCasl specification of streams of type *Elem* as shown in Fig. 1. This specification contains corecursive definitions of functions *odd* and *even*, which given a stream s return the stream of elements of s at odd or even positions, respectively. Furthermore, a function *zip* is defined which merges two streams s_1 and s_2 into a stream which alternatingly contains elements from s_1 and s_2. Finally, the specification contains a theorem (marked as such by the Casl semantic annotation %**implies**) stating that for all streams s, $zip(odd(s), even(s)) = s$.

```
typedecl "Elem"
typedecl "Stream"

consts
"hd" :: "Stream => Elem"
"tl" :: "Stream => Stream"
"even" :: "Stream => Stream"
"zip" :: "Stream => Stream => Stream"
"odd" :: "Stream => Stream"

axioms
odd_hd: "!!s::Stream.(hd(odd s)) = (hd s)"
odd_tl: "!!s::Stream.(tl(odd s)) = odd(tl(tl s))"
even_hd: "!!s::Stream.(hd(even s)) = (hd(tl s))"
even_tl: "!!s::Stream.(tl(even s)) = even(tl(tl s))"
zip_hd: "!!s1::Stream.!!s2::Stream.(hd(zip s1 s2)) = (hd s1)"
zip_tl: "!!s1::Stream.!!s2::Stream.(tl(zip s1 s2)) = zip s2 (tl s1)"

ga_cogenerated_Stream: "!! R :: Stream => Stream => bool.
!! u :: Stream. !! v :: Stream. ! x :: Stream. ! y :: Stream.
R x y --> (hd x = hd y & R (tl x) (tl y)) ==> R u v ==> u = v"

theorem Stream_Zip: "!! s :: Stream . zip (odd s) (even s) = s"
```

Fig. 2. Isabelle translation of the CoCasl specification of streams

Figure 2 shows the automatic translation of this CoCasl specification into an Isabelle theory, generated by Hets. This theory first declares the types Elem| and Stream| together with the observers and additional functions odd|, even| and zip|. This is followed by axioms arising from the coinductive function definitions. Let $\Theta_{Str} = (\{Elem\}, \{hd, tl\})$. The axiom ga_cogenerated_Stream states that every Θ_{Str}-bisimulation R is contained in the equality relation. This axiom constitutes the coinduction proof principle on which the subsequent proofs

```
theorem Stream_Zip: "!! s :: Stream . zip (odd s) (even s) = s"
apply(coinduction)
apply(init)
apply(breakup)
apply(close_or_step)
apply(force_finish)
done

theorem Stream_Zip2: "!! s :: Stream . zip (odd s) (even s) = s"
apply(iterative_coinduction)
done
```

Fig. 3. Two proofs of $zip(odd(s), even(s)) = s$

are based. (The existence part of the finality constraint expressed by the keyword **cofree** is irrelevant for coinductive proofs and presently ignored by the translation.) The theorem $zip(odd(s), even(s)) = s$ is translated as an open goal. Figure 3 shows two proofs of this theorem using the tactics for iterative coinduction. The first proof uses the semiautomatic tactics in order to con-

spec BINTREE1 [**sort** *Elem*] =
    cofree cotype *BinTree* ::= (*left* : *BinTree*; *node* : *Elem*; *right* : *BinTree*)
    op *mirror* : *BinTree*[*Elem*] → *BinTree*[*Elem*];
    vars *t* : *BinTree*[*Elem*];
            • *left*(*mirror*(*t*)) = *mirror*(*right*(*t*))
            • *node*(*mirror*(*t*)) = *node*(*t*)
            • *right*(*mirror*(*t*)) = *mirror*(*left*(*t*))
then %**implies**
    var *t* : *BinTree*[*Elem*]
    •   *mirror*(*mirror*(*t*)) = *t*
end

Fig. 4. CoCasl specification of infinite binary trees

duct the proof step by step. The *coinduction* tactic automatically applies the `ga_cogenerated_Stream` axiom to the current goal, yielding two new subgoals by instantiating the relation variable in the axiom with $R \cup Trans_{\Theta_{Str}}(R_1)$. The first subgoal states that $R_0 = \{(x, y) \mid \exists s :: Stream. x = zip(odd(s), even(s)) \land y = s\} \cup Trans_{\Theta_{Str}}(R_1)$ is a Θ_{Str}-bisimulation; the second subgoal states that R_0 relates $zip(odd(s), even(s))$ and s. The *init* tactic solves this second (trivial) subgoal and transforms the first subgoal into a form to which the *breakup* tactic can be applied.

After the execution of the *breakup* tactic, there are two new subgoals. The first subgoal states that R is mapped into R_0 under hd and tl, i.e. that $hd(x) = hd(y)$ for any $(x, y) \in R$ and that $(tl(x), tl(y)) \in R_0$ for $(x, y) \in R$; the second subgoal makes the corresponding statement for $Trans_{\Theta_{Str}}(R)$ in place of R. The *close-or-step* tactic fails to prove the first subgoal by simplification, and thus applies the *step*-part, instantiating R_1 with $R \cup Trans_{\Theta_{Str}}(R_2)$ and automatically succeeds by assuming $R_2 = \emptyset$ to show that $R_0 = R \cup Trans_{\Theta_{Str}}(R)$ is closed under hd and tl and is hence a bisimulation. The remaining subgoal is trivialized by applying the *force-finish* tactic. The proof is thus finished and can be completed by executing *done*.

The second proof uses the automatic *iterative-coinduction* tactic which combines the smaller tactics and finishes the proof without requiring user interaction.

```
typedecl "BinTree"
typedecl "Elem"

consts
"cons" :: "BinTree => Elem => BinTree => BinTree"
"left" :: "BinTree => BinTree"
"mirror" :: "BinTree => BinTree"
"node" :: "BinTree => Elem"
"right" :: "BinTree => BinTree"

axioms
mirror_left: "!!t::BinTree.(left(mirror t)) = (mirror(right t))"
mirror_node: "!!t::BinTree.(node(mirror t)) = (node t)"
mirror_right: "!!t::BinTree.(right(mirror t)) = (mirror(left t))"

ga_cogenerated_BinTree: "!! R :: BinTree => BinTree => bool.
!! u :: BinTree. !! v :: BinTree. ! x :: BinTree. ! y :: BinTree.
((R x) y) --> (((R (left x)) (left y)) & (node x) = (node y) &
((R (right x)) (right y))) ==> ((R u) v) ==> u = v"

theorem BinTree_Mirror: "!! t :: BinTree. (mirror (mirror t)) = t"
```

Fig. 5. Isabelle translation of the CoCasl specification of infinite binary trees

A CoCasl specification for the cotype of infinite binary trees with nodes labelled in a set *Elem*, together with a corecursively defined function *mirror* which keeps the value of the current node and replaces the left subtree with the mirrored right subtree and the right subtree with the mirrored left subtree, is shown in Fig. 4. Figure 5 contains the corresponding Isabelle theory obtained by automatic translation in Hets. The axiom `ga_cogenerated_BinTree` states that any Θ_{Tree}-bisimulation is contained in the equality relation. The proof goal arising by

translation of the %**implies** part of the CoCasl specification states that *mirror* is self-inverse, i.e. that for all infinite binary trees t, $mirror(mirror(t)) = t$.

Two proofs of this theorem are shown in Fig. 6. The proofs use the tactics in an equivalent way as the proofs in Fig. 3.

```
theorem BinTree_Mirror: "!! t :: BinTree. (mirror (mirror t)) = t"
apply(coinduction)
apply(init)
apply(breakup)
apply(close_or_step)
apply(finish)
done

theorem BinTree_Mirror2: "!! t :: BinTree. (mirror (mirror t)) = t"
apply(iterative_coinduction)
done
```

Fig. 6. Two proofs of $mirror(mirror(t)) = t$

Table 1 shows a selection of theorems which have been proved using the iterative-coinductive proof tactics in a similar manner as in the examples above[1]. The depth of a coinductive proof is the number of iterations required in order to arrive at a bisimulation (including the initial guess). The example goals concerning streams, largely taken from [2], make use of further corecursively defined functions: $swap(a, b)$ is the stream $(a, b, a, b, \ldots)$; $const(a)$ is the stream $(a, a, a, \ldots)$; $iterate(f, a)$ is the stream $(a, f(a), f^2(a), \ldots)$; and $inflist(a, g, f)$ is the stream $(g(a), g(f(a)), g(f^2(a)), \ldots)$. The *bswitch* function interchanges even and odd positions in the stream it receives as its first argument, starting at the first or the second position depending on its boolean second argument. The function *bzip* is defined as in Example 4. Other function names should be self-explanatory.

The proofs of the theorems $zip(s, t) = bzip(s, t, \top)$ and $zip(s, t) = bswitch(zip(t, s), \top)$ are typical examples where the trial bisimulation has to be extended by pairs not previously contained in it; the additional pairs are correctly 'guessed' by the iteration mechanism. As can be seen from Table 1, the proofs presently have to be conducted at the semiautomatic level; however, the proofs do not actually require substantial user interaction, so that further fine-tuning of the *iterative-coinduction* tactic is expected to produce a fully automatic proof of these goals.

The theorems on bitstreams shown in Table 1 mention a function $flop : Bit \to Bit$ which toggles bits, and a function $flip : BitStream \to BitStream$

---

[1] Proof scripts and tactic implementations available under `http://www.informatik.uni-bremen.de/~hausmann/cocasl`

Table 1. Theorems proved by iterative coinduction in Isabelle

Cotype	Theorem	Depth	Automatic
Streams	$zip(even(s), odd(s)) = s$	2	Yes
	$zip3(first(s), second(s), third(s)) = s$	3	Yes
	$zip4(one(s), two(s), three(s), four(s)) = s$	4	Yes
	$zip(const(a), const(b)) = swap(a, b)$	2	Yes
	$zip(s, t) = bzip(s, t, \top)$	2	No
	$zip(even(s), odd(s)) = bswitch(s, \top)$	2	No
	$zip(s, t) = bswitch(zip(t, s), \top)$	2	No
	$odd(zip(s, t)) = s$	1	Yes
	$even(zip(s, t)) = t$	1	Yes
	$iterate(f, f(a)) = map(f, iterate(f, a))$	1	Yes
	$const(a) = odd(swap(a, f))$	1	Yes
	$const(a) = map(identity, const(a))$	1	Yes
	$inflist(a, identity, identity) = const(a)$	1	Yes
	$map(g, iterate(f, a)) = inflist(a, g, f)$	1	Yes
	$map(compose(f, g), l) = map(f, map(g, l))$	1	Yes
	$const(f(a)) = map(f, const(a))$	1	Yes
	$const(a) = even(const(a))$	1	Yes
	$const(a) = iterate(identity, a)$	1	Yes
	$const(a) = swap(a, a)$	1	Yes
BitStreams	$flip(b) = map(flop, b)$	1	No
	$tick = flip(tock)$	1	Yes
	$flip(flip(b)) = b$	1	Yes
NatStreams	$streamadd(s, s) = map(double, s)$	1	Yes
	$streamadd(s, t) = streamadd(s, t)$	1	Yes
Binary Trees	$mirror(mirror(t)) = t$	1	Yes
TreeStreams	$swap(mirror(mirror(t)), t) = const(t)$	2	No

which toggles all bits in a stream; the corecursive definition of $flip$ uses a case distinction over $hd(b)$ in the clause for $hd(flip(b))$, i.e. does not use $flop$. The theorem $flip(b) = map(flop, b)$ for all bit-streams b has to be proved semiautomatically because explicit case distinction needs to be performed in the course of the proof (an approach for further automation of proofs which involve case distinction is described in [4]). Using this theorem in simplification, the goals $tick = flip(tock)$ (where $tick$ and $tock$ are the two alternating bitstreams) and $flip(flip(b)) = b$ can be proved automatically.

Another point where the fully automatic tactic fails is nested coinduction. An example is the theorem $swap(mirror(mirror(t)), t) = const(t)$ for all infinite trees t, where during a coinductive proof over streams, a second coinductive proof – this time over trees – becomes necessary. This requires a semiautomatic proof in which the user explicitly tells the system when to start the second coinductive proof. The iterative coinduction tactics automatically choose the right coinduction principle needed in the current situation.

5 Conclusion

We have proposed a method of coinduction by iterative construction of bisimulations. This method, which postulates only the standard coinduction principle, produces proofs that are similar in spirit to circular induction. As part of the proof support for the algebraic-coalgebraic specification language CoCasl, corresponding proof tactics have been implemented in Isabelle/HOL; iterative coinductive proofs are supported by both an all-out automatic tactic and a set of semiautomatic tactics that allow user-guided initiation, continuation, and completion of the iterative construction.

Compared with circular coinduction as realized in BOBJ [10], our approach is suitable for specifications written in full first-order (and even higher-order) logic, not just conditional equations. Moreover, while the degree of automation that we achieve is comparable to that of BOBJ [10] and CoClam [3], the availability of semiautomatic tactics means that user interaction may help to complete proofs that fail with a completely automatic proof procedure (in particular, missing lemmas appear as open subgoals and can be proved on-the-fly, possibly with another coinduction; cf. the *TreeStreams* example). Last but not least, the realisation of circular coinduction as a proof tactic in Isabelle/HOL means that correctness of the implementation only relies on the rather small and long-tested kernel of Isabelle.

Example proofs have been conducted on CoCasl specifications, automatically translated into Isabelle theories by the Bremen heterogeneuous tool set Hets [5, 6]. Simple proof goals can typically be discharged automatically; typical features that require user interaction are case distinction and nested coinduction. The further automation of case distinction, as in BOBJ, is not expected to cause fundamental difficulties.

Continued work on the CoCasl proof environment includes fine-tuning the automatic proof tactics and extending the implementation (which currently only works for the single-sorted case) to many-sorted coinduction and datatype-valued observers, as well as developing proof support for advanced CoCasl features, in particular CoCasl's modal logic and structured cofree specifications.

Acknowledgements

References

1. M. Bidoit and P. D. Mosses, Casl *user manual*, LNCS, vol. 2900, Springer, 2004.
2. L. Dennis, *Proof planning coinduction*, Ph.D. thesis, Edinburgh University, 1998.

3. L. Dennis, A. Bundy, and I. Green, *Using a generalisation critic to find bisimulations for coinductive proofs*, Automated Deduction, LNAI, vol. 1249, Springer, 1997, pp. 276–290.
4. J. Goguen, K. Lin, and G. Rosu, *Conditional circular coinductive rewriting with case analysis*, WADT 02, LNCS, vol. 2755, Springer, 2003, pp. 216–232.
5. T. Mossakowski, HETCASL *– heterogeneous specification. Language summary*, 2004.
6. _____, *Heterogeneous specification and the heterogeneous tool set*, Habilitation thesis (draft), University of Bremen, 2004.
7. T. Mossakowski, L. Schröder, M. Roggenbach, and H. Reichel, *Algebraic-co-algebraic specification in* CoCASL, J. Logic Algebraic Programming, to appear.
8. P. D. Mosses (ed.), CASL *reference manual*, LNCS, vol. 2960, Springer, 2004.
9. T. Nipkow, L. C. Paulson, and M. Wenzel, *Isabelle/HOL — a proof assistant for higher-order logic*, LNCS, vol. 2283, Springer, 2002.
10. G. Rosu, *Hidden logic*, Ph.D. thesis, University of California at San Diego, 2000.
11. J. Rutten, *Universal coalgebra: A theory of systems*, Theoret. Comput. Sci. **249** (2000), 3–80.