Conditional Circular Coinductive Rewriting with Case Analysis

Joseph Goguen[1], Kai Lin[1], Grigore Roşu[2]

[1] Department of Computer Science & Engineering
University of California at San Diego, USA
[2] Department of Computer Science
University of Illinois at Urbana-Champaign, USA

Abstract: We argue for an algorithmic approach to behavioral proofs, review the hidden algebra approach, develop circular coinductive rewriting for conditional goals, and extend it with case analysis. Some examples are given.

1 Introduction

A natural extension of algebraic specification distinguishes *visible* sorts for data from *hidden* sorts for states, with states *behaviorally equivalent* iff they are indistinguishable under a given set of experiments; we have formalized this as *hidden algebra*, originating in [9] and further developed in [11, 20, 18, 19] and other papers. While standard equational proof techniques like induction are suitable for data, *coinduction* or *context induction* is generally needed for non-trivial behavioral properties, typically requiring extensive human intervention. This is not surprising, since behavioral satisfaction is Π^0_2-hard [5], so that no algorithm can prove [or disprove] all behaviorally true [or false] statements. However, successful technology transfer requires placing less demand on users, as illustrated by the success of model checking. Hence our recent research concerns coinduction algorithms that require no human intervention. The languages we know supporting automated behavioral reasoning are Spike [2], CafeOBJ [6], and BOBJ [10], the first based on context induction, and the other two on forms of coinduction.

The powerful coinduction algorithm now in BOBJ has developed through several stages. The first is to restrict ordinary term rewriting to behavioral rewriting [18], by allowing rules to apply only in certain contexts[1]; this can be used like ordinary rewriting for ordinary equational reasoning, to check behavioral equivalence by computing and comparing behavioral normal forms. Circular coinductive rewriting (CCRW) [10] attempts to prove that a hidden equation holds (in the sense that the two terms cannot be distinguished by any context) by applying behavioral rewriting to both sides, allowing also application of the goal in even more restricted contexts than for rewriting, and generating new goals when rewriting fails to show equivalence; forms of this algorithm have been in BOBJ for more than three years. Conditional circular coinductive rewriting (CCCRW) generalizes CCRW to prove (sets of) conditional equations. Finally, conditional circular coinductive rewriting with case analysis (CCCCRW, or C4RW) adds case analysis, and seems to be the most powerful automated proof technique now available for behavioral equivalence. This paper only discusses details for a simplified version of the C4RW algorithm, showing its correctness by relating its steps to sound inference rules given in Section 3. Some more sophisticated extensions have been implemented in BOBJ, and are briefly sketched in Section 5, but details are left for future papers. These extensions make the algorithm much more powerful in practice, and are needed, for example, in our recent proofs for the alternating bit protocol and the Petersen mutual exclusion algorithm.

[1] Standard equational reasoning is not sound for behavioral satisfaction; see Section 4.

BOBJ's C4RW algorithm takes as input a behavioral specification and a set of hidden sorted conditional equations, and it returns `true`, or `failed`, or else goes into an infinite loop[2]. Here is a simple example, illustrating case analysis in a coinductive proof a conditional equation:

Example 1. Sets with insertion The behavioral theory `SET` has one hidden sort, `Set`, one hidden constant for the empty set, and operations for element membership and insertion. The case definition separates the situation where `X` equals `Y` from that where it does not; this split is applied only when a subterm of the term being reduced matches the pattern, `eq(X,Y)`. BOBJ allows case definitions to be named, reused, and combined with other such definitions.

```
bth SET is sort Set .
  pr NAT .
  op empty : -> Set .
  op _in_ : Nat Set -> Bool  .
  op insert : Nat Set -> Set .
  vars M N : Nat .  var S : Set .
  eq N in empty = false .
  eq N in insert(M, S) = eq(N,M) or N in S .
end
cases CASES for SET is
   vars X Y : Nat .
   context eq(X,Y) .
   case eq X = Y .
   case eq eq(X,Y) = false .
end
cred with CASES  insert(N, S) == S if N in S .
```

BOBJ's C4RW algorithm is called by the `cred` command; notice that the goal here is a conditional equation. An algorithm of [21] first determines that {`in`} is a *cobasis* for `set`-sorted terms, i.e., that two terms are behaviorally equivalent iff they are indistinguishable by experiments with `in`. Next, the condition of the goal is added to the specification as a new equation, with its variables replaced by new constants (see the Condition Elimination rule in Section 3). Then BOBJ attempts to prove the goal by reducing each side to its behavioral normal form and checking for equality; since this fails, the goal is added to the specification as a *circularity*, that can only be applied in a restricted way; then the cobasis is applied to each side, and behavioral rewriting produces

```
    M in eq(M,s) or M in S = M in s,
```

where `n` and `s` are the new constants for `N` and `S`, respectively. After that, case analysis is applied, and since both cases reduce to `true`, the result is proved. All this takes just a few milliseconds, before BOBJ returns `true`. Note that the circularity is not actually used in this proof (but we soon give an example that does use a circularity). □

We have found case analysis essential for larger applications, such as our recent proofs of the alternating bit protocol and the Petersen mutual exclusion algorithm; in addition, we reduced the proof score for liveness a real-time asynchronous data transmission protocol done in CafeOBJ by Futatsugi and Ogata [8], by a factor of about ten. Circularities have also been essential for many non-trivial examples, but here is a simple example, proving an identity that is familiar in functional programming, and also illustrating BOBJ's parameterized module capability:

[2] `failed` may mean that the algorithm could not prove the goal, or that the goal is false, depending on the specification.

Example 2. iter and map Here DATA is defines the interface to STREAM, that is, we consider streams of elements from an arbitrary data structure having some monadic operation f defined on its elements. These streams have the usual head and tail operations, plus _&_, which appends an element to the head of a stream. Its most interesting operations are map and iter, which respectively apply f to all the elements of stream, and create a steam of iterations of f applied to its argument.

```
th DATA is sort Elt .
  op f_ : Elt -> Elt .
end
bth STREAM[X :: DATA] is sort Stream .
  op head_ : Stream -> Elt .
  op tail_ : Stream -> Stream .
  op _&_ : Elt Stream -> Stream .
  op map_ : Stream -> Stream .
  op iter_ : Elt -> Stream .
  var E : Elt . var S : Stream .
  eq head(E & S) = E.
  eq tail(E & S) = S .
  eq head map S = f head S .
  eq tail map S = map tail S .
  eq head iter E = E .
  eq tail iter E = iter f E .
end
cred map iter E == iter f E .
```

The equation to be proved,

```
   map iter E = iter f E
```

often appears in proofs about streams in functional programs. Pure behavioral rewriting fails to prove the goal, so circular coinduction is invoked, with the goal added to the specification in a form that limits its application. The cobasis is determined to consist of head and tail, and so new goals are produced by applying these operations to the original goal. The goal generated by head is directly proved by rewriting, but the goal generated by tail is reduced by behavioral rewriting to

```
   iter f f E = map iter f E
```

By applying the circularity at the top level, the left side reduces to iter f f E, which is the same as the behavioral normal form of the right side of the goal. □

2 Specification and Satisfaction

We first briefly review the basics of algebraic specification, and then review the version of hidden algebra implemented in BOBJ [10], which drops many unnecessary but often accepted restrictions, including that operations cannot multiple hidden arguments, that equations cannot have more than one hidden variable, and that all the operations should preserve behavioral equivalence.

2.1 Preliminaries

The reader is assumed familiar with basic equational logic and algebra. Given an S-sorted signature Σ and an S-indexed set of variables Z, let $T_\Sigma(Z)$ denote the Σ-term algebra over variables in Z. If $V \subseteq S$ then $\Sigma\restriction_V$ is a V-sorted signature consisting of all those operations in Σ defined entirely with sorts in V. We may let $\sigma(X)$ denote the term $\sigma(x_1, ..., x_n)$ when the number of arguments of σ and their order and sorts are not important. If only one argument is important, then to simplify writing we place it at the beginning; for example, $\sigma(t, X)$ is a term having σ as root with only variables as arguments except one, and we do not care which one, which is t. $Der(\Sigma)$ is the derived signature of Σ, which contains all Σ-terms, viewed as operations. If t is a Σ-term and A is a Σ-algebra, then $A_t\colon\ A^{var(t)} \to A$ is the interpretation of t in A, defined as follows: given $\theta\colon var(t) \to A$, let $A_t(\theta)$ be $\theta(t)$, the evaluation of t in A with variables replaced by the values given by θ. If one variable of t, say $\star$, is of special importance, then we may view the evaluation of t in two steps, as $A_t\colon\ A \to (A^{(var(t)-\{\star\})} \to A)$ with the obvious meaning.

2.2 Behavioral Specification and Satisfaction

We generalize the hidden algebra of [9, 11, 19] to include variants such as observational logic [1, 3, 14] and coherent hidden algebra [6, 7]. See [19] for a detailed presentation of many variants, with history, many other concepts, and proofs for some results mentioned. Two important classes of behavioral logic are the *fixed data* and *loose data*, depending on whether or the data universe is assumed fixed (i.e, "built-in"). Due to space limitations, our exposition focuses on the loose data version, but all results also hold for the fixed data version. (But note that validity of case analysis often depends on having a suitable fixed data algebra; for example, the above proof for `SET` requires that sort `Bool` have the usual two truth values.)

Definition 1. *Given disjoint sets V, H called* **visible** *and* **hidden sorts***, a* **hidden (V, H)-signature** *is a many sorted $(V \cup H)$-signature. A* **hidden subsignature of Σ** *is a hidden (V, H)-signature Γ with $\Gamma \subseteq \Sigma$ and $\Gamma\restriction_V = \Sigma\restriction_V$. The* **data signature** *is $\Sigma\restriction_V$, which may be denoted Ω. A visible sorted operation not in Ω is called an* **attribute***, and a hidden sorted operation is called a* **method***.*

Unless otherwise stated, the rest of this paper assumes fixed a hidden signature Σ with a fixed subsignature Γ. Informally, Σ-algebras are universes of possible states of a system, i.e., "black boxes," where one is only concerned with behavior under experiments with operations in Γ, where an experiment is an observation of a system attribute after perturbation; this is formalized below, where the symbol $\star$ is a placeholder (i.e., a variable) for the state being experimented upon.

Definition 2. *A* **Γ-context for sort** *$h \in H$ is a term in $T_\Gamma(\{\star : h\} \cup Z)$ with one occurrence of $\star$, where Z is an infinite set of special variables. A Γ-context with visible result sort is called a* **Γ-experiment***. If c is a context for sort h and $t \in T_{\Sigma,h}$ then $c[t]$ denotes the term obtained from c by substituting t for $\star$; we may also write $c[\star]$ for the context itself.*

Definition 3. *Given a hidden Σ-algebra A with a hidden subsignature Γ, we define for all sorts $s \in V \cup H$ an equivalence relation between elements $a, a' \in A_s$ by $a \equiv_\Sigma^\Gamma a'$ iff $A_c(a)(\theta) = A_c(a')(\theta)$ for all Γ-experiments c and all $(V \cup H)$-sorted maps $\theta\colon var(c) \to A$; we call this relation* **Γ-behavioral equivalence on** *A. We may write $\equiv$ instead of $\equiv_\Sigma^\Gamma$ when Σ and Γ can be inferred from context, and we write $\equiv_\Sigma$ when $\Sigma = \Gamma$. Given an $(V \cup H)$-equivalence $\sim$ on A, an operation σ in $\Sigma_{s_1 \ldots s_n, s}$ is* **congruent**[3] *for $\sim$ iff $A_\sigma(a_1, ..., a_n) \sim A_\sigma(a'_1, ..., a'_n)$ whenever $a_i \sim a'_i$ for $i = 1...n$. An operation σ is* **Γ-behaviorally congruent for** *A iff it is congruent for $\equiv_\Sigma^\Gamma$. We often write just*

[3] This is called "coherent" in [7], where the concept originated.

congruent for behaviorally congruent[4]. *A* **hidden Γ-congruence on** *A is a $(V \cup H)$-equivalence on A which is the identity on visible sorts and for which each operation in Γ is congruent.*

Notice that behavioral equivalence is the identity on visible sorts, because the trivial contexts $\star : v$ are proper experiments for all $v \in V$. The following is the basis for coinduction and other important results, generalizing [11] to operations that have more than one hidden argument or are not behavioral; see [20, 19] for a proof. Since final algebras do not necessarily exist in this setting, existence of a largest hidden Γ-congruence does not depend on them, as in coalgebra [22, 16, 15].

Theorem 1. *Given a hidden subsignature Γ of Σ and a hidden Σ-algebra A, then Γ-behavioral equivalence is the largest hidden Γ-congruence on A.*

Behavioral satisfaction of conditional equations can now be naturally defined in terms of behavioral equivalence:

Definition 4. *A hidden Σ-algebra A* **Γ-behaviorally satisfies** *a Σ-equation $(\forall X)\ t = t'$ `if` $t_1 = t'_1, ..., t_n = t'_n$, say e, iff for each $\theta\colon X \to A$, if $\theta(t_i) \equiv^{\Gamma}_{\Sigma} \theta(t'_i)$ for all $1 \le i \le n$, then $\theta(t) \equiv^{\Gamma}_{\Sigma} \theta(t')$; in this case we write $A \models\!\equiv^{\Gamma}_{\Sigma}\ e$. If E is a set of Σ-equations we then write $A \models\!\equiv^{\Gamma}_{\Sigma}\ E$ when A Γ-behaviorally satisfies each Σ-equation in E. When Σ and/or Γ are clear, we may omit either or both from $\models\!\equiv^{\Gamma}_{\Sigma}$.*

Case analysis is often needed, for example, in our recent proofs of the alternating bit protocol and the Petersen mutual exclusion algorithm. An elegant formulation of case analysis adds a new kind of sentence, which can also be used in specifications:

Definition 5. *Given a hidden signature Σ, a* **Σ-case sentence over variables** *X is a nonempty set $\{C_1, C_2, ..., C_n\}$, written $(\forall X) \bigvee_{i=1}^{n} C_i$, where each C_i for $1 \le i \le n$ is a set of pairs of Σ-terms over variables in X. Given a hidden Σ-algebra A, $A \models\!\equiv (\forall X) \bigvee_{i=1}^{n} C_i$ iff for any $\theta\colon X \to A$ there is some $1 \le i \le n$ such that $\theta(t) \equiv^{\Gamma}_{\Sigma} \theta(t')$ for each $t = t'$ in C_i.*

Definition 6. *A* **behavioral** *(or* **hidden***)* **Σ-specification** *(or* **-theory***) is a triple (Σ, Γ, E) where Σ is a hidden signature, Γ is a hidden subsignature of Σ, and E is a set of Σ-sentences (equations of cases). The operations in $\Gamma - \Sigma\restriction_V$ are called* **behavioral**. *A Σ-algebra A* **behaviorally satisfies** *(or* **is a model of***) a behavioral specification $\mathcal{B} = (\Sigma, \Gamma, E)$ iff $A \models\!\equiv^{\Gamma}_{\Sigma}\ E$, and in this case we write $A \models\!\equiv\ \mathcal{B}$; we write $\mathcal{B} \models\!\equiv\ e$ if $A \models\!\equiv\ \mathcal{B}$ implies $A \models\!\equiv^{\Gamma}_{\Sigma}\ e$. An operation $\sigma \in \Sigma$ is* **behaviorally congruent for** *$\mathcal{B}$ iff σ is Γ-behaviorally congruent for each A such that $A \models\!\equiv\ \mathcal{B}$.*

Many examples of non-Γ-behaviorally congruent operations arise in programming language semantics. For example, if we consider two programs in a given language equivalent iff they both terminate and have same output, then appropriate operations can be defined for Γ to enforce this natural relation of behavioral equivalence. However, to properly define the syntax and the semantics of a programming language, its behavioral specification needs to define various operations which do not preserve this behavioral equivalence, such as its execution environment (two programs may declare a variable x, one instantiate it to 0 and the other to 1, and then never use that variable).

Proposition 1. *If $\mathcal{B} = (\Sigma, \Gamma, E)$ is a behavioral specification, then all operations in Γ and all hidden constants, are behaviorally congruent for $\mathcal{B}$.*

[4] A similar notion is given by Padawitz [17].

Of course, depending on E, other operations may also be congruent. An easy to check criterion for congruence is given in [20], and is further generalized in [4, 21]. As shown in [20], congruent operations can be added to or removed from Γ at our discretion when the equations in E do not have equalities of hidden sort in their conditions, which is often the case.

3 Behavioral Inference

This section introduces five new inference rules that are sound for behavioral equivalence, beyond the usual rules of reflexivity, symmetry, transitivity and substitution; note that they all work on conditional equations. We let $\Vdash\!\!\vdash$ denote the relation we are defining, for deduction from a specification to an equation. Also, if $\mathcal{B}$ is a behavioral specification and Y is a set of variables, we let $\mathcal{B}(Y)$ denote $\mathcal{B}$ with Y adjoined to the signature of $\mathcal{B}$; similarly, if E is a set of equations, we let $\mathcal{B}\{E\}$ denote $\mathcal{B}$ with E adjoined to the equations of $\mathcal{B}$. This notation also allows us to write things like $\mathcal{B}(Y)\{E\}$ and $\mathcal{B}(Y)\{E, E'\}$.

As discussed in the paragraph before Proposition 1, operations are not always congruent. This implies that the congruence rule of equational deduction is not always sound for behavioral reasoning. However, there are important situations where it is sound: a) when applied at the top of a visible term; and b) when applied to behaviorally congruent operations. The reason for the first is that behavioral equivalence is the identity on visible sorts, and for the second is that behaviorally congruent operations preserve the behavioral equivalence relation:

$$\textsf{Congruence}: \begin{cases} a) \quad \dfrac{\mathcal{B} \Vdash\!\!\vdash (\forall X)\ t = t' \ \texttt{if}\ c,\ \ sort(t,t') \in V}{\mathcal{B} \Vdash\!\!\vdash (\forall X, W)\ \sigma(W,t) = \sigma(W,t') \ \texttt{if}\ c,\ \text{ for all } \sigma \in Der(\Sigma)} \\[2ex] b) \quad \dfrac{\mathcal{B} \Vdash\!\!\vdash (\forall X)\ t = t' \ \texttt{if}\ c,\ \ sort(t,t') \in H}{\mathcal{B} \Vdash\!\!\vdash (\forall X, W)\ \delta(\overline{s},t) = \delta(\overline{s},t') \ \texttt{if}\ c,\ \text{ for all } \delta \in \Gamma \text{ and } \overline{s} \in T_\Sigma(W)} \end{cases}$$

where $\overline{s}$ is an appropriate string of Σ-terms over the variables in W. Deduction with this rule plus the usual reflexivity, symmetry and transitivity rules, satisfies an important property given in Proposition 2 below, using the following concepts:

Definition 7. *A Σ-context γ is* **behavioral** *iff all operations on the path to $\star$ in γ are behavioral, and is* **safe** *iff either it is behavioral or there is some behavioral experiment γ' (of visible result) such that $\gamma = \gamma''[\gamma']$ for some appropriate γ''.*

Proposition 2. *If $\mathcal{B} \Vdash\!\!\vdash (\forall X)\ t = t'$ $\texttt{if}$ c then $\mathcal{B} \Vdash\!\!\vdash (\forall X, W)\ \gamma[t] = \gamma[t']$ $\texttt{if}$ c for any appropriate safe Σ-context γ, where W are the variables of γ.*

The *deduction theorem* says that to prove an implication $p \to q$ one can add p to the set of axioms and then prove q. In equational logics, since universal quantifiers bind both the condition and the conclusion of a conditional equation, to make the deduction theorem work one must first unbind the variables in the condition. This is typically done via the "theorem of constants," which adds a new constant to the signature for each variable of interest. Here is a behavioral rule combining these two,

$$\textsf{Condition Elimination:}\ \frac{\mathcal{B}(Y)\{E(c)\} \Vdash\!\!\vdash (\forall X - Y)\ t = t'}{\mathcal{B} \Vdash\!\!\vdash (\forall X)\ t = t' \ \texttt{if}\ c}$$

where Y is the set of variables occurring in c, and $E(c)$ is the set of ground unconditional equations contained in c (arising since c is a conjunction of equalities). In the lower part of the rule above, t, t' and c are all defined over the signature of $\mathcal{B}$ and use variables in X, while in the upper part, t and t' still use variables in X but all their variables in Y are replaced by new constants, thus giving a new behavioral specification $\mathcal{B}(Y)$, where each variable in Y is regarded as a new constant.

A case sentence can be used to derive new equations by providing a substitution from the case statement's variables into terms over the equation's variables. Formally, let $\mathcal{B}$ be a behavioral specification containing the case statement $(\forall Y) \bigvee_{i=1}^{n} C_i$, let φ be a map $Y \to T_{\Sigma}(X)$, and let $V_i = \mathit{var}(\varphi(C_i))$; then the rule is

$$\textsf{Case Analysis: } \frac{\mathcal{B}(V_i)\{(\forall\emptyset)\varphi(C_i)\} \ \Vvdash (\forall X - V_i)\ t = t' \texttt{ if } c, \text{for } 1 \leq i \leq n}{\mathcal{B} \ \Vvdash (\forall X)\ t = t' \texttt{ if } c}$$

This says that to prove $(\forall X)\ t = t' \texttt{ if } c$ by case analysis using $(\forall Y) \bigvee_{i=1}^{n} C_i$, one must provide a substitution instantiating the case statement to the context of the proof task and then, for each case C_i, prove the equational sentence using that case as an axiom, after the relationship between the case's variables and the equation's variables is made explicit by replacing them by constants.

Unrestricted use of the Case Analysis rule can be very expensive, even non-terminating, and it can also be hard to find appropriate substitutions φ. Since our main goal is an automatic and relatively efficient procedure for proving behavioral equivalence, we have developed a simple mechanism to tell BOBJ both when to perform a case analysis and with what substitution. In BOBJ, each case statement comes with a *pattern*, usually denoted by p, which is just a Σ-term with variables. The Case Analysis rule is enabled only if the pattern p matches a subterms of t or t', and then a substitution also comes for free.

Our most powerful rule is *circular coinduction*, but first we recall the important notion of *cobasis*, originating in [20] and later simplified in [18, 12, 21]. For this paper, a cobasis Δ is considered a subset of Γ that generates enough experiments, in the sense that no other experiment can distinguish two states that cannot be distinguished by these experiments. Finding a minimal cobasis seems to be undecidable, but there are *cobasis criteria* that work well in practice [20, 4, 21], and are implemented in BOBJ. In addition, users can also declare their own cobases.

Intuition for circular coinduction can be enhanced by considering its duality with structural induction. Inductive proofs show equality of terms $t(x), t'(x)$ over a given variable x (seen as a constant) by showing $t(\sigma(x))$ equals $t'(\sigma(x))$ for all σ in a basis, while circular coinduction shows terms t, t' behaviorally equivalent by showing equivalence of $\sigma(t)$ and $\sigma(t')$ for all σ in a cobasis. Moreover, coinduction applies cobasis operations at the top, while structural induction applies basis operations at the bottom. Both induction and circular coinduction assume some "frozen" instances of t, t' equal when checking the inductive/coinductive step: for induction, the terms are frozen at the bottom by replacing the induction variable by a constant, so that no other terms can be placed beneath the induction variable, while for coinduction, the terms are frozen at the top, so that they cannot be used as subterms of other terms (with some important but subtle exceptions, which are treated by the Special Context inference rule below).

Freezing terms at the top is elegantly handled by a simple trick. Suppose every specification has a special visible sort b, and for each (hidden or visible) sort s in the specification, a special congruent operation $[\_] \colon\ s \to b$. No equations are assumed for these operations and no user defined sentence can refer to them; they are there for technical reasons. Thus, with the inference rules introduced so far, for any behavioral specification $\mathcal{B}$ and any conditional equation $(\forall X)\ t = t' \texttt{ if } c$, it is necessarily the case that $\mathcal{B} \ \Vvdash (\forall X)\ t = t' \texttt{ if } c$ if and only if $\mathcal{B} \ \Vvdash (\forall X)\ [t] = [t'] \texttt{ if } c$. The circular coinduction rule preserves this property. Let Δ be a cobasis for $\mathcal{B}$, and assume that the sort of t and t' is hidden.

$$\textsf{Circular Coinduction}: \frac{\begin{array}{c}\mathcal{B}\{(\forall X)\ [t] = [t']\ \mathtt{if}\ c\} \Vdash (\forall X, W)\ [\delta(t, W)] = [\delta(t', W)]\ \mathtt{if}\ c,\\ \text{for all appropriate } \delta \in \Delta\end{array}}{\mathcal{B} \Vdash (\forall X)\ t = t'\ \mathtt{if}\ c}$$

We call the equation $(\forall X)\ [t] = [t']$ $\mathtt{if}\ c$ added to $\mathcal{B}$ a **circularity**; it could just as well have been called a coinduction hypothesis or a co-hypothesis, but we find the first name more intuitive because, from a coalgebraic point of view, coinduction is all about finding circularities.

Another way to look at the circular coinduction rule is through the lense of context induction [13]. To clarify the discussion in this paragraph, we replace the operations $[\star]$ by $C[\star]$. Then our rule says that to show $(\forall X)\ t = t'$ $\mathtt{if}\ c$, one can assume $(\forall X)\ C[t] = C[t']$ $\mathtt{if}\ c$ and then show $(\forall X, W)\ C[\delta(t, W)] = C[\delta(t'], W)$ $\mathtt{if}\ c$ for each $\delta \in \Delta$, which, if one thinks of $C[\delta(\star, W)$ as $(C; \delta)(\star, W)$, is just an induction scheme on contexts. In fact, this is how we prove the soundness of this rule (see Theorem 2 below).

The restriction of the application of circularities to the top of proof goals enforced by the operations $[\star]$ excludes many important situations (e.g., see Example 5). We begin our consideration of when this restriction can be lifted with the following:

Definition 8. *A Γ–Δ context $\gamma[\star]$ is called* **special** *iff for any Δ–experiment $C[\star]$, there exists a Δ–experiment $D[\star]$ such that $C[\gamma[\star]] = D[\star]$ and the size of $D[\star]$ is not bigger than the size of $C[\star]$.*

The next rule allows circularities to be used inside of special contexts:

$$\textsf{Special Context}: \frac{\mathcal{B} \Vdash (\forall X)\ [t] = [t']\ \mathtt{if}\ c}{\mathcal{B} \Vdash (\forall X, W)\ [\gamma(t, W)] = [\gamma(t', W)]\ \mathtt{if}\ c,\ \text{when } \gamma \text{ is a special context}}$$

We now describe two kinds of special contexts, but will consider other more general kinds in future publications.

Definition 9. *An operation f in Γ–Δ is* **context collapsed** *iff for any Δ–experiment $C[\star]$, one of the following two conditions holds:*

1. *there exists an attribute g in Δ and a context $D[\star]$ such that $C[\,f(W)\,] = D[\,g(W)\,]$ and $D[\star]$ does not involve hidden sorts.*
2. *$C[\,f(W)\,] = t$ where t does not involve hidden sorts.*

It is not hard to see that f is context collapsed if both of the following are satisfied:

1. For any attribute g in Δ and any variable x of hidden sort, $g(f(x, V), W) = t$, where t contains no hidden sorts, or $t = D[\,g'(x, W)\,]$ where g' is another attribute in Δ and $D[\star]$ does not involve hidden sorts.
2. For any non-attribute operation g in Δ and any variable x on a hidden sort, $g(f(x, V), W) = x$, or $g(f(x, V), W) = C[\,x\,]$ where C is a context made by context collapsed operations.

Definition 10. *An operation f in Γ–Δ is* **context preserved** *iff both of the following are satisfied:*

1. *For any attribute g in Δ and any variable x of hidden sort, $g(f(x, V), W) = t$ where t involves no hidden sorts, or $t = D[\,g'(x, W)\,]$ where g' is an attribute in Δ and $D[\star]$ involves no hidden sorts.*

2. *For any non-attribute operation g in Δ and any variable x on a hidden sort,*

$$\begin{array}{ll} g(f(x,V),W) = C[\,f(g'(x,V,W),V,W)] & \textit{or} \\ g(f(x,V),W) = C[f(x,V,W)] & \textit{or} \\ g(f(x,V),W) = C[g'(x,V,W)] & \textit{or} \\ g(f(x,V),W) = x & \end{array}$$

where $C[\star]$ is a context of context collapsed operations and g' is in Δ.

A context consisting of context collapsed operation and context preserved operations obviously satisfies the condition in the Special Context rule, so that circularities can be applied under it.

One should be extremely careful in checking that contexts are special. For example, the cobasis operations in Δ cannot be special, because otherwise the Circular Coinduction rule would prove everything. As shown by Example 3, even behavioral operations in $\Gamma - \Delta$ cannot all be special. The examples and theory in this paper show that the issues involved are more subtle than we realized in [18], which described an overly optimistic algorithm. The following is the main result of this paper; its soundness assertions are used to justify the steps of the C4RW algorithm.

Theorem 2. *The usual equational inference rules,* Reflexivity, Symmetry, Transitivity, *and* Substitution*, as well as the new rules above,* Congruence, Condition Elimination, Case Analysis, Circular Coinduction *and* Special Context*, are all sound. By soundness here we mean that if $\mathcal{B} \Vdash (\forall X)\ t = t'$ `if` c and $sort(t,t') \neq b$, or if $\mathcal{B} \Vdash (\forall X)\ [t] = [t']$ `if` c, then $\mathcal{B} \models (\forall X)\ t = t'$ `if` c.*

4 Behavioral Rewriting

Behavioral rewriting relates to the first five rules of Theorem 2 as ordinary term rewriting relates to equational logic: it provides a simple, efficient and automatic procedure for checking equalities. To simplify the exposition, we treat only unconditional rewriting, but the generalization to conditional rewriting is similar to that for ordinary term rewriting. The fact that some operations may not be behaviorally congruent requires modifying ordinary term rewriting.

Definition 11. *A* **Σ-rewrite rule** *is a triple $(\forall Y)\ l \to r$, where $l, r \in T_\Sigma(Y)$. A* **behavioral Σ-rewriting system** *is a triple (Σ, Γ, R), where Σ is a hidden signature, Γ is a hidden subsignature of Σ, and R is a set of Σ-rewrite rules.*

Definition 12. *The* **behavioral (term) rewriting relation** *associated to a behavioral rewriting system $\mathcal{R}$ is the smallest relation $\Rrightarrow$ such that:*

1. *$\theta(l) \Rrightarrow \theta(r)$ for each $(\forall Y)\ l \to r$ in $\mathcal{R}$ and $\theta \colon Y \to T_\Sigma(X)$.*
2. *if $t \Rrightarrow t'$ and $sort(t,t') \in V$ then $\sigma(W,t) \Rrightarrow \sigma(W,t')$ for all $\sigma \in Der(\Sigma)$ and all appropriate variables W, and*
3. *if $t \Rrightarrow t'$ and $sort(t,t') \in H$ then $\delta(W,t) \Rrightarrow \delta(W,t')$ for all $\delta \in \Gamma$ and all appropriate variables W.*

When $\mathcal{R}$ is important, we write $\Rrightarrow_{\mathcal{R}}$ instead of $\Rrightarrow$.

Behavioral rewriting only applies a rule to a hidden redex if only behavioral[5] operations occur on the path from that redex towards the root until a visible sort is found; if no

[5] We suggest declaring as many operations as possible behavioral; in particular, all congruent operations should be behavioral [20]. Those who don't like this may substitute "behavioral or congruent" for "behavioral" through the rest of this paper.

visible operation is found, rewriting is still allowed if all operations on the path from the redex to the root are behavioral. We can formulate an equivalent definition of behavioral rewriting with the following:

Proposition 3. *$t \Rrightarrow t'$ iff there is a rewrite rule $(\forall Y)\ l \to r$ in R, a safe Σ-context γ, and a substitution θ such that $t = \gamma[\theta(l)]$ and $t' = \gamma[\theta(r)]$.*

Behavioral rewriting is implemented in CafeOBJ [6] and BOBJ [10]. Confluence and termination of behavioral rewriting are interesting subjects for research, but we do not focus on them here, except to notice that termination of ordinary rewriting produces termination of behavioral rewriting, because $\Rrightarrow$ is a subrelation of the usual term rewriting relation, so that any termination criterion for ordinary rewriting applies to behavioral rewriting. Many classical results generalize:

Proposition 4. *If $\mathcal{R} = (\Sigma, \Gamma, R)$ is a behavioral Σ-rewriting system and $\mathcal{B} = (\Sigma, \Gamma, E)$ is its associated behavioral specification, i.e., if $E = \{(\forall Y)\ l = r \mid (\forall Y)\ l \to r \in R\}$, and if $\Rrightarrow$ and $\equiv_{Eq}$ are the behavioral rewriting and equational derivability (using the first five rules in Theorem 2) relations on $\mathcal{R}$ and $\mathcal{B}$, respectively, then*

1. *$\Rrightarrow\ \subseteq\ \equiv_{Eq}$,*
2. *If $\Rrightarrow$ is confluent then $\equiv_{Eq}\ =\ \overset{*}{\Rrightarrow};\overset{*}{\Lleftarrow}$, and*
3. *If $\Rrightarrow$ is canonical then $t \equiv_{Eq} t'$ iff $bnf_{\mathcal{B}}(t) = bnf_{\mathcal{B}}(t')$, where $bnf_{\mathcal{B}}(u)$ is the behavioral normal form of a Σ-term u.*

We now extend behavioral rewriting to take account of special contexts, for use in the algorithm of the next section:

Definition 13. *$\Rrightarrow^{\sharp}$ is defined for behavioral rewriting systems extended with the special sorts b and operations $[\_]$, by extending $\Rrightarrow$ minimally such that if $[t] \Rrightarrow^{\sharp} [t']$ then $[\gamma(t, W)] \Rrightarrow^{\sharp} [\gamma(t', W)]$ for each special context (see Definition 8). Given a behavioral rewriting system $\mathcal{R}$ with its associated behavioral specification $\mathcal{B}$, we let $bnf^{\sharp}_{\mathcal{R}}(t)$ and/or $bnf^{\sharp}_{\mathcal{R}}(t')$ denote the normal form of a term t under the rewriting relation $\Rrightarrow^{\sharp}$.*

Soundness of $\Rrightarrow^{\sharp}$ follows from Proposition 4 and soundness of the Special Context rule (Theorem 2, which also says what we mean by soundness in the context of the special sort b and operations $[\_]$).

5 The C4RW Algorithm

A simple way to automate behavioral reasoning is just to behaviorally rewrite the two terms to normal forms, and then compare them, as suggested by Proposition 4. Although this is too weak to prove most interesting properties, the C4RW algorithm combines it with circular coinduction and case analysis in a harmonious and automatic way, for proving hidden properties, which are usually the most interesting and difficult. Intuitively, to prove a hidden conditional equation $(\forall X)\ t = t'$ `if` c, one applies the circular coinduction rule and tries to prove the resulting apparently more complex properties. The algorithm maintains a list of goals, which is reduced when a goal is proved, and is increased when new goals are generated by the coinduction or case analysis rules. The algorithm terminates when either a visible proof task cannot be proved, in which case failed is returned, or when the set of goals becomes empty, in which case true is returned. The proof goals are stored in bracketed form to control their application. We first describe the main procedure, C4RW, which has a set, $\mathcal{G}$, of hidden equations with visible conditions as initial goals.

The loop at step 1 processes each goal in $\mathcal{G}$, and terminates when $\mathcal{G}$ becomes empty or when a failed is returned at step 9. Step 2 removes goals from $\mathcal{G}$, and step 3 puts them (in frozen or bracketed form) into the original specification. These frozen versions of goals can then be used in subsequent proofs, much as induction hypotheses are used in inductive proofs by induction, but "at the top" instead of "at the bottom" (see the discussion in Section 3).

Steps 4 and 5 prepare for applying the Circular Coinduction rule. Since it generates new conditional proof obligations, each with the same condition, and since all these will later be subject to Condition Elimination, for efficiency step 4 first generates new constants for the variables in the condition, and then step 5 calculates the set of ground unconditional equations that will later be added to the specification by Condition Elimination. Steps 6–11 apply the Circular Coinduction rule. For each appropriate operation δ in the cobasis Δ, step 7 tries to prove that $[\delta(t, W)]$ equals $[\delta(t', W)]$, by first applying the Condition Elimination rule ($\mathcal{B}\{E_{\theta(c)}\}$), then using behavioral rewriting on both terms, and finally checking equality with the procedure ProveEq, which is explained below. Notice that behavioral rewriting can use the frozen equation; more precisely, the frozen equation is applied (as a rewrite rule) if the term $[\delta(t, W)]$ reduces to an instance (via a substitution) of $[t]$ (of course, if the condition holds). This is equivalent to saying that the equation $(\forall X)\ t = t'$ `if` c can only be applied on the top when reducing the terms $\delta(t, W)$ generated by circular coinduction. If the procedure ProveEq does not return true, meaning that it was not able to prove the two b-sorted terms equal, then the algorithm returns failed if the cobasis operation was visible (step 8), or else it adds a new (hidden) goal to $\mathcal{G}$, as required by the Circular Coinduction rule.

```
procedure C4RW($\mathcal{B}, \Delta, \mathcal{G}$)
(can modify its $\mathcal{B}$ and $\mathcal{G}$ arguments)
INPUT:   - a behavioral theory $\mathcal{B} = (\Sigma, \Gamma, E)$
         - a cobasis $\Delta$ of $\mathcal{B}$
         - a set $\mathcal{G}$ of hidden $\Sigma$-equations of visible conditions (in bracket form)
OUTPUT: true if a proof of $\mathcal{B} \models\!\equiv \mathcal{G}$ is found; otherwise failed or non-terminating
 1. while there is some $e := (\forall X)\ [t] = [t']$ if $c$ in $\mathcal{G}$ do
 2.     let $\mathcal{G}$ be $\mathcal{G} - \{e\}$
 3.     let $\mathcal{B}$ be $\mathcal{B}\{e\}$
 4.     let $\theta$ be a substitution on $X$ assigning new constants to the variables in $c$;
        add the new constants to $\mathcal{B}$
 5.     let $E_{\theta(c)}$ be the set of visible ground equations in $\theta(c)$
 6.     for each $\delta \in \Delta$ appropriate for $e$ do
 7.         if ProveEq($\mathcal{B}$, $bnf^{\sharp}_{\mathcal{B}\{E_{\theta(c)}\}}([\delta(t, W)])$, $bnf^{\sharp}_{\mathcal{B}\{E_{\theta(c)}\}}([\delta(t', W)])$, $\theta(c)$, $E_{\theta(c)}$) $\neq$ true
 8.         then if $\delta$ is an attribute
 9.             then return failed
10.             else let $\mathcal{G}$ be $\mathcal{G} \cup \{(\forall X, W)\ bnf^{\sharp}_{\mathcal{B}}([\delta(t, W)]) = bnf^{\sharp}_{\mathcal{B}}([\delta(t', W)])$ if $c\}$
11.     endfor
12. endwhile
13. return true
```

We next discuss the procedure ProveEq, which takes as arguments a behavioral specification, two b-sorted terms, u, u', and the ground version of the equation's condition (with its variables replaced by new constants) together with the set of ground equations it generates; it returns **true**, or **failed**, or loops forever. Step 1 returns true if the two terms are equal, and steps 2–6 check whether any case statement in $\mathcal{B}$ can be applied. Remember that the Case Analysis rule requires a substitution of the variables in the case statement into terms over the variables of the equation to derive, and that we use a pattern in BOBJ which automatically selects a substitution, τ (step 3). Step 4 checks whether the case analysis L can show the two terms equivalent, using the procedure CaseAnalysis described below. If no case sentence can show the terms u, u' equivalent, then step 7 returns failed.

```
procedure ProveEq($\mathcal{B}, u, u', \theta(c), E_{\theta(c)}$)
INPUT:  - a behavioral theory $\mathcal{B}$
        - two terms $u$ and $u'$ of visible sort $b$
        - a ground visible condition $\theta(c)$ and its ground equations $E_{\theta(c)}$
OUTPUT: true if a proof of $\mathcal{B} \models\!\!\equiv (\forall var(u,u'))\ u = u'$ if $\theta(c)$ is found;
        otherwise failed or non-terminating
1. if $u = u'$ then return true
2. for each case sentence $(p, L)$ in $\mathcal{B}$ do
3.     if $p$ matches a subterm of $u$ or $u'$ with substitution $\tau$
4.     then if CaseAnalysis($L, \tau, \mathcal{B}, u, u', \theta(c), E_{\theta(c)}$)
5.          then return true
6. endfor
7. return failed
```

The CaseAnalysis procedure just applies the Case Analysis rule. For each case C, it first adds a new constant for each variable in C (step 2) and generates the ground equations of C (step 3). Steps 4–5 check the top derivation in the CaseAnalysis rule: step 4 checks whether the condition of the equational sentence became false (to keep the presentation short we have not introduced an inference rule for false conditions), and if this is not the case, then step 5 recursively checks whether u and u' became equal under the new assumptions)since this recursion may not terminate, some care may be required when defining case statements).

```
procedure CaseAnalysis($L, \tau, \mathcal{B}, u, u', \theta(c), E_{\theta(c)}$)
1. for each case $C$ in $L$ do
2.     let $\eta$ be $\tau$ with a new constant substituted for each variable in $C$
3.     let $E_{\eta(C)}$ be the set of visible ground equations in $\eta(C)$
4.     if $bnf_{\mathcal{B}\{E_{\eta(C)}\}}(\theta(c)) \neq$ false and
5.        ProveEq($\mathcal{B}\{E_{\eta(C)}\}, bnf^{\sharp}_{\mathcal{B}\{E_{\eta(C)}, E_{\theta(c)}\}}(u), bnf^{\sharp}_{\mathcal{B}\{E_{\eta(C)}, E_{\theta(c)}\}}(u'), \theta(c), E_{\theta(c)}$) $\neq$ true
6.     then return failed
7. endfor
8. return true
```

To take full advantage of behavioral rewriting, one must carefully orient the new equations added at step 10 as rewrite rules; the success of c4rw depends on how well this is done. The following orientation procedure has worked very well in practice: If both directions are valid as rewrite rules (i.e., both sides have the same variables), then orient so that the left side has less symbols than the right side; if the terms have the same number of symbols, then orient with right side smaller than the left side under the lexicographic path ordering induced by the order in which operations are declared. The following is a more precise description:

1. if only one direction is valid, then use it;
2. if both directions are valid, but one of $\delta(t, W)$ and $\delta(t', W)$, say t_1 has more symbols than the other, say t_2, then add the rule $[t_1] \to [t_2]$ `if` c to $\mathcal{G}$;
3. if both directions are valid and have the same number of symbols, but $t_1 \gg t_2$, then add $[t_1] \to [t_2]$ `if` c to $\mathcal{G}$, where $\gg$ is the lexicographic path ordering induced by the operation declaration ordering, defined by

$f(t_1, ..., t_n) \gg t_i$ for all $1 \le i \le n$;
$f(t_1, ..., t_n) \gg f(s_1, ..., s_n)$ if $\langle t_1, ..., t_n \rangle \gg \langle s_1, ..., s_n \rangle$ in lexicographic order;
$f(t_1, ..., t_n) \gg g(s_1, ..., s_m)$ if $f > g$ and $f(t_1, ..., t_n) \gg s_i$ for all $1 \le i \le m$.

Example 3. An Invalid Coinduction This shows how the unrestricted use of circularities can give rise to incorrect results. Notice that `odd` is a congruent operation not in the cobasis, which for streams, consists of just `head` and `tail`.

```
bth FOO is pr STREAM[NAT] .
  op odd_ : Stream -> Stream .
  var S : Stream .
  eq head odd S = head S .
  eq tail odd S = odd tail tail S .
  ops a b : -> Stream .
  eq head a = head b .
  eq tail a = odd a .
  eq tail tail b = odd b .
end
set cred trace on .
cred odd b == a .
```

The "proof" goes as follows, using the cobasis {`head`, `tail`} as usual:
```
     head odd b == head a
```
follows by behavioral reduction, and
```
     tail odd b == tail a
```
reduces to
```
     odd odd b == odd a
```
which follows (illegitimately) by applying the circularity inside the context `odd`. To show that the rresult really is false, one may take `a` to be the stream which begins with 001 and then has all 2s, and `b` to be the sequence which also begins with 001, and then continues with all 0s. (Of course, BOBJ's C4RW algorithm also fails to prove it; in fact, it goes into an infinite loop during behavioral rewriting when given this input.) □

Example 4. Two definitions of Fibonacci, plus Evenness Here is a not so trivial example. The goal of the first invocation of C4RW via `cred` is to show equality of two different definitions of the stream of all Fibonacci numbers; for this, the algorithm generates an unending stream of new circularities, thus illustrating how C4RW itself can fail to terminate. The `zip` function interleaves two streams. The second goal involves both a conditional goal and a circularity, and it succeeds.

```
bth 2FIBO is pr STREAM[NAT] .
  ops fib fib' : Nat Nat -> Stream .
  vars N N' : Nat . vars S S' : Stream .
  eq head fib(N, N') = N .
  eq tail fib(N, N') = fib(N', N + N') .
  eq head fib'(N, N') = N .
  eq head tail fib'(N, N') = N' .
  op zip : Stream Stream -> Stream .
  eq head zip(S, S') = head S .
  eq tail zip(S, S') = zip(S', tail S) .
  op add_ : Stream -> Stream .
  eq tail tail fib'(N, N') = add zip(fib'(N, N'), tail fib'(N, N')).
  eq head add S = head S + head tail S .
  eq tail add S = add tail tail S .
end
```

```
  set cred trace on .
  cred fib(N, N') == fib'(N, N') .
  bth EVENNESS is pr 2FIBO + STREAM[BOOL] * (sort Stream to BStream) .
    op all-true : -> BStream .
    eq head all-true = true .
    eq tail all-true = all-true .
    op even?_ : Nat -> Bool .
    op even?_ : Stream -> BStream .
    vars M N : Nat . var S : Stream .
    eq even? 0 = true .
    eq even? s 0 = false .
    eq even? s s N = even? N .
    eq head even? S = even? head S .
    eq tail even? S = even? tail S .
    eq even?(M + N) = true if even?(M) and even?(N) .
  end
  cred even? fib(M, N) == all-true if even?(M) and even?(N) .
```

The last equation is really a lemma that should be proved by induction; it is needed in the proof that if `fib` is given two even arguments, then all its values are even. □

Example 5. Two definitions for iteration The goal here is to prove the equivalence of two ways to produce a stream of increasingly many copies of a function applied to an element.

```
bth MAP-ITER [X :: DATA] is pr STREAM[X] .
  ops (iter1_) (iter2_) : Elt -> Stream .
  var E : Elt . var S : Stream .
  eq head iter1 E = E .
  eq tail iter1 E = iter1 f E .
  eq head iter2 E = E .
  eq tail iter2 E = map iter2(E) .
end
cred iter1 E == iter2 E .
```

The C4RW algorithm generates two circularities, one of which is applied in a special context, i.e., not at the top, in fact, under `map`. Hence this is an example that actually requires the use of special contexts. □

6 Conclusions and Future Research

We believe that the C4RW algorithm, especially with its use of special contexts, is the most powerful algorithm now available for proving behavioral properties of complex systems. However, much can be done to improve it. First, the conditions for contexts to be special in Section 3 are only the beginning of what could be a long journey, parallel to the one followed in research on automatic induction algorithms. In fact, it would probably be useful to combine the C4RW algorithm with some automatic induction methods. In any case, we will consider more powerful conditions for special contexts in future publications.

Another topic that seems worth exploring is adding conditions to case statements; the idea is that after the pattern is matched, the case split would only be applied if the condition is satisfied. This could make the application of case splits more precise, as well as reduce the computation needed for some large examples.

Finally, more should be done on the duality between induction and circular coinduction. In particular, since we are talking about sophisticated algorithms that generate new hypotheses, not just about basic forms of induction and coinduction, the very notion of duality may need some careful explication.

References

1. G. Bernot, M. Bidoit, and T. Knapik. Observational specifications and the indistinguishability assumption. *Theoretical Computer Science*, 139:275–314, 1995.
2. N. Berregeb, A. Bouhoula, and M. Rusinowitch. Observational proofs with critical contexts. In *Fundamental Approaches to Software Engineering*, volume 1382 of *LNCS*, pages 38–53. Springer, 1998.
3. M. Bidoit and R. Hennicker. Behavioral theories and the proof of behavioral properties. *Theoretical Computer Science*, 165(1):3–55, 1996.
4. M. Bidoit and R. Hennicker. Observer complete definitions are behaviourally coherent. In K. Futatsugi, J. Goguen, and J. Meseguer, editors, *OBJ/CafeOBJ/Maude at Formal Methods '99*, pages 83–94. Theta, 1999.
5. S. Buss and G. Roşu. Incompleteness of behavioral logics. In H. Reichel, editor, *Proceedings of Coalgebraic Methods in Computer Science*, volume 33 of *Electronic Notes in Theoretical Computer Science*, pages 61–79. Elsevier Science, 2000.
6. R. Diaconescu and K. Futatsugi. *CafeOBJ Report: The Language, Proof Techniques, and Methodologies for Object-Oriented Algebraic Specification*. World Scientific, 1998. AMAST Series in Computing, volume 6.
7. R. Diaconescu and K. Futatsugi. Behavioral coherence in object-oriented algebraic specification. *Journal of Universal Computer Science*, 6(1):74–96, 2000.
8. K. Futatsugi and K. Ogata. Rewriting can verify distributed real-time systems – how to verify in CafeOBJ. In Y. Toyama, editor, *Proc. Int. Workshop on Rewriting in Proof and Computation*, pages 60–79. Tohoku University, 2001.
9. J. Goguen. Types as theories. In G.M. Reed, A.W. Roscoe, and R.F. Wachter, editors, *Topology and Category Theory in Computer Science*, pages 357–390. Oxford, 1991.
10. J. Goguen, K. Lin, and G. Roşu. Circular coinductive rewriting. In *Proceedings, Automated Software Engineering*, pages 123–131. IEEE, 2000.
11. J. Goguen and G. Malcolm. A hidden agenda. *Theoretical Computer Science*, 245(1):55–101, 2000.
12. J. Goguen and G. Roşu. Hiding more of hidden algebra. In *Proceeding, FM'99*, volume 1709 of *LNCS*, pages 1704–1719. Springer, 1999.
13. R. Hennicker. Context induction: a proof principle for behavioral abstractions. *Formal Aspects of Computing*, 3(4):326–345, 1991.
14. R. Hennicker and M. Bidoit. Observational logic. In *Proceedings, AMAST'98*, volume 1548 of *LNCS*, pages 263–277. Springer, 1999.
15. B. Jacobs. Mongruences and cofree coalgebras. In M. Nivat, editor, *Algebraic Methodology and Software Technology (AMAST95)*, pages 245–260. Springer, 1995. LNCS 936.
16. B. Jacobs and J. Rutten. A tutorial on (co)algebras and (co)induction. *Bulletin of European Association for Theoretical Computer Science*, 62:222–259, 1997.
17. P. Padawitz. Towards the one-tiered design of data types and transition systems. In *Proceedings, WADT'97*, volume 1376 of *LNCS*, pages 365–380. Springer, 1998.
18. G. Roşu. Behavioral coinductive rewriting. In K. Futatsugi, J. Goguen, and J. Meseguer, editors, *OBJ/CafeOBJ/Maude at Formal Methods '99*, pages 179–196. Theta, 1999.
19. G. Roşu. *Hidden Logic*. PhD thesis, University of California, San Diego, 2000.
20. G. Roşu and J. Goguen. Hidden congruent deduction. In R. Caferra and G. Salzer, editors, *Automated Deduction in Classical and Non-Classical Logics*, volume 1761 of *LNAI*, pages 252–267. Springer, 2000.
21. G. Roşu and J. Goguen. Circular coinduction. In *International Joint Conference on Automated Reasoning (IJCAR'01)*. 2001.
22. J. Rutten. Universal coalgebra: a theory of systems. *Theoretical Computer Science*, 249:3–80, 2000.