Automated Proving of the Behavioral Attributes *

Gheorghe Grigoraş
grigoras@info.uaic.ro

Dorel Lucanu
dlucanu@info.uaic.ro

Georgiana Caltais
gcaltais@info.uaic.ro

Eugen-Ioan Goriac
egoriac@info.uaic.ro

Faculty of Computer Science
Alexandru Ioan Cuza University
Iaşi, Romania

Abstract

Behavioral equivalence is indistinguishably under experiments: two elements are behavioral equivalent iff each experiment returns the same value for the two elements. Behavioral equivalence can be proved by coinduction. CIRC *is a theorem prover which implements circular coinduction, an efficient coinductive technique. Equational attributes refer properties like associativity, commutativity, unity, etc. If these attributes are behaviorally satisfied, then we refer them as behavioral attributes. Two problems regarding these properties are important: expressing the commutativity as a rewrite rule leads to non-termination and their use as attributes requires a careful handling in the proving process. In this paper we present how these attributes are automatically checked in* CIRC *and we prove that this extension is sound.*

1 Introduction

Proving properties of systems involving infinite amount of information became a subject of high interest in computer science in the last years. Lazy functional programs, concurrency, transition and reactive systems, software verification and analysis are only several fields where the infinite data structures and other infinite objects are frequently used. The specification of such systems is given in different settings: process algebra (see, e.g., [16]), coalgebra [1, 12], behavioral equational logic [4, 9, 18, 17], type theory [7], temporal logic [8] and so on. The most known proof techniques used to prove properties for these systems are bisimilarity, coinduction, context induction, circular coinduction, coinductive types. Among the tools supporting (some of) these proof techniques we mention here Coq (coinductive types) [3], Isabelle/HOL [11], CIRC (circular coinduction) [14], BOBJ (circular coinduction) [18], Concurrency Workbench (bisimilarity) [6].

In this paper we refer the behavioral equational logic as a specification language, the circular coinduction proof technique [19], and its implementation CIRC [13]. We present the mechanism CIRC uses for automated proving of the attributes *associativity*, *commutativity*, *identity* and/or *idempotency* (ACUI) characterizing behaviorally defined operators over infinite data structures.

Motivating example Let us consider infinite binary trees with information in nodes from the boolean ring $Z_2 = (Z_2, +, \times, 0, 1)$. Since these trees carry infinite information we prove properties over them in terms of *behavioral equivalence*. Infinite binary trees are behaviorally specified by means of a *hidden* sort *Tree* for trees, a *visible* sort Z_2 for the information in the nodes, the equational specification of Z_2 and the following three *observers*:

- *root* : *Tree* $\to Z_2$, returning the information from the root of the tree;
- *left*, *right* : *Tree* $\to$ *Tree*, returning the left, respectively the right subtrees of a tree

Two trees are behaviorally equivalent if they cannot be distinguished under all possible experiments, *i.e.*, $T_1 \equiv T_2$ if $C[T_1] = C[T_2]$ for each experiment C. Here are several examples of experiments: $root(*{:}Tree)$, $root(left(*{:}Tree))$, $root(right(*{:}Tree))$, $root(left(left(*{:}Tree)))$ and so on. In CIRC, $*{:}Tree$ is a generic notation used for representing variables of hidden sort (in our case, the sort *Tree*). Note

*This work is partially supported by the PNII grant CNCSIS IDEI 393

that an experiment always returns a visible (data) value in Z_2.

The operations over trees are coinductively defined by means of the observers. For instance, the addition of trees is defined as follows:

$$\mathit{root}(T_1 + T_2) = \mathit{root}(T_1) + \mathit{root}(T_2)$$
$$\mathit{left}(T_1 + T_2) = \mathit{left}(T_1) + \mathit{left}(T_2)$$
$$\mathit{right}(T_1 + T_2) = \mathit{right}(T_1) + \mathit{right}(T_2)$$

Note that the operator $+$ is overloaded: it denotes the addition in the boolean ring and the addition of infinite trees. CIRC uses circular coinduction to prove behavioral equivalence $T_1 \equiv T_2$. Briefly, the algorithm works as follows: First try to deduce $T_1 \equiv T_2$ using the equations of the specification as rewrite rules. If it succeeds, then the algorithm successfully terminates. Otherwise, a frozen form of $T_1 \equiv T_2$, $\boxed{T_1} = \boxed{T_2}$, is added as coinductive hypothesis to the specification and three new subgoals are generated: *root*(T_1) = *root*(T_2), *left*(T_1) = *left*(T_2), *right*(T_1) = *right*(T_2). By freezing the coinductive hypothesis, we forbid its use under contexts, avoiding in this way unsound deductions. In order to allow the use of the coinductive hypotheses, the goals are handled in the frozen form. If $T_1 \equiv T_2$ expresses the commutativity of the addition, *i.e.*, $T + T' = T' + T$, then the specification becomes nonterminating after the frozen form of the coinductive hypothesis is added. Therefore the above algorithm does not work for properties such as commutativity. We decided to handle the properties like commutativity, associativity, unity and idempotency as distinguished goals. When provided a goal such as

```
op _+_ : Tree Tree -> Tree [assoc comm] .
```

expressing the associativity and commutativity of the addition operator, the expanding consist of the following actions:

1. add to the specification a new operation $\_ +^{AC} \_$ which is declared with the same attributes as the goal (here, associativity and commutativity);
2. add to the specification a set of frozen equations that express the freezing of the coinductive hypotheses in terms of the new operator:

$$\boxed{T_1 + T_2} = \boxed{T_1 +^{AC} T_2} \tag{1}$$
$$\boxed{T_1 + (T_2 + T_3)} = \boxed{T_1 +^{AC} T_2 +^{AC} T_3} \tag{2}$$
$$\boxed{(T_1 + T_2) + T_3} = \boxed{T_1 +^{AC} T_2 +^{AC} T_3} \tag{3}$$

   where the parentheses in the right hand side are no longer necessary because the operator $+^{AC}$ is associative and commutative;
3. compute the new subgoals, in their frozen form, corresponding to the equations defining the operational attributes:

$$root(T1 + T2) = root(T2 + T1)$$
$$left(T1 + T2) = left(T2 + T1)$$
$$right(T1 + T2) = right(T2 + T1)$$
$$root(T1 + (T2 + T3)) = root((T1 + T2) + T3)$$
$$left(T1 + (T2 + T3)) = left((T1 + T2) + T3)$$
$$right(T1 + (T2 + T3)) = right((T1 + T2) + T3)$$

We noticed that it is not sufficient to add the equation in step 2. Sometimes, due to the new equations, our term rewriting system is not confluent, so the equalities obtained by applying the Knuth-Bendix completion procedure [2] need to be added as well.

For instance, the term $\boxed{\mathit{left}(T_1) + \mathit{left}(T_2 + T_3)}$ can be reduced to both:

- $\boxed{T_1' = \mathit{left}(T_1) +^{AC} (\mathit{left}(T_2) + \mathit{left}(T_3))}$, according to (1) and the definition of the addition
- $\boxed{T_2' = \mathit{left}(T_1) +^{AC} \mathit{left}(T_2) +^{AC} \mathit{left}(T_3)}$, by applying the definition of the addition and (2)

These two terms form a critical pair, therefore we need to add the equation $T_1' = T_2'$ in order to obtain a confluent term rewriting system.

In this paper we formally present the extension of CIRC with the capability of automatically proving behavioral attributes and we prove that the extension is sound.

The paper is organized as follows. Section 2 briefly recalls the behavioral algebraic specifications and the notion of behavioral equivalence and introduces the infinite binary trees as the running example. Section 3 presents procedures used by CIRC to automatically prove certain behavioral properties. Section 4 describes the mechanism of proving behavioral attributes.

2 Behavioral Algebraic Specifications

We assume the reader familiar with basics of many sorted algebraic specifications [10] and only briefly recall our notation.

Let Σ be an algebraic signature consisting of a set S of *sorts* and an $S^* \times S$-indexed set $Op(\Sigma) = (Op(\Sigma)_{w,s} \mid w \in S^*, s \in S)$ of *operations*. Let $\mathcal{X}$ be a fixed S-indexed set of *variables*. $T_\Sigma(\mathcal{X})$ is the Σ-algebra of terms with variables in $\mathcal{X}$. A Σ-*equation* is a sentence $(\forall X)\ t = t'$ `if` c, where t and t' are Σ-terms over variables $X \subseteq \mathcal{X}$ having the same result sort, and c is the *condition* of the equation consisting of a finite set of pairs (u_i, v_i) of terms over variables X. The condition can be empty, case in which the equation is unconditional and written as $(\forall X)\ t = t'$.

Given a set E of Σ-equations, we say that a Σ-equation e is *deducible* (inferable) from E, and write $E \models e$, if e can be obtained by applying the following rules for a finite number of times:

1. *Assumption*. $E \models_\Sigma e$, for each e in E.
2. *Reflexivity*. $E \models_\Sigma (\forall X)\, t = t$.
3. *Transitivity*. If $E \models_\Sigma (\forall X)\, t_1 = t_2$ and $E \models_\Sigma (\forall X)\, t_2 = t_3$, then $E \models_\Sigma (\forall X)\, t_1 = t_3$.
4. *Substitution*. Given $e \in E$ such that e is either $(\forall X)\, t_1 = t_2$ `if` c or $(\forall X)\, t_2 = t_1$ `if` c, and a substitution $\theta : T_\Sigma(X) \to T_\Sigma(Y)$ such that $E \models (\forall Y)\theta(u_i) = \theta(v_i)$ for each (u_i, v_i) in c,
   then $E \models (\forall Y)\theta(t_1) = \theta(t_2)$.
5. *Congruence*. Given a context $t_0 \in T_\Sigma(Y \cup \{*\})$ with $* \notin Y$, $e \in E$ such that e is either $(\forall X)\, t_1 = t_2$ `if` c or $(\forall X)\, t_2 = t_1$ `if` c,
   if $E \models e$ then $E \models (\forall X \cup Y)\, t_0[t_1] = t_0[t_2]$ `if` c.

In the last rule, $t_0[t_i]$ denotes the term obtained from t_0 by replacing t_i for the distinguished variable $*$. We omit to write the subscript Σ for the deduction relation whenever it is understood from the context.

A *derivative* is a term $\delta \in T_\Sigma(\mathcal{X} \cup \{*{:}h\})$, where $*{:}h$ is a special variable of sort h. A *behavioral specification* is a pair $(\mathcal{B}, \Delta)$, where $\mathcal{B} = (S, \Sigma, E)$ is a many sorted equational specification and Δ is a set of derivatives. We distinguish two disjoint subsets $V, H \subseteq S$, where H is the subset of *hidden sorts* h corresponding to the star variables in the derivatives, and $V = S \setminus H$ is the subset of *visible sorts*. We assume that the equations E have only visible conditions. A Δ*-experiment* for the hidden sort $h \in H$ is inductively defined as follows: each derivative for the hidden sort $h \in H$ with visible result sort is a Δ-experiment for h; if C is a Δ-experiment for h' and δ a behavioral operation for h with result sort h', then $C[\delta]$ is a Δ-experiment for h. As above, $C[\delta]$ denotes the term obtained from C by replacing δ for the distinguished variable $*{:}h'$. A Δ-experiment $C[*{:}h]$ can be seen as a partially defined *equation transformer* $e \mapsto C[e]$: if e is an equation $(\forall X)\, t = t'$ `if` c of sort s, then $C[e]$ is the equation $(\forall X \cup Y)\, C[t] = C[t']$ `if` c, where Y is the set of non-star variables occurring in $C[*{:}s]$. Moreover, $\Delta[e] = \{\delta[e] \mid \delta \in \Delta\}$.

The notion of *behavioral equivalence* is an inherently semantic one: there is a behavioral equivalence relation on each model which can be defined as “indistinguishably under experiments”. For technical simplicity, we here prefer to avoid introducing models, so we give an alternative proof theoretic definition. Let $(\mathcal{B}, \Delta)$ be a behavioral specification. We say that $\mathcal{B}$ *behaviorally satisfies* an equation e, written $\mathcal{B} \Vdash e$, iff:

- $\mathcal{B} \models e$ if e is visible, and
- $\mathcal{B} \models C[e]$ for each appropriate Δ-experiment C if e is hidden.

The *behavioral equivalence* of $\mathcal{B}$, is the set of equations $\{e \mid \mathcal{B} \Vdash e\}$ [19].

Example: Infinite Binary Trees. We use the CIRC syntax for presenting the behavioral specifications. Since CIRC extends Maude [5] with behavioral features, the equational part of these specifications uses only the Maude syntax. The behavioral specification of infinite binary trees can be specified as follows. First, the specification module of the boolean ring Z_2 is given:

```
theory BRING is
  sort Z2 .
  ops 0 1 : -> Z2 .
  op _+_ : Z2 Z2 -> Z2 [assoc comm id: 0] .
  op _x_ : Z2 Z2 -> Z2 [assoc comm] .
  op ~_ : Z2 -> Z2 .
  eq 1 + 1 = 0 .
  eq (0 x X:Z2) = 0 .
  eq (1 x X:Z2) = X:Z2 .
  eq ~ 0 = 1 .
  eq ~ 1 = 0 .
  eq ~ ~ X:Z2 = X:Z2 .
endtheory
```

The module with the equations specifying the infinite binary trees is as follows:

```
theory EQ-TREE is
  including BRING .

  sort Tree .
  vars T T1 T2 : Tree .
  var X : Z2 .

  op root : Tree -> Z2 .
  ops left right : Tree -> Tree .

  op zero : -> Tree .         op one : -> Tree .
  eq root(zero) = 0 .         eq root(one) = 1 .
  eq left(zero) = zero .      eq left(one) = one .
  eq right(zero) = zero .     eq right(one) = one .

  op ~_ : Tree -> Tree .       op thue : -> Tree .
  eq root(~T)=~root(T) .       eq root(thue) = 0 .
  eq left(~T)=~left(T) .       eq left(thue) = thue.
  eq right(~T)=~right(T) . eq right(thue)=thue+
                                              one .
  op _+_ : Tree Tree -> Tree .
  eq root(T1 + T2) = root(T1) + root(T2) .
  eq left(T1 + T2) = left(T1) + left(T2) .
  eq right(T1 + T2) = right(T1) + right(T2) .
  ...
endtheory
```

The derivatives (behavioral operations) for infinite trees are declared in a separate CIRC theory module, which extends the functionality of a Maude theory module.

```
ctheory TREE is
  including EQ-TREE .
  derivative root(*:Tree) .
  derivative left(*:Tree) .
  derivative right(*:Tree) .
endctheory
```

The sort *Tree* is a hidden sort while the sort Z_2 is visible with respect to TREE specification. Recall that the result sort of the experiments is always visible. In this situation, an experiment is defined as:

1. *root(*:Tree)* is an experiment;
2. if $C[*{:}\mathit{Tree}]$ is an experiment then $C[\mathit{left}(*{:}\mathit{Tree})]$ and $C[\mathit{right}(*{:}\mathit{Tree})]$ are experiments.

For instance, the terms $root(*{:}Tree)$, $root(left(*{:}Tree))$, $root(right(*{:}Tree))$, $root(left(left(*{:}Tree)))$ are several examples of experiments. Two trees T_1 and T_2 are *behaviorally equivalent* iff *root*$(T_1) =$ *root*(T_2), *root*(*left*$(T_1)) =$ *root*(*left*$(T_2))$, *root*(*right*$(T_1)) =$ *root*(*right*$(T_2))$, and so on.

3 CIRC

CIRC is a tool for automated inductive and coinductive theorem proving, created as a behavioral extension of Maude.

The circular coinduction engine of CIRC implements the proof system presented in [19] by the reduction rules given in Fig. 1. Since the equational deduction is recursive enumerable, CIRC uses the decidable entailment $\mathcal{E} \vdash_{\rightarrow\leftarrow} (\forall X)t = t'$ if $\wedge_{i\in I}(u_i = v_i)$ iff $\mathrm{nf}(t) = \mathrm{nf}(t')$, where $\mathrm{nf}(t)$ is computed as follows:

– the variables X are turned into fresh constants;

– the condition equalities $u_i = v_i$ are added as equations to the specification;

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

The reduction rules are defined over triples $(\mathcal{B}, \mathcal{F}, \mathcal{G})$, where $\mathcal{B}$ represents the (original) algebraic specification, $\mathcal{F}$ is the set of frozen axioms and $\mathcal{G}$ is the current set of proof obligations.

[Done]

$(\mathcal{B}, \mathcal{F}, \emptyset) \Rightarrow \cdot$

[Reduce]

$(\mathcal{B}, \mathcal{F}, \mathcal{G} \cup \{\boxed{e}\}) \Rightarrow (\mathcal{B}, \mathcal{F}, \mathcal{G})$ if $\mathcal{B} \cup \mathcal{F} \vdash_{\rightarrow\leftarrow} e$

[Derive]

$(\mathcal{B}, \mathcal{F}, \mathcal{G} \cup \{\boxed{e}\}) \Rightarrow (\mathcal{B}, \mathcal{F}, \mathcal{G} \cup \{\boxed{\Delta(e)}\})$

if $\mathcal{B} \cup \mathcal{F} \not\vdash_{\rightarrow\leftarrow} e$ and e is hidden

[Normalize]

$(\mathcal{B}, \mathcal{F}, \mathcal{G} \cup \{\boxed{e}\}) \Rightarrow (\mathcal{B}, \mathcal{F}, \mathcal{G} \cup \{\boxed{\mathrm{nf}(e)}\})$

[Fail]

$(\mathcal{B}, \mathcal{F}, \mathcal{G} \cup \{\boxed{e}\}) \Rightarrow$ *fail* if $\mathcal{B} \cup \mathcal{F} \not\vdash_{\rightarrow\leftarrow} e$ and e is visible

Figure 1. Circular Coinduction in CIRC

A brief description of the rules is as follows:

[Done] – is applied whenever the set of proof obligations is empty and indicates the termination of the process.

[Reduce] – is applied whenever the current goal is a $\vdash_{\rightarrow\leftarrow}$-consequence of $\mathcal{B} \cup \mathcal{F}$ and operates by removing $\boxed{e}$ from the set of goals.

[Derive] – is applied when the current goal e is hidden and it is not a $\vdash_{\rightarrow\leftarrow}$-consequence. The current goal is added to the specification and its derivatives to the set of goals. $\Delta(e)$ denotes the set $\{\delta[e] \mid \delta \in \Delta\}$.

[Normalize] – removes the current goal from the set of proof obligations and adds its normal form as a new goal. The normal form $\mathrm{nf}(e)$ of an equation e of the form $(\forall X)t = t'$ if $\wedge_{i\in I}(u_i = v_i)$ is $(\forall X)\mathrm{nf}(t) = \mathrm{nf}(t')$ if $\wedge_{i\in I}(u_i = v_i)$, where the constants from the normal forms are turned back into the corresponding variables.

[Fail] – stops the reduction process with failure whenever the current goal e is visible and the corresponding normal forms are different.

The wrapping operator $\boxed{\_} : s \rightarrow Frozen$ is implemented in CIRC as the operator `[* _ *]`. For an equation $(\forall X)\ t = t'$ if c, the corresponding frozen equation is: $(\forall X)$`[*` t `*]` $=$ `[*` t' `*]` if c, where `[* _ *]` $: Sort(t) \rightarrow Frozen$.

A proof of the following theorem can be found in [15].

Theorem 1 ***(soundness of CIRC)*** *Let $(\mathcal{B}, \Delta)$ be a behavioral specification and let $\mathcal{G}$ be a set of frozen equations. If $(\mathcal{B}, \emptyset, \mathcal{G}) \Rightarrow^* (\mathcal{B}, \mathcal{F}, \emptyset)$ applying the reduction rules in Fig. 1, then $\mathcal{B} \Vdash \mathcal{G}$.*

We present a session in CIRC for simultaneous proving that $thue + one = {\sim}thue$ and ${\sim}{\sim}T = T$. After entering the specification, we need to add these properties as goals:

```
Maude> (add goal thue + one = ~ thue .)
Goal added: thue + one = ~ thue
Maude> (add goal ~ ~ T:Tree = T:Tree .)
Goal added: ~ ~ T:Tree = T:Tree
```

Then we introduce the coinduction command to automatically prove the two goals.

```
Maude> (coinduction .)
Proof succeeded.
  Number of derived goals: 9
  Number of proving steps performed: 45
  Maximum number of proving steps is set to: 256
Proved properties:
  ~ thue + one = thue
  thue + one = ~ thue
  ~ ~ T:Tree = T:Tree
```

It is worth noting that CIRC discovered and automatically proved a new lemma: ${\sim}thue + one = thue$. The circular coinduction cannot terminate in some cases, therefore there is a parameter which sets the maximum number of reduction steps (here 256). The rest of the output is self-explanatory. We may see the complete proof of the above properties:

```
Maude> (show proof .)
[* right(~ thue + one) *] = [* right(thue) *]
-------------------------------------- [Reduce]
[* right(~ thue + one) *] = [* right(thue) *]

[* left(~ thue + one) *] = [* left(thue) *]
-------------------------------------- [Reduce]
[* left(~ thue + one) *] = [* left(thue) *]

[* root(~ thue + one) *] = [* root(thue) *]
-------------------------------------- [Reduce]
[* root(~ thue + one) *] = [* root(thue) *]

[* root(~ thue + one) *] = [* root(thue) *]
[* left(~ thue + one) *] = [* left(thue) *]
```

```
[*right(~thue+one)*]=[*right(thue)*]
------------------------------------- [Derive]
[*~thue+one*] = [*thue*]

[*~thue+one*] = [*thue*]
------------------------------------- [Normalize]
[*right(thue+one)*]=[*right(~thue)*]

[*left(thue+one)*] = [*left(~thue)*]
------------------------------------- [Reduce]
[*left(thue+one)*] = [*left(~thue)*]

[*root(thue+one)*] = [*root(~thue)*]
------------------------------------- [Reduce]
[*root(thue+one)*] = [*root(~thue)*]

[*right(~~T)*] = [*right(T)*]
------------------------------------- [Reduce]
[*right(~~T)*] = [*right(T)*]

[*left(~~T)*] = [*left(T)*]
------------------------------------- [Reduce]
[*left(~~T)*] = [*left(T)*]

[*root(~~T)*] = [*root(T)*]
------------------------------------- [Reduce]
[*root(~~T)*] = [*root(T)*]

[*root(thue+one)*] = [*root(~thue)*]
[*left(thue+one)*] = [*left(~thue)*]
[*right(thue+one)*]=[*right(~thue)*]
------------------------------------- [Derive]
[*thue+one*] = [*~thue*]

[*root(~~T)*] = [*root(T)*]
[*left(~~T)*] = [*left(T)*]
[*right(~~T)*] = [*right(T)*]
------------------------------------- [Derive]
[*~~T*] = [*T*]
```

We further provide an example on how CIRC handles proving properties that do not hold. Let us try to prove that $\sim zero = zero$:

```
Maude> (add goal ~ zero = zero .)
Goal added: ~ zero = zero
```

We use the following two commands in order to see all the proof details and, respectively, start the coinduction algorithm:

```
Maude> (set show details on .)
Maude> (coinduction .)
Hypo ~ zero = zero added and coexpanded to
1.  root(~ zero) = root(zero)
2.  left(~ zero) = left(zero)
3.  right(~ zero) = right(zero)
Goal root(~ zero) = root(zero) reduced to
    1 = 0
Visible goal 1 = 0 failed during coinduction.
```

Note that after the expansion, one of the derived goals is `root(~zero) = root(zero)`, which is reduced according to the specification to `1 = 0`. This means that the initial goal failed to be proved.

All the examples from this paper can be tested with the on-line version of the CIRC tool (http://fsl.cs.uiuc.edu/index.php/Special:CircOnline).

4 Proving Behavioral Attributes

Let $\mathcal{B}$ be the behavioral specification and let us consider a new type of goals noted by $W(op)$ where op is an operation defined in $\mathcal{B}$ and W is any combination of the following attributes A - associativity, C - commutativity, I - idempotency, U -unity. In fact the goal is to prove the properties in W for the operation op. For example AC$(+)$ is the task *to prove that the operation + is associative and commutative*, ACU(+) is the goal *to prove that the operation + is associative, commutative and has unity* and so on. We denote by $Eqn(W)$ the set of equations corresponding to W and by $Fr(W)$ the set of equations corresponding to freezing W. According to the attributes, the equations in $Eqn(W)$ for a general operator *op* are:

$$\begin{aligned}
A &: (\forall X, Y, Z)(X\ op\ Y)\ op\ Z = X\ op\ (Y\ op\ Z)\\
C &: (\forall X, Y) X\ op\ Y = Y\ op\ X\\
U &: (\forall X) X\ op\ 0 = X\\
&\quad (\forall X) 0\ op\ X = X\\
I &: (\forall X) X\ op\ X = X
\end{aligned}$$

and the equations in $Fr(W)$ are:

$$\begin{aligned}
A &: \boxed{X\ op\ (Y\ op\ Z)} = \boxed{X\ op^W Y\ op^W Z}\\
&\quad \boxed{(X\ op\ Y)\ op\ Z} = \boxed{X\ op^W Y\ op^W Z}\\
C, U, I &: \boxed{X\ op\ Y} = \boxed{X\ op^W Y}
\end{aligned}$$

If E is a set of equations, then let $KB(E)$ denote the completion of E obtained by applying Knuth-Bendix completion procedure[2]. Now we extend CIRC with a new deduction rule:

[Derive-atts]

$$(\mathcal{B}, \mathcal{F}, \mathcal{G} \cup \{W(op)\}) \Rightarrow$$
$$(\mathcal{B} \cup \{op^W\}, \mathcal{F} \cup KB(Fr(W)), \mathcal{G} \cup \boxed{\Delta(Eqn(W))}\ )$$

Theorem 2 *Let $(\mathcal{B}, \Delta)$ be a behavioral specification and W(op) a goal. If $(\mathcal{B}, \emptyset, \{W(op)\}) \Rightarrow^\star (\mathcal{B} \cup \{op^W\}, \mathcal{F}, \emptyset)$ using all the deduction rules introduced, then $\mathcal{B} \Vdash Eqn(W)$.*

Proof. We have:

$$(\mathcal{B}, \emptyset, \{W(op)\}) \Rightarrow^{\text{[Derive-atts]}}$$
$$(\mathcal{B} \cup \{op^W\}, KB(Fr(W)), \boxed{\Delta(Eqn(W))}\ ) \Rightarrow^\star$$
$$(\mathcal{B} \cup \{op^W\}, \mathcal{F}, \emptyset)$$

Let $\mathcal{B}_1$ denote the specification $\mathcal{B} \cup \{op^W\} \cup KB(Fr(W))$. By Theorem 1, $\mathcal{B}_1 \Vdash \Delta(Eqn(W))$. We show that $\mathcal{B} \cup \{Eqn(W)\} \Vdash \boxed{e} \Leftrightarrow \mathcal{B}_1 \Vdash \boxed{e}$ for each frozen $\mathcal{B}$ equation $\boxed{e}$. The direct implication follow from the fact $\mathcal{B}_1 \Vdash Eqn(W)$. For the inverse implication, we assume that $\mathcal{B}_1 \Vdash \boxed{e}$. It follows that $\mathcal{B} \cup \{op^W\} \cup Fr(W) \cup Eqn(W) \Vdash \boxed{e}$ by the monotonicity and cut rule of $\Vdash$. If π is a proof (in the equational deduction system) of $\boxed{e}$ from $\mathcal{B}_1 \cup \{Eqn(W)\}$, then we can construct a proof π' by replacing in each term the occurrences of op^W with *op* and in each step using an equation from *Fr(W)* with a corresponding equation from *Eqn(W)*. For instance, $x + y = x +^{AC} y$ is replaced with the use of x + y = y + x and (x + y) + z = x + (y + z). It is easy to see now that π' is in fact a proof of $\boxed{e}$ from $\mathcal{B} \cup \{Eqn(W)\}$. We have now $\mathcal{B} \cup \{Eqn(W)\} \Vvdash \Delta(Eqn(W))$, which implies $\mathcal{B} \Vvdash Eqn(W)$ by Theorem 7 in [15]. □

As an example, we present the CIRC dialog resulted while proving that $+$ is both associative and commutative and has the identity (unit) element *zero*:

```
Maude> (add goal (op _+_ : Tree Tree -> Tree
                  [assoc comm id: zero] .) .)
Maude> (coinduction .)
Proof succeeded.
  Number of derived goals: 12
  Number of proving steps performed: 52
  Maximum number of proving steps is set to: 256
Maude> (show proof .)
...
[*root((X + Y)+ Z)*] = [*root(X + Y + Z)*]
[*left((X + Y)+ Z)*] = [*left(X + Y + Z)*]
[*right((X + Y)+ Z)*] = [*right(X + Y + Z)*]
[*root(X + Y)*] = [*root(Y + X)*]
[*left(X + Y)*] = [*left(Y + X)*]
[*right(X + Y)*] = [*right(Y + X)*]
[*root(zero + X)*] = [*root(X)*]
[*left(zero + X)*] = [*left(X)*]
[*right(zero + X)*] = [*right(X)*]
[*root(X + zero)*] = [*root(X)*]
[*left(X + zero)*] = [*left(X)*]
[*right(X + zero)*] = [*right(X)*]
----------------------------------- [Derive-atts]
  op _+_ : Tree Tree -> Tree
           [assoc comm id: zero] .
```

The output of the proof is not complete, only the first applied deduction rule, [Derive-atts], is presented here. We can see that all twelve derived goals are generated by this rule. Obviously, all these new goals are proved in this case using only [Reduce]. The rule [Derive-atts] may interfere with the the other rules of circular coinduction during the automated proving process, increasing in this way the power of the prover.

5 Conclusion

In this paper we presented some examples of using CIRC, a theorem prover implementing the circular coinduction principle, in order to prove a set of properties over infinite data structures. The main contribution is providing a new technique for proving behavioral attributes based on rewriting modulo commutativity, associativity, unity and/or idempotency.

References

[1] J. Adámek. Introduction to coalgebra. *Theory and Applications of Categories*, 14(8):157–199, 2005.

[2] L. Bachmair, N. Dershowitz, and D. Plaisted. Completion without failure. In *Resolution of Equations in Algebraic Structures*, volume 2, pages 1–30. Academic Press, 1989.

[3] Y. Bertot and E. Komendantskaya. Inductive and coinductive components of corecursive functions in coq. *Electron. Notes Theor. Comput. Sci.*, 203(5):25–47, 2008.

[4] M. Bidoit, R. Hennicker, and A. Kurz. Observational logic, constructor-based logic, and their duality. *Theoretical Computer Science*, 3(298):471–510, 2003.

[5] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. L. Talcott, editors. *All About Maude - A High-Performance Logical Framework, How to Specify, Program and Verify Systems in Rewriting Logic*, volume 4350 of *Lecture Notes in Computer Science*. Springer, 2007.

[6] R. Cleaveland, J. Parrow, and B. Steffen. The concurrency workbench: a semantics-based tool for the verification of concurrent systems. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 15(1):36–72, 1993.

[7] T. Coquand. Infinite objects in type theory. In H. Barendregt and T. Nipkow, editors, *TYPES*, volume 806 of *Lecture Notes in Computer Science*, pages 62–78. Springer, 1993.

[8] E. A. Emerson. Model checking and the mu-calculus. In *DIMACS Series in Discrete Mathematics*, pages 185–214. American Mathematical Society, 1997.

[9] J. Goguen and G. Malcolm. A hidden agenda. *Theoretical Computer Science*, 245(1):55–101, August 2000.

[10] J. Goguen and J. Meseguer. Completeness of Many-Sorted Equational Logic. *Houston Journal of Mathematics*, 11(3):307–334, 1985.

[11] D. Hausmann, T. Mossakowski, and L. Schröder. Iterative circular coinduction for cocasl in isabelle/hol. In M. Cerioli, editor, *FASE*, volume 3442 of *Lecture Notes in Computer Science*, pages 341–356. Springer, 2005.

[12] B. Jacobs and J. Rutten. A tutorial on (co)algebras and (co)induction. *Bulletin of the European Association for Theoretical Computer Science*, 62:222–259, 1997.

[13] D. Lucanu, E.-I. Goriac, G. Caltais, and G. Roşu. CIRC : A behavioral verification tool based on circular coinduction. In *CALCO'09*, LNCS, 2009. To appear.

[14] D. Lucanu and G. Roşu. Circ : A circular coinductive prover. In T. Mossakowski, U. Montanari, and M. Haveraaen, editors, *CALCO*, volume 4624 of *Lecture Notes in Computer Science*, pages 372–378. Springer, 2007.

[15] D. Lucanu and G. Roşu. Circular Coinduction with Special Contexts. Technical Report UIUCDCS-R-2009-3039, University of Illinois at Urbana-Champaign, 2009. Submitted.

[16] R. Milner. *Communicating and Mobile Systems: the π-calculus*. Cambridge University Press, 1999.

[17] T. Mossakowski, H. Reichel, M. Roggenbach, and L. Schroder. Algebraic-coalgebraic specification in CoCASL. In *Proceedings of WADT'02*, volume 2755 of *LNCS*, pages 376–392. Springer, 2002.

[18] G. Roşu. *Hidden Logic*. PhD thesis, University of California at San Diego, 2000.

[19] G. Roşu and D. Lucanu. Circular Coinduction –A Proof Theoretical Foundation. In *CALCO'09*, LNCS. Springer, 2009. To appear.