Circular Coinduction-based Techniques for Proving Behavioral Properties

Dorel Lucanu

Al. I. Cuza Univ., Iaşi, RO

Nov. 6, 2008, INRIA/LORIA
Plan
What is CIRC?

- CIRC is a metalanguage application implemented as an extension of Full Maude:
  - comes with a new way of execution (proving goals expressing behavioral equivalences)
  - provides a parser for a language syntax that extends Maude’s
- implements Circularity Principle (CP) for both coinduction and induction (the later partially implemented)
- extends CP for coinduction with simplification and special context (in progress)
- uses strategies for specifying proof tactics
What is CIRC?

- CIRC is a metalanguage application implemented as an extension of Full Maude:
  - comes with a new way of execution (proving goals expressing behavioral equivalences)
  - provides a parser for a language syntax that extends Maude’s
- implements Circularity Principle (CP) for both coinduction and induction (the later partially implemented)
- extends CP for coinduction with simplification and special context (in progress)
- uses strategies for specifying proof tactics
What is CIRC?

- CIRC is a metalanguage application implemented as an extension of Full Maude:
  - comes with a new way of execution (proving goals expressing behavioral equivalences)
  - provides a parser for a language syntax that extends Maude’s
- implements Circularity Principle (CP) for both coinduction and induction (the later partially implemented)
- extends CP for coinduction with simplification and special context (in progress)
- uses strategies for specifying proof tactics
What is CIRC?

- CIRC is a metalanguage application implemented as an extension of Full Maude:
  - comes with a new way of execution (proving goals expressing behavioral equivalences)
  - provides a parser for a language syntax that extends Maude’s
- implements Circularity Principle (CP) for both coinduction and induction (the later partially implemented)
  - extends CP for coinduction with simplification and special context (in progress)
- uses strategies for specifying proof tactics
What is CIRC?

- CIRC is a metalanguage application implemented as an extension of Full Maude:
  - comes with a new way of execution (proving goals expressing behavioral equivalences)
  - provides a parser for a language syntax that extends Maude’s
- implements Circularity Principle (CP) for both coinduction and induction (the later partially implemented)
- extends CP for coinduction with simplification and special context (in progress)
- uses strategies for specifying proof tactics
What is CIRC?

- CIRC is a metalanguage application implemented as an extension of Full Maude:
  - comes with a new way of execution (proving goals expressing behavioral equivalences)
  - provides a parser for a language syntax that extends Maude’s
- implements Circularity Principle (CP) for both coinduction and induction (the later partially implemented)
- extends CP for coinduction with simplification and special context (in progress)
- uses strategies for specifying proof tactics
Plan
Example: Streams 1/3

- consider:
  - two datatypes: \( \text{Bit} = \{0, 1\} \) and \( \text{BitStream} \ (S = b_0 b_1 b_2 \ldots) \)
  - two behavioral operations:
    - \( \text{hd} : \text{BitStream} \rightarrow \text{Bit} \) \( (\text{hd}(S) = b_0) \)
    - \( \text{tl} : \text{BitStream} \rightarrow \text{BitStream} \) \( (\text{tl}(S) = b_1 b_2 \ldots) \)
  - another operation:
    - \( \text{zip} : \text{BitStream} \times \text{BitStream} \rightarrow \text{Bit} \)
    - \( \text{zip}(b_0 b_1 \ldots, b'_0 b'_1 \ldots) = b_0 b'_0 b_1 b'_1 \ldots \)
  - define behavioral equivalence \( \equiv \) over \( \text{BitStream} \) by:
    - \( S_1 \equiv S_2 \) iff \( \text{hd}(S_1) = \text{hd}(S_2) \) and \( \text{tl}(S_1) \equiv \text{tl}(S_2) \)
  - how do we prove that \( \text{zip}(0^\infty, 1^\infty) \equiv (01)^\infty \)?

\[
W^\infty \overset{\text{def}}{=} W : W : W : \ldots
\]
Example: Streams 1/3

Consider:

- Two datatypes: Bit = \{0, 1\} and BitStream (S = b_0 b_1 b_2 \ldots)
- Two behavioral operations:
  \[ hd : \text{BitStream} \rightarrow \text{Bit} \quad (\text{hd}(S) = b_0) \]
  \[ tl : \text{BitStream} \rightarrow \text{BitStream} \quad (\text{tl}(S) = b_1 b_2 \ldots) \]
- Another operation:
  \[ \text{zip} : \text{BitStream} \times \text{BitStream} \rightarrow \text{Bit} \]
  \[ \text{zip}(b_0 b_1 \ldots, b'_0 b'_1 \ldots) = b_0 b'_0 b_1 b'_1 \ldots \]

Define behavioral equivalence \( \equiv \) over BitStream by:

\[ S_1 \equiv S_2 \text{ iff } \text{hd}(S_1) = \text{hd}(S_2) \text{ and } \text{tl}(S_1) \equiv \text{tl}(S_2) \]

How do we prove that \( \text{zip}(0^\infty, 1^\infty) \equiv (01)^\infty? \)

---

\( W^\infty \overset{\text{def}}{=} W : W : W : \ldots \)
Example: Streams 1/3

- consider:
  - two datatypes: $\text{Bit} = \{0, 1\}$ and $\text{BitStream} (S = b_0 b_1 b_2 \ldots)$
  - two behavioral operations:
    - $hd : \text{BitStream} \rightarrow \text{Bit}$ \hspace{1cm} ($hd(S) = b_0$)
    - $tl : \text{BitStream} \rightarrow \text{BitStream}$ \hspace{1cm} ($tl(S) = b_1 b_2 \ldots$)
  - another operation:
    - $zip : \text{BitStream} \times \text{BitStream} \rightarrow \text{Bit}$
      \hspace{1cm} ($zip(b_0 b_1 \ldots, b'_0 b'_1 \ldots) = b_0 b'_0 b_1 b'_1 \ldots$)

- define behavioral equivalence $\equiv$ over $\text{BitStream}$ by:
  \hspace{1cm} $S_1 \equiv S_2$ iff $hd(S_1) = hd(S_2)$ and $tl(S_1) \equiv tl(S_2)$

- how do we prove that $zip(0^\infty, 1^\infty) \equiv (01)^\infty$?\footnote{$w^\infty \overset{\text{def}}{=} w : w : w : \ldots$}
Example: Streams 1/3

▶ consider:
  ▶ two datatypes: Bit = \{0, 1\} and BitStream (S = b_0 b_1 b_2 ...)
  ▶ two behavioral operations:
    \[
    \begin{align*}
    & \text{hd} : \text{BitStream} \to \text{Bit} \\
    & \text{tl} : \text{BitStream} \to \text{BitStream}
    \end{align*}
    \]
    \( (\text{hd}(S) = b_0) \)
    \( (\text{tl}(S) = b_1 b_2 ...) \)
  ▶ another operation:
    \[
    \begin{align*}
    & \text{zip} : \text{BitStream} \times \text{BitStream} \to \text{Bit} \\
    & \text{zip}(b_0 b_1 ... , b'_0 b'_1 ...) = b_0 b'_0 b_1 b'_1 ...
    \end{align*}
    \]
▶ define behavioral equivalence \( \equiv \) over BitStream by:
\[
S_1 \equiv S_2 \text{ iff } \text{hd}(S_1) = \text{hd}(S_2) \text{ and } \text{tl}(S_1) \equiv \text{tl}(S_2)
\]
▶ how do we prove that \( \text{zip}(0^\infty, 1^\infty) \equiv (01)^\infty ? \)

---

\(^1\) \( W^\infty \) def = \( W : W : W : \ldots \)
Example: Streams 1/3

▶ consider:
  ▶ two datatypes: Bit = \{0, 1\} and BitStream \(S = b_0 b_1 b_2 \ldots\)
  ▶ two behavioral operations:
    \(hd : \text{BitStream} \to \text{Bit} \quad (hd(S) = b_0)\)
    \(tl : \text{BitStream} \to \text{BitStream} \quad (tl(S) = b_1 b_2 \ldots)\)
  ▶ another operation:
    \(zip : \text{BitStream} \times \text{BitStream} \to \text{Bit} \quad zip(b_0 b_1 \ldots, b'_0 b'_1 \ldots) = b_0 b'_0 b_1 b'_1 \ldots\)

▶ define behavioral equivalence \(\equiv\) over BitStream by:
\(S_1 \equiv S_2 \text{ iff } hd(S_1) = hd(S_2) \text{ and } tl(S_1) \equiv tl(S_2)\)

▶ how do we prove that \(zip(0^\infty, 1^\infty) \equiv (01)^\infty?\)

\[1 \quad W^\infty \overset{\text{def}}{=} W : W : W : \ldots\]
Example: Streams 2/3

(fth BITSTREAM .
  sort Bit BitStream .
  var S S' : BitStream .
  var B : Bit .

  ops 0 1 : -> Bit . --- constants of sort Bit

  op hd : BitStream -> Bit . --- the derivatives
  op tl : BitStream -> BitStream . --- (observers)

  op zip : BitStream BitStream -> BitStream .
  eq hd(zip(S, S')) = hd(S) .
  eq tl(zip(S, S')) = zip(S', tl(S)) .

  op _:_ : Bit BitStream -> BitStream .
  eq hd(B : S)) = B .
  eq tl(B : S)) = S .
endfth)
Example: Streams 2/3

(fth BITSTREAM .
  sort Bit BitStream .
  var S S' : BitStream .
  var B : Bit .

  ops 0 1 : -> Bit . --- constants of sort Bit

  op hd : BitStream -> Bit .           --- the derivatives
  op tl : BitStream -> BitStream .     (observers)

  op zip : BitStream BitStream -> BitStream .
  eq hd(zip(S, S')) = hd(S) .
  eq tl(zip(S, S')) = zip(S', tl(S)) .

  op _:_: : Bit BitStream -> BitStream .
  eq hd(B : S)) = B .
  eq tl(B : S)) = S .
endfth)
Example: Streams 2/3

(fth BITSTREAM .
    sort Bit BitStream .
    var S S' : BitStream .
    var B : Bit .

    ops 0 1 : -> Bit . --- constants of sort Bit

    op hd : BitStream -> Bit . --- the derivatives
    op tl : BitStream -> BitStream . --- (observers)

    op zip : BitStream BitStream -> BitStream .
    eq hd(zip(S, S')) = hd(S) .
    eq tl(zip(S, S')) = zip(S', tl(S)) .

    op _:_ : Bit BitStream -> BitStream .
    eq hd(B : S)) = B .
    eq tl(B : S)) = S .
endfth)
Example: Streams 2/3

(fth BITSTREAM .
    sort Bit BitStream .
    var S S' : BitStream .
    var B : Bit .

    ops 0 1 : -> Bit . --- constants of sort Bit

    op hd : BitStream -> Bit . --- the derivatives
    op tl : BitStream -> BitStream . --- (observers)

    op zip : BitStream BitStream -> BitStream .
    eq hd(zip(S, S')) = hd(S) .
    eq tl(zip(S, S')) = zip(S', tl(S)) .

    op _:_ : Bit BitStream -> BitStream .
    eq hd(B : S)) = B .
    eq tl(B : S)) = S .
endfth)
Example: Streams 2/3

(fth BITSTREAM .
    sort Bit BitStream .
    var S S' : BitStream .
    var B : Bit .

    ops 0 1 : -> Bit . --- constants of sort Bit

    op hd : BitStream -> Bit . --- the derivatives
    op tl : BitStream -> BitStream . --- (observers)

    op zip : BitStream BitStream -> BitStream .
    eq hd(zip(S, S')) = hd(S) .
    eq tl(zip(S, S')) = zip(S', tl(S)) .

    op _:_ : Bit BitStream -> BitStream .
    eq hd(B : S)) = B .
    eq tl(B : S)) = S .
endfth)
Example: Streams 3/3

(cth BITSTREAM-0-1 is including BITSTREAM .
  ops zeroes ones blink : -> BitStream .

  eq hd(zeroes) = 0 . eq tl(zeroes) = zeroes .
  eq hd(ones) = 1 . eq tl(ones) = ones .
  eq hd(blink) = 0 . eq tl(blink) = 1 : blink .

  der hd(*:BitStream) .
  der tl(*:BitStream) .
endcth)

(add goal zip(zeroes, ones) = blink .)

Demo: streams-blink.maude
Example: Streams 3/3

(cth BITSTREAM-0-1 is including BITSTREAM .
ops zeroes ones blink : -> BitStream .

  eq hd(zeroes) = 0 .  eq tl(zeroes) = zeroes .
  eq hd(ones)   = 1 .  eq tl(ones)   = ones .
  eq hd(blink)  = 0 .  eq tl(blink)  = 1 : blink .

  der hd(*:BitStream) .
  der tl(*:BitStream) .
endcth)

(add goal zip(zeroes, ones) = blink .)

Demo: streams-blink.maude
Example: Streams 3/3

(cth BITSTREAM-0-1 is including BITSTREAM .
   ops zeroes ones blink : ~> BitStream .

   eq hd(zeroes) = 0 .  eq tl(zeroes) = zeroes .
   eq hd(ones) = 1 .  eq tl(ones) = ones .
   eq hd(blink) = 0 .  eq tl(blink) = 1 : blink .

   der hd(*:BitStream) .
   der tl(*:BitStream) .
endcth)

(add goal zip(zeroes, ones) = blink .)

Demo: streams-blink.maude
Example: Extended Regular Expressions 1/3

Let $\textit{Alph}$ be an alphabet.

The extended regular expressions over $\textit{Alph}$:

$$R ::= \varepsilon | \emptyset | A | R_1 + R_2 | R_1 \# R_2 | R^* | R_1 \cap R_2 | ¬R$$

where $A$ ranges over $\textit{Alph}$.

The behavioral operations (derivatives)[Brzozowski]:

- $\texttt{epsIn}$, testing the membership of $\varepsilon$ to an ERE, and
- $\texttt{\{\}_}$, which takes an ERE $R$ and a letter $a$ and returns an expression characterized by $L(R\{a\}) = \{w | aw \in L(R)\}$.

the behavioral equivalence:

$$R \equiv R' \text{ iff } \text{epsIn}R = \text{epsIn}R' \text{ and } (\forall a)R\{a\} \equiv R'\{a\}$$

Theorem

Together with the B–ERE behavioral specification, CIRC becomes a fully automatic decision procedure for the equivalence of EREs.
Example: Extended Regular Expressions 2/2

th ALPH is
  sort Alph .                     --- the alphabet
  ops a b : -> Alph .
endth

th ERE is
  inc ALPH + BOOL .
  sort Ere .                      --- regular expressions

  var R R1 R2 : Ere .   var A B : Alph .

  op _{ _}_: Ere Alph -> Ere .    --- letters derivatives
  op epsIn_: Re -> Bool .        --- epsilon membership derivative

  subsort Alph < Ere .           --- a letter
  eq epsIn A = false .
  eq B { A } = if A == B then epsilon else empty fi .

  op epsilon : -> Ere .          --- the empty word
  eq epsilon { A } = empty .
  eq epsIn epsilon = true .

  op _+_ : Ere Ere -> Ere [assoc comm] . --- union
  eq ( R1 + R2 ){ A } = (R1 { A }) + (R2 { A }) .
  eq epsIn ( R1 + R2 ) = epsIn R1 or epsIn R2 .
...
endth     ***>

Demo: ere.maude
Plan
Behavioral Specifications: Syntax

A behavioral specification is a triple \((D, B, \Delta)\), where

- \(D = (S_D, \Sigma_D, E_D)\) specifies data,
- \(B = (S, \Sigma, E)\) specifies further the behavioral operations
- there is an inclusion \(D \hookrightarrow B\),
- the elements of \(S_D\) are visible sorts; e.g., \texttt{Elt}
- the elements of \(S \setminus S_D\) are hidden sorts; e.g., \texttt{Stream}
- \(\Delta \subseteq \text{Der}(\Sigma)\) whose operations are called derivatives (behavioral opns) \(\Delta\); e.g., \(\text{hd}(*:\text{Stream}), \text{tl}(*:\text{Stream})\)
- \(*\) is a special variable of a hidden sort
- other names for derivatives: destructors, observers
- the derivatives are used to define the experiments:
  - each \(\delta \in \Delta\) of visible sort is an experiment; e.g., \(\text{hd}(*:\text{Stream})\)
  - if \(\gamma[*:h]\) is an experiment and \((\delta : w \rightarrow h) \in \Delta\),
    - then \(\gamma[\delta/*]\) is an experiment;
  - e.g., \(\text{hd}(\text{tl}(*:\text{Stream})), \text{hd}(\text{tl}(\text{tl}(*:\text{Stream})))\),...
Behavioral Specifications of EREs

(cth B-ERE
    including ERE .
    --- any Maude declaration is allowed here
    der epsIn(*:Ere) .
    der *:Ere a .
    der *:Ere b .

endcth)
Behavioral Specifications of EREs

(cth B-ERE
   including ERE .
   --- any Maude declaration is allowed here

   der epsIn(*:Ere) .
   der *:Ere a .
   der *:Ere b .

endcth)
Behavioral Specifications: Semantics

Let \((\mathcal{D}, \mathcal{B}, \Delta)\) be a behavioral spec.

A model is a \(\mathcal{B}\)-model \(M\) together with the behavioral equivalence \(\equiv_M\):

1. \((\forall v \in S_{\mathcal{D}})\ a \equiv_M, v\ b \iff a = b\) (for visible sorts \(\equiv_M\) is equality)
2. \((\forall h \in S - S_{\mathcal{D}})\ a \equiv_M, h\ b \iff (\forall \gamma[*]) [\gamma]_M(a) = [\gamma]_M(b)\)
   (for hidden sorts is the non-distinguish-ability under experiments)

E.g., for streams, \(s \equiv s'\) iff \((\forall i)[hd]_M([tl^i]_M(s)) = [hd]_M([tl^i]_M(s'))\)

\(M\) behavioral satisfies \((\forall X)t = t'\) iff for all \(\theta : X \to M, \theta^*(t) \equiv_M \theta^*(t')\),
where \(\theta^*\) is the extension of \(\theta\) to terms. We write \(M \models (\forall X)t = t'\).

**Notation.** \(D = M|_\mathcal{D}\) (the restriction of \(M\) to \(\mathcal{D}\))
A Model for the “Streams of Bits”

\[ \text{Stream}_M = \mathbb{N}^\infty \]
\[ \text{hd}_M(n_0 : n_1 : n_2 : \ldots) = n_0 \mod 2 \]
\[ \text{tl}_M(n_0 : n_1 : n_2 : \ldots) = n_1 : n_2 : \ldots \]
\[ \text{zip}_M(n_0 : n_1 : n_2 : \ldots, n'_0 : n'_1 : n'_2 : \ldots) = n_0 : n'_0 : n_1 : n'_1 : \ldots \]

\[ w^\infty \overset{\text{def}}{=} w : w : w : \ldots \]

\[ (0 : 1)^\infty \equiv_M 2 : (3 : 4)^\infty \]
A Model for the Regular Expression

\[
[\text{Re}]_M = \mathcal{P}(\{a, b\}^*)
\]

\[
[\text{epsilon}]_M = \{\varepsilon\}
\]

\[
[+]_M(L_1, L_2) = L_1 \cup L_2
\]

\[
\ldots
\]

\[
[-\{-\}]_M(L, x) = \{w \mid xw \in L\}
\]

\[
[\text{epsIn}]_M(L) = (\varepsilon \in L)
\]

\[
L_1 \equiv_M L_2 \text{ iff } L_1 = L_2
\]
Behavioral equivalence lifted up to specifications

Let \((\mathcal{D}, \mathcal{B}, \Delta)\) be a behavioral spec, where \(D = (S_D, \Sigma_D, E_D)\), \(B = (S, \Sigma, E)\)

- consider a sound inference relation \(E \vdash (\forall X) t = t'\)
  - \(\vdash\) could be the equational deduction (complete, but non-practical)
  - \(E \vdash (\forall X) t = t'\) iff \(nf_E(t) = nf_E(t')\), where the normal forms are computed using the equations as rewrite rules, e.g., oriented from left to right (practical, but incomplete)

- define \(E \Vdash (\forall X) t = t'\) as follows:
  - if \(t, t'\) are of visible sort, then \(E \Vdash (\forall X) t = t'\) iff \(E \vdash (\forall X) t = t'\)
  - if \(t, t'\) are of hidden sort, then \(E \Vdash (\forall X) t = t'\) iff \((\forall \gamma)E \vdash (\forall X) \gamma[t] = \gamma[t']\)

Note. We may consider conditional equations, as well.
Soundness

Theorem

Let \((D, B, \Delta)\) be a behavioral spec, where \(D = (S_D, \Sigma_D, E_D)\), \(B = (S, \Sigma, E)\). If \(E \models (\forall X) t = t'\), then \(E \equiv (\forall X) t = t'\).

Note. \(E \equiv e\) iff for each model \(M\) of \((D, B, \Delta)\), \(M \models e\).

Notation. We write \(t \equiv_E t'\), or \(t \equiv t'\) when \(E\) is understood from the context, for \(E \models (\forall X) t = t'\).
Plan
How to prove behavioral equivalence

goal: $t \equiv_E t'$

- coinduction
  - define a relation $R$
  - show that $R \subseteq \equiv_E$
  - show that $E \vdash t \; R \; t'$
- context induction (Hennicker, 1990)
  - uses the inductive definition of the experiments
- both above methods need manual intervention
- circular coinduction
  - first time implemented in BOBJ (Goguen, Roșu, Lin, 1999)
  - at that time Maude did not include reflective capabilities
  - now implemented in CIRC (Lucanu & Roșu, 2007)
- iterative circular coinduction for CoCASL in Isabelle/HOL (Haussman et al., 2005)
Circularity principle

- generalizes circular coinductive deduction
- assume that each equation of interest (to be proved) \( e \) admits
  - a frozen form \( \llbracket e \rrbracket \) and
  - a set of derived equations, its derivatives, \( \text{Der}(e) \)
- the **circularity principle** says that the following statement is valid:
  if from hypotheses \( \mathcal{H} \) together with \( \llbracket e \rrbracket \) we can deduce \( \text{Der}(e) \),
  then \( e \) is a consequence of \( \mathcal{H} \)
- structural induction can also be seen as an instance of the circularity principle
Circular Coinduction in CIRC(1/4)

- an example:
  - the initial goal is $\text{zip}(0^\infty, 1^\infty) \equiv (01)^\infty$
  - compute $\text{hd}(\text{zip}(0^\infty, 1^\infty)) = 0$; $\text{hd}((01)^\infty) = 0$. They are equal. OK.
  - compute $\text{tl}(\text{zip}(0^\infty, 1^\infty)) = \text{zip}(1^\infty, 0^\infty)$; $\text{tl}((01)^\infty) = 1 : (01)^\infty)$. $\text{zip}(1^\infty, 0^\infty) \equiv 1 : (01)^\infty)$ is the new goal.
  - compute $\text{hd}(\text{zip}(1^\infty, 0^\infty)) = 1$; $\text{hd}(1 : (01)^\infty)) = 1$. OK.
  - compute $\text{tl}(\text{zip}(1^\infty, 0^\infty)) = \text{zip}(0^\infty, 1^\infty)$, $\text{tl}(1 : (01)^\infty)) = (01)^\infty$. $\text{zip}(0^\infty, 1^\infty) \equiv (01)^\infty$ should be the new goal but . . .
  - . . . because it is equal to the initial goal (a circularity was found), we conclude that $\text{zip}(0^\infty, 1^\infty) \equiv (01)^\infty$ holds.
Circular Coinduction in CIRC(1/4)

- an example:
  - the initial goal is $\text{zip}(0^\infty, 1^\infty) \equiv (01)^\infty$
  - compute $\text{hd}(\text{zip}(0^\infty, 1^\infty)) = 0$; $\text{hd}((01)^\infty) = 0$. They are equal. OK.
  - compute $\text{tl}(\text{zip}(0^\infty, 1^\infty)) = \text{zip}(1^\infty, 0^\infty)$; $\text{tl}((01)^\infty) = 1 : (01)^\infty$.
    $\text{zip}(1^\infty, 0^\infty) \equiv 1 : (01)^\infty$ is the new goal.
  - compute $\text{hd}(\text{zip}(1^\infty, 0^\infty)) = 1$; $\text{hd}(1 : (01)^\infty)) = 1$. OK.
  - compute $\text{tl}(\text{zip}(1^\infty, 0^\infty)) = \text{zip}(0^\infty, 1^\infty)$; $\text{tl}(1 : (01)^\infty)) = (01)^\infty$.
    $\text{zip}(0^\infty, 1^\infty) \equiv (01)^\infty$ should be the new goal but ...
  - ... because it is equal to the initial goal (a circularity was found), we conclude that $\text{zip}(0^\infty, 1^\infty) \equiv (01)^\infty$ holds.
Circular Coinduction in CIRC(1/4)

an example:

- the initial goal is $\text{zip}(0^\infty, 1^\infty) \equiv (01)^\infty$
- compute $\text{hd}(\text{zip}(0^\infty, 1^\infty)) = 0; \text{hd}((01)^\infty) = 0$. They are equal. OK.
- compute $\text{tl}(\text{zip}(0^\infty, 1^\infty)) = \text{zip}(1^\infty, 0^\infty); \text{tl}((01)^\infty) = 1 : (01)^\infty)$. $\text{zip}(1^\infty, 0^\infty) \equiv 1 : (01)^\infty) \equiv 1 : (01)^\infty)$ is the new goal.
- compute $\text{hd}(\text{zip}(1^\infty, 0^\infty)) = 1; \text{hd}(1 : (01)^\infty)) = 1$. OK.
- compute $\text{tl}(\text{zip}(1^\infty, 0^\infty)) = \text{zip}(0^\infty, 1^\infty), \text{tl}(1 : (01)^\infty)) = (01)^\infty$. $\text{zip}(0^\infty, 1^\infty) \equiv (01)^\infty$ should be the new goal but . . .
- . . .because it is equal to the initial goal (a circularity was found), we conclude that $\text{zip}(0^\infty, 1^\infty) \equiv (01)^\infty$ holds.
Circular Coinduction in CIRC(1/4)

▶ an example:

▶ the initial goal is \( \text{zip}(0^\infty, 1^\infty) \equiv (01)^\infty \)

▶ compute \( \text{hd}(\text{zip}(0^\infty, 1^\infty)) = 0; \text{hd}((01)^\infty) = 0 \). They are equal. OK.

▶ compute \( \text{tl}(\text{zip}(0^\infty, 1^\infty)) = \text{zip}(1^\infty, 0^\infty); \text{tl}((01)^\infty) = 1 : (01)^\infty) \).
\( \text{zip}(1^\infty, 0^\infty) \equiv 1 : (01)^\infty \) is the new goal.

▶ compute \( \text{hd}(\text{zip}(1^\infty, 0^\infty)) = 1; \text{hd}(1 : (01)^\infty)) = 1 \). OK.

▶ compute \( \text{tl}(\text{zip}(1^\infty, 0^\infty)) = \text{zip}(0^\infty, 1^\infty), \text{tl}(1 : (01)^\infty)) = (01)^\infty \).
\( \text{zip}(0^\infty, 1^\infty) \equiv (01)^\infty \) should be the new goal but . . .

▶ . . . because it is equal to the initial goal (a circularity was found), we conclude that \( \text{zip}(0^\infty, 1^\infty) \equiv (01)^\infty \) holds.
Circular Coinduction in CIRC(1/4)

▶ an example:

▶ the initial goal is $\text{zip}(0^\infty, 1^\infty) \equiv (01)^\infty$
▶ compute $hd(\text{zip}(0^\infty, 1^\infty)) = 0$; $hd((01)^\infty) = 0$. They are equal. OK.
▶ compute $tl(\text{zip}(0^\infty, 1^\infty)) = \text{zip}(1^\infty, 0^\infty)$; $tl((01)^\infty) = 1 : (01)^\infty$.
$\text{zip}(1^\infty, 0^\infty) \equiv 1 : (01)^\infty$ is the new goal.
▶ compute $hd(\text{zip}(1^\infty, 0^\infty)) = 1$; $hd(1 : (01)^\infty)) = 1$. OK.
▶ compute $tl(\text{zip}(1^\infty, 0^\infty)) = \text{zip}(0^\infty, 1^\infty)$; $tl(1 : (01)^\infty)) = (01)^\infty$.
$\text{zip}(0^\infty, 1^\infty) \equiv (01)^\infty$ should be the new goal but . . .
▶ . . . because it is equal to the initial goal (a circularity was found), we conclude that $\text{zip}(0^\infty, 1^\infty) \equiv (01)^\infty$ holds.
Circular Coinduction in CIRC(1/4)

- an example:
  - the initial goal is $\text{zip}(0^\infty, 1^\infty) \equiv (01)^\infty$
  - compute $\text{hd}(\text{zip}(0^\infty, 1^\infty)) = 0$; $\text{hd}((01)^\infty) = 0$. They are equal. OK.
  - compute $\text{tl}(\text{zip}(0^\infty, 1^\infty)) = \text{zip}(1^\infty, 0^\infty)$; $\text{tl}((01)^\infty) = 1 : (01)^\infty)$. $\text{zip}(1^\infty, 0^\infty) \equiv 1 : (01)^\infty$ is the new goal.
  - compute $\text{hd}(\text{zip}(1^\infty, 0^\infty)) = 1$; $\text{hd}(1 : (01)^\infty)) = 1$. OK.
  - compute $\text{tl}(\text{zip}(1^\infty, 0^\infty)) = \text{zip}(0^\infty, 1^\infty)$; $\text{tl}(1 : (01)^\infty)) = (01)^\infty$. $\text{zip}(0^\infty, 1^\infty) \equiv (01)^\infty$ should be the new goal but . . .
  - . . . because it is equal to the initial goal (a circularity was found), we conclude that $\text{zip}(0^\infty, 1^\infty) \equiv (01)^\infty$ holds.
Circular Coinduction in CIRC(1/4)

- an example:
  - the initial goal is $\text{zip}(0^\infty, 1^\infty) \equiv (01)^\infty$
  - compute $\text{hd}(\text{zip}(0^\infty, 1^\infty)) = 0$; $\text{hd}((01)^\infty) = 0$. They are equal. OK.
  - compute $\text{tl}(\text{zip}(0^\infty, 1^\infty)) = \text{zip}(1^\infty, 0^\infty)$; $\text{tl}((01)^\infty) = 1 : (01)^\infty)$. $\text{zip}(1^\infty, 0^\infty) \equiv 1 : (01)^\infty$ is the new goal.
  - compute $\text{hd}(\text{zip}(1^\infty, 0^\infty)) = 1$; $\text{hd}(1 : (01)^\infty)) = 1$. OK.
  - compute $\text{tl}(\text{zip}(1^\infty, 0^\infty)) = \text{zip}(0^\infty, 1^\infty)$, $\text{tl}(1 : (01)^\infty)) = (01)^\infty$. $\text{zip}(0^\infty, 1^\infty) \equiv (01)^\infty$ should be the new goal but . . .
  - . . .because it is equal to the initial goal (a circularity was found), we conclude that $\text{zip}(0^\infty, 1^\infty) \equiv (01)^\infty$ holds.
Circular Coinduction in CIRC(2/4)

The frozen form of equation \((\forall X) \ t = t' \text{ if } c\) is

\[
(\forall X) \llbracket t \rrbracket = \llbracket t' \rrbracket \text{ if } c,
\]

where \(\llbracket \cdot \rrbracket : \text{sort}(t) \to \text{new}\) is a new operation and \(\text{new}\) is a new sort.

The set \(\text{Der}_\Delta(e)\) is

\[
\{(\forall X) \llbracket \delta[t/\ast:h] \rrbracket = \llbracket \delta[t'/\ast:h] \rrbracket \text{ if } c \mid \delta[\ast:h] \in \Delta, h = \text{sort}(t)\}.
\]

**Note.** 1. CIRC tool uses the notation \(fr(t)\) for \(\llbracket t \rrbracket\).

2. Conditions \(c\) in equations must of of the form \(\land_i t_i = t'_i\), where \(t_i\) and \(t'_i\) are of visible sort.
Circular Coinduction in CIRC(3/4)

[Normalize]:
\[(\mathcal{E}, \mathcal{G} \cup \{[\forall X] t = t' \text{ if } c\}, \mathcal{S}) \Rightarrow \\
(\mathcal{E}, \mathcal{G} \cup \{[\forall X] nf(t) = nf(t') \text{ if } c\}, \mathcal{S})\]

[EqRed]:
\[(\mathcal{E}, \mathcal{G} \cup \{[e]\}, \mathcal{S}) \Rightarrow (\mathcal{E}, \mathcal{G}, \mathcal{S}) \text{ if } \mathcal{E} \vdash [e]\]

[CoindFail]:
\[(\mathcal{E}, \mathcal{G} \cup \{[e]\}, \mathcal{S}) \Rightarrow \text{failure} \text{ if } \mathcal{E} \nvdash [e] \text{ and } e \text{ is visible}\]

[CCStep]:
\[(\mathcal{E}, \mathcal{G} \cup \{[\forall X][t] = [t'] \text{ if } c\}, \mathcal{S}) \Rightarrow \\
(\mathcal{E} \cup \{[\forall X][t] = [t'] \text{ if } c\}, \\
\mathcal{G} \cup \text{Der}_\Delta([\forall X] t = t' \text{ if } c), \mathcal{S})\]

if \(\mathcal{E} \nvdash (\forall X)[t] = [t'] \text{ if } c\) and \(t, t'\) are of a hidden sort
Circular Coinduction in CIRC(4/4)

[Simpl]:
\[(\mathcal{E}, \mathcal{G} \cup \{e\}, S) \Rightarrow (\mathcal{E}, \mathcal{G} \cup \{(\forall X)\theta(u) = \theta(u')\}, S) \]
if \((S \text{ is } S' \cup \{(\forall X) t = t' \text{ if } u = u' \land \text{scond}\}) \land \)
\((e \text{ is } (\forall X)\theta(t) = \theta(t')) \text{ for some } \theta \land \)
\(\mathcal{E} \vdash (\forall X)\theta([\text{scond}])\)

[Comm]:
\[(\mathcal{E}, \mathcal{G} \cup \{\_op\_ : \text{Stream Stream} \rightarrow \text{Stream} \ [\text{comm}]\}, S) \Rightarrow \]
\((\mathcal{E} \cup \{\_opComm\_ : \text{Stream Stream} \rightarrow \text{Stream \ [comm]}, \)
\((\forall x, y) [x \_op\ y] = [x \_opComm\ y] \}
\(\mathcal{G} \cup \text{Der}(x \_op\ y = y \_op\ x), S)\)

**Note.** Similar rules to [Comm] are added for associativity, idempotency, and identity. Combinations are also possible, but these require some workaround; the details will be given somewhere else.
Correctness of CIRC

Theorem
Let \((\mathcal{D}, \mathcal{B} = (\Sigma, E), \Delta)\) be a behavioral specification, \(e\) a \(\Sigma\)-equation, and \(S\) a set of simplification equations. If \((E, \llbracket e \rrbracket) \Rightarrow^* (E, \emptyset)\) using the procedure above, then \(E \models e\).

Remarks
- the termination is not guaranteed
- the procedure may fail even if the eqn is beh satisfied (false negative answers)
- behavioral equivalence problem is \(\Pi^0_2\)-complete (even for streams)

There are cases when CIRC together with an appropriate beh spec supplies a fully automatic procedure, e.g., extended regular expressions (ERE)
Circular Coinduction

Coinduction Step

[CCStep]:

\[(\mathcal{E}, \mathcal{G} \cup \{ (\forall X)\llbracket t \rrbracket = \llbracket t' \rrbracket \text{ if } c \}, \mathcal{S}) \Rightarrow \]

\[(\mathcal{E} \cup \{ (\forall X)\llbracket t \rrbracket = \llbracket t' \rrbracket \text{ if } c \},\]

\[\mathcal{G} \cup \text{Der}_\Delta((\forall X) t = t' \text{ if } c), \mathcal{S})\]

if \(\mathcal{E} \not\vdash (\forall X)\llbracket t \rrbracket = \llbracket t' \rrbracket \text{ if } c\) and \(t, t'\) are of a hidden sort

- implements Circularity Principle for coinduction

\[
\mathcal{E} \cup \{ (\forall X)\llbracket t \rrbracket = \llbracket t' \rrbracket \text{ if } c \} \vdash \mathcal{G} \cup \text{Der}_\Delta((\forall X) t = t' \text{ if } c)
\]

\[
\mathcal{E} \vdash \mathcal{G} \cup \{ (\forall X)\llbracket t \rrbracket = \llbracket t' \rrbracket \text{ if } c \}
\]

- in terms of proving theory: [CCStep] tries to discover new helpful lemmas; if in the end all these lemmas are proved using the frozen hypothesis, then the initial goals hold

- in terms of bisimulation: [CCStep] tries to discover new bisimilar pairs
Circular coinduction

Simplification Step

[Simpl]:
\[(\mathcal{E}, \mathcal{G} \cup \{[el] \}, S) \Rightarrow (\mathcal{E}, \mathcal{G} \cup \{(\forall X)[\theta(u)] = [\theta(u')]\}, S)\]
if \(S \text{ is } S' \cup \{(\forall X)t = t' \text{ if } u = u' \land \text{ scond}\}\) \land
\((e \text{ is } (\forall X)\theta(t) = \theta(t'))\) for some \(\theta\) \land
\(\mathcal{E} \vdash (\forall X)\theta([\text{scond}])\)

- it is used to simplify goals; a simplification equation can be thought as
  \[(\forall X)\frac{t = t'}{u = u'}\] if scond

- an example of simplification equation is
  \[S_1 + S_2 = S'\]
  \[\frac{S_1 = S'}{S_2 = [0]}\] if \(S_2 = [0]\)

- the correctness is given by the following Modus Ponens like rule:
  \[(\text{MP})\quad (\mathcal{E} \cup S) \vdash (\forall X)t = t' \text{ if } c, \quad \mathcal{E} \vdash c\]
  \[\mathcal{E} \vdash (\forall X)t = t'\]
  provided that \(S\) is sound for \(\mathcal{E}\), i.e., \(\mathcal{E} \models S\).
Special Contexts

- restricting application of circularities to the top of proof goals using the operation \( \llbracket \_ \rrbracket \) excludes many important situations

- e.g., if \( f = 1 : \text{zip}(f, f), \ g = 1 : \text{zip}(g, g) \) and the goal is \( f = g \), then after a derivation with \( \text{tl} \), we obtain \( \text{zip}(f, f) = \text{zip}(g, g) \); now the frozen hypothesis should be applied under the contexts \( \text{zip}(\ast : \text{Stream}, S : \text{Stream}) \) and \( \text{zip}(S : \text{Stream}, \ast : \text{Stream}) \)

- a context is defined similarly to experiments, but the result sort could also be hidden

- under which context is it safe to use the frozen hypothesis?

  [in progress]

- adding the equation

  \[ \llbracket \gamma[X:h] \rrbracket = \llbracket \gamma[X':h] \rrbracket \text{ if } \llbracket X:h \rrbracket := \llbracket X':h \rrbracket \land X:h \neq X':h \]

  to \( \mathcal{E} \), solve the above problem

- can the special contexts be automatically computed?[in progress]
Proof Strategies

- the core of CIRC is a set of reduction rules (nondeterministic procedure)
- a proof tactic means apply these rules in a controlled way
- CIRC uses a strategy language, ROC!, to specify proof tactics

 SYNASC 2008

```act ::= r | act ▼ act | act ◦ act | act !```

- the (extended) coinduction strategy is like
  ([Comm] ▼ [Normalize] ▼ [Simpl] ▼ [EqRed] ▼ [CCstep])!
Plan
Conclusion

A short history of CIRC

- February-March 2006: a first version, as a Maude application, developed by G. Roșu, A. Popescu and D. Lucanu at UIUC
- Autumn 2006: the first major refactoring as Maude metalanguage application (D. Lucanu) [CALCO 2007]
- Spring 2007: regular strategies are added (D. Lucanu, G. Roșu, Gh. Grogoraș) [WRS 2007]
- Autumn 2007: CIRC become a funded project (PN II ID 393, Romanian Government); G. Caltais (Goriac) and E. Goriac enjoy CIRC team
- Spring 2008: the second major refactoring based on patterns and the strategy language ROC! [WRLA 2008, SYNASC 2008]
Conclusion

- CIRC tool implements in Maude Circularity Principle using reflection property of RWL
- CIRC extends Circularity Principle with other capabilities: simplification, special contexts, case analysis
- the proof tactics for CIRC can be described using rewriting strategies written in ROC!
- the theoretical background of CIRC is behavioral logic, known also as hidden logic
- CIRC can be easily extended with other capabilities, e.g., combination of induction with coinduction
- case studies: streams (over rings, fields), extended regular expressions, infinite trees, equivalence of programs [in progress]
- things to do: refactor induction engine, finding automatically special contexts, bisimulation of lts specified as rewrite theories, application to program equivalence, integration ...
Thanks!

circ.jpg