Circular Coinduction-based Techniques for Proving Behavioral Properties

Dorel Lucanu

Al. I. Cuza Univ., Iași, 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: $\texttt{Bit} = \{0, 1\}$ and $\texttt{BitStream}$ $(S = b_0 b_1 b_2 \ldots)$
  - two behavioral operations:

    $hd : \texttt{BitStream} \to \texttt{Bit} \quad (hd(S) = b_0)$

    $tl : \texttt{BitStream} \to \texttt{BitStream} \quad (tl(S) = b_1 b_2 \ldots)$
  - another operation:

    $zip : \texttt{BitStream}\ \texttt{BitStream} \to \texttt{Bit}$

    $zip(b_0 b_1 \ldots, b'_0 b'_1 \ldots) = b_0 b'_0 b_1 b'_1 \ldots$
- define behavioral equivalence $\equiv$ over $\texttt{BitStream}$ by: $S_1 \equiv S_2$ iff $hd(S_1) = hd(S_2)$ and $tl(S_1) \equiv tl(S_2)$
- how do we prove that $zip(0^\infty, 1^\infty) \equiv (01)^\infty$? [1]

[1] $w^\infty \stackrel{\text{def}}{=} w : w : w : \ldots$

Example: Streams 1/3

- consider:
  - two datatypes: Bit $= \{0, 1\}$ and BitStream ($S = b_0 b_1 b_2 \ldots$)
  - two behavioral operations:

$$\begin{array}{lll} hd : \texttt{BitStream} \to \texttt{Bit} & (hd(S) = b_0) \\ tl : \texttt{BitStream} \to \texttt{BitStream} & (tl(S) = b_1 b_2 \ldots) \end{array}$$

  - another operation:

$$zip : \texttt{BitStream BitStream} \to \texttt{Bit}$$
$$zip(b_0 b_1 \ldots, b'_0 b'_1 \ldots) = b_0 b'_0 b_1 b'_1 \ldots$$

- define behavioral equivalence $\equiv$ over BitStream by: $S_1 \equiv S_2$ iff $hd(S_1) = hd(S_2)$ and $tl(S_1) \equiv tl(S_2)$
- how do we prove that $zip(0^\infty, 1^\infty) \equiv (01)^\infty$? [1]

[1] $w^\infty \stackrel{\text{def}}{=} w : w : w : \ldots$

Example: Streams 1/3

- consider:
  - two datatypes: $\texttt{Bit} = \{0,1\}$ and $\texttt{BitStream}$ $(S = b_0 b_1 b_2 \ldots)$
  - two behavioral operations:

    $$\begin{array}{ll} hd : \texttt{BitStream} \to \texttt{Bit} & (hd(S) = b_0) \\ tl : \texttt{BitStream} \to \texttt{BitStream} & (tl(S) = b_1 b_2 \ldots) \end{array}$$

  - another operation:

    $$zip : \texttt{BitStream}\ \texttt{BitStream} \to \texttt{Bit}$$

    $$zip(b_0 b_1 \ldots, b'_0 b'_1 \ldots) = b_0 b'_0 b_1 b'_1 \ldots$$

- define behavioral equivalence $\equiv$ over $\texttt{BitStream}$ by:
  $S_1 \equiv S_2$ iff $hd(S_1) = hd(S_2)$ and $tl(S_1) \equiv tl(S_2)$
- how do we prove that $zip(0^\infty, 1^\infty) \equiv (01)^\infty$? [1]

[1] $w^\infty \stackrel{\text{def}}{=} w : w : w : \ldots$

Example: Streams 1/3

- consider:
  - two datatypes: `Bit` $= \{0,1\}$ and `BitStream` $(S = b_0 b_1 b_2 \ldots)$
  - two behavioral operations:

    $$\begin{array}{lll} hd : \texttt{BitStream} \to \texttt{Bit} & (hd(S) = b_0) \\ tl : \texttt{BitStream} \to \texttt{BitStream} & (tl(S) = b_1 b_2 \ldots) \end{array}$$
  - another operation:

    $$zip : \texttt{BitStream}\ \texttt{BitStream} \to \texttt{Bit}$$
    $$zip(b_0 b_1 \ldots, b'_0 b'_1 \ldots) = b_0 b'_0 b_1 b'_1 \ldots$$
- define behavioral equivalence $\equiv$ over `BitStream` by:
  $S_1 \equiv S_2$ iff $hd(S_1) = hd(S_2)$ and $tl(S_1) \equiv tl(S_2)$
- how do we prove that $zip(0^\infty, 1^\infty) \equiv (01)^\infty$? [1]

[1] $w^\infty \stackrel{\text{def}}{=} w : w : w : \ldots$

Example: Streams 1/3

- consider:
  - two datatypes: $\texttt{Bit} = \{0, 1\}$ and $\texttt{BitStream}$ $(S = b_0 b_1 b_2 \ldots)$
  - two behavioral operations:

    $$\begin{array}{ll} hd : \texttt{BitStream} \to \texttt{Bit} & (hd(S) = b_0) \\ tl \ : \texttt{BitStream} \to \texttt{BitStream} & (tl(S) \ = b_1 b_2 \ldots) \end{array}$$
  - another operation:

    $$zip : \texttt{BitStream BitStream} \to \texttt{Bit}$$

    $$zip(b_0 b_1 \ldots, b'_0 b'_1 \ldots) = b_0 b'_0 b_1 b'_1 \ldots$$
- define behavioral equivalence $\equiv$ over $\texttt{BitStream}$ by: $S_1 \equiv S_2$ iff $hd(S_1) = hd(S_2)$ and $tl(S_1) \equiv tl(S_2)$
- how do we prove that $zip(0^\infty, 1^\infty) \equiv (01)^\infty$? [1]

[1] $w^\infty \stackrel{\text{def}}{=} w : w : w : \ldots$

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 *Alph* be an alphabet.
The extended regular expressions over *Alph*:

$$R ::= \varepsilon \mid \emptyset \mid A \mid R_1 + R_2 \mid R_1 \# R_2 \mid R^* \mid R_1 \cap R_2 \mid \neg R$$

where A ranges over *Alph*.
The behavioral operations (derivatives)[Brzozowski]:

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

the behavioral equivalence:

$\mathtt{R} \equiv \mathtt{R}'$ iff $\mathtt{epsInR} = \mathtt{epsinR}'$ and $(\forall \mathtt{a})\, \mathtt{R}\{\mathtt{a}\} \equiv \mathtt{R}'\{\mathtt{a}\}$

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 $(\mathcal{D}, \mathcal{B}, \Delta)$, where

- $\mathcal{D} = (S_D, \Sigma_D, E_D)$ specifies data,
- $\mathcal{B} = (S, \Sigma, E)$ specifies further the behavioral operations
- there is an inclusion $\mathcal{D} \hookrightarrow \mathcal{B}$,
- the elements of S_D are visible sorts; e.g., `Elt`
- the elements of $S - S_D$ are hidden sorts; e.g., `Stream`
- $\Delta \subseteq \mathtt{Der}(\Sigma)$ whose operations are called derivatives (behavioral opns) Δ; e.g., $\mathtt{hd}(* : \mathtt{Stream})$, $\mathtt{tl}(* : \mathtt{Stream})$
  $*$ 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., $\mathtt{hd}(* : \mathtt{Stream})$
  if $\gamma[* : h]$ is an experiment and $(\delta : w \to h) \in \Delta$,
  then $\gamma[\delta/*]$ is an experiment;
  e.g., $\mathtt{hd}(\mathtt{tl}(* : \mathtt{Stream})), \mathtt{hd}(\mathtt{tl}(\mathtt{tl}(* : \mathtt{Stream}))), \ldots$

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 $\equiv_M$:

- $(\forall v \in S_D)\ a \equiv_{M,v} b$ iff $a = b$ (for visible sorts $\equiv_M$ is equality)
- $(\forall h \in S - S_D)\ a \equiv_{M,h} b$ 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 $(\forall i)[\![\mathtt{hd}]\!]_{\mathtt{M}}([\![\mathtt{tl}^{\mathtt{i}}]\!]_{\mathtt{M}}(\mathtt{s})) = [\![\mathtt{hd}]\!]_{\mathtt{M}}([\![\mathtt{tl}^{\mathtt{i}}]\!]_{\mathtt{M}}(\mathtt{s}'))$

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

Notation. $D = M|_{\mathcal{D}}$ (the restriction of M to $\mathcal{D}$)

A Model for the "Streams of Bits"

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

$w^\infty \stackrel{\text{def}}{=} w : w : w : \ldots$

$(0 : 1)^\infty \equiv_M 2 : (3 : 4)^\infty$

A Model for the Regular Expression

$[\![\mathtt{Re}]\!]_M = \mathcal{P}(\{a, b\}^*)$
$[\![\mathtt{epsilon}]\!]_M = \{\varepsilon\}$
$[\![+]\!]_M(L_1, L_2) = L_1 \cup L_2$
$\ldots$

$[\![\_\{\_\}]\!]_M(L, x) = \{w \mid xw \in L\}$
$[\![\mathtt{epsIn}]\!]_M(L) = (\varepsilon \in L)$

$L_1 \equiv_M L_2$ iff $L_1 = L_2$

Behavioral equivalence lifted up to specifications

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

- 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 $(D, \mathcal{B}, \Delta)$ be a behavioral spec, where $D = (S_D, \Sigma_D, E_D)$, $\mathcal{B} = (S, \Sigma, E)$. If $E \Vdash (\forall X)\, t = t'$, then $E \models\!\!\equiv (\forall X)\, t = t'$.

Note. $E \models\!\!\equiv e$ iff for each model M of $(D, \mathcal{B}, \Delta)$, $M \models\!\!\equiv e$.

Notation. We write $t \equiv_E t'$, or $t \equiv t'$ when E is understood from the context, for $E \models\!\!\equiv (\forall X) t = t'$.

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 $[\![e]\!]$ and
  - a set of derived equations, its derivatives, $Der(e)$
- the circularity principle says that the following statement is valid: if from hypotheses $\mathcal{H}$ together with $[\![e]\!]$ we can deduce $Der(e)$, 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 $zip(0^\infty, 1^\infty) \equiv (01)^\infty$
  - compute $hd(zip(0^\infty, 1^\infty)) = 0$; $hd((01)^\infty) = 0$. They are equal. OK.
  - compute $tl(zip(0^\infty, 1^\infty)) = zip(1^\infty, 0^\infty)$; $tl((01)^\infty) = 1 : (01)^\infty)$. $zip(1^\infty, 0^\infty) \equiv 1 : (01)^\infty)$ is the new goal.
  - compute $hd(zip(1^\infty, 0^\infty)) = 1$; $hd(1 : (01)^\infty)) = 1$. OK.
  - compute $tl(zip(1^\infty, 0^\infty)) = zip(0^\infty, 1^\infty)$, $tl(1 : (01)^\infty)) = (01)^\infty$. $zip(0^\infty, 1^\infty) \equiv (01)^\infty$ should be the new goal but ...
  - ... because it is equal to the initial goal (a circularity was found), we conclude that $zip(0^\infty, 1^\infty) \equiv (01)^\infty$ holds.

Circular Coinduction in CIRC(1/4)

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

Circular Coinduction in CIRC(1/4)

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

Circular Coinduction in CIRC(1/4)

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

Circular Coinduction in CIRC(1/4)

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

Circular Coinduction in CIRC(1/4)

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

Circular Coinduction in CIRC(1/4)

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

Circular Coinduction in CIRC(2/4)

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

$$(\forall X)[\![t]\!] = [\![t']\!] \texttt{ if } c,$$

where $[\![\_]\!] : \mathit{sort}(t) \to \texttt{new}$ is a new operation and
$\texttt{new}$ is a new sort

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

$$\{(\forall X)\ [\![\delta[t/*{:}h]]\!] = [\![\delta[t'/*{:}h]]\!] \texttt{ if } c \mid \delta[*{:}h \in \Delta, h = \mathit{sort}(t)\}.$$

Note. 1. CIRC tool uses the notation $\mathit{fr}(t)$ for $[\![t]\!]$.
2. Conditions c in equations must of of the form $\wedge_i t_i = t'_i$, where t_i and t'_i are of visible sort.

Circular Coinduction in CIRC(3/4)

[Normalize]:

$(\mathcal{E}, \mathcal{G} \cup \{[\![(\forall X)t = t' \texttt{ if } c]\!]\}, \mathcal{S}) \Rightarrow$
$(\mathcal{E}, \mathcal{G} \cup \{[\![(\forall X)nf(t) = nf(t') \texttt{ if } c]\!]\}, \mathcal{S})$

[EqRed]:

$(\mathcal{E}, \mathcal{G} \cup \{[\![e]\!]\}, \mathcal{S}) \Rightarrow (\mathcal{E}, \mathcal{G}, \mathcal{S})$ if $\mathcal{E} \vdash [\![e]\!]$

[CoindFail]:

$(\mathcal{E}, \mathcal{G} \cup \{[\![e]\!]\}, \mathcal{S}) \Rightarrow \mathit{failure}$ if $\mathcal{E} \not\vdash [\![\mathrm{e}]\!]$ and e is visible

[CCStep]:

$(\mathcal{E}, \mathcal{G} \cup \{(\forall X)[\![t]\!] = [\![t']\!] \texttt{ if } c\}, \mathcal{S}) \Rightarrow$
$(\mathcal{E} \cup \{(\forall X)[\![t]\!] = [\![t']\!] \texttt{ if } c\},$
$\mathcal{G} \cup Der_{\Delta}((\forall X)t = t' \texttt{ if } c), \mathcal{S})$
if $\mathcal{E} \not\vdash (\forall X)[\![t]\!] = [\![t']\!] \texttt{ if } c$ and t, t' are of a hidden sort

Circular Coinduction in CIRC(4/4)

[Simpl]:

$$(\mathcal{E}, \mathcal{G} \cup \{[\![e]\!]\}, \mathcal{S}) \Rightarrow (\mathcal{E}, \mathcal{G} \cup \{(\forall X)[\![\theta(u)]\!] = [\![\theta(u')]\!]\}, \mathcal{S})$$
$$\text{if } (\mathcal{S} \text{ is } \mathcal{S}' \cup \{(\forall X)\, t = t' \texttt{ if } u = u' \wedge \mathit{scond}\}) \wedge$$
$$(e \text{ is } (\forall X)\theta(t) = \theta(t')) \text{ for some } \theta \wedge$$
$$\mathcal{E} \vdash (\forall X)\theta([\![\mathit{scond}]\!])$$

[Comm]:

$$(\mathcal{E}, \mathcal{G} \cup \{\_op\_ : \texttt{Stream Stream} \to \texttt{Stream [comm]}\}, \mathcal{S}) \Rightarrow$$
$$(\mathcal{E} \cup \{\ \_opComm\_ : \texttt{Stream Stream} \to \texttt{Stream [comm]},$$
$$(\forall x, y)[\![x\ op\ y]\!] = [\![x\ opComm\ y]\!]\}$$
$$\mathcal{G} \cup \mathit{Der}(x\ op\ y = y\ op\ x)\}, \mathcal{S})$$

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 Σ-equation, and $\mathcal{S}$ a set of simplification equations. If $(E, [\![e]\!]) \Rightarrow^\star (\mathcal{E}, \emptyset)$ using the procedure above, then $E \Vdash e$.

Remarks

- the termination is not guaranteed
- the procedure may fail even if the eqn is beh satisfied (false negative answers)
- behavioral equivalence problem is Π_2^0-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)

Coinduction Step

[CCStep]:

$(\mathcal{E}, \mathcal{G} \cup \{(\forall X)[\![t]\!] = [\![t']\!] \ \mathtt{if} \ c\}, \mathcal{S}) \Rightarrow$
$(\mathcal{E} \cup \{(\forall X)[\![t]\!] = [\![t']\!] \ \mathtt{if} \ c\},$
$\mathcal{G} \cup Der_\Delta((\forall X)t = t' \ \mathtt{if} \ c), \mathcal{S})$
if $\mathcal{E} \not\Vdash (\forall X)[\![t]\!] = [\![t']\!] \ \mathtt{if} \ c$ and t, t' are of a hidden sort

- implements Circularity Principle for coinduction

$$\frac{\mathcal{E} \cup \{(\forall X)[\![t]\!] = [\![t']\!] \ \mathtt{if} \ c\} \Vdash \mathcal{G} \cup Der_\Delta((\forall X)t = t' \ \mathtt{if} \ c)}{\mathcal{E} \Vdash \mathcal{G} \cup \{(\forall X)[\![t]\!] = [\![t']\!] \ \mathtt{if} \ c\}}$$

- 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

Simplification Step

[Simpl]:

$$(\mathcal{E}, \mathcal{G} \cup \{[\![e]\!]\}, \mathcal{S}) \Rightarrow (\mathcal{E}, \mathcal{G} \cup \{(\forall X)[\![\theta(u)]\!] = [\![\theta(u')]\!]\}, \mathcal{S})$$

$$\text{if } (\mathcal{S} \text{ is } \mathcal{S}' \cup \{(\forall X)\, t = t' \ \mathtt{if}\ u = u' \wedge \mathit{scond}\}) \wedge$$

$$(e \text{ is } (\forall X)\theta(t) = \theta(t')) \ \text{ for some } \theta \wedge$$

$$\mathcal{E} \vdash (\forall X)\theta([\![\mathit{scond}]\!])$$

- it is used to simplify goals; a simplification equation can be thought as
$$(\forall X)\frac{t = t'}{u = u'} \quad \text{if } \mathit{scond}$$
- an example of simplification equation is
$$\frac{S_1 + S_2 = S'}{S_1 = S'} \quad \text{if } S_2 = [0]$$
- the correctness is given by the following Modus Ponens like rule:
$$(MP)\ \frac{(\mathcal{E} \cup \mathcal{S}) \vdash (\forall X)t = t' \ \mathtt{if}\ c, \quad \mathcal{E} \vdash c}{\mathcal{E} \vdash (\forall X)t = t'}$$

provided that $\mathcal{S}$ is sound for $\mathcal{E}$, i.e., $\mathcal{E} \models \mathcal{S}$.

Special Contexts

- restricting application of circularities to the top of proof goals using the operation $[\![\_]\!]$ excludes many important situations
- e.g., if $f = 1 : \mathtt{zip}(f, f),\ \ g = 1 : \mathtt{zip}(g, g)$ and the goal is $f = g$, then after a derivation with $\mathtt{tl}$, we obtain $\mathtt{zip}(f, f) = \mathtt{zip}(g, g)$; now the frozen hypothesis should be applied under the contexts $\mathtt{zip}(*{:}\mathtt{Stream}, S{:}\mathtt{Stream})$ and $\mathtt{zip}(S{:}\mathtt{Stream}, *{:}\mathtt{Stream})$
- 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
  $$[\![\gamma[X{:}h]]\!] = [\![\gamma[X'{:}h]]\!]\ \mathtt{if}\ [\![X{:}h]\!] := [\![X'{:}h]\!] \wedge X{:}h \neq X'{:}h$$
  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)

  $act ::= r \mid act \triangleright act \mid act \circ act \mid act\,!$
- the (extended) coinduction strategy is like

  ([Comm] $\triangleright$ [Normalize] $\triangleright$ [Simpl] $\triangleright$ [EqRed] $\triangleright$ [CCstep])!

Plan

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, fieds), 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