Coinductive Methods in Computer Science
(and Beyond)

Filippo Bonchi
CNRS, Ens-Lyon

Udine 9/5/2016

Coinduction

Mathematical principle (dual to induction) to prove properties of infinite structures like, e.g., infinite lists, infinite trees, or the behaviour of a control system

Coinduction is nowadays recognised as one of the fundamental principle of computer science
Computer Science

Proving properties of computing systems

- Different kinds of properties
  Qualitative, Quantitative, Safety, Liveness
- Different kinds of systems
  Concurrent, Probabilistic, Hybrid, ...
- Different kinds of proofs
  Fully-automated (Model Checking)
  Semi-automated (Proof assistant)

... and beyond...

in economy

Pierre Lescanne, Matthieu Perrinel:

Samson Abramsky, Viktor Winschel:

Samson Abramsky, Alexander Kurz, Pierre Lescanne, Viktor Winschel:
... and beyond...

in analysis...

Dusko Pavlovic, Martín Hötzel Escardó:
*Calculus in Coinductive Form.* LICS 1998: 408-417

Ichiro Hasuo, Bart Jacobs, Milad Niqui:
*Coalgebraic Representation Theory of Fractals.*

and ecology.

Michael Hauhs, Baltasar Trancón y Widemann:
*Applications of Algebra and Coalgebra in Scientific Modelling:*
*Illustrated with the Logistic Map.*

Multidisciplinary origins


- **Concurrency Theory**

  *Communication and Concurrency.*
  Prentice Hall.

  *Concurrency on automata and infinite sequences.*
  In Lecture Notes in Computer Science, vol. 104. Springer.

- **Philosophical Logic**

  BENTHEM, J. V. 1976.
  *Modal correspondence theory.*

- **Set Theory**

  *Set theory with free construction principles.*
Coinduction was not necessary for automata theory

But many classical results can be proved by mean of coinduction

Proofs are usually simpler and nicer

Novel results (and algorithms) can follow from a coinductive perspective

Plan

1. Fixpoint Theorems, Coinduction and Algorithms
2. Up-to Techniques
3. HKC: Hopcroft and Karp up-to Congruence
4. Weighted Automata
5. A Coinductive Proof of Moessner Theorem
At the black-board

1. Posets and complete lattices
2. Knaster-Tarski fixpoint theorem and coinduction
3. Kleene fixpoint theorem
4. DFAs, language equivalence as greatest fixpoint
5. Partition Refinement algorithm

\[ \text{Naive}(x, y) \]

\begin{enumerate}
\item \( R \) is empty; \textit{todo} is empty;
\item insert \((x, y)\) in \textit{todo};
\item while \textit{todo} is not empty, do
  \begin{enumerate}
  \item extract \((x', y')\) from \textit{todo};
  \item if \((x', y')\) \in \( R \) then continue;
  \item if \( o(x') \neq o(y') \) then return \textit{false};
  \item for all \( a \in A \),
    insert \((t_a(x'), t_a(y'))\) in \textit{todo};
  \item insert \((x', y')\) in \( R \);
  \end{enumerate}
\item return \textit{true};
\end{enumerate}

\textit{todo} = \emptyset

\( R = \emptyset \)
**Naive(x, y)**

1. *R* is empty; *todo* is empty;
2. insert (*x*, *y*) in *todo*;
3. while *todo* is not empty, do
   3.1 extract (*x’, y’*) from *todo*;
   3.2 if (*x’, y’*) \( \in R \) then continue;
   3.3 if \( o(x’) \neq o(y’) \) then return false;
   3.4 for all \( a \in A \),
      insert \( (t_a(x’), t_a(y’)) \) in *todo*;
4. return true;

\[ \text{todo} = \{ (x, u) \} \]
\[ R = \emptyset \]

\[ x \xrightarrow{a} y \xleftrightarrow{a} z \]
\[ u \xrightarrow{a} v \xleftrightarrow{a} w \]
Naive algorithm for checking the equivalence of states

(1) $R$ is empty; todo is empty;
(2) insert $(x, y)$ in todo;
(3) while todo is not empty, do
   (3.1) extract $(x', y')$ from todo;
   (3.2) if $(x', y') \in R$ then continue;
   (3.3) if $o(x') \neq o(y')$ then return false;
   (3.4) for all $a \in A$,
      insert $(t_a(x'), t_a(y'))$ in todo;
   (3.5) insert $(x', y')$ in $R$;
(4) return true;

\[
\begin{align*}
\text{todo} & = \emptyset \quad (x, u) \\
R & = \emptyset \\
\end{align*}
\]

\[
\begin{align*}
x & \xrightarrow{a} y \xleftarrow{a} z \\
u & \xrightarrow{a} v \xleftarrow{a} w
\end{align*}
\]
Naive algorithm for checking the equivalence of states

\[ \text{Naive}(x, y) \]

(1) \( R \) is empty; \( todo \) is empty;
(2) insert \((x, y)\) in \( todo \);
(3) while \( todo \) is not empty, do
   (3.1) extract \((x', y')\) from \( todo \);
   (3.2) if \( (x', y') \in R \) then continue;
   (3.3) if \( o(x') \neq o(y') \) then return false;
   (3.4) for all \( a \in A \),
       insert \((t_a(x'), t_a(y'))\) in \( todo \);
(3.5) insert \((x', y')\) in \( R \);
(4) return true;

\[ \begin{array}{c}
     x \xleftarrow{a} y \xleftrightarrow{a} z \\
     u \xrightarrow{a} v \xleftrightarrow{a} w
\end{array} \]
Naive$(x,y)$

(1) $R$ is empty; $todo$ is empty;
(2) insert $(x,y)$ in $todo$;
(3) while $todo$ is not empty, do
  (3.1) extract $(x',y')$ from $todo$;
  (3.2) if $(x',y') \in R$ then continue;
  (3.3) if $o(x') \neq o(y')$ then return false;
  (3.4) for all $a \in A$, insert $(t_a(x'), t_a(y'))$ in $todo$;
(3.5) insert $(x',y')$ in $R$;
(4) return true;

\[
\begin{align*}
\text{todo} &= \{(y,v)\} \quad (x,u) \\
R &= \{(x,u)\}
\end{align*}
\]

\[
\begin{align*}
x &\xrightarrow{a} \overline{y} \quad a \\
\overline{y} &\xleftarrow{a} z
\end{align*}
\]

\[
\begin{align*}
u &\xrightarrow{a} \overline{v} \quad a \\
\overline{v} &\xleftarrow{a} w
\end{align*}
\]
Naive($x,y$)

(1) $R$ is empty; todo is empty;
(2) insert $(x,y)$ in todo;
(3) while todo is not empty, do
   (3.1) extract $(x',y')$ from todo;
   (3.2) if $(x',y') \in R$ then continue;
   (3.3) if $o(x') \neq o(y')$ then return false;
   (3.4) for all $a \in A$,
       insert $(t_a(x'), t_a(y'))$ in todo;
(3.5) insert $(x',y')$ in $R$;
(4) return true;

\begin{align*}
\begin{array}{c}
todo = \{(y,v)\} \\
R = \{(x,u)\}
\end{array}
\end{align*}

\begin{align*}
\begin{array}{c}
x \xrightarrow{a} y \quad y \xleftarrow{a} z \\
\quad 1 & 1 \\
\end{array}
\end{align*}

Naive($x,y$)

(1) $R$ is empty; todo is empty;
(2) insert $(x,y)$ in todo;
(3) while todo is not empty, do
   (3.1) extract $(x',y')$ from todo;
   (3.2) if $(x',y') \in R$ then continue;
   (3.3) if $o(x') \neq o(y')$ then return false;
   (3.4) for all $a \in A$,
       insert $(t_a(x'), t_a(y'))$ in todo;
(3.5) insert $(x',y')$ in $R$;
(4) return true;

\begin{align*}
\begin{array}{c}
todo = \emptyset \\
R = \{(x,u)\}
\end{array}
\end{align*}

\begin{align*}
\begin{array}{c}
x \xrightarrow{a} y \quad y \xleftarrow{a} z \\
\quad 1 & 1 \\
\end{array}
\end{align*}
Naive(x, y)

(1) \( R \) is empty; \( todo \) is empty;
(2) insert \((x, y)\) in \( todo \);
(3) while \( todo \) is not empty, do
   (3.1) extract \((x', y')\) from \( todo \);
   (3.2) if \((x', y')\) is already in \( R \) then continue;
   (3.3) if \( o(x') \neq o(y') \) then return \textit{false};
   (3.4) for all \( a \in A \),
      insert \((t_a(x'), t_a(y'))\) in \( todo \);
(3.5) insert \((x', y')\) in \( R \);
(4) return \textit{true};
Naive($x, y$)

(1) $R$ is empty; $todo$ is empty;
(2) insert ($x, y$) in $todo$;
(3) while $todo$ is not empty, do
(3.1) extract ($x', y'$) from $todo$;
(3.2) if ($x', y'$) $\in$ $R$ then continue;
(3.3) if $o(x') \neq o(y')$ then return false;
(3.4) for all $a \in A$,
insert ($t_a(x'), t_a(y')$) in $todo$;
(3.5) insert ($x', y'$) in $R$;
(4) return true;

\[
\begin{align*}
todo &= \{(z, w)\} \quad (y, v) \\
R &= \{(x, u)\}
\end{align*}
\]
\[
\text{Naive}(x, y)
\]

(1) \(R\) is empty; \(todo\) is empty;
(2) insert \((x, y)\) in \(todo\);
(3) while \(todo\) is not empty, do
  (3.1) extract \((x', y')\) from \(todo\);
  (3.2) if \((x', y') \in R\) then continue;
  (3.3) if \(o(x') \neq o(y')\) then return \(false\);
  (3.4) for all \(a \in A\),
    insert \((t_a(x'), t_a(y'))\) in \(todo\);
  (3.5) insert \((x', y')\) in \(R\);
(4) return \(true\);

\[
\text{Naive}(x, y)
\]

(1) \(R\) is empty; \(todo\) is empty;
(2) insert \((x, y)\) in \(todo\);
(3) while \(todo\) is not empty, do
  (3.1) extract \((x', y')\) from \(todo\);
  (3.2) if \((x', y') \in R\) then continue;
  (3.3) if \(o(x') \neq o(y')\) then return \(false\);
  (3.4) for all \(a \in A\),
    insert \((t_a(x'), t_a(y'))\) in \(todo\);
  (3.5) insert \((x', y')\) in \(R\);
(4) return \(true\);
Naive\((x, y)\)

(1) \(R\) is empty; \(todo\) is empty;
(2) insert \((x, y)\) in \(todo\);
(3) while \(todo\) is not empty, do
(3.1) extract \((x', y')\) from \(todo\);
(3.2) if \((x', y')\) ∈ \(R\) then continue;
(3.3) if \(o(x') \neq o(y')\) then return false;
(3.4) for all \(a \in A\),
    insert \((t_a(x'), t_a(y'))\) in \(todo\);
(3.5) insert \((x', y')\) in \(R\);
(4) return true;

\[
\begin{align*}
todo &= \emptyset & (z, w) \\
R &= \{(x, u), (y, v)\} & \begin{array}{c}
x \xrightarrow{a} y \xleftrightarrow{a} z \\
\end{array} \\
\end{align*}
\]
Naive$(x, y)$

1. $R$ is empty; $todo$ is empty;
2. insert $(x, y)$ in $todo$;
3. while $todo$ is not empty, do
   3.1 extract $(x', y')$ from $todo$;
   3.2 if $(x', y') \in R$ then continue;
   3.3 if $o(x') \neq o(y')$ then return false;
3.4 for all $a \in A$,
   insert $(t_a(x'), t_a(y'))$ in $todo$;
3.5 insert $(x', y')$ in $R$;
4. return true;

$todo = \emptyset$  
\[
\begin{array}{c|c|c}
1 & 2 \\
\hline
x & \overrightarrow{a} & \overleftarrow{a} \\
\mid & \mid & \mid \\
\end{array}
\]

$R = \{(x, u), (y, v)\}$  
\[
\begin{array}{c|c|c}
1 & 2 \\
\hline
x & \overrightarrow{a} & \overleftarrow{a} \\
\mid & \mid & \mid \\
\end{array}
\]

Naive$(x, y)$

1. $R$ is empty; $todo$ is empty;
2. insert $(x, y)$ in $todo$;
3. while $todo$ is not empty, do
   3.1 extract $(x', y')$ from $todo$;
   3.2 if $(x', y') \in R$ then continue;
   3.3 if $o(x') \neq o(y')$ then return false;
3.4 for all $a \in A$,
   insert $(t_a(x'), t_a(y'))$ in $todo$;
3.5 insert $(x', y')$ in $R$;
4. return true;

$todo = \{(x, u)\}$  
\[
\begin{array}{c|c|c}
1 & 2 \\
\hline
x & \overrightarrow{a} & \overleftarrow{a} \\
\mid & \mid & \mid \\
\end{array}
\]

$R = \{(x, u), (y, v)\}$  
\[
\begin{array}{c|c|c}
1 & 2 \\
\hline
x & \overrightarrow{a} & \overleftarrow{a} \\
\mid & \mid & \mid \\
\end{array}
\]


**Naive**($x, y$)

1. $R$ is empty; todo is empty;
2. insert $(x, y)$ in todo;
3. while todo is not empty, do
   3.1 extract $(x', y')$ from todo;
   3.2 if $(x', y') \in R$ then continue;
   3.3 if $o(x') \neq o(y')$ then return false;
   3.4 for all $a \in A$, 
       insert $(t_a(x'), t_a(y'))$ in todo;
3.5 insert $(x', y')$ in $R$;
4. return true;

\[ x \xrightarrow{a} y \xleftarrow{a} z \]
\[ R = \{(x, u), (y, v), (z, w)\} \]
\[ todo = \{(x, u)\} \quad (z, w) \]

$R$ is not empty,

pseudo code:

```plaintext
Naive(x, y)

1. R is empty; todo is empty;
2. insert (x, y) in todo;
3. while todo is not empty, do
   3.1 extract (x', y') from todo;
   3.2 if (x', y') \in R then continue;
   3.3 if o(x') \neq o(y') then return false;
   3.4 for all a \in A,
       insert (t_a(x'), t_a(y')) in todo;
3.5 insert (x', y') in R;
4. return true;
```

**Naive**($x, y$)

1. $R$ is empty; todo is empty;
2. insert $(x, y)$ in todo;
3. while todo is not empty, do
   3.1 extract $(x', y')$ from todo;
   3.2 if $(x', y') \in R$ then continue;
   3.3 if $o(x') \neq o(y')$ then return false;
   3.4 for all $a \in A$, 
       insert $(t_a(x'), t_a(y'))$ in todo;
3.5 insert $(x', y')$ in $R$;
4. return true;

\[ x \xrightarrow{a} y \xleftarrow{a} z \]
\[ R = \{(x, u), (y, v), (z, w)\} \]
\[ todo = \{(x, u)\} \quad (z, w) \]

$R$ is not empty,
Naive\((x, y)\)
(1) \(R\) is empty; \(todo\) is empty;
(2) insert \((x, y)\) in \(todo\);
(3) while \(todo\) is not empty, do
  (3.1) extract \((x', y')\) from \(todo\);
  (3.2) if \((x', y')\) \(\in\) \(R\) then continue;
  (3.3) if \(o(x') \neq o(y')\) then return false;
  (3.4) for all \(a \in A\),
      insert \((t_a(x'), t_a(y'))\) in \(todo\);
  (3.5) insert \((x', y')\) in \(R\);
(4) return true;

\(todo\) = \{(x, u)\}  \quad \{z, w\}
\begin{align*}
R &= \{(x, u), (y, v), (z, w)\} \\
&\begin{array}{c|c|c|
\hline
1 & 2 & 3 \\
\hline
\end{array}
\begin{array}{c}
\hline
u \xrightarrow{a} v \quad \leftrightarrow \quad a \quad \xrightarrow{a} w \\
\hline
\end{array}
\begin{array}{c}
\hline
x \xrightarrow{a} y \quad \leftrightarrow \quad a \quad \xrightarrow{a} z \\
\hline
\end{array}
\end{align*}
Naive($x, y$)

(1) $R$ is empty; todo is empty;
(2) insert $(x, y)$ in todo;
(3) while todo is not empty, do
(3.1) extract $(x', y')$ from todo;
(3.2) if $(x', y') \in R$ then continue;
(3.3) if $o(x') \neq o(y')$ then return false;
(3.4) for all $a \in A$,
insert $(t_a(x'), t_a(y'))$ in todo;
(3.5) insert $(x', y')$ in $R$;
(4) return true;

\[
\begin{array}{c|c|c|c}
\text{todo} = \emptyset & (x, u) & R = \{(x, u), (y, v), (z, w)\} \\
\hline
1 & 2 & 3 \\
\end{array}
\]
Naive$(x, y)\$

1. $R$ is empty; $todo$ is empty;
2. insert $(x, y)$ in $todo$;
3. while $todo$ is not empty, do
   3.1. extract $(x', y')$ from $todo$;
   3.2. if $(x', y') \in R$ then continue;
   3.3. if $o(x') \neq o(y')$ then return false;
   3.4. for all $a \in A$,
       insert $(t_a(x'), t_a(y'))$ in $todo$;
   3.5. insert $(x', y')$ in $R$;
4. return true;

$todo = \emptyset$ \hspace{1cm} $(x, u)$

$R = \{(x, u), (y, v), (z, w)\}$

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$</td>
<td>$\rightarrow$</td>
<td>$\rightarrow$</td>
<td>$\rightarrow$</td>
</tr>
<tr>
<td>$a$</td>
<td>$\rightarrow$</td>
<td>$\rightarrow$</td>
<td>$\rightarrow$</td>
</tr>
<tr>
<td>$\downarrow$</td>
<td>$\downarrow$</td>
<td>$\downarrow$</td>
<td>$\downarrow$</td>
</tr>
<tr>
<td>$y$</td>
<td>$\downarrow$</td>
<td>$\downarrow$</td>
<td>$\downarrow$</td>
</tr>
<tr>
<td>$a$</td>
<td>$\rightarrow$</td>
<td>$\rightarrow$</td>
<td>$\rightarrow$</td>
</tr>
<tr>
<td>$z$</td>
<td>$\downarrow$</td>
<td>$\downarrow$</td>
<td>$\downarrow$</td>
</tr>
<tr>
<td>$a$</td>
<td>$\rightarrow$</td>
<td>$\rightarrow$</td>
<td>$\rightarrow$</td>
</tr>
<tr>
<td>$u$</td>
<td>$\rightarrow$</td>
<td>$\rightarrow$</td>
<td>$\rightarrow$</td>
</tr>
<tr>
<td>$v$</td>
<td>$\rightarrow$</td>
<td>$\rightarrow$</td>
<td>$\rightarrow$</td>
</tr>
<tr>
<td>$a$</td>
<td>$\rightarrow$</td>
<td>$\rightarrow$</td>
<td>$\rightarrow$</td>
</tr>
<tr>
<td>$w$</td>
<td>$\downarrow$</td>
<td>$\downarrow$</td>
<td>$\downarrow$</td>
</tr>
<tr>
<td>$a$</td>
<td>$\rightarrow$</td>
<td>$\rightarrow$</td>
<td>$\rightarrow$</td>
</tr>
</tbody>
</table>

Regular Expressions

e::= 0, 1, a, e+e, ee, e*
Brzozowski derivatives defines a DA (RE,o,t)

<table>
<thead>
<tr>
<th></th>
<th>$e$</th>
<th>$f$</th>
<th>$e$</th>
<th>$e'$</th>
<th>$f$</th>
<th>$f'$</th>
<th>$e$</th>
<th>$e'$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$\downarrow$</td>
<td>$\downarrow$</td>
<td>$\downarrow$</td>
<td>$\downarrow$</td>
<td>$\downarrow$</td>
<td>$\downarrow$</td>
<td>$\downarrow$</td>
<td>$\downarrow$</td>
</tr>
<tr>
<td>$e+e$</td>
<td>$\downarrow$</td>
<td>$\downarrow$</td>
<td>$\downarrow$</td>
<td>$\downarrow$</td>
<td>$\downarrow$</td>
<td>$\downarrow$</td>
<td>$\downarrow$</td>
<td>$\downarrow$</td>
</tr>
<tr>
<td>$e+f$</td>
<td>$\downarrow$</td>
<td>$\downarrow$</td>
<td>$\downarrow$</td>
<td>$\downarrow$</td>
<td>$\downarrow$</td>
<td>$\downarrow$</td>
<td>$\downarrow$</td>
<td>$\downarrow$</td>
</tr>
<tr>
<td>$e+f$</td>
<td>$\downarrow$</td>
<td>$\downarrow$</td>
<td>$\downarrow$</td>
<td>$\downarrow$</td>
<td>$\downarrow$</td>
<td>$\downarrow$</td>
<td>$\downarrow$</td>
<td>$\downarrow$</td>
</tr>
<tr>
<td>$ef$</td>
<td>$\downarrow$</td>
<td>$\downarrow$</td>
<td>$\downarrow$</td>
<td>$\downarrow$</td>
<td>$\downarrow$</td>
<td>$\downarrow$</td>
<td>$\downarrow$</td>
<td>$\downarrow$</td>
</tr>
<tr>
<td>$e^*$</td>
<td>$\downarrow$</td>
<td>$\downarrow$</td>
<td>$\downarrow$</td>
<td>$\downarrow$</td>
<td>$\downarrow$</td>
<td>$\downarrow$</td>
<td>$\downarrow$</td>
<td>$\downarrow$</td>
</tr>
</tbody>
</table>

|   |   |   |   |   |
|---|---|---|---|
| $0 \overset{a}{\rightarrow} 0$ | $1 \overset{a}{\rightarrow} 0$ | $a \overset{a}{\rightarrow} 1$ | $b \not\overset{a}{\rightarrow} 1$ | $e \overset{a}{\rightarrow} e'$ |
| $f \overset{a}{\rightarrow} f'$ | $e+f \overset{a}{\rightarrow} e'+f'$ |
| $e \overset{a}{\rightarrow} e'$ | $e^* \overset{a}{\rightarrow} e'$ |
Kleene Algebra

We can prove the soundness of Kleene Algebra Axiomatization by mean of coinduction

Commutativity: $e + f \sim f + e$

$R = \{ (e+f,f+e) \mid e,f \in \text{RE} \}$ is a bisimulation:

1. $e+f \downarrow \iff e \downarrow \text{or } f \downarrow \iff f+e \downarrow$
2. $e+f \overset{R}{\sim} f+e$

Exercises 1

a) Let $(X,o,t)$ be a DFA and $x \sim y$. Let $R$ be the relation computed by Naive$(x,y)$. Let $P$ the partition computed by Partition Refinement. Prove that $R \subseteq P$

b) Execute Naive on the following DFA. How many pairs are explored?

c) Prove that $(\text{RE},+0)$ is an idempotent monoid:
   \[ e+(f+g) \sim (e+f)+g \]
   \[ e+0 \sim e \]
   \[ e+e \sim e \]

d) OPTIONAL
   Use coinduction to prove distributivity:
   \[ e(f+g) \sim ef+eg \]

The solutions are due for the lesson of tomorrow Tuesday 10th!!!!