Coinductive Graph Representation

Celia Picard

IRIT - UPS
Team : ACADIE
Thesis advisor : Ralph Matthes

CaCos - 26/07/2012
Genesis: certified model transformation
Genesis: certified model transformation
Genesis: certified model transformation

Transformation

model → certification → model

CaCos - 26/07/2012  Celia Picard  Coinductive Graph Representation
Genesis: certified model transformation

Transformation

model → certification → model
Genesis: certified model transformation
Genesis: certified model transformation
Genesis: certified model transformation

Transformation

model

Coq

certification

model
Genesis: certified model transformation

Coq

Transformation

model → model
Genesis: certified model transformation

Coq

Transformation

model

model
Genesis: certified model transformation

Coq

Transformation

model

model
Genesis: certified model transformation

Coinductive type

Transformation

Coq
Coinductive representation
A first attempt

Definition

\[
t : T \quad l : \text{list} (\text{Graph } T) \\
\overline{\text{mk}_\text{Graph} t \ l : \text{Graph } T}
\]

Examples

**Finite**\(_\text{Graph}\) =
\[
\text{mk}_\text{Graph} 0 \ [\text{mk}_\text{Graph} 1 \ [\text{Finite}_\text{Graph}]]
\]

**Infinite**\(_\text{Graph}\)_\(_n\) =
\[
\text{mk}_\text{Graph} \ n \ [\text{Infinite}_\text{Graph}\_\(_{n+1}\)]
\]

A first function

We would like to define the function (with \(f\) of type \(T \rightarrow U\)): \(\text{applyF2G } f \ (\text{mk}_\text{Graph} t \ l) = \text{mk}_\text{Graph} (f \ t) \ (\text{map} \ (\text{applyF2G } f) \ l)\)

but... forbidden!
Coinductive representation
A first attempt

Definition

\[ t : T \quad l : \text{list (Graph } T) \]
\[ \text{mk}_\text{Graph} \ t \ l : \text{Graph } T \]

Examples

\[ \text{Finite}_\text{Graph} = \]
\[ \text{mk}_\text{Graph} \ 0 \ [\text{mk}_\text{Graph} \ 1 \ [\text{Finite}_\text{Graph}]] \]
\[ \text{Infinite}_\text{Graph}_n = \]
\[ \text{mk}_\text{Graph} \ n \ [\text{Infinite}_\text{Graph}_{n+1}] \]

A first function

We would like to define the function (with \( f \) of type \( T \rightarrow U \)):
\[ \text{applyF2G} \ f \ (\text{mk}_\text{Graph} \ t \ l) = \text{mk}_\text{Graph} \ (f \ t) \ (\text{map} \ (\text{applyF2G} \ f) \ l) \]
but... forbidden!
Coinductive representation
A first attempt

**Definition**

\[
t : T \quad l : \text{list} \ (\text{Graph} \ T) \\
\overline{\text{mk}_\text{Graph} \ t \ l : \text{Graph} \ T}
\]

**Examples**

- \(\text{Finite}_\text{Graph} = \text{mk}_\text{Graph} \ 0 \ [\text{mk}_\text{Graph} \ 1 \ [\text{Finite}_\text{Graph}]]\)
- \(\text{Infinite}_\text{Graph}_n = \text{mk}_\text{Graph} \ n \ [\text{Infinite}_\text{Graph}_{n+1}]\)

**A first function**

We would like to define the function (with \(f\) of type \(T \rightarrow U\)):

\[
\text{applyF2G} \ f \ (\text{mk}_\text{Graph} \ t \ l) = \text{mk}_\text{Graph} \ (f \ t) \ (\text{map} \ (\text{applyF2G} \ f) \ l)
\]

but... forbidden!
The problem

Guard condition

Explanation of the idea

Objective: ensure that we can get more information on the structure in a finite amount of time (productivity rule).

Restrictive solution offered by Coq: a corecursive call must always be a constructor argument.

On a small example: filter on streams

Problem/solution

Problem: applyF2G actually semantically correct!

Solution: overcome guardedness condition (not change it)
The problem

Guard condition

Explanation of the idea

Objective: ensure that we can get more information on the structure in a finite amount of time (productivity rule). Restrictive solution offered by Coq: a corecursive call must always be a constructor argument.

On a small example: filter on streams

Problem/solution

Problem: applyF2G actually semantically correct!
Solution: overcome guardedness condition (not change it)
The problem
Guard condition

Explanation of the idea

**Objective**: ensure that we can get more information on the structure in a finite amount of time (productivity rule).
Restrictive solution offered by Coq: a corecursive call must always be a constructor argument.

On a small example: filter on streams

Problem/solution

**Problem**: applyF2G actually semantically correct!
**Solution**: overcome guardedness condition (not change it)
Outline

1. A Functional Equivalent to Lists
2. A Coinductive Graph Representation
3. Related Work and Conclusions
Outline

1. A Functional Equivalent to Lists
   • Definition of \textit{ilist}
   • Capturing Permutations on \textit{ilist}

2. A Coinductive Graph Representation

3. Related Work and Conclusions
The idea

Using **functions** instead of inductive types to represent lists
A list = a **shape** (specified by number of **positions**) and a
**function**: positions → $T$ (**container** view)

Example for the list [10 ; 22 ; 5]

First problem : represent set of $n$ elements ($n$ **indeterminate**):
family of sets $\text{Fin}$ such that $\forall n, \text{card} \{i \mid i : \text{Fin } n\} = n$
Implementation of *ilist*

**Implementation**

The function: 
\[ \text{ilist}_n \ (T : \text{Set}) \ (n : \mathbb{N}) = \text{Fin} \ n \rightarrow T \]

The *ilist*: 
\[ \text{ilist} \ (T : \text{Set}) = \Sigma(n : \mathbb{N}).\text{ilist}_n \ T \ n \]

**Lemma**: There is a bijection between *ilist* and *list*.

**An equivalence on *ilist***

\[ \forall l_1 \ l_2 : \text{ilist} \ T, \text{ilist}_\text{rel}_R \ l_1 \ l_2 \iff \]
\[ \forall h : \text{lg} \ l_1 = \text{lg} \ l_2, \forall i : \text{Fin} \ (\text{lg} \ l_1), R \ (\text{fct} \ l_1 \ i) \ (\text{fct} \ l_2 \ i'_h) \]

where \( \text{lg} \) and \( \text{fct} \) are projections on *ilist*, \( R \) is a relation on \( T \) and \( i'_h \) is \( i \), converted from type \( \text{Fin} \ (\text{lg} \ l_1) \) to type \( \text{Fin} \ (\text{lg} \ l_2) \)

**Tools**

Replacement for map: 
\[ \text{imap} \ f \ l = \langle \text{lg} \ l, f \circ (\text{fct} \ l) \rangle \]
Implementation of *ilst*

### Implementation

The function:  
\[ \text{ilstn} (T : Set) (n : \mathbb{N}) = \text{Fin} n \to T \]

The *ilst*:  
\[ \text{ilst} (T : Set) = \sum (n : \mathbb{N}). \text{ilstn} T n \]

**Lemma**: There is a bijection between *ilst* and *list*.

### An equivalence on *ilst*

\[ \forall l_1, l_2 : \text{ilst} T, \text{ilst}_\text{rel}_R l_1 l_2 \iff \forall h : \text{lg} l_1 = \text{lg} l_2, \forall i : \text{Fin} (\text{lg} l_1), R (\text{fct} l_1 i) (\text{fct} l_2 i'_h) \]

where *lg* and *fct* are projections on *ilst*, *R* is a relation on *T* and *i'_h* is *i*, converted from type *Fin (lg l_1)* to type *Fin (lg l_2)*

### Tools

Replacement for *map*:  
\[ \text{imap} f l = \langle \text{lg} l, f \circ (\text{fct} l) \rangle \]
Implementation of \textit{ilist}

\textbf{Implementation}

The function: \textit{ilistn} \((T : \text{Set}) (n : \mathbb{N}) = \text{Fin} \ n \rightarrow T\)

The \textit{ilist}: \textit{ilist} \((T : \text{Set}) = \Sigma(n : \mathbb{N}).\text{ilistn} \ T \ n\)

Lemma: There is a bijection between \textit{ilist} and \textit{list}.

\textbf{An equivalence on \textit{ilist}}

\(\forall l_1 \ l_2 : \textit{ilist} \ T, \textit{ilist\_rel}_R \ l_1 \ l_2 \iff \forall h : \text{lg} \ l_1 = \text{lg} \ l_2, \forall i : \text{Fin} \ (\text{lg} \ l_1), R \ (\text{fct} \ l_1 \ i) \ (\text{fct} \ l_2 \ i'_h)\)

where \textit{lg} and \textit{fct} are projections on \textit{ilist}, \(R\) is a relation on \(T\) and \(i'_h\) is \(i\), converted from type \(\text{Fin} \ (\text{lg} \ l_1)\) to type \(\text{Fin} \ (\text{lg} \ l_2)\)

\textbf{Tools}

Replacement for \textit{map}: \textit{imap} \(f \ l = \langle \text{lg} \ l, f \circ (\text{fct} \ l)\rangle\)
Outline

1. A Functional Equivalent to Lists
   - Definition of ilist
   - Capturing Permutations on ilist

2. A Coinductive Graph Representation

3. Related Work and Conclusions
Capturing permutations on *i*list

Permutations on *i*list with decidability

The idea for comparing $l_1$ and $l_2$

$$\forall t, \text{card}\ \{i \mid R\ (fct\ l_1\ i)\ t\} = \text{card}\ \{i \mid R\ (fct\ l_2\ i)\ t\}$$

Implementation: counting elements

$$\forall l_1\ l_2, \text{iperm\_occ}_{Rd} l_1\ l_2 \iff \forall t, \text{nbocc}_{Rd} t\ l_1 = \text{nbocc}_{Rd} t\ l_2$$

where $\text{nbocc}_{Rd} t\ l$ gives the number of occurrences of $t$ in $l$.

The problem

*iperm\_occ* needs **decidability**. Cannot always be assumed.
Capturing permutations on *ilist*

Inductive definitions of permutations on *ilist*- Definitions

\[ \text{iperm\_ind}_R \ l_1 \ l_2 \iff \begin{cases} 
\lg l_1 = \lg l_2 = 0 \\
\exists i_1 \exists i_2, R (fct l_1 i_1) (fct l_2 i_2) \land \\
\text{iperm\_ind}_R (\text{remEl} l_1 i_1) (\text{remEl} l_2 i_2) 
\end{cases} \land \\
\text{iperm\_ind}'_R l_1 l_2 \iff \lg l_1 = \lg l_2 \land (\forall i_1 \exists i_2, R (fct l_1 i_1) (fct l_2 i_2) \land \\
\text{iperm\_ind}'_R (\text{remEl} l_1 i_1) (\text{remEl} l_2 i_2)) \land \\
\text{iperm\_ind}''_R l_1 l_2 \iff \lg l_1 = \lg l_2 \land (\forall i_2 \exists i_1, R (fct l_1 i_1) (fct l_2 i_2) \land \\
\text{iperm\_ind}''_R (\text{remEl} l_1 i_1) (\text{remEl} l_2 i_2)) \land \\
\text{remEl} l i \text{ removes the } i^{th} \text{ element of } l. \]

where *remEl l i* removes the *i*th element of *l*. 

![Diagram of list with element t3 removed](image)
Capturing permutations on \textit{ilist}
Inductive definitions of permutations on \textit{ilist}- Results

Theorem of equivalence between definitions

\[ \forall l_1 \ l_2, \text{iperm\_ind}_R \ l_1 \ l_2 \Leftrightarrow \text{iperm\_ind}'_R \ l_1 \ l_2 \Leftrightarrow \text{iperm\_ind}''_R \ l_1 \ l_2 \]

Proof not straightforward since one definition can be seen as a special case of the others.

Usefulness of having various definitions: some properties easier to prove on one than on the other and vice versa.

Other properties

Preservation of equivalence, decidability, monotonicity.

Definition with skeleton: \textit{skel\_type}

Equivalent to \textit{iperm\_ind} with witness of the permutation used.
Capturing permutations on *ilist*
Definition using bijective functions and comparison between definitions

**Definition of *iperm_bij***

Idea: use a bijective function in the same style as *ilist_rel*.

\[
\forall f, g, \text{bij } f, g \iff (\forall t, g(f(t)) = t) \land (\forall u, f(g(u)) = u)
\]

\[
\forall l_1, l_2, \text{iperm_bij}_R l_1, l_2 \iff \exists f, g, \text{bij } f, g \land \forall i, R(fct(l_1, i))(fct(l_2(f(i))))
\]

**Equivalence between definitions**

- We can show that \(\forall l_1, l_2, \text{iperm_ind}_R l_1, l_2 \iff \text{iperm_bij}_R l_1, l_2\)
- Permutations on lists by Contejean equivalent to ours

**Comparison between definitions**

*iperm_ind* captures better **intuition** than *iperm_bij* but inductive. Contejean’s definition on *list*.

We prefer definition on *ilist* \(\Rightarrow\) our choice is *iperm_ind*. 

---

CaCos - 26/07/2012
Celia Picard
Coinductive Graph Representation

13/33
Capturing permutations on *ilist*
Definition using bijective functions and comparison between definitions

**Definition of *iperm_bij***

Idea: use a bijective function in the same style as *ilist_rel*.

\[
\forall f, g, \text{bij } f, g \iff (\forall t, g(f \ t) = t) \land (\forall u, f(g \ u) = u)
\]

\[
\forall l_1, l_2, \text{iperm_bij}_R \ l_1 \ l_2 \iff \exists f, g, \text{bij } f, g \land (\forall i, R(fct \ l_1 \ i)(fct \ l_2 \ (f \ i)))
\]

**Equivalence between definitions**

- We can show that \(\forall l_1, l_2, \text{iperm_ind}_R \ l_1 \ l_2 \iff \text{iperm_bij}_R \ l_1 \ l_2\)
- Permutations on lists by Contejean equivalent to ours

**Comparison between definitions**

*iperm_ind* captures better **intuition** than *iperm_bij* but inductive. Contejean’s definition on *list*.

We prefer definition on *ilist* ⇒ our choice is *iperm_ind*.
Outline

1. A Functional Equivalent to Lists

2. A Coinductive Graph Representation
   - New Graph Representation
   - A More Liberal Bisimulation Relation on Graph
     - Need For a More Liberal Relation on Graph
     - A Relation On Graph Using iperm_ind
     - Relations On Graph Using iperm_bij
     - The Final Relation Over Graph

3. Related Work and Conclusions
New graph representation
Definition of *Graph*

**Graph and applyF2G (coinductive)**

\[
\begin{align*}
\text{Graph} & : \quad t : T \quad l : \text{ilist}(\text{Graph } T) \\
& \quad \quad \Rightarrow \quad \text{mk}_\text{Graph} \; t \; l : \text{Graph } T \\
\text{applyF2G} & : \quad \text{applyF2G} \; f \; (\text{mk}_\text{Graph} \; t \; l) = \\
& \quad \quad \Rightarrow \quad \text{mk}_\text{Graph} \; (f \; t) \; (\text{imap} \; (\text{applyF2G} \; f) \; l)
\end{align*}
\]

**Bisimulation relation on Graph: Geq**

Why?
*Graph* is **infinite**
⇒ “=” not usable

\[
\begin{align*}
R \; (\text{label } g_1) \; (\text{label } g_2) \quad \text{ilist}\_\text{rel}_{\text{Geq}_R} \; (\text{sons } g_1) \; (\text{sons } g_2) \\
& \quad \quad \Rightarrow \quad \text{Geq}_R \; g_1 \; g_2
\end{align*}
\]

where *label* and *sons* are the projections on *Graph*
New graph representation

Finiteness

Notion of finiteness

\[ \text{Finiteness} : \forall g, \ G\_\text{finite}_R \ g \iff \exists gs, \ G\_\text{all} (\text{element\_of}_R \ gs) \ g \]

with \( Gall \) universal quantification on \( Graph \) and \( \text{element\_of} \) list membership modulo \( Geq \)

Redefinition of the examples from the beginning

\[
\begin{align*}
\text{Finite\_Graph} & := \text{mk\_Graph} \ 0 \ [\text{mk\_Graph} \ 1 \ [\text{Finite\_Graph}]] \\
\text{Infinite\_Graph}_n & := \text{mk\_Graph} \ n \ [\text{Infinite\_Graph}_{n+1}]
\end{align*}
\]

Proofs of finiteness

\[ G\_\text{finite}\_\text{Finite\_Graph} : \text{rather easy proof} \]

\[ \forall n, \ \neg G\_\text{finite}\_\text{Infinite\_Graph}_n : \text{we use unbounded labels} \]

labels and \#sons bounded \( \Rightarrow \) proofs of infinity much harder
A Functional Equivalent to Lists

A Coinductive Graph Representation

1. A Functional Equivalent to Lists

2. A Coinductive Graph Representation
   - New Graph Representation
     - A More Liberal Bisimulation Relation on Graph
       - Need For a More Liberal Relation on Graph
       - A Relation On Graph Using iperm_ind
       - Relations On Graph Using iperm_bij
       - The Final Relation Over Graph

3. Related Work and Conclusions
Need for a more liberal relation on *Graph*

**The problem**

These pairs of graphs are not bisimulated through $\text{Geq}$:

$$
\begin{align*}
0 & \leftrightarrow 0 \\
1 & \leftrightarrow 1 \\
1 & \leftrightarrow 0
\end{align*}
$$

**Solution**

- Define a new equivalence relation on *Graph* using *permutations* on *ilist*
- Define a new equivalence relation on *Graph* using the previous one and taking into account *rotations*
A relation on \textit{Graph} using \textit{iperm\_ind}

**Definition of \textit{GPerm} (coinductive)**

\[
R(\text{label } g_1)(\text{label } g_2) \quad \text{iperm\_ind}_{GPerm_R}(\text{sons } g_1)(\text{sons } g_2) \\
GPerm_R \ g_1 \ g_2
\]

The problem: proof that \textit{GPerm} preserves reflexivity

**Lemma:** \(\forall R, \ R \text{ reflexive} \Rightarrow \forall g, \ GPerm_R \ g \ g\)

Proof (by coinduction): We must prove that
\[
R(\text{label } g)(\text{label } g) \quad \text{ok} \quad \text{has to be inductive}
\]

\[
R(\text{label } g)(\text{label } g) \quad \text{iperm\_ind}_{GPerm_R}(\text{sons } g)(\text{sons } g)
\]

**Mendler-style definition (coinductive and impredicative)**

\[
\mathcal{R} \subseteq GPerm_{\text{mend}}_R \quad R(\text{label } g_1)(\text{label } g_2) \quad \text{iperm\_ind}_{\mathcal{R}}(\text{sons } g_1)(\text{sons } g_2)
\\
GPerm_{\text{mend}}_R \ g_1 \ g_2
\]

Preserves equivalence
A relation on Graph using iperm_ind

**Definition of GPerm (coinductive)**

\[
R \ (\text{label } g_1) \ (\text{label } g_2) \quad \text{iperm\_ind}_{GPerm_R} \ (\text{sons } g_1) \ (\text{sons } g_2) \\
\hline
GPerm_R \ g_1 \ g_2
\]

**The problem: proof that GPerm preserves reflexivity**

**Lemma:** \(\forall R, \ R \text{ reflexive} \Rightarrow \forall g, \ GPerm_R \ g \ g\)

**Proof (by coinduction):** We must prove that

\[
R \ (\text{label } g) \ (\text{label } g) \quad \text{iperm\_ind}_{GPerm_R} \ (\text{sons } g) \ (\text{sons } g)
\]

ok has to be inductive

**Mendler-style definition (coinductive and impredicative)**

\[
\mathcal{R} \subseteq GPerm_{mend} \quad R \ (\text{label } g_1) \ (\text{label } g_2) \quad \text{iperm\_ind}_{\mathcal{R}} \ (\text{sons } g_1) \ (\text{sons } g_2)
\]

\[
GPerm_{mend} \ g_1 \ g_2
\]

Preserves equivalence
A relation on *Graph* using *iperm_ind*

**Definition of *GPerm* (coinductive)**

\[
R (\text{label } g_1) (\text{label } g_2) \quad \text{iperm\_ind}_{GPerm_R} (\text{sons } g_1) (\text{sons } g_2) \\
\Rightarrow \quad GPerm_R g_1 g_2
\]

**The problem: proof that *GPerm* preserves reflexivity**

**Lemma:** \( \forall R, \ R \text{ reflexive} \Rightarrow \forall g, \ GPerm_R g g \)

**Proof (by coinduction):** We must prove that

\[
R (\text{label } g) (\text{label } g) \land \text{iperm\_ind}_{GPerm_R} (\text{sons } g) (\text{sons } g) \\
\land \quad \text{ok} \quad \text{has to be inductive}
\]

**Mendler-style definition (coinductive and impredicative)**

\[
\mathcal{R} \subseteq GPerm_{mend_R} \quad R (\text{label } g_1) (\text{label } g_2) \quad \text{iperm\_ind}_{\mathcal{R}} (\text{sons } g_1) (\text{sons } g_2) \\
\Rightarrow \quad GPerm_{mend_R} g_1 g_2
\]

Preserves equivalence
A relation on *Graph* using *iperm_ind*

An equivalent approach based on observation - The idea

Using **inductive trees** to observe coinductive graphs until a certain **depth**.

⇒ **no more mixing** of inductive and coinductive types

Observed

until depth 5
A relation on Graph using \textit{iperm\_ind}

An equivalent approach based on observation - Definitions

\begin{align*}
\text{iTree (inductive)}: & \quad \frac{t : T \quad l : \text{ilist} \ (\text{iTree} \ T)}{\text{mk\_iTree} \ t \ l : \text{iTree} \ T} \\
\text{TPerm (inductive)}: & \quad \frac{R \ (\text{labeliT} \ t_1) \ (\text{labeliT} \ t_2) \ \text{iperm\_ind}_{\text{TPerm}_R} \ (\text{sonsiT} \ t_1) \ (\text{sonsiT} \ t_2)}{\text{TPerm}_R \ t_1 \ t_2} \\
\text{G2iT:} & \\
\quad & \quad \forall T, \text{nat} \rightarrow \text{Graph} \ T \rightarrow \text{iTree} \ T \\
\quad & \quad \text{G2iT} \ T \ 0 \ (\text{mk\_Graph} \ t \ l) := \text{mk\_Tree} \ t \\
\quad & \quad \text{G2iT} \ T \ (n + 1) \ (\text{mk\_Graph} \ t \ l) := \text{mk\_Tree} \ t \ (\text{imap} \ (\text{G2iT} \ n) \ l) \\
\quad & \quad \equiv_{R,n} : \forall n \ g_1 \ g_2, \ g_1 \equiv_{R,n} \ g_2 \leftrightarrow \text{TPerm}_R \ (\text{G2iT} \ n \ g_1) \ (\text{G2iT} \ n \ g_2) \\
\quad & \quad \text{GTPerm:} \ \forall g_1 \ g_2, \ \text{GTPerm}_R \ g_1 \ g_2 \leftrightarrow \forall n, \ g_1 \equiv_{R,n} \ g_2
\end{align*}
A relation on *Graph* using *iperm_ind*

An equivalent approach based on observation - Main theorem

The theorem

\[ \forall g_1, g_2, \text{GPerm}_\text{mend}_R g_1 g_2 \iff \text{GTPerm}_R g_1 g_2 \]

Proof

(Direction \(\Rightarrow\)) easy (induction on \(n\))

(Direction \(\Leftarrow\)) proved using the lemma:

\[ \forall g_1, g_2, \text{GTPerm}_R g_1 g_2 \Rightarrow \text{iperm}_\text{ind}_{\text{GTPerm}_R} (\text{sons } g_1) (\text{sons } g_2) \]

Modulo non-constructive axiom: *Infinite Pigeonhole Principle*
A relation on \( \text{Graph} \) using \( \text{iperm\_ind} \)

An equivalent approach based on observation - Main theorem

The theorem

\[ \forall g_1, g_2, \text{GPerm\_mend}_R g_1 g_2 \iff \text{GPerm}_R g_1 g_2 \]

Proof

[Direction \( \Rightarrow \)] easy (induction on n)

[Direction \( \Leftarrow \)] proved using the lemma:

\[ \forall g_1, g_2, \text{GPerm}_R g_1 g_2 \Rightarrow \text{iperm\_ind}_{\text{GPerm}_R} (\text{sons } g_1) (\text{sons } g_2) \]

Modulo non-constructive axiom: \textbf{Infinite Pigeonhole Principle}
Definitions

- Direct definition:

\[
R \left( \text{label } g_1 \right) \left( \text{label } g_2 \right) \ 	ext{iperm\_bij}_{\text{GPerm\_bij}_R} \left( \text{sons } g_1 \right) \left( \text{sons } g_2 \right) \ \\ \text{GPerm\_bij}_R \ g_1 \ g_2
\]

- Need an impredicative one for proofs of equivalence:

\[
R \subseteq \text{GPerm\_bij\_mend}_R \ \\ R\left( \text{label } g_1 \right) \left( \text{label } g_2 \right) \ 	ext{iperm\_bij}_R \left( \text{sons } g_1 \right) \left( \text{sons } g_2 \right) \ \\ \text{GPerm\_bij\_mend}_R \ g_1 \ g_2
\]

Results

- Equivalence relations

\[
\text{GPerm\_mend} \iff \text{GPerm\_bij\_mend} \iff \text{GPerm\_bij}
\]
Summary of the obtained notions

- **GPerm**
- **GTPerm**
- **GPerm\_mend**
- **GPerm\_bij\_mend**
- **GPerm\_bij**

- nested induction
- with infinite pigeonhole principle
- flexibility of Mendler style
- no nested induction
The final relation over *Graph*

The idea

- Change in the "point of view" for the observation of the graph
- Single-rooted graph $\Rightarrow$ **path from the root to all nodes**
- Change in the root $\Rightarrow$ both roots in the same cycle $\Rightarrow$
  $g_1 \subset g_2 \land g_2 \subset g_1$
- Only for a "**general**" view:

![Diagram showing the graph relations](image-url)
The final relation over \textit{Graph}

Definitions

Non-strict Inclusion

General definition (inductive):

\[
\forall g_{in} \ g_{out}, \ GinG_{R_G}^* \ g_{in} \ g_{out} \iff \begin{cases} R_G \ g_{in} \ g_{out} \\ \exists i, \ GinG_{R_G}^* \ g_{in} \ (\text{fct} \ (\text{sons} \ g_{out}) \ i) \end{cases}
\]

Instantiation: \( GinGP_R := GinG_{GPerm\_mend_R}^* \)

The final relation

\[
\forall g_1 \ g_2, \ GeqPerm_R \ g_1 \ g_2 \iff GinGP_R \ g_1 \ g_2 \land GinGP_R \ g_2 \ g_1
\]

Preserves equivalence

\[
\begin{array}{ccc}
1 & \xrightarrow{\text{fct}} & 2 \\
0 & \xrightarrow{\text{fct}} & 1
\end{array}
\quad \iff \quad
\begin{array}{ccc}
1 & \xrightarrow{\text{fct}} & 0 \\
2 & \xrightarrow{\text{fct}} & 1
\end{array}
\quad \iff \quad
\begin{array}{ccc}
1 & \xrightarrow{\text{fct}} & 0 \\
0 & \xrightarrow{\text{fct}} & 1
\end{array}
\]
Outline

1. A Functional Equivalent to Lists
2. A Coinductive Graph Representation
3. Related Work and Conclusions
Related work

Permutations

- Contejean: treats the same problem for lists
- Standard library: requires decidability or Leibniz equality

Graph representation

- Erwig: inductive directed graph representation; each node is added with its successors and predecessors
- Courcelle: inductive representation as regular expressions
Related work

Guardedness issues

- **Bertot & Komendantskaya**: same approach with *streams* represented by functions
- **Dams**: defines *everything coinductively* and restricts the finite parts with properties of finiteness
- **Niqui**: solution using *category theory* but not usable here
- **Danielsson**: experimental solution to the problem in *Agda* adding one constructor for each problematic function
- **Nakata & Uustalu**: *Mendler-style* definition
Conclusions

Achievements

- **Complete solution** to the guardedness problem in the case of lists
- **Permutations** captured for *ilist*
- Complete **representation of graphs** in Coq, many tools
- Quite **liberal equivalence** relation on *Graph*
- Various extensions in order to **represent models** (non-connected graphs, multiplicities)
- Completely formalized in Coq: [www.irit.fr/~Celia.Picard/These/](http://www.irit.fr/~Celia.Picard/These/)

Publications (with R. Matthes)

- Permutations in Coinductive Graph Representation - CMCS’12
Perspectives

Extension of the representation

New **finiteness criterion** using spanning trees

Generalization

- generalize the solutions for **any inductive type**
- apply expertise to other problems

Extend links

- **containers:**
  - morphism coming with categorical notion of container
  - notion of quotient types for permutations
  - possibility of representing graphs as containers

- **process algebras**
Perspectives - Certified model transformation

Extension

Deepen notion of forest of graphs

Applications

- A first direct application:
  - instantiation of the graphs for finite automata
  - certified transformations: minimization, determinization

- Metamodel representation (inheritance with polymorphism)
Summary

What has been done

- Library for functional equivalent to lists
- Full representation of graphs with liberal equivalence relation
- Fully proved in Coq

What remains to be done

- Extend and generalize the representation
- Extend links with existing work
- Follow the idea of representing and transformings models

Thanks for your attention.
**Fin** - a type family for finite indexed sets

**Problem**: represent a set of \( n \) elements for \( n \) indeterminate

**Solution**: we represent a **family of sets** parameterized by the number of their elements.

We use a common solution (Altenkirch, McBride & McKinna):

**Fin** of type \( \mathbb{N} \rightarrow \text{Set} \) with 2 constructors:

- `first (k : \mathbb{N}) : Fin (k + 1)`
- `succ (k : \mathbb{N}) : Fin k \rightarrow Fin (k + 1)`

**Lemmas**:

- \( \forall n, \ \text{card} \ \{ i \mid i : Fin \ n \} = n \)
- \( \forall n \ m, Fin \ n = Fin \ m \Rightarrow n = m \)
### Multiplicities representation

**Presentation**

**Final goal:** represent big metamodels, perform and certify transformations on them  
**Partial goal:** represent multiplicities  
**Solution:** extend \( ilist \) to include bounds.

### PropMult

Indicates whether a natural number fits a multiplicity condition:

\[ \forall (\inf : \mathbb{N}) (\sup : \text{option } \mathbb{N}) (i : \mathbb{N}), \]

\[ \text{PropMult } \inf \text{ sup } n \Leftrightarrow \begin{cases} i \geq \inf \land i \leq s & \text{if } \sup = \text{Some } s \\ i \geq \inf & \text{if } \sup = \text{None} \end{cases} \]

### ilistMult

\[ \text{ilistnMult } T \inf \text{ sup } n := \{i : \text{ilistn } T \ n \mid \text{PropMult } \inf \text{ sup } n\} \]

\[ \text{ilistMult } T \inf \text{ sup } := \Sigma(n : \mathbb{N}).\text{ilistnMult } T \inf \text{ sup } n \]
A relation on Graph using iperm_ind
An impredicative definition

The impredicative definition: \( GPerm_{imp} \)

\[
\forall g_1 \ g_2, GPerm_{imp}_R \ g_1 \ g_2 \iff \exists R, \left( \forall g_1' \ g_2', R \ g_1' \ g_2' \Rightarrow R (\text{label } g_1') (\text{label } g_2') \land iperm_{ind}_R (\text{sons } g_1') (\text{sons } g_2') \right) \land R \ g_1 \ g_2
\]

where variable \( R \) ranges over relations on Graph \( T \)

Tools and definitions

Coinduction principle: \( (\forall g_1 \ g_2, \ R \ g_1 \ g_2 \Rightarrow R (\text{label } g_1) (\text{label } g_2) \land iperm_{ind}_R (\text{sons } g_1) (\text{sons } g_2)) \Rightarrow \forall g_1 \ g_2, \ R \ g_1 \ g_2 \Rightarrow GPerm_{imp}_R \ g_1 \ g_2 \)

Unfolding principle: \( \forall g_1 \ g_2, \ GPerm_{imp}_R \ g_1 \ g_2 \Rightarrow R (\text{label } g_1) (\text{label } g_2) \land iperm_{ind}_{GPerm_{imp}_R} (\text{sons } g_1) (\text{sons } g_2) \)

Constructor: \( \forall g_1 \ g_2, \ R (\text{label } g_1) (\text{label } g_2) \land iperm_{ind}_{GPerm_{imp}_R} (\text{sons } g_1) (\text{sons } g_2) \Rightarrow GPerm_{imp}_R \ g_1 \ g_2 \)
A relation on \textit{Graph} using \textit{iperm\_ind}

An equivalent approach based on observation - Main theorem

The theorem

\[ \forall g_1, g_2, \text{GPerm} \_mend_R \ g_1 \ g_2 \iff \text{GTPerm}_R \ g_1 \ g_2 \]

Proof

[Direction \( \Rightarrow \)] easy (induction on \( n \))

[Direction \( \Leftarrow \)] proved using the lemma:

\[ \forall g_1, g_2, \text{GTPerm}_R \ g_1 \ g_2 \Rightarrow \text{iperm\_ind}_{\text{GTPerm}_R} (\text{sons } g_1) (\text{sons } g_2) \]

Modulo non-constructive axiom: \textit{Infinite Pigeonhole Principle}
A relation on *Graph* using *iperm_ind*

An equivalent approach based on observation - Main theorem

The theorem

\[ \forall g_1, g_2, GPerm_{mend} R g_1 g_2 \iff GTPerm_R g_1 g_2 \]

Proof

[Direction \(\Rightarrow\)] easy (induction on \(n\))

[Direction \(\Leftarrow\)] proved using the lemma:

\[ \forall g_1, g_2, GTPerm_R g_1 g_2 \Rightarrow iperm\_ind_{GTPerm_R} (sons g_1) (sons g_2) \]

Modulo non-constructive axiom: **Infinite Pigeonhole Principle**
A relation on Graph using iperm\_ind
An equivalent approach based on observation - Main theorem

The theorem
\[ \forall g_1, g_2, \mathrm{GPerm\_mend}_R \ g_1 \ g_2 \iff \mathrm{GTPerm}_R \ g_1 \ g_2 \]

Proof
[Direction \( \Rightarrow \)] easy (induction on n)
[Direction \( \Leftarrow \)] proved using the lemma:
\[ \forall g_1, g_2, \mathrm{GTPerm}_R \ g_1 \ g_2 \Rightarrow \mathrm{iperm\_ind}_{\mathrm{GTPerm}_R} (\ \mathrm{sons} \ g_1) (\ \mathrm{sons} \ g_2) \]
Modulo non-constructive axiom: Infinite Pigeonhole Principle
A relation on $\text{Graph}$ using $\text{iperm\_ind}$

An equivalent approach based on observation - Main theorem

The theorem

$\forall g_1, g_2, \text{GPerm\_mend}_R \ g_1 \ g_2 \Leftrightarrow \text{GTPerm}_R \ g_1 \ g_2$

Proof

[Direction $\Rightarrow$] easy (induction on n)

[Direction $\Leftarrow$] proved using the lemma:

$\forall g_1, g_2, \text{GTPerm}_R \ g_1 \ g_2 \Rightarrow \text{iperm\_ind}_{\text{GTPerm}_R}(\text{sons \ } g_1)(\text{sons \ } g_2)$

Modulo non-constructive axiom: $\text{Infinite Pigeonhole Principle}$
A relation on Graph using iperm_ind
An equivalent approach based on observation - Main theorem

The theorem

\[ \forall g_1, g_2, GPerm_{\text{mend}} R g_1 g_2 \iff GTPerm_R g_1 g_2 \]

Proof

[Direction \( \Rightarrow \)] easy (induction on n)

[Direction \( \Leftarrow \)] proved using the lemma:

\[ \forall g_1, g_2, GTPerm_R g_1 g_2 \Rightarrow iperm_{\text{ind}}_{GTPerm_R} (\text{sons } g_1) (\text{sons } g_2) \]

Modulo non-constructive axiom: **Infinite Pigeonhole Principle**
A representation of a wider class of graphs

We would like to represent graphs like this one:

![Graph Diagram]
A representation of a wider class of graphs

Solution: fictitious nodes.

\[ \text{AllGraph using Graph: } \text{AllGraph } T := \text{Graph (option } T) \]
A representation of a wider class of graphs

Other solution: forest.

\[ \text{AllGraph: } \text{AllGraph } T := \text{list (Graph } T) \]