Coinductive Graph Representation

Celia Picard

IRIT - UPS
Team : ACADIE
Thesis advisor : Ralph Matthes

12/05/2011

Outline

The Problem

A first representation

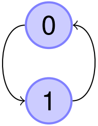
Context: certified model transformations (Coq)

Aim: representing metamodels as graphs and graphs using coinductive types (to directly represent navigability in loops)

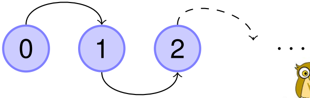
First attempt: constructor (coinductive):
$mk\_G : nat \rightarrow (list\ Graph) \rightarrow Graph$

Examples:

Finite graph:
$Finite\_Graph =$
$mk\_G\ 0\ [mk\_G\ 1\ [Finite\_Graph]]$



Infinite graph:
$Infinite\_Graph_n =$
$mk\_G\ n\ [Infinite\_Graph_{n+1}]$



The Problem
Guard condition

An example

We would like to define the function (with f of type $nat \to nat$):

$$applyF2G\ f\ (mk\_G\ n\ l) = mk\_G\ (f\ n)\ (map\ (applyF2G\ f)\ l)$$

but... forbidden !

Explanation: Coq's guard condition

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**.

Why is it a problem?

The definition above actually is semantically correct!

The Problem
Guard condition

An example

We would like to define the function (with f of type $nat \to nat$):

$$applyF2G\ f\ (mk\_G\ n\ l) = mk\_G\ (f\ n)\ (map\ (applyF2G\ f)\ l)$$

but... forbidden !

Explanation: Coq's guard condition

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.**

Why is it a problem?

The definition above actually is semantically correct!

The Problem
Guard condition

An example

We would like to define the function (with f of type $nat \to nat$):

$$applyF2G\ f\ (mk\_G\ n\ l) = mk\_G\ (f\ n)\ (map\ (applyF2G\ f)\ l)$$

but... forbidden !

Explanation: Coq's guard condition

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.**

Why is it a problem?

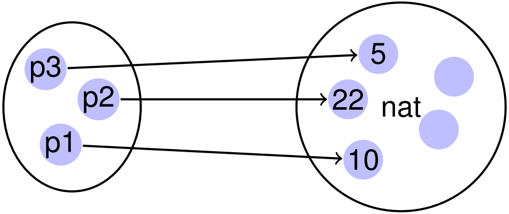
The definition above actually is semantically correct!

The Solution: *ilist*

The idea

Using **functions** instead of inductive types to represent lists

Example for the list [10 ; 22 ; 5]



First problem : represent a set of n elements

The Solution: *ilist*

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 $nat \rightarrow Set$ with 2 constructors:

$$\begin{array}{lll} first & (k : nat) : & Fin\,(k+1) \\ succ & (k : nat) : & Fin\ k \rightarrow Fin\,(k+1) \end{array}$$

Lemmas :

- $\forall n,\ card\ \{i \mid i : Fin\ n\} = n$ (not formalizable in Coq)
- $\forall n\ m,\ n = m \Leftrightarrow Fin\ n = Fin\ m$

The Solution: *ilist*

ilist implementation

Implementation

The function : $ilistn\ (T : Set)\ (n : nat) = Fin\ n \rightarrow T$

The *ilist* : $ilist\ (T : Set) = \Sigma(n : nat).ilistn\ T\ n$

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

An equivalence on *ilist*

$\forall l_1\ l_2 : ilist\ T, ilist\_rel\ R\ l_1\ l_2 \Leftrightarrow$
$\forall h : lgti\ l_1 = lgti\ l_2 \rightarrow (\forall i : Fin\ (lgti\ l_1), R\ (fcti\ l_1\ i)\ (fcti\ l_2\ i'_h))$
where *lgti* and *fcti* are projections on *ilist*, R is a relation on T and i'_h is i, converted from type $Fin\ (lgti\ l_1)$ to type $Fin\ (lgti\ l_2)$

Tools

Replacement for map: $imap\ f\ l = \langle (lgti\ l), (f \circ (fcti\ l)) \rangle$
Universal quantification: $iall\ T\ P\ l : Prop = \forall i, P\ (fcti\ l\ i)$

The Solution: *ilist*

ilist implementation

Implementation

The function : $ilistn\ (T : Set)\ (n : nat) = Fin\ n \rightarrow T$

The *ilist* : $ilist\ (T : Set) = \Sigma(n : nat).ilistn\ T\ n$

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

An equivalence on *ilist*

$\forall l_1\ l_2 : ilist\ T, ilist\_rel\ R\ l_1\ l_2 \Leftrightarrow$
$\forall h : lgti\ l_1 = lgti\ l_2 \rightarrow (\forall i : Fin\ (lgti\ l_1), R\ (fcti\ l_1\ i)\ (fcti\ l_2\ i'_h))$
where *lgti* and *fcti* are projections on *ilist*, R is a relation on T and i'_h is i, converted from type $Fin\ (lgti\ l_1)$ to type $Fin\ (lgti\ l_2)$

Tools

Replacement for map: $imap\ f\ l = \langle (lgti\ l), (f \circ (fcti\ l))\rangle$
Universal quantification: $iall\ T\ P\ l : Prop = \forall i, P\ (fcti\ l\ i)$

The Solution: *ilist*

ilist implementation

Implementation

The function : $ilistn\ (T : Set)\ (n : nat) = Fin\ n \rightarrow T$

The *ilist* : $ilist\ (T : Set) = \Sigma(n : nat).ilistn\ T\ n$

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

An equivalence on *ilist*

$\forall l_1\ l_2 : ilist\ T, ilist\_rel\ R\ l_1\ l_2 \Leftrightarrow$
$\forall h : lgti\ l_1 = lgti\ l_2 \rightarrow (\forall i : Fin\ (lgti\ l_1), R\ (fcti\ l_1\ i)\ (fcti\ l_2\ i'_h))$
where *lgti* and *fcti* are projections on *ilist*, R is a relation on T and i'_h is i, converted from type $Fin\ (lgti\ l_1)$ to type $Fin\ (lgti\ l_2)$

Tools

Replacement for map: $imap\ f\ l = \langle (lgti\ l), (f \circ (fcti\ l)) \rangle$
Universal quantification: $iall\ T\ P\ l : Prop = \forall i, P\ (fcti\ l\ i)$

New Graph Representation

Definition of *Graph*

Graph and *applyF2G* (coinductive)

Graph : $mk\_G : nat \rightarrow (ilist\ Graph) \rightarrow Graph$

applyF2G :

$applyF2G\ f\ (mk\_Graph\ n\ l) = mk\_G\ (f\ n)\ (imap\ (applyF2G\ f)\ l)$

Equivalence on *Graph*

Geq bisimulation relation on *Graph*

$\forall g_1\ g_2 : Graph,\ Geq\ g_1\ g_2 \Leftrightarrow$

$label\ g_1 = label\ g_2 \wedge ilist\_rel\ Geq\ (sons\ g_1)\ (sons\ g_2)$

where *label* and *sons* are the projections on *Graph*

Universal quantification on *Graph*

$\forall P,\ \forall g,\ G\_all\ P\ g \Leftrightarrow P\ g \wedge iall\ (G\_all\ P)\ (sons\ g)$

New Graph Representation

Finiteness

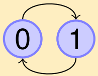
Notion of finiteness

List membership of an element of *Graph*:

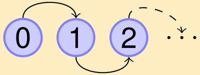
$P\_finite\ (lg : list\ Graph)\ (g : Graph) := \exists y,\ y \in lg\ \wedge\ Geq\ g\ y$

Finiteness : $\forall g,\ G\_finite\ g \Leftrightarrow \exists lg,\ G\_all\ (P\_finite\ lg)\ g$

Redefinition of the examples from the beginning



$Finite\_Graph := mk\_Graph\ 0\ [\![mk\_Graph\ 1\ [\![Finite\_Graph]\!]]\!]$



$Infinite\_Graph_n := mk\_Graph\ n\ [\![Infinite\_Graph_{n+1}]\!]$

Proofs of finiteness

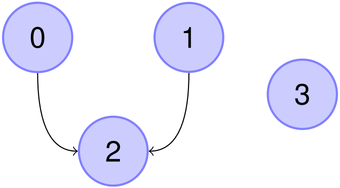
$G\_finite\ Finite\_Graph$: rather easy proof

$\forall n,\ \neg\ G\_finite\ Infinite\_Graph_n$: we use unbounded labels

labels and #sons bounded $\Rightarrow$ proofs of infiniteness much harder

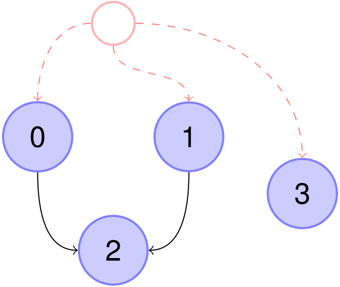
A Representation of a Wider Class of Graphs

We would like to represent graphs like this one:



A Representation of a Wider Class of Graphs

Solution: fictitious nodes.



AllGraph using *Graph*: *AllGraph T* := *Graph* (*option T*)

Multiplicity Representation

Presentation

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

PropMult

Indicates whether a natural number fits a multiplicity condition:
$\forall (inf : nat)\ (sup : option\ nat)\ (i : nat),$
$[sup = Some\ s]\ i \geq inf \wedge i \leq s \qquad [sup = None]\ i \geq inf$

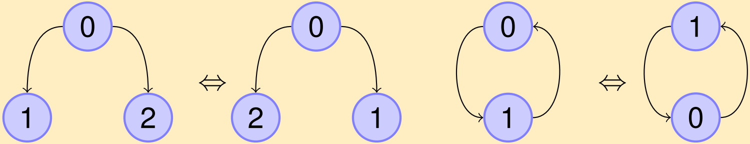
ilistMult

$ilistnMult\ T\ inf\ sup\ n := \{i : ilistn\ T\ n \mid PropMult\ inf\ sup\ n\}$
$ilistMult\ T\ inf\ sup := \Sigma(n : nat).ilistnMult\ T\ inf\ sup\ n$

Need for a more Liberal Relation on *Graph*

The problem

These pairs of graphs are not bisimulated through *Geq*:



Solution

- Define a new equivalence relation on *ilist* for permutations
- Define a new equivalence relation on *Graph* using the previous equivalence on *ilist* and taking into account rotations

Capturing Permutations on *ilist*

Permutations on *ilist* with decidability

The idea

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

But not possible in Coq because there is no *card* operation

Implementation: counting elements

$\forall\ l_1\ l_2,\ ilist\_perm\_occ_{R_d}\ l_1\ l_2 \Leftrightarrow \forall t, nb\_occ_{R_d}\ t\ l_1 = nb\_occ_{R_d}\ t\ l_2$
where $(nb\_occ\ t\ l)$ gives the number of occurences of t in l.

The problem

ilist_perm_occ needs decidability. Cannot be assumed for *Geq*.

Capturing Permutations on *ilist*

Inductive definition of permutations on *ilist*

$$\forall\ l_1\ l_2,\ ilist\_perm_R\ l_1\ l_2$$

$$\Leftrightarrow \begin{cases} lgti\ l_1 = lgti\ l_2 = 0 & \text{or} \\ \exists\ i_1\ i_2, R\ (fcti\ l_1\ i_1)\ (fcti\ l_2\ i_2) \wedge & \\ ilist\_perm_R\ (removeElement\ l_1\ i_1)\ (removeElement\ l_2\ i_2) & \end{cases}$$

$$\Leftrightarrow \begin{array}{l} lgti\ l_1 = lgti\ l_2\ \wedge\ (\forall i_1, \exists i_2, R\ (fcti\ l_1\ i_1)\ (fcti\ l_2\ i_2) \\ \wedge\ ilist\_perm_R\ (removeElement\ l_1\ i_1)\ (removeElement\ l_2\ i_2)) \end{array}$$

where *removeElement l i* removes the i^{th} element of l.

The proof of equivalence is not straightforward since one definition can be seen as a particular case of the other.

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

A Relation On *Graph* Using *ilist_perm*

An unsuccessful attempt

Definition of *GPerm* (coinductive)

$\forall\, g_1\ g_2,\ GPerm_R\ g_1\ g_2 \Leftrightarrow$
$R\ (label\ g_1)\ (label\ g_2)\ \wedge\ ilist\_perm_{GPerm_R}\ (sons\ g_1)\ (sons\ g_2)$

The problem: proof that *GPerm* preserves reflexivity

Lemma: $\forall\, R,\ R\ reflexive \Rightarrow \forall\, g,\ GPerm_R\ g\ g$

Proof (by coinduction): We must prove that

$$\underbrace{R\ (label\ g)\ (label\ g)}_{\text{ok}}\ \wedge\ \underbrace{ilist\_perm_{GPerm_R}\ (sons\ g)\ (sons\ g)}_{\text{has to be inductive}}$$

A Relation On *Graph* Using *ilist_perm*

An impredicative definition

The impredicative definition: implementation of $GPerm_R\ g_1\ g_2$

$\exists \mathcal{R},\ \Big(\forall\ g_1'\ g_2',\ \mathcal{R}\ g_1'\ g_2' \Rightarrow R\ (label\ g_1')\ (label\ g_2')\ \wedge$
$ilist\_perm_{\mathcal{R}}\ (sons\ g_1')\ (sons\ g_2')\Big)\ \wedge\ \mathcal{R}\ g_1\ g_2$
where variable $\mathcal{R}$ ranges over relations on *Graph T*

Tools and definitions

Coinduction principle: $(\forall\ g_1\ g_2,\ \mathcal{R}\ g_1\ g_2 \Rightarrow$
$R\ (label\ g_1)\ (label\ g_2) \wedge ilist\_perm_{\mathcal{R}}\ (sons\ g_1)\ (sons\ g_2)) \Rightarrow$
$\forall\ g_1\ g_2,\ \mathcal{R}\ g_1\ g_2 \Rightarrow GPerm_R\ g_1\ g_2$

Unfolding principle: $\forall\ g_1\ g_2,\ GPerm_R\ g_1\ g_2 \Rightarrow$
$R\ (label\ g_1)\ (label\ g_2) \wedge ilist\_perm_{GPerm_R}\ (sons\ g_1)\ (sons\ g_2)$

Constructor: $\forall\ g_1\ g_2,\ R\ (label\ g_1)\ (label\ g_2) \wedge$
$ilist\_perm_{GPerm_R}\ (sons\ g_1)\ (sons\ g_2) \Rightarrow GPerm_R\ g_1\ g_2$

A Relation On *Graph* Using *ilist_perm*

Mendler-style definition

Definition (coinductive)

$\forall\ g_1\ g_2,\ GPermMendler_R\ g_1\ g_2 \Leftrightarrow \forall X, X \subset GPermMendler_R \wedge R\ (label\ g_1)\ (label\ g_2)\ \wedge\ ilist\_perm_X\ (sons\ g_1)\ (sons\ g_2)$

Properties

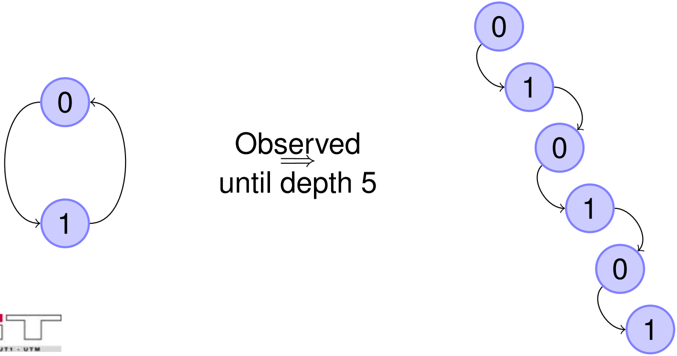
- Equivalent to *GPerm*
- Preserves equivalence

A Relation On *Graph* Using *ilist_perm*

An equivalent approach based on observation - The idea

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

$\Rightarrow$ no more mixing of inductive and coinductive types



A Relation On *Graph* Using *ilist_perm*

An equivalent approach based on observation - Definitions

TreeG (inductive): $mk\_TreeG : T \to ilist\ (TreeG\ T) \to TreeG\ T$

TPerm (inductive): $\forall\ t_1\ t_2,\ TPerm_R\ t_1\ t_2 \Leftrightarrow$
$R\ (labelT\ t_1)\ (labelT\ t_2) \wedge ilist\_perm_{TPerm_R}\ (sonsT\ t_1)\ (sonsT\ t_2)$

Graph2TreeG:

$$
\begin{array}{l}
Graph2TreeG : \forall\ T, nat \to Graph\ T \to TreeG\ T \\
Graph2TreeG\ T\ 0\ g := mk\_TreeG\ (label\ g)\ [\![]\!] \\
Graph2TreeG\ T\ (n+1)\ (mk\_Graph\ t\ l) := \\
\qquad mk\_TreeG\ t\ (imap\ (Graph2TreeG\ n)\ l)
\end{array}
$$

$\equiv_{R,n}$: $\forall\ n\ g_1\ g_2, g_1 \equiv_{R,n} g_2 \Leftrightarrow$
$TPerm_R\ (Graph2TreeG\ n\ g_1)\ (Graph2TreeG\ n\ g_2)$

GTPerm: $\forall\ g_1\ g_2, (GTPerm_R\ g_1\ g_2 \Leftrightarrow \forall n, g_1 \equiv_{R,n} g_2)$

A Relation On *Graph* Using *ilist_perm*

An equivalent approach based on observation - Main theorem(1/2)

The theorem

$\forall\ g_1\ g_2,\ GPerm_R\ g_1\ g_2 \Leftrightarrow GTPerm_R\ g_1\ g_2$

Proof

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

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

$\forall g_1\ g_2,\ GTPerm_R\ g_1\ g_2 \Rightarrow ilist\_perm_{GTPerm_R}\ (sons\ g_1)\ (sons\ g_2)$

A Relation On *Graph* Using *ilist_perm*

An equivalent approach based on observation - Main theorem (2/2)

The theorem

$\forall\ g_1\ g_2, GPerm_R\ g_1\ g_2 \Leftrightarrow GTPerm_R\ g_1\ g_2$

The auxiliary lemma

$\forall g_1\ g_2, GTPerm_R\ g_1\ g_2 \Rightarrow ilist\_perm_{GTPerm_R}\ (sons\ g_1)\ (sons\ g_2)$

Proof of the lemma

Main problem: problem of continuity. The unfolding gives:

$\forall g_1\ g_2, (\forall n, g_1 \equiv_{R,n} g_2) \Rightarrow ilist\_perm_{\cap_n \equiv_{R,n}}\ (sons\ g_1)\ (sons\ g_2)$

$\Rightarrow$ we have to "fix" a permutation $\forall n$.

A Relation On *Graph* Using *ilist_perm*

An equivalent approach based on observation - Main theorem (2/2)

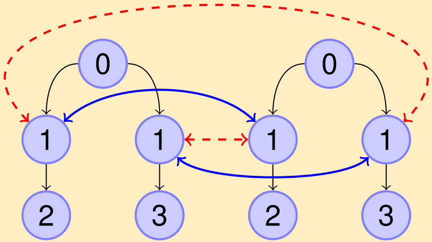
The theorem

$\forall\ g_1\ g_2,\ GPerm_R\ g_1\ g_2 \Leftrightarrow GTPerm_R\ g_1\ g_2$

The auxiliary lemma

$\forall g_1\ g_2,\ GTPerm_R\ g_1\ g_2 \Rightarrow ilist\_perm_{GTPerm_R}\ (sons\ g_1)\ (sons\ g_2)$

Proof of the lemma



A Relation On *Graph* Using *ilist_perm*

An equivalent approach based on observation - Main theorem (2/2)

The theorem

$\forall\ g_1\ g_2, GPerm_R\ g_1\ g_2 \Leftrightarrow GTPerm_R\ g_1\ g_2$

The auxiliary lemma

$\forall g_1\ g_2, GTPerm_R\ g_1\ g_2 \Rightarrow ilist\_perm_{GTPerm_R}\ (sons\ g_1)\ (sons\ g_2)$

Proof of the lemma

$\Rightarrow$ use of infinite pigeonhole principle

Need to manipulate permutations $\Rightarrow$ certificates:

$cert\_type\ 0 := unit$

$cert\_type\ (n+1) := (Fin\ (n+1) \times Fin\ (n+1)) \times cert\_type\ n$

A Relation On *Graph* Using *ilist_perm*

An equivalent approach based on observation - Main theorem (2/2)

The theorem

$\forall\ g_1\ g_2, GPerm_R\ g_1\ g_2 \Leftrightarrow GTPerm_R\ g_1\ g_2$

The auxiliary lemma

$\forall g_1\ g_2, GTPerm_R\ g_1\ g_2 \Rightarrow ilist\_perm_{GTPerm_R}\ (sons\ g_1)\ (sons\ g_2)$

Proof of the lemma

And we "include" them in *ilist_perm*:

$$\forall\ l_1\ l_2\ H_{lgti}\ c,\ ilist\_perm\_cert_R\ l_1\ l_2\ H_{lgti}\ c \Leftrightarrow$$
$$\begin{cases} lgti\ l_1 = 0 \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{or} \\ \exists\ i_1\ i_2\ c', R\ (fcti\ l_1\ i_1)\ (fcti\ l_2\ i_2)\ \wedge \text{“}c = ((i_1, i_2), c')\text{”}\ \wedge \\ ilist\_perm\_cert_R\ (removeElement\ l_1\ i_1) \\ \qquad\qquad\qquad\qquad (removeElement\ l_2\ i_2)\ H'_{lgti}\ c' \end{cases}$$

(equivalent to *ilist_perm*) / notion of continuity

A Relation On *Graph* Using *ilist_perm*

An equivalent approach based on observation - Main theorem (2/2)

The theorem

$\forall\ g_1\ g_2, GPerm_R\ g_1\ g_2 \Leftrightarrow GTPerm_R\ g_1\ g_2$

The auxiliary lemma

$\forall g_1\ g_2, GTPerm_R\ g_1\ g_2 \Rightarrow ilist\_perm_{GTPerm_R}\ (sons\ g_1)\ (sons\ g_2)$

Proof of the lemma

We prove:

$\forall n \exists c : cert\_type\ (lgti\ (sons\ g_1)),$
$ilist\_perm\_cert_{\equiv_{R,n}}\ (sons\ g_1)\ (sons\ g_2)\ H_{lgti}\ c \quad (H_1)$

Axiom of functional choice $\Rightarrow \phi$:

$\forall n, ilist\_perm\_cert_{\equiv_{R,n}}\ (sons\ g_1)\ (sons\ g_2)\ H_{lgti}\ (\phi\ n)\ (H_2)$

Infinite pigeonhole principle $\Rightarrow$ the “good” permutation c_0 such that: $\forall\ n\ \exists\ n', n' \geq n \wedge \phi\ n' = c_0$ (H_3).

A Relation On *Graph* Using *ilist_perm*

An equivalent approach based on observation - Main theorem (2/2)

The theorem

$\forall\ g_1\ g_2,\ GPerm_R\ g_1\ g_2 \Leftrightarrow GTPerm_R\ g_1\ g_2$

The auxiliary lemma

$\forall g_1\ g_2,\ GTPerm_R\ g_1\ g_2 \Rightarrow ilist\_perm_{GTPerm_R}\ (sons\ g_1)\ (sons\ g_2)$

Proof of the lemma

Using *ilist_perm* equivalent to *ilist_perm_cert*, goal becomes:

$ilist\_perm\_cert_{GTPerm_R}\ (sons\ g_1)\ (sons\ g_2)\ H_{lgti}\ c_0$

Continuity: $\forall n,\ ilist\_perm\_cert_{\equiv_{R,n}}\ (sons\ g_1)\ (sons\ g_2)\ H_{lgti}\ c_0$

Using H_2 and H_3:

$\forall n\ \exists n',\ n' \geq n \wedge ilist\_perm\_cert_{\equiv_{R,n'}}\ (sons\ g_1)\ (sons\ g_2)\ H_{lgti}\ c_0$

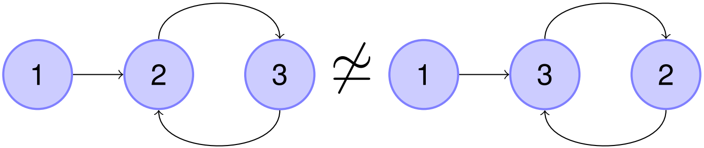
$\equiv_{R,n'} \subset \equiv_{R,n} \Rightarrow \forall n,\ ilist\_perm\_cert_{\equiv_{R,n}}(sons\ g_1)(sons\ g_2)\ H_{lgti}\ c_0$

$\square$

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 $\Rightarrow$ $g_1 \subset g_2 \wedge g_2 \subset g_1$
- Only for a “general” view:



The Final Relation Over *Graph*

Definitions

Inclusion

General definition (inductive):

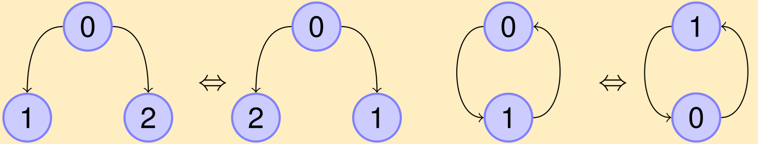
$$\forall g_{in}\ g_{out},\ GinG^{*}_{R_G}\ g_{in}\ g_{out} \Leftrightarrow \begin{cases} R_G\ g_{in}\ g_{out} & or \\ \exists i,\ GinG^{*}_{R_G}\ g_{in}\ (fcti\ (sons\ g_{out})\ i) \end{cases}$$

Instantiation: $GinGP_R := GinG^{*}_{GPerm_R}$

The final relation (coinductive)

$$\forall\ g_1\ g_2,\ GeqPerm_R\ g_1\ g_2 \Leftrightarrow GinGP_R\ g_1\ g_2 \wedge GinGP_R\ g_2\ g_1$$

Preserves equivalence



Related Work

Guardedness issues

- Bertot and Komendantskaya: same approach with streams
- Dams: defines everything coinductively and restricts the finite parts with properties of finiteness
- Niqui: general solution using category theory
- Danielsson: experimental solution to the problem in Agda (add constructors for each problematic function)
- Nakata and Uustalu: Mendler-style definition

Graph representation

- Erwig: inductive directed graph representation. Each node is added with its successors and predecessors.

Permutations

- Contejean: treats the same problem for lists

Conclusions and Perspectives

- Done so far:
  - Complete solution to overcome the guardedness condition in the case of lists
  - Permutations captured for *ilist*
  - Quite liberal equivalence relation on *Graph*
  - Completely formalised in Coq (available at: `www.irit.fr/~Celia.Picard/Coq/Permutations/`)
- Current work:
  - implementation of a small certified model transformation: finite automata minimization (done by a student)
  - use of *ilist* (and *ilistMult*) in infinite triangles
- Future work : equivalence with work by Contejean
- Perspectives:
  - More general solution for any inductive type
  - Deepening of coinductive representation of metamodels

Thanks for your attention. Questions ?

Conclusions and Perspectives

- Done so far:
  - Complete solution to overcome the guardedness condition in the case of lists
  - Permutations captured for *ilist*
  - Quite liberal equivalence relation on *Graph*
  - Completely formalised in Coq (available at: `www.irit.fr/~Celia.Picard/Coq/Permutations/`)
- Current work:
  - implementation of a small certified model transformation: finite automata minimization (done by a student)
  - use of *ilist* (and *ilistMult*) in infinite triangles
- Future work : equivalence with work by Contejean
- Perspectives:
  - More general solution for any inductive type
  - Deepening of coinductive representation of metamodels

Thanks for your attention. Questions ?