Permutations in Coinductive Graph Representation

Celia Picard and Ralph Matthes

Institut de Recherche en Informatique de Toulouse (IRIT)
affiliated with University of Toulouse and C.N.R.S.
Team : ACADIE

CMCS'12 in Tallinn, Estonia
research partly supported by the CLIMT project (ANR-11-BS02-016)
(some small typographical improvements applied to this presentation on April 3)

01/04/2012

Outline

The Problem
A first representation

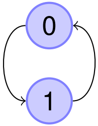
Context: certified model transformations (Coq)

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

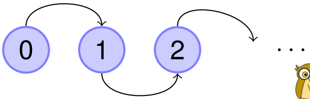
First attempt: coinductive constructor (for coinductive rose trees): $mk\_G : T \to list\ (Graph\ T) \to Graph\ T$

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 $T \to T'$):

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

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 $T \to T'$):

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

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 $T \to T'$):

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

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 container view of lists

ilist implementation

Implementation using **functions** to represent lists

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

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

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

An equivalence on *ilist*

$\forall l_1\ l_2 : ilist\ T, ilist\_rel_R\ l_1\ l_2 \Leftrightarrow$
$\exists h : lg\ l_1 = lg\ l_2 \to (\forall i : Fin\ (lg\ l_1), R\ (fct\ l_1\ i)\ (fct\ l_2\ i'_h))$ where lg and fct are projections on *ilist*, 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: $imap\ f\ l = \langle lg\ l, f \circ (fct\ l)\rangle$

The Solution: *ilist* – the container view of lists

ilist implementation

Implementation using **functions** to represent lists

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

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

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

An equivalence on *ilist*

$\forall l_1\ l_2 : ilist\ T, ilist\_rel_R\ l_1\ l_2 \Leftrightarrow$
$\exists h : lg\ l_1 = lg\ l_2 \to (\forall i : Fin\ (lg\ l_1), R\ (fct\ l_1\ i)\ (fct\ l_2\ i'_h))$ where
lg and *fct* are projections on *ilist*, 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: $imap\ f\ l = \langle lg\ l, f \circ (fct\ l)\rangle$

The Solution: *ilist* – the container view of lists

ilist implementation

Implementation using **functions** to represent lists

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

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

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

An equivalence on *ilist*

$\forall l_1\ l_2 : ilist\ T, ilist\_rel_R\ l_1\ l_2 \Leftrightarrow$
$\exists h : lg\ l_1 = lg\ l_2 \to (\forall i : Fin\ (lg\ l_1), R\ (fct\ l_1\ i)\ (fct\ l_2\ i'_h))$ where *lg* and *fct* are projections on *ilist*, 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: $imap\ f\ l = \langle lg\ l, f \circ (fct\ l)\rangle$

New Graph Representation
Definition of *Graph*

Graph (coinductive definition)

Graph : $mk\_G : T \to ilist(Graph\ T) \to Graph\ T$

applyF2G (corecursive definition)

applyF2G with $f : T \to T'$:
$applyF2G\ f\ (mk\_G\ t\ l) = mk\_G\ (f\ t)\ (imap\ (applyF2G\ f)\ l)$

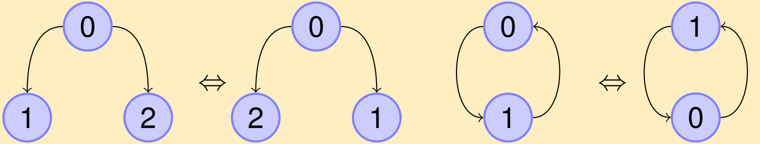
Equivalence on *Graph* (coinductively defined relation)

Geq generic coinductive notion of bisimilarity on *Graph*
$\forall g_1\ g_2 : Graph\ T,\ Geq_R\ g_1\ g_2 \Leftrightarrow$
$R\ (label\ g_1)\ (label\ g_2)\ \wedge\ ilist\_rel_{Geq_R}\ (sons\ g_1)\ (sons\ g_2)$
where *label* and *sons* are the projections on *Graph*

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*

Inductive definition of permutations on *ilist* (*iperm* and *iperm′* in the paper)

$$\forall l_1\ l_2,\ iperm_R\ l_1\ l_2$$

$$\Leftrightarrow \begin{cases} lg\ l_1 = lg\ l_2 = 0 & \text{or} \\ \exists i_1\ i_2, R\ (fct\ l_1\ i_1)\ (fct\ l_2\ i_2) \wedge & \\ iperm_R\ (remEl\ l_1\ i_1)\ (remEl\ l_2\ i_2) & \end{cases}$$

$$\Leftrightarrow \begin{array}{l} lg\ l_1 = lg\ l_2\ \wedge\ (\forall i_1 \exists i_2, R\ (fct\ l_1\ i_1)\ (fct\ l_2\ i_2) \\ \wedge\ iperm_R\ (remEl\ l_1\ i_1)\ (remEl\ l_2\ i_2)) \end{array}$$

where $remEl\ l\ i$ 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.

Capturing Permutations on *ilist*

Definition using bijective functions and comparison between definitions

Definition of *ipermb*

Idea : use a bijective function to define *ipermb* in the same style as *ilist_rel*. $\forall f\ g, bij\ f\ g \Leftrightarrow (\forall t, g(f\ t) = t) \wedge (\forall u, f(g\ u) = u)$
$\forall l_1\ l_2, ipermb_R\ l_1\ l_2 \Leftrightarrow \exists f\ g, bij\ f\ g \wedge (\forall i, R\ (fct\ l_1\ i)\ (fct\ l_2\ (f\ i)))$

Equivalence between definitions

- We can show that $\forall l_1\ l_2, iperm_R\ l_1\ l_2 \Leftrightarrow ipermb_R\ l_1\ l_2$
- Permutations on lists by Contejean equivalent to ours

Comparison between definitions

iperm (specially first def.) captures better the intuition than *ipermb* but is inductive. Contejean's definition is on lists. We prefer a definition on *ilist* $\Rightarrow$ our choice is *iperm* (first variant)

A Relation on *Graph* Using *iperm*

An unsuccessful attempt

Definition of *GPerm* (coinductive)

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

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

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

A Relation on *Graph* Using *iperm*

An impredicative definition — the type-theoretic way of getting a final coalgebra

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

$$\exists \mathcal{R},\ \Big(\forall\ g'_1\ g'_2,\ \mathcal{R}\ g'_1\ g'_2 \Rightarrow R\ (label\ g'_1)\ (label\ g'_2)\ \wedge$$
$$iperm_{\mathcal{R}}\ (sons\ g'_1)\ (sons\ g'_2)\Big)\ \wedge\ \mathcal{R}\ g_1\ g_2$$

where variable $\mathcal{R}$ ranges over relations on *Graph T*

Tools and definitions

Coinduction principle:

$$\Big(\forall\ g_1\ g_2,\ \mathcal{R}\ g_1\ g_2 \Rightarrow R\ (label\ g_1)\ (label\ g_2) \wedge$$
$$iperm_{\mathcal{R}}\ (sons\ g_1)\ (sons\ g_2)\Big) \Rightarrow\ \mathcal{R} \subseteq GPerm_R$$

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

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

A Relation On *Graph* Using *iperm*

Mendler-style definition — inspired by work of Keiko Nakata and Tarmo Uustalu

Definition (coinductive)

$\forall\ g_1\ g_2,\ \mathit{GPermMendler}_R\ g_1\ g_2 \Leftrightarrow \forall \mathcal{R}, \mathcal{R} \subseteq \mathit{GPermMendler}_R \wedge R\ (\mathit{label}\ g_1)\ (\mathit{label}\ g_2)\ \wedge\ \mathit{iperm}_{\mathcal{R}}\ (\mathit{sons}\ g_1)\ (\mathit{sons}\ g_2)$

Properties

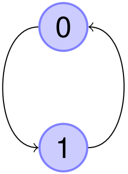
- Natively properly supported by Coq since only $\mathcal{R}$ enters the inductive predicate and not the relation $\mathit{GPermMendler}_R$
- Equivalent to *GPerm* (the impredicative implementation)
- Preserves equivalence — without Coq problems

A Relation On *Graph* Using *iperm*

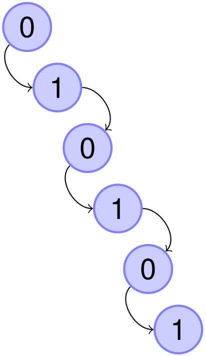
An equivalent approach based on observation - The idea

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

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



Observed $\Longrightarrow$ until depth 5



A Relation On *Graph* Using *iperm*

An equivalent approach based on observation of “rose trees” - Definitions

iTree (inductive): $mk\_iTree : T \to ilist\ (iTree\ T) \to iTree\ T$

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

G2iT:
$G2iT : \forall\ T, nat \to Graph\ T \to iTree\ T$
$G2iT\ T\ 0\ g := mk\_iTree\ (label\ g)\ [\![]\!]$
$G2iT\ T\ (n+1)\ (mk\_G\ t\ l) := mk\_iTree\ t\ (imap\ (G2iT\ n)\ l)$

$\equiv_{R,n}$: $\forall\ n\ g_1\ g_2, g_1 \equiv_{R,n} g_2 \Leftrightarrow TPerm_R\ (G2iT\ n\ g_1)\ (G2iT\ n\ g_2)$

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 *iperm*

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:

$GTPerm_R\ g_1\ g_2 \Rightarrow iperm_{GTPerm_R}\ (sons\ g_1)\ (sons\ g_2)$

A Relation On *Graph* Using *iperm*

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

$GTPerm_R\ g_1\ g_2 \Rightarrow iperm_{GTPerm_R}\ (sons\ g_1)\ (sons\ g_2)$

Proof of the lemma

Main problem: problem of continuity. The unfolding gives:

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

$\Rightarrow$ we need to “fix” a permutation that works for all n.

A Relation On *Graph* Using *iperm*

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

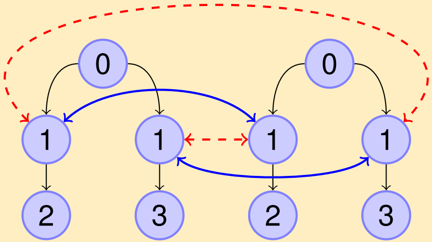
The theorem

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

The auxiliary lemma

$GTPerm_R\ g_1\ g_2 \Rightarrow iperm_{GTPerm_R}\ (sons\ g_1)\ (sons\ g_2)$

Proof of the lemma



A Relation On *Graph* Using *iperm*

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

$GTPerm_R\ g_1\ g_2 \Rightarrow iperm_{GTPerm_R}\ (sons\ g_1)\ (sons\ g_2)$

Proof of the lemma

$\Rightarrow$ use of infinite pigeonhole principle

Need to manipulate permutations $\Rightarrow$ “certificates”:

$skel\_type\ 0 := unit$

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

A Relation On *Graph* Using *iperm*

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

$$GTPerm_R\ g_1\ g_2 \Rightarrow iperm_{GTPerm_R}\ (sons\ g_1)\ (sons\ g_2)$$

Proof of the lemma

And we "include" them in *iperm*:

$$\forall\ l_1\ l_2\ H_{lgti}\ s,\ \ iperm\_skel_R\ l_1\ l_2\ H_{lgti}\ s \Leftrightarrow$$

$$\begin{cases} lg\ l_1 = 0 & \text{or} \\ \exists\ i_1\ i_2\ s',\ R\ (fct\ l_1\ i_1)\ (fct\ l_2\ i_2)\ \wedge \text{“}s = ((i_1, i_2), s')\text{”}\ \wedge & \\ iperm\_skel_R\ (remEl\ l_1\ i_1)\ (remEl\ l_2\ i_2)\ H'_{lgti}\ s' & \end{cases}$$

(equivalent to *iperm*) / notion of continuity

A Relation On *Graph* Using *iperm*

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

$GTPerm_R\ g_1\ g_2 \Rightarrow iperm_{GTPerm_R}\ (sons\ g_1)\ (sons\ g_2)$

Proof of the lemma

We first get:

$\forall n \exists s : skel\_type\ (lg\ (sons\ g_1)),$
$iperm\_skel_{\equiv_{R,n}}\ (sons\ g_1)\ (sons\ g_2)\ H_{lg}\ s$

The version of the infinite pigeonhole principle we want to use:

$\forall m\, \forall P : \mathbb{N} \to skel\_type\ m \to Prop,$
$(\forall n \exists s : skel\_type\ m, P\ n\ s) \to$
$\exists s_0 : skel\_type\ m, (\forall n \exists n', n' \geq n \wedge P\ n'\ s_0)$

A Relation On *Graph* Using *iperm*

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

$GTPerm_R\ g_1\ g_2 \Rightarrow iperm_{GTPerm_R}\ (sons\ g_1)\ (sons\ g_2)$

Proof of the lemma

Using *iperm* equivalent to *iperm_skel*, goal becomes:

$iperm\_skel_{GTPerm_R}\ (sons\ g_1)\ (sons\ g_2)\ H_{lg}\ s_0$

Continuity: $\forall n, iperm\_skel_{\equiv_{R,n}}\ (sons\ g_1)\ (sons\ g_2)\ H_{lg}\ s_0$

Using what the infinite pigeon hole principle says about s_0:

$\forall n\ \exists n', n' \geq n\ \wedge\ iperm\_skel_{\equiv_{R,n'}}\ (sons\ g_1)\ (sons\ g_2)\ H_{lg}\ s_0$

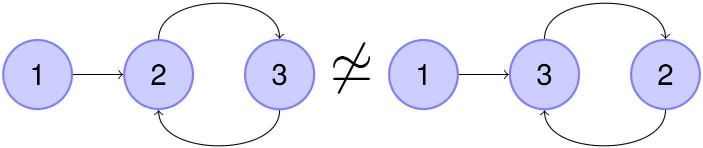
$\equiv_{R,n'} \subset \equiv_{R,n} \Rightarrow \forall n, iperm\_skel_{\equiv_{R,n}} (sons\ g_1)(sons\ g_2)\ H_{lg}\ s_0$

□

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):

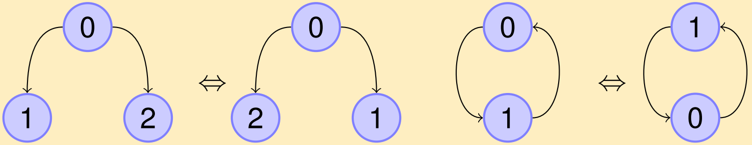
$$\forall g_{in}\ g_{out},\ GinG^*_{R_G}\ g_{in}\ g_{out} \Leftrightarrow \begin{cases} R_G\ g_{in}\ g_{out} & or \\ \exists i,\ GinG^*_{R_G}\ g_{in}\ (fct\ (sons\ g_{out})\ i) \end{cases}$$

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

The final relation

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

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: solution using category theory but not usable here
- 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/`)
  - But also completely described in mathematical language, see the forthcoming thesis by Celia
- Current and future work :
  - Instantiation of the graphs for finite automata (several contributions by quite some researchers to this ETAPS)
  - More general solution for any inductive type — the container view may be most helpful
  - Use of the present work to represent transformations in the ANR CLIMT project on categorical and logical methods in model transformation (Grenoble/Toulouse)