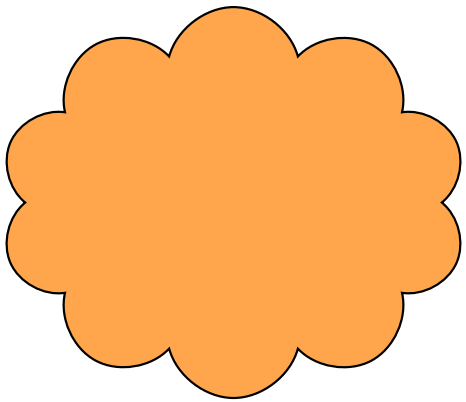
Coinductive Graph Representation

Celia Picard

IRIT - UPS
Team : ACADIE
Thesis advisor : Ralph Matthes

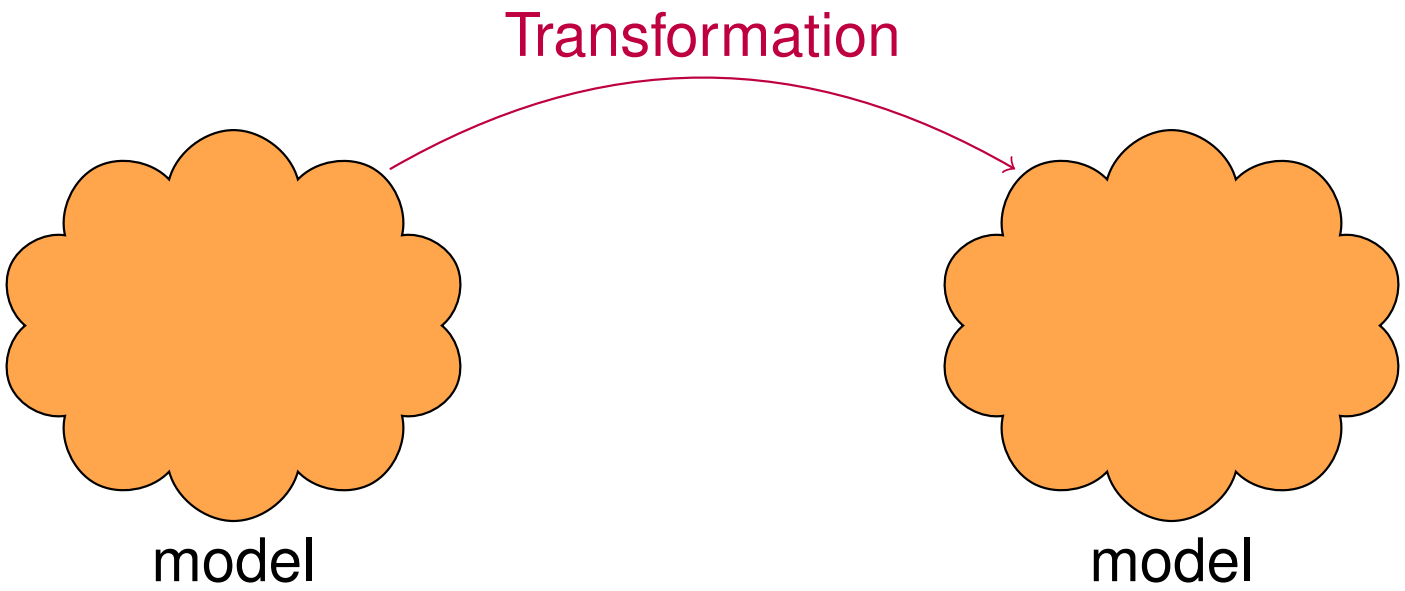
CaCos - 26/07/2012

Genesis: certified model transformation

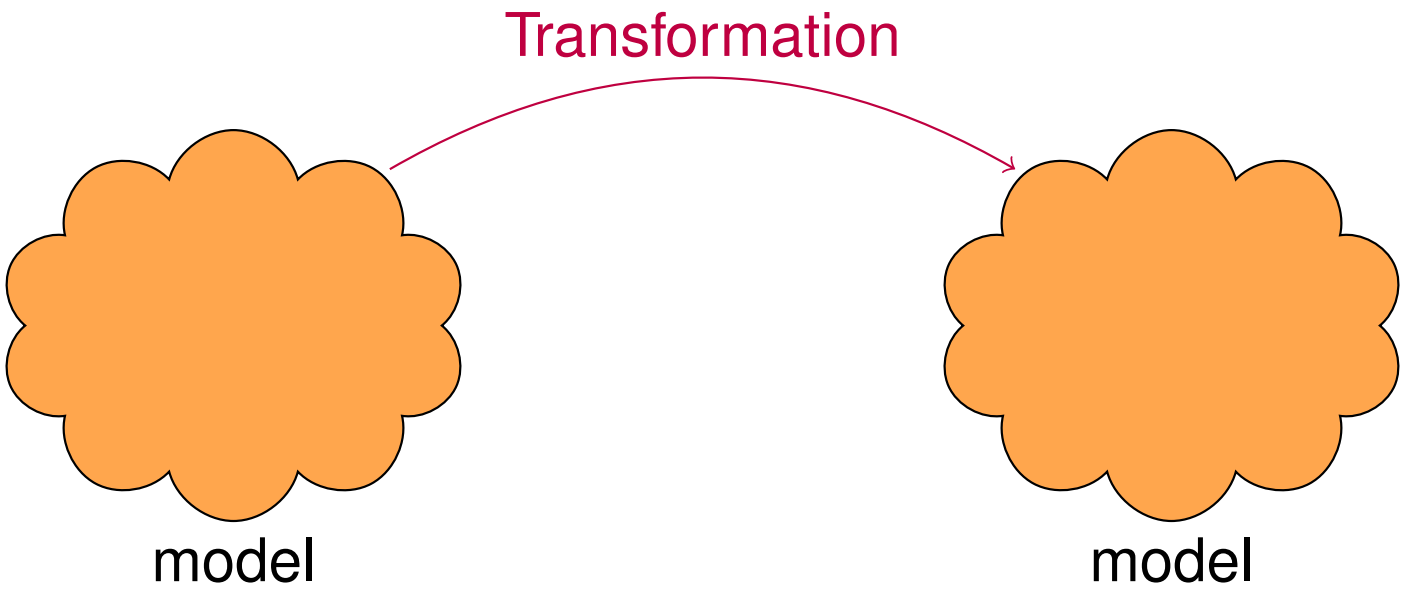


model

Genesis: certified model transformation

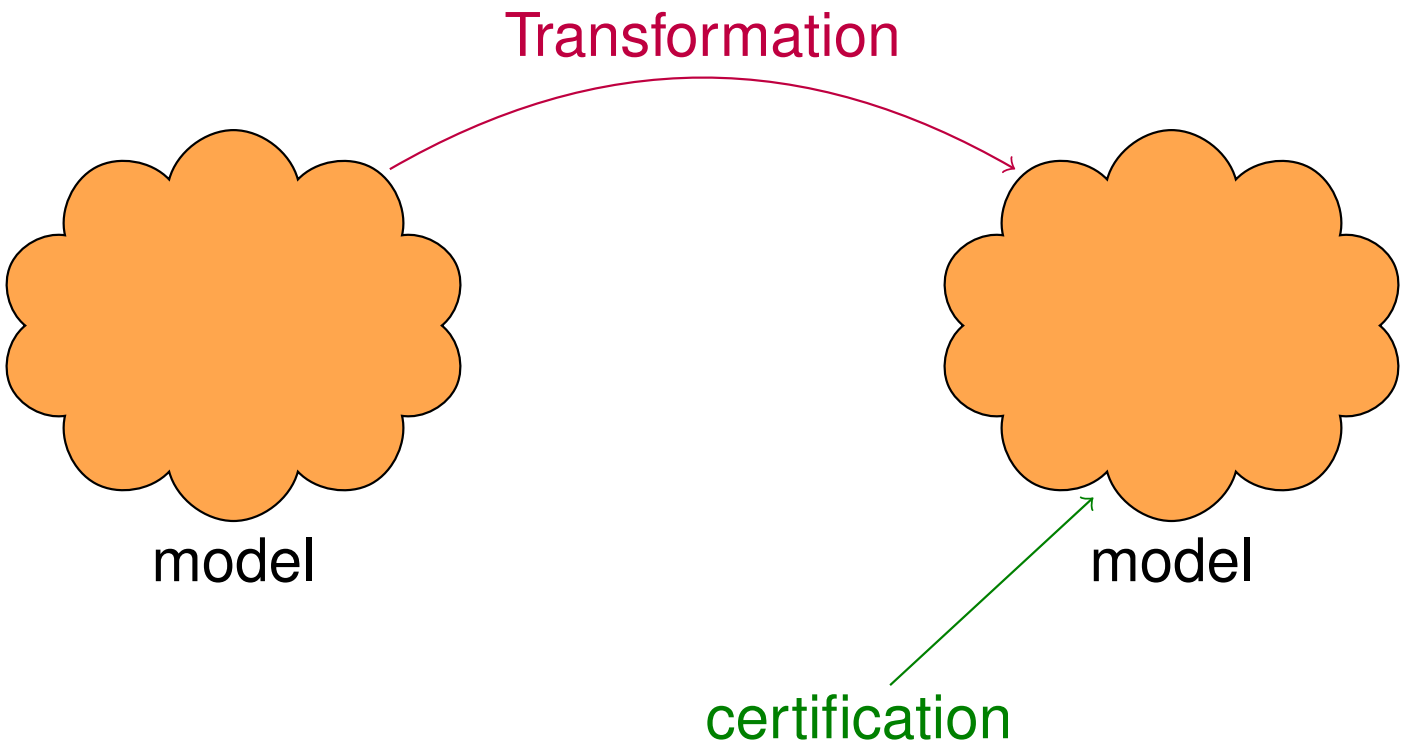


Genesis: certified model transformation

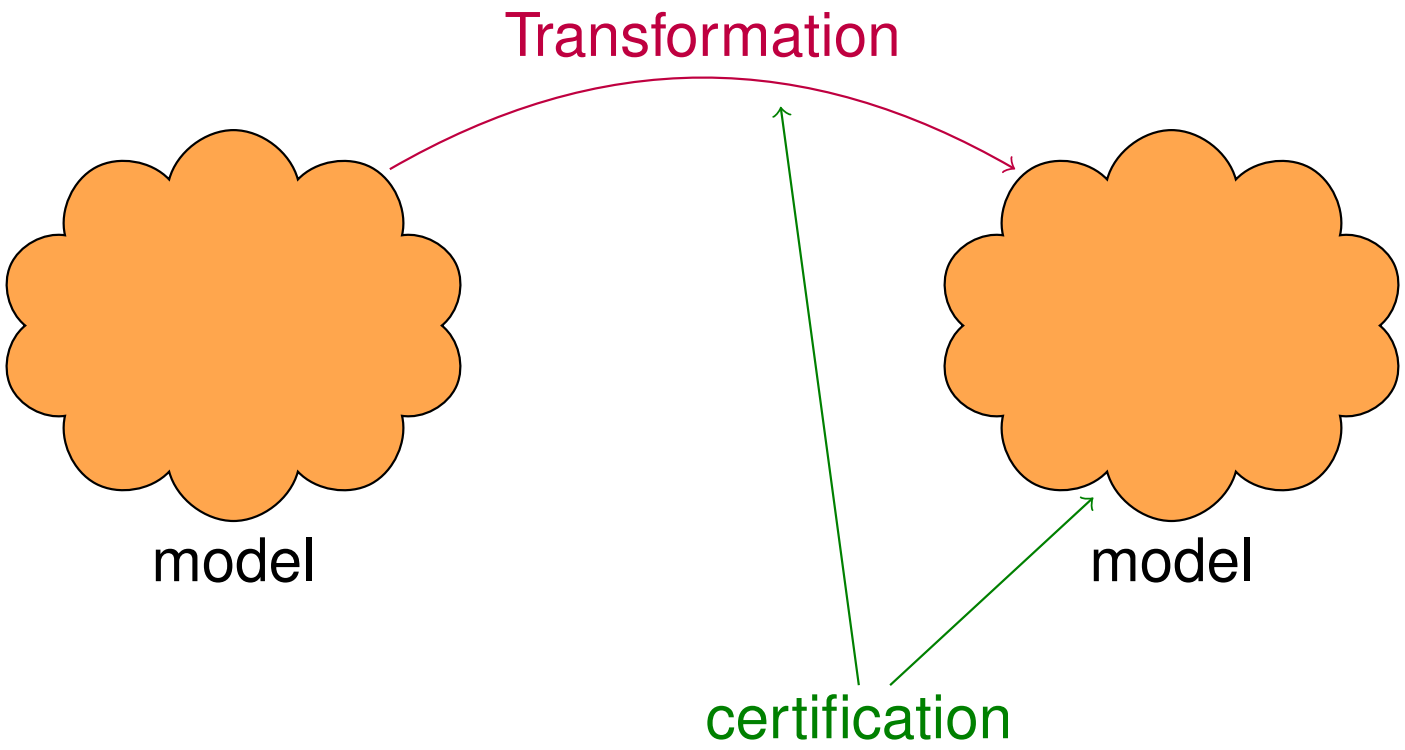


certification

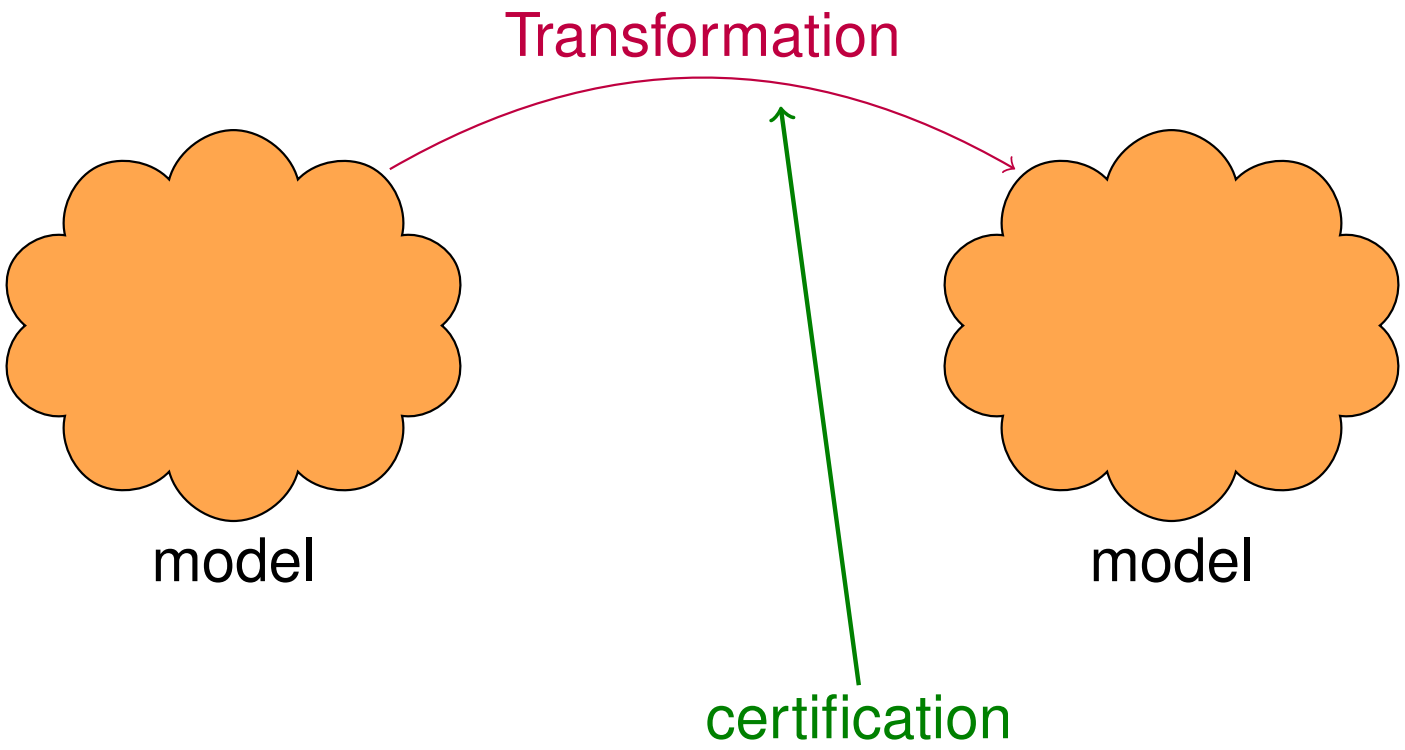
Genesis: certified model transformation



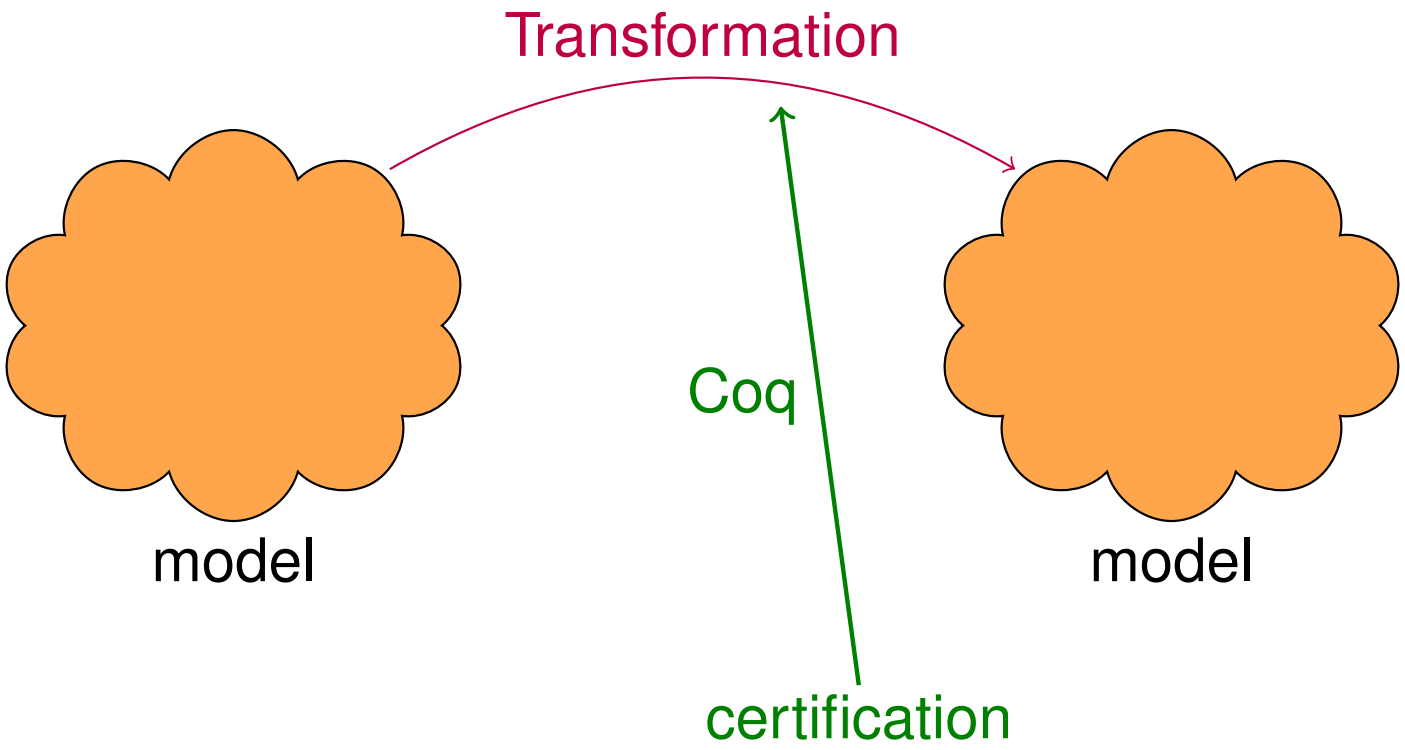
Genesis: certified model transformation



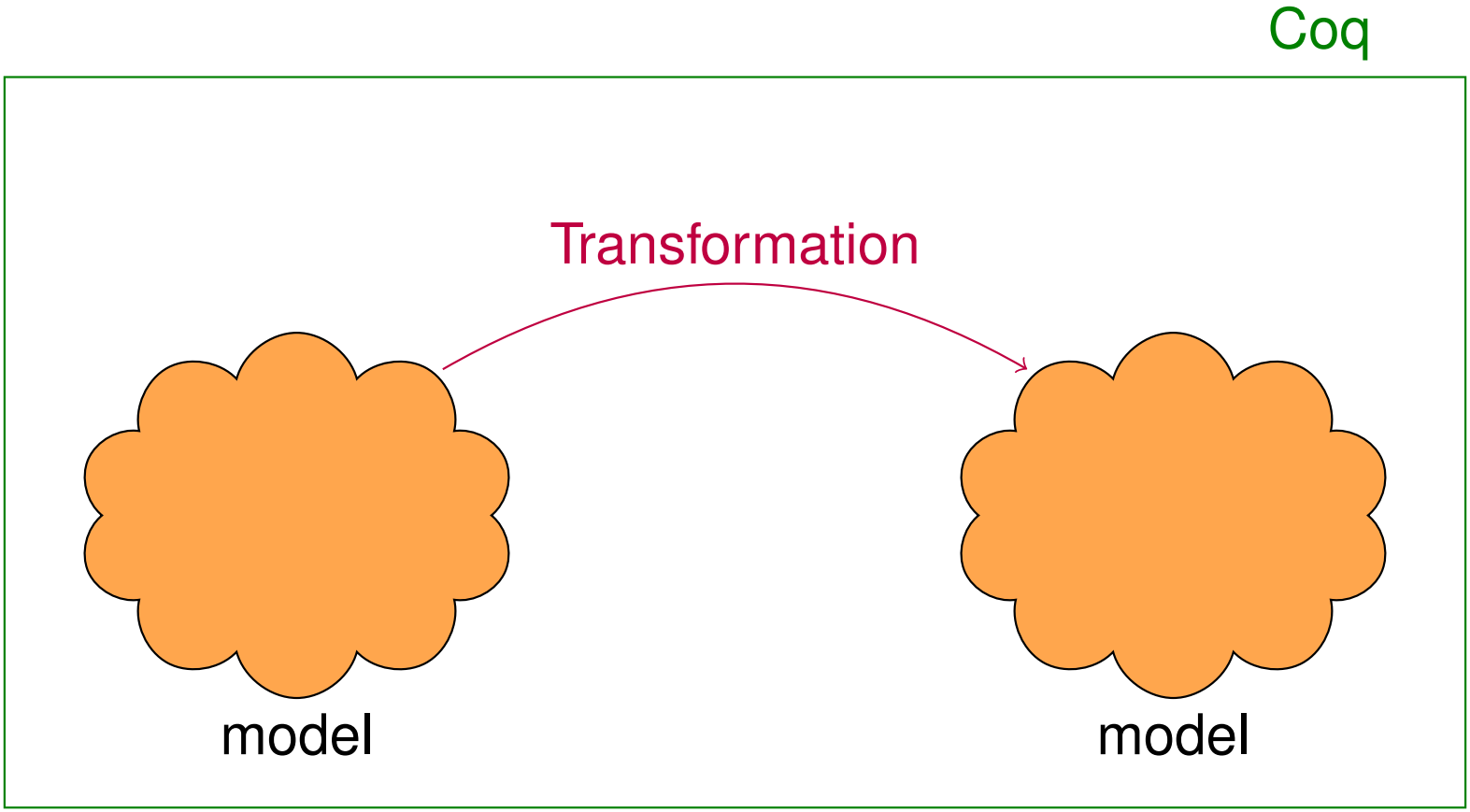
Genesis: certified model transformation



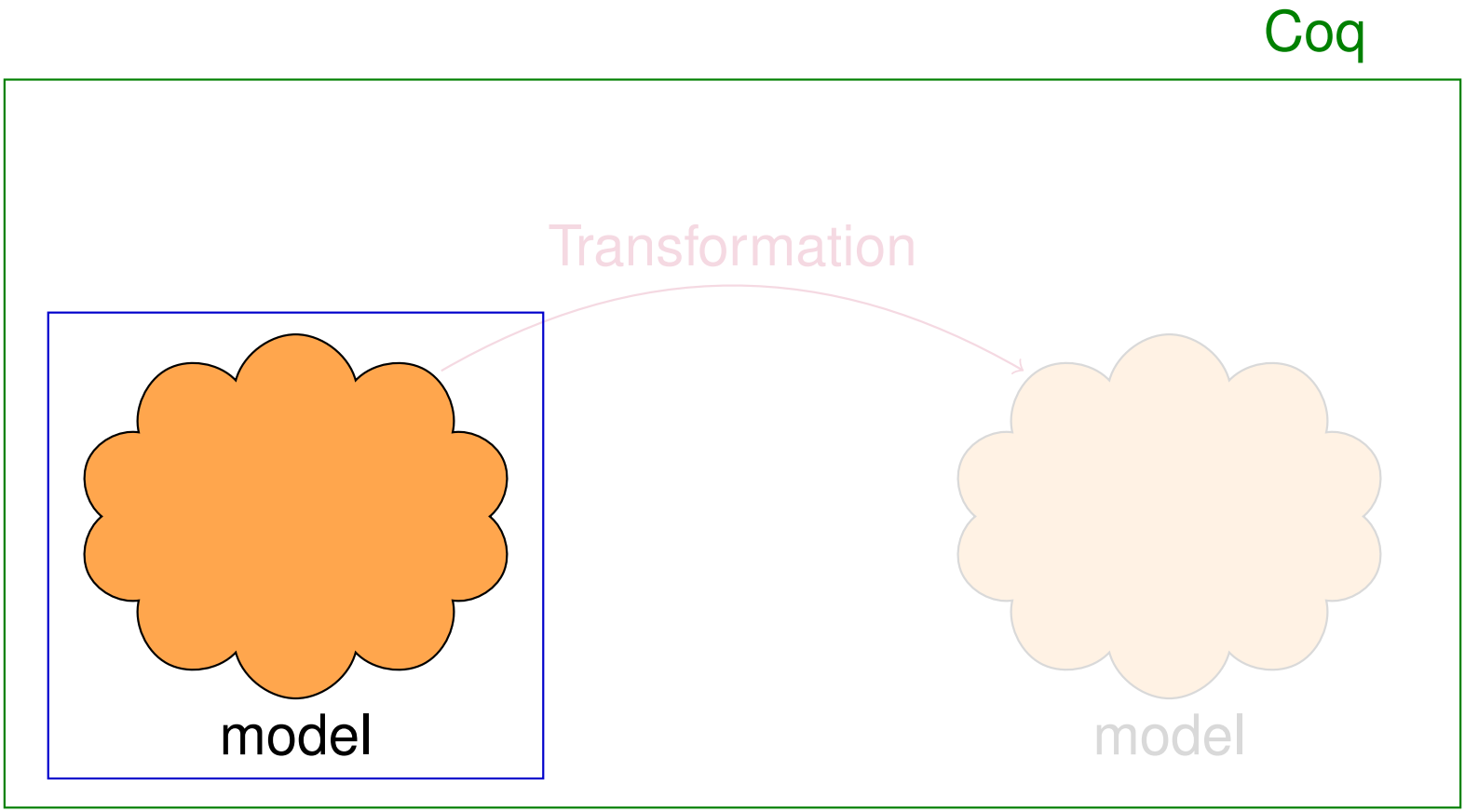
Genesis: certified model transformation



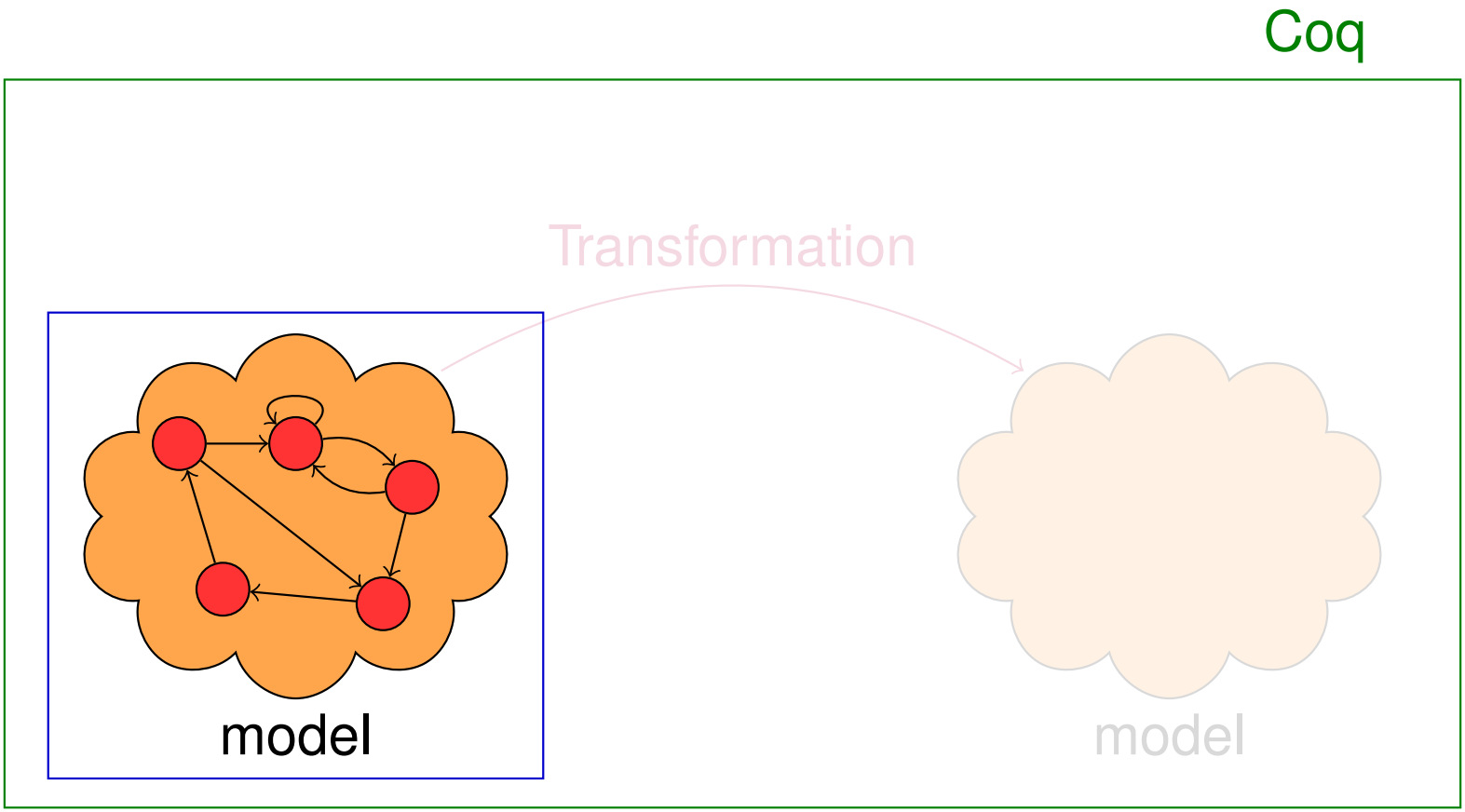
Genesis: certified model transformation



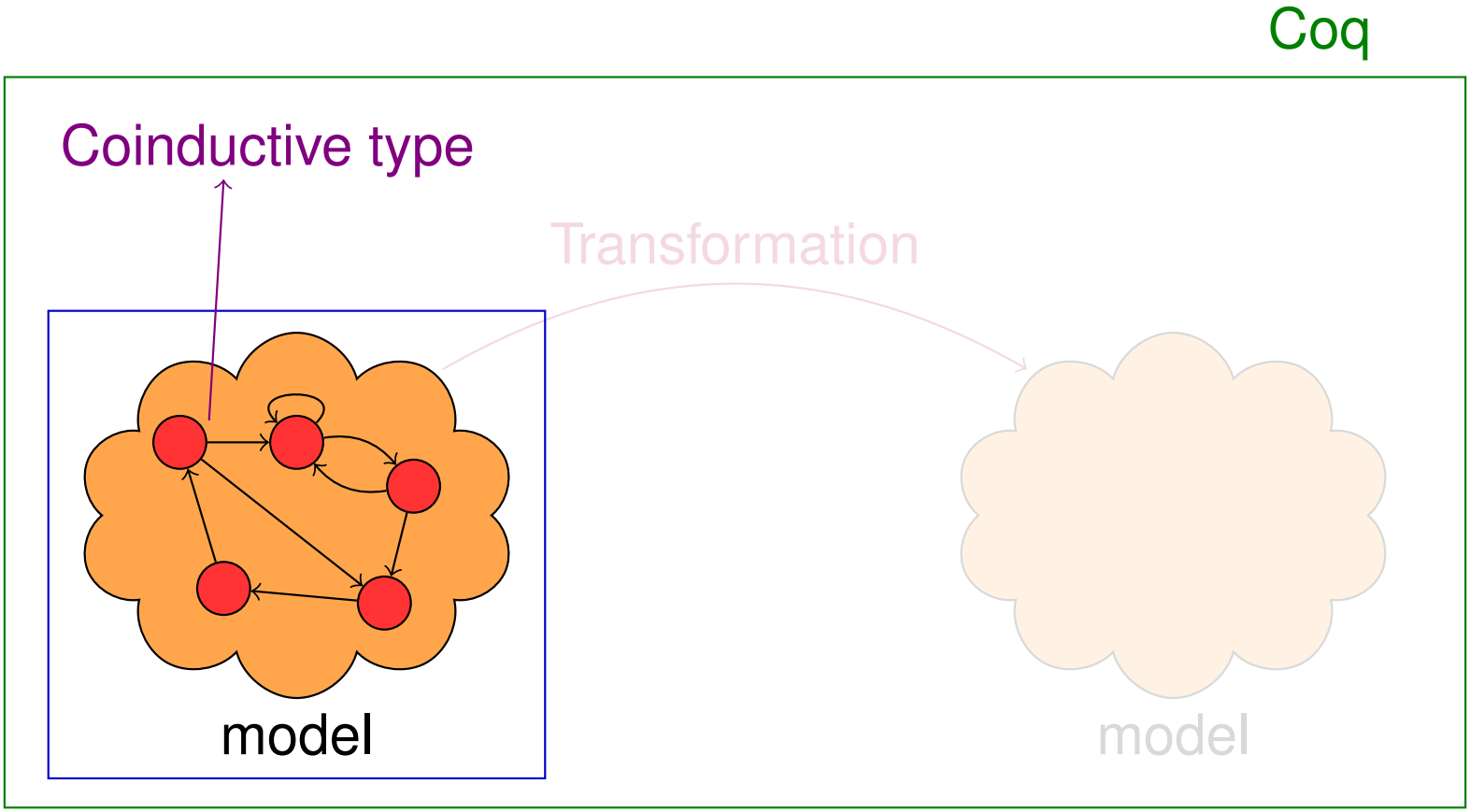
Genesis: certified model transformation



Genesis: certified model transformation



Genesis: certified model transformation



Coinductive representation

A first attempt

Definition

$$\frac{t : T \qquad l : list\ (Graph\ T)}{mk\_Graph\ t\ l : Graph\ T}$$

Examples

$Finite\_Graph =$
$mk\_Graph\ 0\ [mk\_Graph\ 1\ [Finite\_Graph]]$
$Infinite\_Graph_n =$
$mk\_Graph\ n\ [Infinite\_Graph_{n+1}]$

A first function

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

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

but... forbidden !

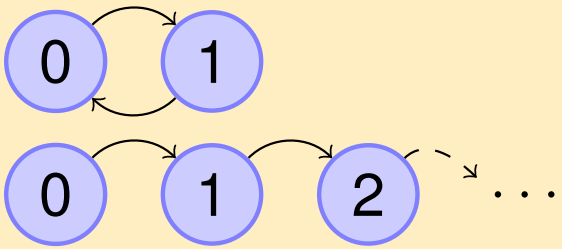
Coinductive representation

A first attempt

Definition

$$\frac{t : T \qquad l : list\ (Graph\ T)}{mk\_Graph\ t\ l : Graph\ T}$$

Examples



$Finite\_Graph =$
$mk\_Graph\ 0\ [mk\_Graph\ 1\ [Finite\_Graph]]$
$Infinite\_Graph_n =$
$mk\_Graph\ n\ [Infinite\_Graph_{n+1}]$

A first function

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

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

but... forbidden !

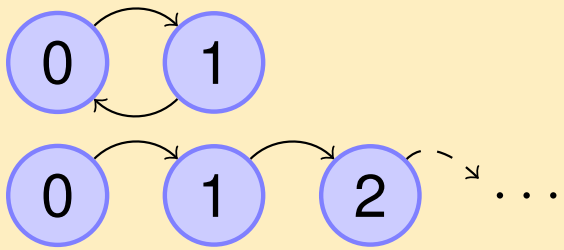
Coinductive representation

A first attempt

Definition

$$\frac{t : T \qquad l : list\ (Graph\ T)}{mk\_Graph\ t\ l : Graph\ T}$$

Examples



$Finite\_Graph =$
$mk\_Graph\ 0\ [mk\_Graph\ 1\ [Finite\_Graph]]$
$Infinite\_Graph_n =$
$mk\_Graph\ n\ [Infinite\_Graph_{n+1}]$

A first function

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

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

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

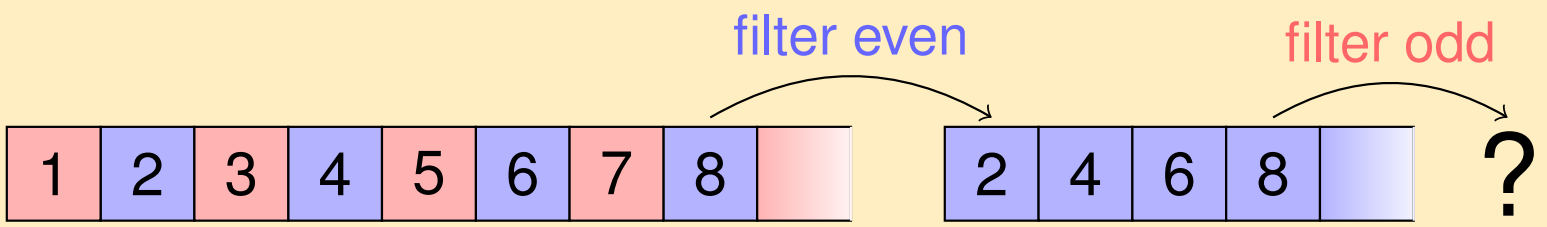
Problem: *applyF2G* actually **semantically correct**!
Solution: overcome guardedness condition (not change it)

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

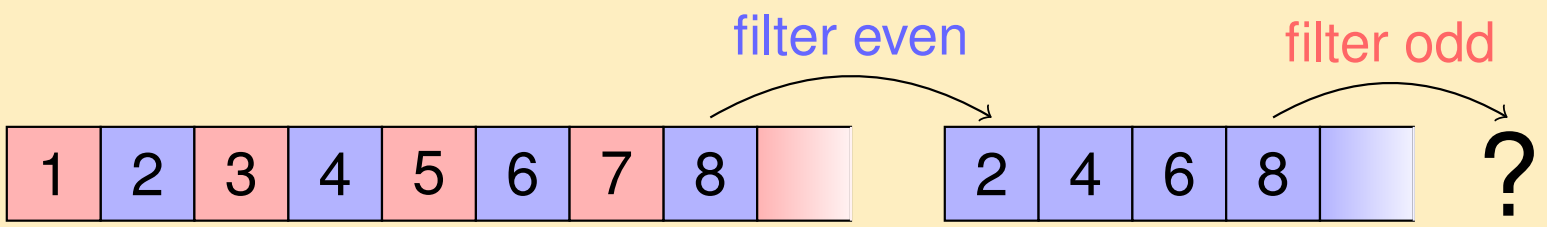
Problem: *applyF2G* actually **semantically correct**!
Solution: overcome guardedness condition (not change it)

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)

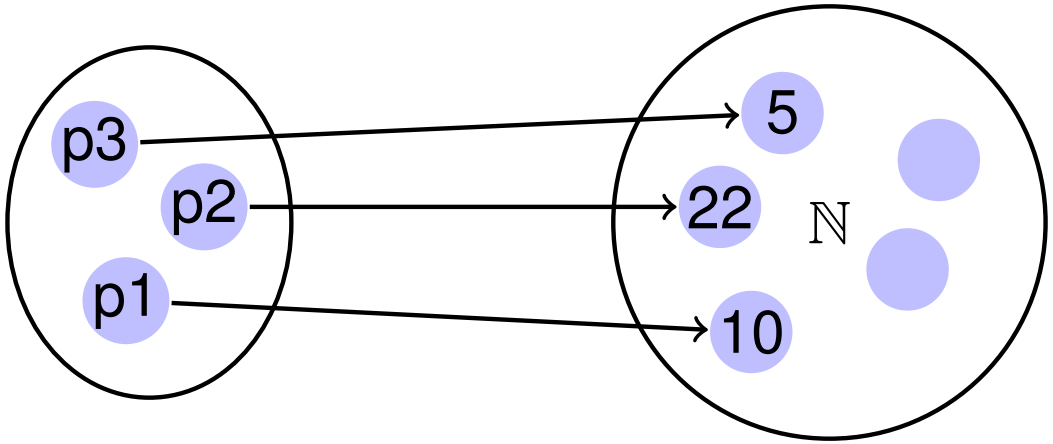
Outline

Outline

The idea

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

Example for the list [10 ; 22 ; 5]



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

Implementation of *ilist*

Implementation

The function : $ilistn\ (T : Set)\ (n : \mathbb{N}) = Fin\ n \to T$

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

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

An equivalence on *ilist*

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

Implementation of *ilist*

Implementation

The function : $ilistn\ (T : Set)\ (n : \mathbb{N}) = Fin\ n \to T$

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

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

An equivalence on *ilist*

$\forall l_1\ l_2 : ilist\ T, ilist\_rel_R\ l_1\ l_2 \Leftrightarrow$
$\forall h : lg\ l_1 = lg\ l_2, \forall i : Fin\ (lg\ l_1), R\ (fct\ l_1\ i)\ (fct\ l_2\ i'_h)$

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*: $imap\ f\ l = \langle lg\ l, f \circ (fct\ l)\rangle$

Implementation of *ilist*

Implementation

The function : $ilistn\ (T : Set)\ (n : \mathbb{N}) = Fin\ n \to T$

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

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

An equivalence on *ilist*

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

Outline

Capturing permutations on *ilist*

Permutations on *ilist* with decidability

The idea for comparing l_1 and l_2

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

Implementation: counting elements

$\forall l_1\ l_2,\ iperm\_occ_{R_d}\ l_1\ l_2 \Leftrightarrow \forall t, nbocc_{R_d}\ t\ l_1 = nbocc_{R_d}\ t\ l_2$
where $nbocc_{R_d}\ t\ l$ gives the number of occurrences of t in l.

The problem

iperm_occ needs **decidability**. Cannot always be assumed.

Capturing permutations on *ilist*

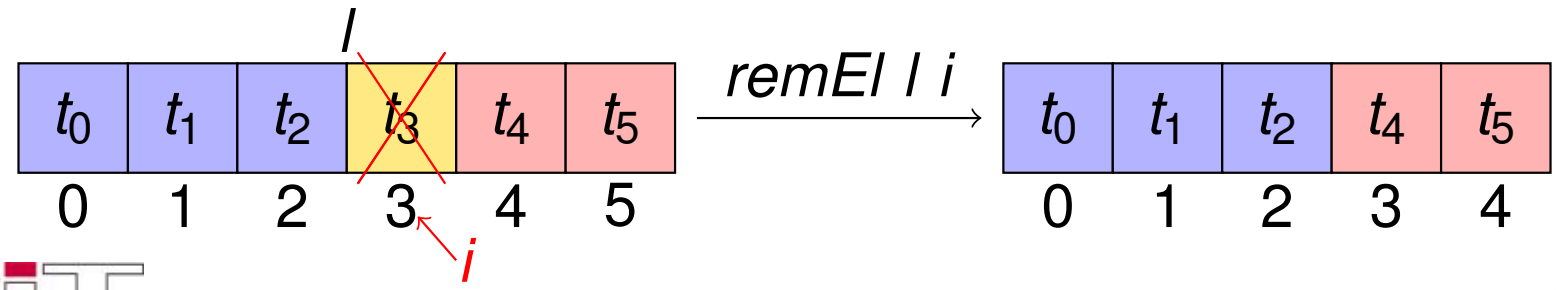
Inductive definitions of permutations on *ilist*- Definitions

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

$$iperm\_ind'_R\ l_1\ l_2 \Leftrightarrow lg\ l_1 = lg\ l_2 \wedge \big(\forall i_1 \exists i_2, R\ (fct\ l_1\ i_1)\ (fct\ l_2\ i_2) \wedge iperm\_ind'_R\ (remEl\ l_1\ i_1)\ (remEl\ l_2\ i_2)\big)$$

$$iperm\_ind''_R\ l_1\ l_2 \Leftrightarrow lg\ l_1 = lg\ l_2 \wedge \big(\forall i_2 \exists i_1, R\ (fct\ l_1\ i_1)\ (fct\ l_2\ i_2) \wedge iperm\_ind''_R\ (remEl\ l_1\ i_1)\ (remEl\ l_2\ i_2)\big)$$

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



Capturing permutations on *ilist*

Inductive definitions of permutations on *ilist*- Results

Theorem of equivalence between definitions

$\forall l_1\ l_2, iperm\_ind_R\ l_1\ l_2 \Leftrightarrow iperm\_ind'_R\ l_1\ l_2 \Leftrightarrow iperm\_ind''_R\ l_1\ l_2$

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: *skel_type*

Equivalent to *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*.

$\forall f\ g, bij\ f\ g \Leftrightarrow (\forall t, g(f\ t) = t) \wedge (\forall u, f(g\ u) = u)$

$\forall l_1\ l_2, iperm\_bij_R\ l_1\ l_2 \Leftrightarrow \exists f\ g, bij\ f\ g \wedge \forall i, R\ (fct\ l_1\ i)\ (fct\ l_2\ (f\ i))$

Equivalence between definitions

- We can show that $\forall l_1\ l_2, iperm\_ind_R\ l_1\ l_2 \Leftrightarrow iperm\_bij_R\ l_1\ l_2$
- 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*.

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*.

$\forall f\ g, bij\ f\ g \Leftrightarrow (\forall t, g(f\ t) = t) \wedge (\forall u, f(g\ u) = u)$

$\forall l_1\ l_2, iperm\_bij_R\ l_1\ l_2 \Leftrightarrow \exists f\ g, bij\ f\ g \wedge \forall i, R\ (fct\ l_1\ i)\ (fct\ l_2\ (f\ i))$

Equivalence between definitions

- We can show that $\forall l_1\ l_2, iperm\_ind_R\ l_1\ l_2 \Leftrightarrow iperm\_bij_R\ l_1\ l_2$
- 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*.

Outline

New graph representation

Definition of *Graph*

Graph and *applyF2G* (coinductive)

Graph : $$\frac{t : T \qquad l : ilist\ (Graph\ T)}{mk\_Graph\ t\ l : Graph\ T}$$

applyF2G : $applyF2G\ f\ (mk\_Graph\ t\ l) = mk\_Graph\ (f\ t)\ (imap\ (applyF2G\ f)\ l)$

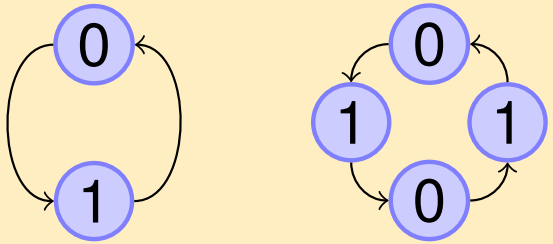
Bisimulation relation on *Graph*: *Geq*

Why ?

Graph is **infinite**

$\Rightarrow$ "=" not usable

Example:



$$\frac{R\ (label\ g_1)\ (label\ g_2) \qquad ilist\_rel_{Geq_R}\ (sons\ g_1)\ (sons\ g_2)}{Geq_R\ g_1\ g_2}$$

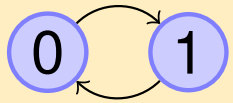
where *label* and *sons* are the projections on *Graph*

New graph representation
Finiteness

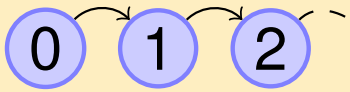
Notion of finiteness

Finiteness : $\forall g,\ G\_finite_R\ g \Leftrightarrow \exists gs,\ G\_all\ (element\_of_R\ gs)\ g$
with *Gall* universal quantification on *Graph* and *element_of* list membership modulo *Geq*

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

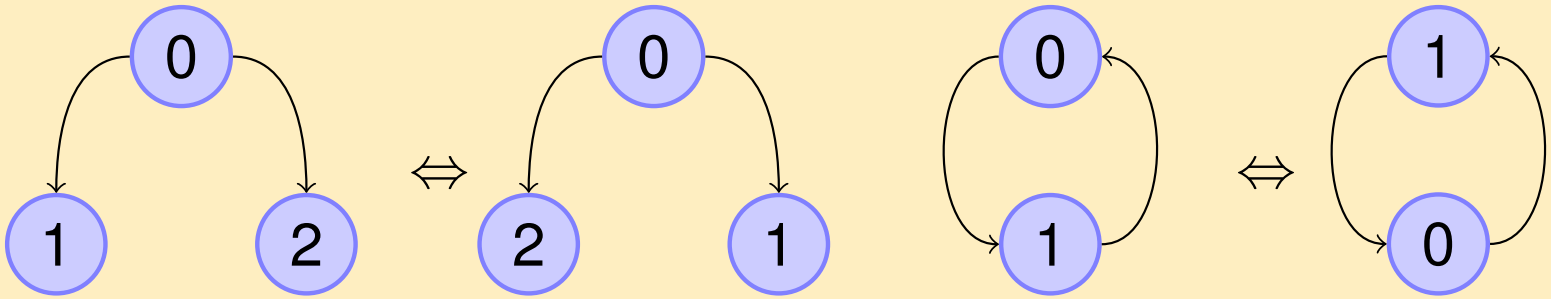
$G\_finite_=\ Finite\_Graph$: rather easy proof
$\forall n, \neg\ G\_finite_=\ Infinite\_Graph_n$: we use **unbounded labels**
labels and #sons bounded $\Rightarrow$ proofs of infinity much harder

Outline

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

A relation on *Graph* using *iperm_ind*

Definition of *GPerm* (coinductive)

$$\frac{R\ (label\ g_1)\ (label\ g_2) \qquad iperm\_ind_{GPerm_R}\ (sons\ g_1)\ (sons\ g_2)}{GPerm_R\ g_1\ g_2}$$

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

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

Mendler-style definition (coinductive and impredicative)

$$\frac{\mathcal{R} \subseteq GPerm\_mend_R \quad R\ (label\ g_1)\ (label\ g_2) \quad iperm\_ind_{\mathcal{R}}\ (sons\ g_1)\ (sons\ g_2)}{GPerm\_mend_R\ g_1\ g_2}$$

Preserves equivalence

A relation on *Graph* using *iperm_ind*

Definition of *GPerm* (coinductive)

$$\frac{R\ (label\ g_1)\ (label\ g_2) \qquad iperm\_ind_{GPerm_R}\ (sons\ g_1)\ (sons\ g_2)}{GPerm_R\ g_1\ g_2}$$

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

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

Mendler-style definition (coinductive and impredicative)

$$\frac{\mathcal{R} \subseteq GPerm\_mend_R \qquad R\ (label\ g_1)\ (label\ g_2) \qquad iperm\_ind_{\mathcal{R}}\ (sons\ g_1)\ (sons\ g_2)}{GPerm\_mend_R\ g_1\ g_2}$$

Preserves equivalence

A relation on *Graph* using *iperm_ind*

Definition of *GPerm* (coinductive)

$$\frac{R\ (label\ g_1)\ (label\ g_2) \qquad iperm\_ind_{GPerm_R}\ (sons\ g_1)\ (sons\ g_2)}{GPerm_R\ g_1\ g_2}$$

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

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

Mendler-style definition (coinductive and impredicative)

$$\frac{\mathcal{R} \subseteq GPerm\_mend_R \quad R\ (label\ g_1)\ (label\ g_2) \quad iperm\_ind_{\mathcal{R}}\ (sons\ g_1)\ (sons\ g_2)}{GPerm\_mend_R\ g_1\ g_2}$$

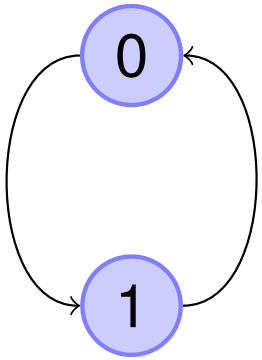
Preserves equivalence

A relation on *Graph* using *iperm_ind*

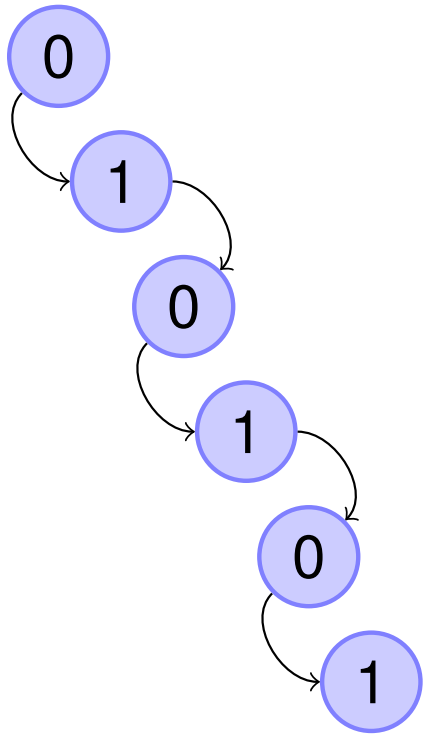
An equivalent approach based on observation - The idea

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

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



Observed $\Longrightarrow$ until depth 5



A relation on *Graph* using *iperm_ind*

An equivalent approach based on observation - Definitions

iTree (inductive): $\dfrac{t : T \qquad l : ilist\ (iTree\ T)}{mk\_iTree\ t\ l : iTree\ T}$

TPerm (inductive):

$$\frac{R\ (labeliT\ t_1)\ (labeliT\ t_2) \quad iperm\_ind_{TPerm_R}\ (sonsiT\ t_1)\ (sonsiT\ t_2)}{TPerm_R\ t_1\ t_2}$$

G2iT:

$G2iT : \forall T, nat \to Graph\ T \to iTree\ T$

$G2iT\ T\ 0\ (mk\_Graph\ t\ l) := mk\_Tree\ t\ [\![]\!]$

$G2iT\ T\ (n+1)\ (mk\_Graph\ t\ l) := mk\_Tree\ t\ (imap\ (G2iT\ n)\ l)$

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

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_ind*

An equivalent approach based on observation - Main theorem

The theorem

$\forall g_1\ g_2, GPerm\_mend_R\ g_1\ g_2 \Leftrightarrow GTPerm_R\ g_1\ g_2$

Proof

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

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

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

Modulo non-constructive axiom: **Infinite Pigeonhole Principle**

A relation on *Graph* using *iperm_ind*

An equivalent approach based on observation - Main theorem

The theorem

$\forall g_1\ g_2, GPerm\_mend_R\ g_1\ g_2 \Leftrightarrow GTPerm_R\ g_1\ g_2$

Proof

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

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

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

Modulo non-constructive axiom: **Infinite Pigeonhole Principle**

Relations on *Graph* using *iperm_bij*

Definitions

- Direct definition:

$$\frac{R\ (label\ g_1)\ (label\ g_2) \quad iperm\_bij_{GPerm\_bij_R}\ (sons\ g_1)\ (sons\ g_2)}{GPerm\_bij_R\ g_1\ g_2}$$

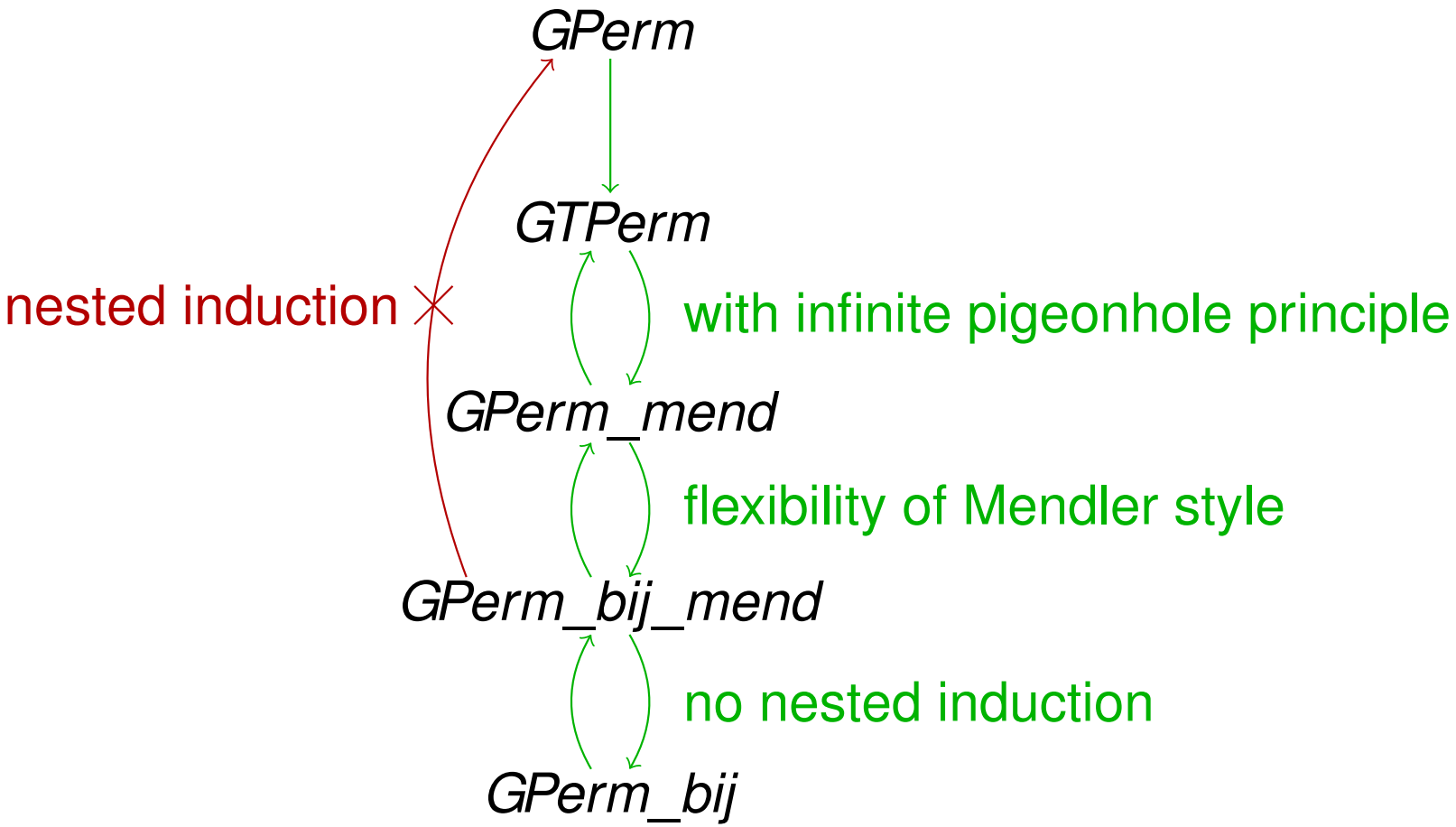
- Need an impredicative one for proofs of equivalence:

$$\frac{\mathcal{R} \subseteq GPerm\_bij\_mend_R \quad R(label\ g_1)(label\ g_2) \quad iperm\_bij_{\mathcal{R}}(sons\ g_1)(sons\ g_2)}{GPerm\_bij\_mend_R\ g_1\ g_2}$$

Results

- Equivalence relations
- $GPerm\_mend \Leftrightarrow GPerm\_bij\_mend \Leftrightarrow GPerm\_bij$

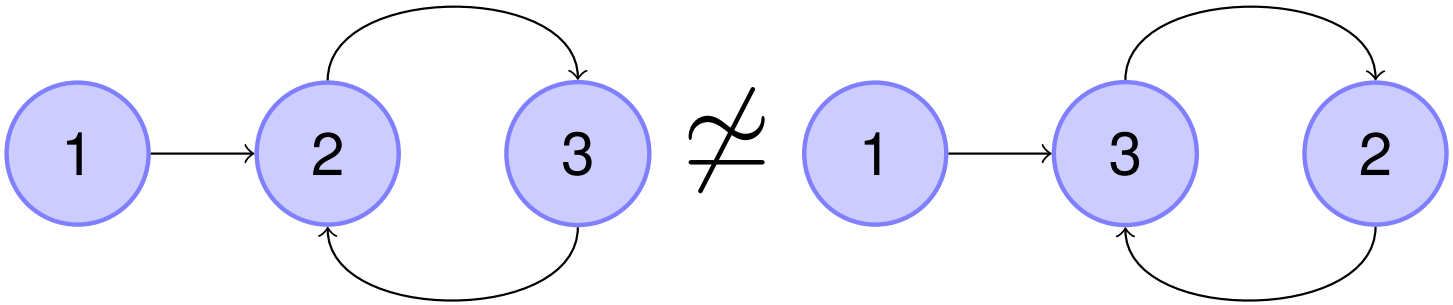
Summary of the obtained notions



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

Non-strict Inclusion

General definition (inductive):

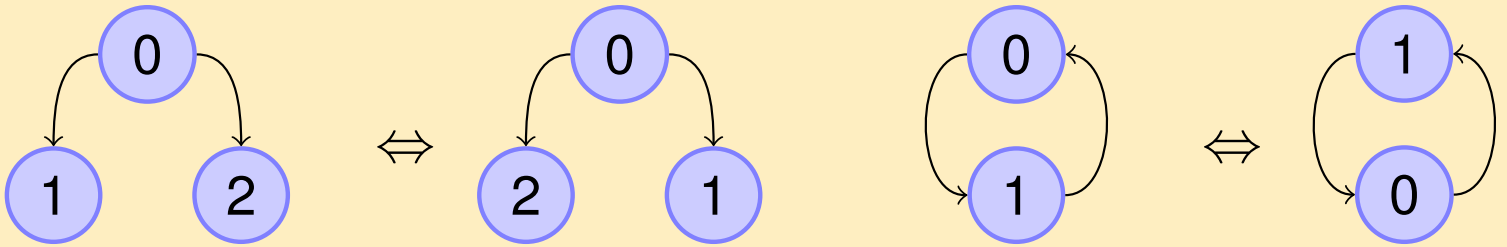
$$\forall g_{in}\ g_{out},\ GinG^{\star}_{R_G}\ g_{in}\ g_{out} \Leftrightarrow \begin{cases} R_G\ g_{in}\ g_{out} & \text{or} \\ \exists i,\ GinG^{\star}_{R_G}\ g_{in}\ (fct\ (sons\ g_{out})\ i) \end{cases}$$

Instantiation: $GinGP_R := GinG^{\star}_{GPerm\_mend_R}$

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



Outline

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/

Publications (with R. Matthes)

- Coinductive Graph Representation : the Problem of Embedded Lists - ECEASST, Vol. 39, 2011
- 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 $\mathbb{N} \to \mathit{Set}$ with 2 constructors:

$$\begin{array}{lll} \mathit{first} & (k : \mathbb{N}) : & \mathit{Fin}\ (k+1) \\ \mathit{succ} & (k : \mathbb{N}) : & \mathit{Fin}\ k \to \mathit{Fin}\ (k+1) \end{array}$$

Lemmas :

- $\forall n,\ \mathfrak{card}\ \{i \mid i : \mathit{Fin}\ n\} = n$
- $\forall n\ m, \mathit{Fin}\ n = \mathit{Fin}\ m \Rightarrow n = m$

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:

$\forall (inf : \mathbb{N})\ (sup : option\ \mathbb{N})\ (i : \mathbb{N}),$

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

ilistMult

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

A relation on *Graph* using *iperm_ind*
An impredicative definition

The impredicative definition: *GPerm_imp*

$$\forall g_1\ g_2, GPerm\_imp_R\ g_1\ g_2 \Leftrightarrow \exists \mathcal{R}, \Big(\forall g_1'\ g_2', \mathcal{R}\ g_1'\ g_2' \Rightarrow R\ (label\ g_1')\ (label\ g_2') \wedge iperm\_ind_{\mathcal{R}}\ (sons\ g_1')\ (sons\ g_2')\Big) \wedge \mathcal{R}\ g_1\ g_2$$

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

Tools and definitions

Coinduction principle: $\big(\forall g_1\ g_2,\ \mathcal{R}\ g_1\ g_2 \Rightarrow R\ (label\ g_1)\ (label\ g_2) \wedge iperm\_ind_{\mathcal{R}}\ (sons\ g_1)\ (sons\ g_2)\big) \Rightarrow \forall g_1\ g_2,\ \mathcal{R}\ g_1\ g_2 \Rightarrow GPerm\_imp_R\ g_1\ g_2$

Unfolding principle: $\forall g_1\ g_2,\ GPerm\_imp_R\ g_1\ g_2 \Rightarrow R\ (label\ g_1)\ (label\ g_2) \wedge iperm\_ind_{GPerm\_imp_R}\ (sons\ g_1)\ (sons\ g_2)$

Constructor: $\forall g_1\ g_2,\ R\ (label\ g_1)\ (label\ g_2) \wedge iperm\_ind_{GPerm\_imp_R}\ (sons\ g_1)\ (sons\ g_2) \Rightarrow GPerm\_imp_R\ g_1\ g_2$

A relation on *Graph* using *iperm_ind*

An equivalent approach based on observation - Main theorem

The theorem

$\forall g_1\ g_2, GPerm\_mend_R\ g_1\ g_2 \Leftrightarrow GTPerm_R\ g_1\ g_2$

Proof

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

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

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

Modulo non-constructive axiom: **Infinite Pigeonhole Principle**

A relation on *Graph* using *iperm_ind*

An equivalent approach based on observation - Main theorem

The theorem

$\forall g_1\ g_2, GPerm\_mend_R\ g_1\ g_2 \Leftrightarrow GTPerm_R\ g_1\ g_2$

Proof

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

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

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

Modulo non-constructive axiom: **Infinite Pigeonhole Principle**

A relation on *Graph* using *iperm_ind*

An equivalent approach based on observation - Main theorem

The theorem

$\forall g_1\ g_2, GPerm\_mend_R\ g_1\ g_2 \Leftrightarrow GTPerm_R\ g_1\ g_2$

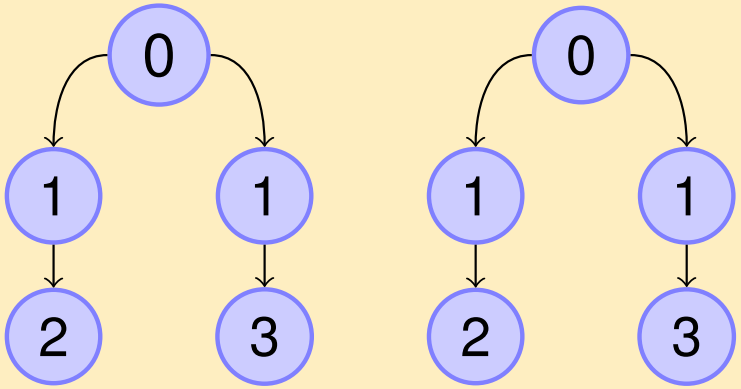
Proof

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

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

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

Modulo non-constructive axiom: **Infinite Pigeonhole Principle**



A relation on *Graph* using *iperm_ind*

An equivalent approach based on observation - Main theorem

The theorem

$\forall g_1\ g_2, GPerm\_mend_R\ g_1\ g_2 \Leftrightarrow GTPerm_R\ g_1\ g_2$

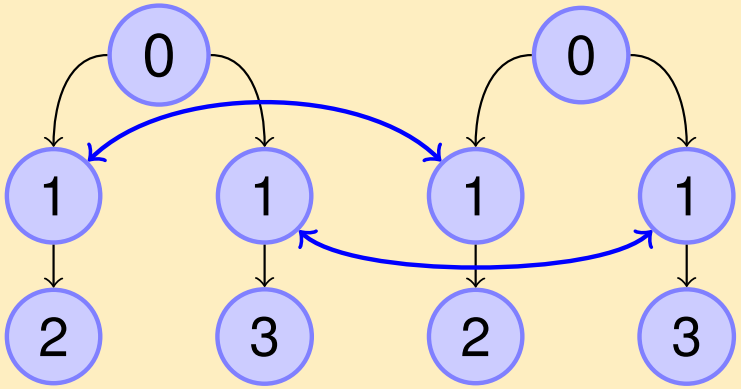
Proof

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

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

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

Modulo non-constructive axiom: **Infinite Pigeonhole Principle**



A relation on *Graph* using *iperm_ind*

An equivalent approach based on observation - Main theorem

The theorem

$\forall g_1\ g_2, GPerm\_mend_R\ g_1\ g_2 \Leftrightarrow GTPerm_R\ g_1\ g_2$

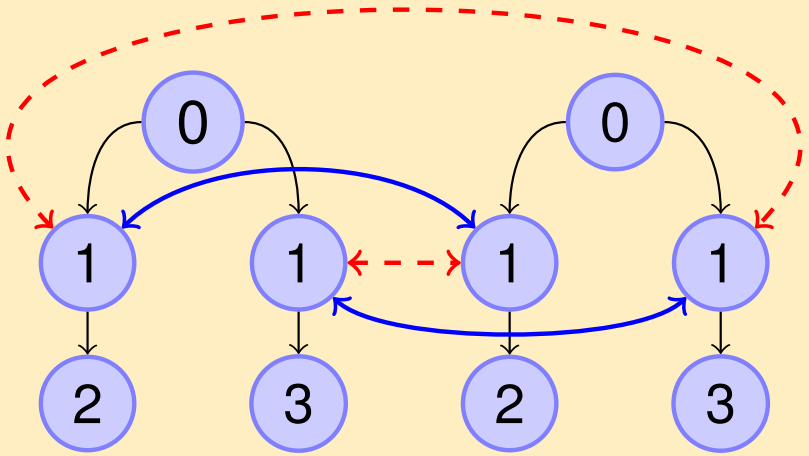
Proof

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

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

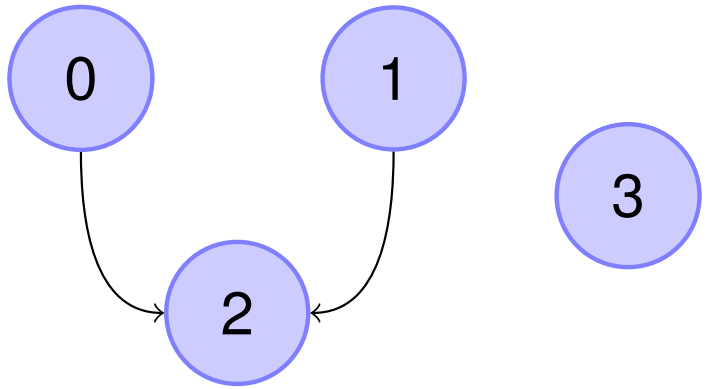
$\forall g_1\ g_2, GTPerm_R\ g_1\ g_2 \Rightarrow iperm\_ind_{GTPerm_R}\ (sons\ g_1)\ (sons\ g_2)$

Modulo non-constructive axiom: **Infinite Pigeonhole Principle**



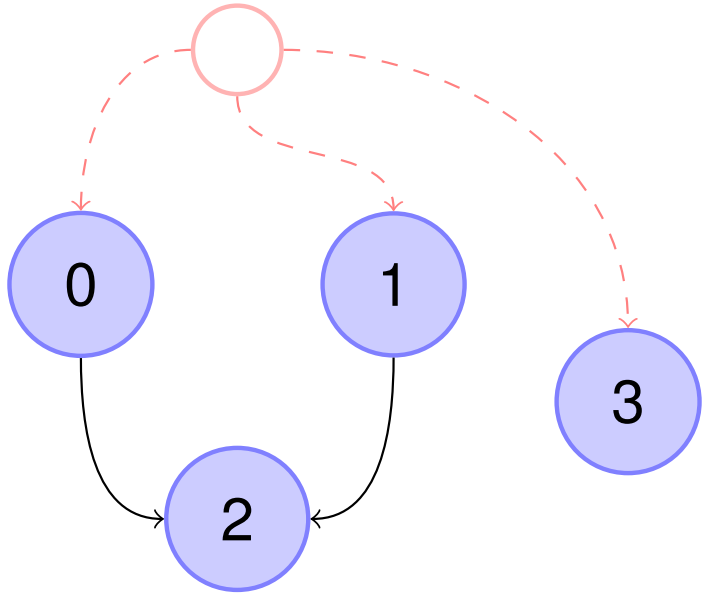
A representation of a wider class of graphs

We would like to represent graphs like this one:



A representation of a wider class of graphs

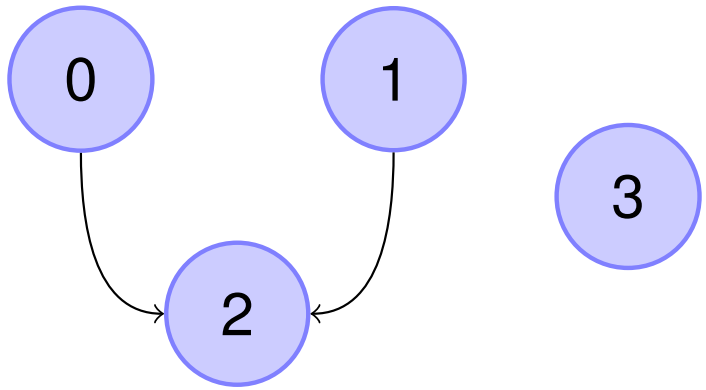
Solution: fictitious nodes.



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

A representation of a wider class of graphs

Other solution: forest.



AllGraph: $AllGraph\ T := list\ (Graph\ T)$