Equality and equivalence relations in formal proofs

Pierre CORBINEAU

DCS day, Autrans, 26-27 march 2009

Outline

1. A short introduction to myself
2. Equality and equivalence relations in Coq

Curriculum

1998-2002 Student at ENS, rue d'Ulm

spring 2000 Stage (4 months) with Rance Cleaveland
SUNY Stony Brook (NY, USA)
first contact with *model-checking*

2001-2005 Ph.D. student at Université Paris-Sud
with Christine Paulin-Mohring and Claude Marché
Automated reasoning in Type Theory

2005-2008 Post-Doc Radboud Universiteit Nijmegen
with Herman Geuvers and Henk Barendregt
Languages and interfaces for formal proofs

Recherche topic: formal proofs

- Computer-hosted and -handled object
- explicit et detailed description of a reasoning process
- Can be checked mechanically

Proof Assistants for :

- Formalising mathematics (4 colours Theorem)
- Critical software and system verification (CompCert)

Problems with formal proofs :

- Lengthy and tedious work: little automation
- Complicated and arbitrary Proof Language
- Disposable write-only Proofs

Research contributions: Ph.D.

Pragmatic approach:

1. Metatheoretical justification
2. Implementation and distribution

Thesis: Automating reasoning in Coq

- Equational logic
  congruence tactic implemented and released with Coq
- Intuitionnistic first-order logic
  firstorder tactic implemented and released with Coq
- Importing proofs from external automated tools
  Method using computational reflection
  Prototype for rewriting with CiME

Impact : Widely used procedures (CompCert. . .) A3PAT and DeCert Projects (CNAM, LRI)

Research contributions: Post-doc

Development of innovative proof interfaces

- The C-zar proof language
  Simple langage with few instructions
  Explicit logic based langage
  Increased readability
- Proof interfaces: The Wiki way
  A Wiki-Coq prototype
  Collaboration and outreach platform
  Project proposals (STREP – refused , Dutch – accepted)

Metatheoretical research :

- Enriched pattern-matching constructs for Type Theory
  Objective: programming and easier proofs with dependently-typed objects

The C-zar proof language

```
Lemma double_div2: forall n, div2 (double n) = n.
proof.




end proof.
Qed.
```

The C-zar proof language

```
Lemma double_div2: forall n, div2 (double n) = n.
proof.
  let n:nat.
  per induction on n.








  end induction.
end proof.
Qed.
```

The C-zar proof language

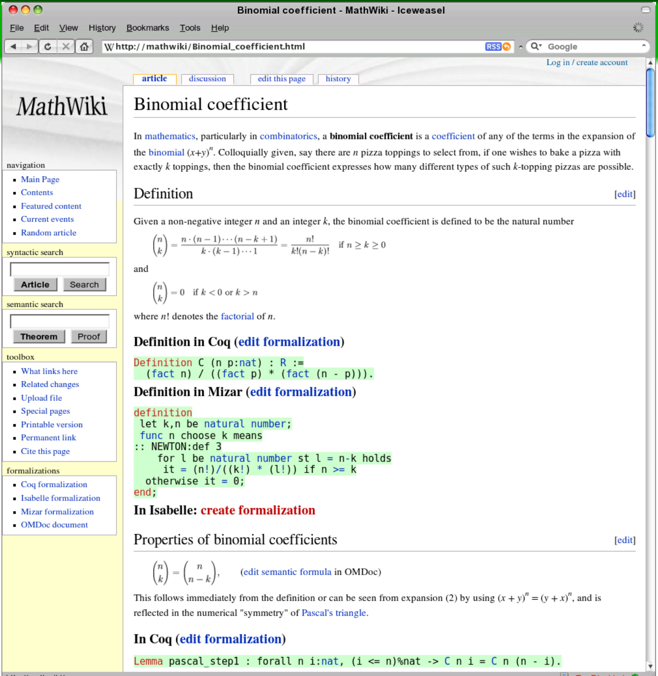
```
Lemma double_div2: forall n, div2 (double n) = n.
proof.
  let n:nat.
  per induction on n.
    suppose it is 0.

    suppose it is (S m) and Hrec:thesis for m.




  end induction.
end proof.
Qed.
```

The C-zar proof language

```
Lemma double_div2: forall n, div2 (double n) = n.
proof.
  let n:nat.
  per induction on n.
    suppose it is 0.
      thus (0=0).
    suppose it is (S m) and Hrec:thesis for m.
      have (div2 (double (S m))
            = div2 (S (S (double m)))).
           ~= (S (div2 (double m))).
      thus ~= (S m) by Hrec.
  end induction.
end proof.
Qed.
```

MathWiki

Wiki + proof assistants



Binomial coefficient - MathWiki - Iceweasel

File Edit View History Bookmarks Tools Help

http://mathwiki/Binomial_coefficient.html

Log in / create account

article | discussion | edit this page | history

MathWiki

navigation
- Main Page
- Contents
- Featured content
- Current events
- Random article

syntactic search
Article | Search

semantic search
Theorem | Proof

toolbox
- What links here
- Related changes
- Upload file
- Special pages
- Printable version
- Permanent link
- Cite this page

formalizations
- Coq formalization
- Isabelle formalization
- Mizar formalization
- OMDoc document

Binomial coefficient

In mathematics, particularly in combinatorics, a **binomial coefficient** is a coefficient of any of the terms in the expansion of the binomial $(x+y)^n$. Colloquially given, say there are n pizza toppings to select from, if one wishes to bake a pizza with exactly k toppings, then the binomial coefficient expresses how many different types of such k-topping pizzas are possible.

Definition [edit]

Given a non-negative integer n and an integer k, the binomial coefficient is defined to be the natural number

$$\binom{n}{k} = \frac{n\cdot(n-1)\cdots(n-k+1)}{k\cdot(k-1)\cdots 1} = \frac{n!}{k!(n-k)!} \quad \text{if } n \geq k \geq 0$$

and

$$\binom{n}{k} = 0 \quad \text{if } k < 0 \text{ or } k > n$$

where $n!$ denotes the factorial of n.

Definition in Coq (edit formalization)

```
Definition C (n p:nat) : R :=
  (fact n) / ((fact p) * (fact (n - p))).
```

Definition in Mizar (edit formalization)

```
definition
 let k,n be natural number;
 func n choose k means
:: NEWTON:def 3
    for l be natural number st l = n-k holds
     it = (n!)/((k!) * (l!)) if n >= k
  otherwise it = 0;
end;
```

In Isabelle: create formalization

Properties of binomial coefficients [edit]

$$\binom{n}{k} = \binom{n}{n-k}, \quad \text{(edit semantic formula in OMDoc)}$$

This follows immediately from the definition or can be seen from expansion (2) by using $(x+y)^n = (y+x)^n$, and is reflected in the numerical "symmetry" of Pascal's triangle.

In Coq (edit formalization)

```
Lemma pascal_step1 : forall n i:nat, (i <= n)%nat -> C n i = C n (n - i).
```

Outline

Equational reasoning in Coq

The *standard* equality in Coq.

- Equality is defined inductively as

```
Inductive eq (A:Type) (x:A) : A -> Prop :=
    refl_equal : eq A x x.
```

- Equality states the identity of two objects of the same type
- Equality allows replacement in any well typed context:

```
eq_ind : forall (A:Type) (x:A) (P:A -> Prop),
    P x -> forall y : A, x = y -> P y
```

- The following are equivalent:
  1. There exist a closed term `t:eq B u v`
  2. $u =_\beta v$ (`u` and `v` compute into the same value)

Limit #1: intensional vs extensional

A frequent problem in system verification : execution traces.

- infinite traces datatype:

```
CoInductive trace (A:Type) : Type :=
    Cons : A -> trace A -> trace A.
```

- If we define two similar traces:

```
CoFixpoint a := Cons nat 42 a.
CoFixpoint b := Cons nat 42 (Cons nat 42 b).
```

- We can prove that `a=a` and `b=b`
- But we cannot prove that `a=b` !
- `a` and `b` are observationally (extensionally) the same, but not intensionally (as fixpoint definitions).

We need to use an equivalence relation.

Limit #1: second attempt

What if `trace A` is defined as `nat -> A` ?

- Suppose we have a primality test

  `is_prime : nat -> bool`

- If we define two similar traces:

```
Definition a (n:nat) := 42.
Definition b (n:nat) :=
    if is_prime n then 42 else 42.
```

- Again we can prove that `a=a` and `b=b`
- But again we cannot prove that `a=b` !
- Same problem with probability distributions

We need to use an equivalence relation.

Limit #2: inconsistent axioms

How would you represent integer polynomials ?

- Easy :

```
Inductive poly :=
    Null : poly | mXp : poly -> nat -> poly.
```

- Now we want to identify identical polynomials:

```
Axiom Null_Null : mXp Null 0 = Null.
```

- Now we can prove that Null_Null is inconsistent !

We need to use an equivalence relation.

What is a setoid ?

A setoid is defined as :

- A carrier type A
- An equivalence relation $\approx_A : A \to A \to Prop$ i.e.
  - reflexive : $\forall a : A, a \approx_A a$
  - symmetric : $\forall a, b : A, a \approx_A b \to b \approx_A a$
  - transitive : $\forall a, b, c : A, a \approx_A b \to b \approx_A c \to a \approx_A c$

Examples:

- `Prop` quotiented by `<->`
- `poly` quotiented by `mXp Null 0` $\approx$ `Null`
- `A -> B` quotiented by extensional equivalence

One setoid leads to another

A setoid morphism is defined as :

- A function $f : A \to B$
- An proof of $\forall a_1, a_2 : A, a_1 \approx_A a_2 \to f(a_1) \approx_B f(a_2)$

Morphisms turn equivalent input into equivalent output.
Examples:

- The function that chops leading zeros off polynomials
- The `tail` function on traces (both definitions)
- A predicate `P:A -> Prop` is a morphism from $=_A$ to `<->`
- The composition of morphisms is a morphism

From total to partial setoids

An natural definition for $\approx_{A \to B}$ is:
$f \approx_{A \to B} g \iff \forall a_1, a_2 : A, a_1 \approx_A a_2 \to f(a_1) \approx_B g(a_2)$

- Good news: f is a morphism if, and only if $f \approx_{A \to B} f$
- Bad news: some functions are not morphisms
  - $\approx_{A \to B}$ is not reflexive
  - $A \to B / \approx_{A \to B}$ is not a setoid

Solution: drop the reflexivity conditions and work with partial equivalence relations and partial setoids

Partial setoids

A partial equivalence relation is:

- symmetric : $\forall a, b : A, a \approx_A b \to b \approx_A a$
- transitive : $\forall a, b, c : A, a \approx_A b \to b \approx_A c \to a \approx_A c$
- not reflexive in general

Theorem

If $A/\approx_A$ and $B/\approx_B$ are partial setoids, then $A \to B/\approx_{A\to B}$ is too.

Partial setoids are the correct notion:

$$\frac{f \approx_{A\to B} g \quad x \approx_A y}{f(x) \approx_B g(y)} \ \textsc{Congr}$$

The congruence-closure algorithm

Satisfiability of finite sets of equalities and inequalities [Downey,Sethi,Tarjan,1980]

- Uses Union-Find structures for equivalence classes of terms
- Merges classes containing equivalent terms
- Tries to build a model of the given constraints
- Supports only one total equivalence relation

Implemented in `congruence` tactic.

Congruence-closure for Partial setoids

All relations are by definition stable w.r.t. equality :

$$\frac{x = y \quad y \approx_A z}{x \approx_A z}\ \textsc{Stable-L} \qquad \frac{x \approx_A y \quad y = z}{x \approx_A z}\ \textsc{Stable-R}$$

Idea: Equivalence classes of terms for setoid relations implemented as classes of equality classes

- Mark individual equality classes as reflexive:

$$\frac{x \approx_A x \quad x = y}{y \approx_A y}\ \textsc{Stable}$$

Beyond ground equations

Use congruence closure in an iterative semi-decision

1. Propagate all constraints
2. Check for contradiction
3. Generate instances for quantified hypotheses
4. Go back to step 1

Instances generation: an efficient E-matching algorithm
Work in the *Prop*/ $\iff$ setoid to mix in some propositional reasoning.

Further work

- Prove completeness of the method
- Implement the procedure
- Find a satisfactory strategy for instances
- Study propositional extensions
- Study reflexion rule

Use it on actual proofs.

Thank you for your attention