Theorem and proof transfer for a universal collaborative repository of formal proofs

Théo Zimmermann

πr^2 project-team, Inria
&
IRIF, Paris Diderot University

Logipedia meeting
Tuesday, 22 January 2019

Concepts have multiple definitions (and multiple names)

- Equivalent characterizations
- Different foundations
- Varied levels of efficiency
- Invented by different people

Why try to select a canonical one?

Theorems have multiple statements (and zero, one, or several names)

- Order of hypotheses
- Equivalent formulation of hypotheses
- Associativity, commutativity, symmetry...
- Specializations, generalizations

Why try to eliminate duplicates?

Duplication is unavoidable (and sometimes good), so we need instead:

- Good search
- Good automation
- Concept alignment

Finding concept alignments

- Automation should automatically find potential alignements.
- Try to prove transfer properties.
- Ask humans to prove other transfer properties (or rate the likelihood of the conjecture).

Using concept alignments

- To transfer theorems: the proof still depends on the transfered theorem, and its dependencies.
- To transfer proofs: more difficult but can avoid complex dependencies.

Using concept alignments

- To transfer theorems: the proof still depends on the transfered theorem, and its dependencies.
- To transfer proofs: more difficult but can avoid complex dependencies.

Sometimes a theorem can be transfered but not its proof. For instance, if the statement contains two related operators but the proof needs to unfold them and they do not unfold in the same way.

$(\mathbb{N}, |)$ is isomorphic to multisets of prime numbers equipped with multiset inclusion.

$n|m \triangleq \exists p, p \cdot n = m$ while $n \sqsubseteq m \triangleq \forall x \in n, x \in m$

How to transfer

[1999] **C. Dubois, M. Jaume**, *Reuse of formal developments: some experiments within Coq*

[2000] **N. Magaud, Y. Bertot**, *Changing Data Structures in Type Theory: A Study of Natural Numbers*

[2001] **G. Barthe, O. Pons**, *Type Isomorphisms and Proof Reuse in Dependent Type Theory*

[2004] **E.B. Johnsen, C. Lüth**, *Theorem Reuse by Proof Term Transformation*

[2012] **B. Huffman, O. Kunčar**, *Lifting and Transfer: A Modular Design for Quotients in Isabelle/HOL*

[2013] **P. Lammich**, *Automatic Data Refinement*

[2013] **C. Cohen, M. Dénes, A. Mörtberg**, *Refinements for free!*

[2015] **T. Zimmermann, H. Herbelin**, *Automatic and Transparent Transfer of Theorems along Isomorphisms in the Coq Proof Assistant*

[2017] **R. Cauderlier, C. Dubois**, *Focalize and dedukti to the rescue for proof interoperability*

[2017] **N. Tabareau, É. Tanter, S. Matthieu**, *Equivalences for Free!*

[2018] **M.M. Moscato, C.G.L. Pombo, C.A. Munoz, M.A. Feliú**, *Boosting the Reuse of Formal Specifications*

Sorry to those I forgot...

Transfer 101

How to relate the first line to the second?

```
∀ x : A, A.eq (A.add x A.zero) x
```

```
∀ x : B, B.eq (B.add x B.zero) x
```

Transfer 101

How to relate the first line to the second?

```
∀ x : A, A.eq (A.add x A.zero) x
```

```
∀ x : B, B.eq (B.add x B.zero) x
```

First transformation:

```
A.all (λ x. A.eq (A.add x A.zero) x)
→
B.all (λ x. B.eq (B.add x B.zero) x)
```

Transfer 101

(from Huffman and Kunčar)

$$\frac{\Gamma \vdash (A \Rightarrow\!\!\!\!\mapsto B)\ f\ g \quad \Gamma \vdash A\ x\ y}{\Gamma \vdash B\ (f\ x)\ (g\ y)}\textsc{App}$$

$$\frac{\Gamma, A\ x\ y \vdash B\ (f\ x)\ (g\ y)}{(\Gamma \vdash A \Rightarrow\!\!\!\!\mapsto B)\ (\lambda x.f\ x)\ (\lambda y.g\ y)}\textsc{Abs}$$

$$\frac{A\ x\ y \in \Gamma}{\Gamma \vdash A\ x\ y}\textsc{Var}$$

Transfer 101

Thanks to:

```
((~ => →) => →) A.all B.all
```

we get down to:

```
x, x', x ~ x' |- A.eq (A.add x A.zero) x
                 → B.eq (B.add x' B.zero) x'
```

Transfer 101

Thanks to:

```
((~ => →) => →) A.all B.all
```

we get down to:

```
x, x', x ~ x' |- A.eq (A.add x A.zero) x
                 → B.eq (B.add x' B.zero) x'
```

Thanks to:

```
(~ => ~ => →) A.eq B.eq
```

we get down to:

```
x, x', x ~ x' |- A.add x A.zero ~ B.add x' B.zero
x, x', x ~ x' |- x ~ x'
```

Transfer 101

Thanks to:

```
(~ => ~ => ~) A.add B.add
```

we get down to:

```
x, x', x ~ x' |- x ~ x'
x, x', x ~ x' |- A.zero ~ B.zero
```