

Invertible Transformations of Types and Their Applications to Security (updated talk from Types 2018) EU Types, Aarhus

Sergei Soloviev, IRIT, France
Jan Malalkhovski, IRIT, France, and ITMO University, Russia

09/10/2018

About this talk

This talk is an updated version of the talk presented at “Types 2018”:

- We continue to work on this topic.
- To our mind, it deserves more discussion, and this workshop is a more appropriate place than “general purpose” Types conference.

Recall: Isos, Autos, Retractions

- As usual, $t : A \rightarrow B, t^{-1} : B \rightarrow A$ are mutually inverse isomorphisms if $t^{-1} \circ t \equiv id_A$ and $t \circ t^{-1} \equiv id_B$.
- E.g., $\lambda z : B_1 \rightarrow (B_2 \rightarrow C). \lambda x_2 : B_2. \lambda x_1 : B_1. (zx_1 x_2)$ is an isomorphism from $B_1 \rightarrow (B_2 \rightarrow C)$ to $B_2 \rightarrow (B_1 \rightarrow C)$
- It works for different systems of Type Theory and λ -calculus.
- An automorphism is an isomorphism $t : A \rightarrow A$.
- Non-trivial automorphism
 $\lambda z : B \rightarrow (B \rightarrow C). \lambda x_2 : B. \lambda x_1 : B. (zx_1 x_2)$
- I.e., $B_1 = B_2$ above.

Isos, Autos, Retractions

- **Remark.** If one-side invertibility is considered, we speak about retractions:
- in $t : A \rightarrow B, t' : B \rightarrow A$ where $t' \circ t = id_A$ the terms t' is a retraction, and t is a coretraction.
- **Example.** (Adding fictive parameters.)

$$t = \lambda y : C. \lambda x : A. x : A \rightarrow (C \rightarrow A)$$

$$z : C \vdash t' = \lambda u : C \rightarrow A. (uz) : (C \rightarrow A) \rightarrow A$$

- Of course it works with more arguments: $A_1 \rightarrow \dots \rightarrow A_k \rightarrow A$ is a retract to $A_1 \rightarrow \dots \rightarrow A_n \rightarrow A$ ($k \leq n$).

Isos, Autos, Retractions

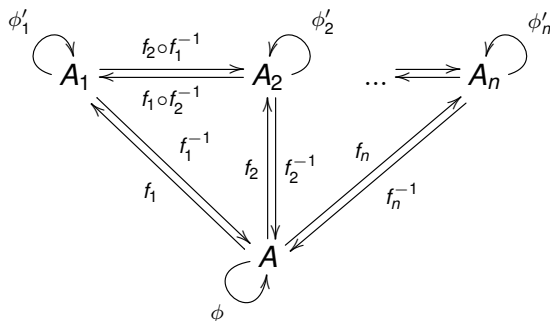
- For each type A the automorphisms $A \rightarrow A$ form the group of automorphisms $Aut(A)$.
- It may be seen as subcategory of the groupoid of isomorphisms .
- i.e., the subcategory of the category of types and deductions - or terms - with the same objects and only isomorphisms as morphisms.
- If we fix a type, we may consider also the groupoid $Gr(A)$ of types isomorphic to A .

Isos, Autos, Retractions

- To distinguish: the isomorphism relation \sim and the isomorphisms as morphisms.
- $A \sim B$ iff $\exists t : A \rightarrow B$. (t is an iso).
- The equivalence class w.r.t \sim of A may contain one element while $Aut(A)$ is non trivial: e.g., for $X \rightarrow (X \rightarrow X)$ (X is a type variable).
- Also, $Aut(A)$ may be trivial ($Aut(A) = \{id\}$) while the \sim -equivalence class is non-trivial: e.g., for $X \rightarrow (Y \rightarrow Z)$.
- **Remark.** To describe retractions “globally” is a much more difficult task.

Isos, Autos, Retractions

In type theories below the number of types isomorphic to A is finite, so the groupoid may be represented by the diagram



where A_1, \dots, A_n are all types that are isomorphic (but not equal) to A (and to each other), f_i, f_i^{-1} denote the fixed isomorphisms and their inverses, ϕ denotes an arbitrary automorphism of A and $\phi'_i = f_i \circ \phi \circ f_i^{-1}$ ($1 \leq i \leq n$).

Isos, Autos, Retractions

- If retractions are involved, the algebraic structures are less “nice” but still (of course) there are some.
- For example, there is the monoid of endomorphisms $A \rightarrow A$. Let $End(A)$ be this monoid.
- If A_0 is a retract of A , $t : A_0 \rightarrow A$, $t' : A \rightarrow A_0$, there is also the semigroup homomorphism $End(A_0) \rightarrow End(A)$ defined by $f \mapsto t \circ f \circ t'$.
- It is not a monoid homomorphism, id is not preserved.

Automorphism Groups

- **Second order λ -calculus without constants.**
- The types are now considered up to renaming of bound variables.
- Simple types are included, and $\bar{\forall}.A$ denotes the universal closure of any type A (quantification over all free variables).

Theorem

For every finite group G there exists a type A_G in simply typed λ -calculus such that the group $\text{Aut}(\bar{\forall}.A_G)$ is isomorphic to G .

- **Remark.** The question about optimal presentation may be considered separately.

Automorphism groups

- In the case with **dependent product** (e.g., Z. Luo's typed logical framework LF) the universal quantification is modelled by dependent product.
- Instead of $\forall X.B(X)$ we write $(X : Type)B(X)$. We can obtain a **similar theorem**: any finite group may be represented in LF .
- The case of simply typed λ -calculus is rather more limited:

Theorem

The groups $Aut(A)$ for simple types A are (up to isomorphism of groups) exactly the groups that may be obtained from symmetric groups by cartesian product and wreath product.

- By an old Jordan's theorem: exactly the groups of automorphisms of finite trees. (Not C_3 for example.)

Applications to Security

- **What may be the use** of λ -terms, isomorphisms and automorphisms in a security-oriented picture?
- **A general idea:** lambda terms may be seen as derived combinators. They may be used to create any program from more elementary building blocks of code.
- To use them “intelligently” to hide (and thus protect) the way how the main program is built from these elementary blocks; if necessary, even the specification may be hidden.
- Isos and autos to encrypt and decrypt (invertibility).

Applications to Security

- Consider some type S . The closed terms $F : S$ represent combinators that may take other terms as arguments.
- Let $f_{1 \div n}$ abbreviate f_1, \dots, f_n .
- If we take $F \equiv \lambda f_{1 \div n} : X \rightarrow X \lambda x : X. (f_{\sigma(1)}(\dots(f_{\sigma(n)}x)\dots))$ (σ a permutation of $\{1, \dots, n\}$), and apply to some concrete $\phi_{1 \div n}$, it will combine them in any desired order. The $\phi_{1 \div n}$ themselves may be even coding functions, as in [?].
- If $F \equiv \lambda f_{1 \div n} : X \rightarrow X \lambda x : X. f_i x$ then one of ϕ will be selected, etc.
- If $\Phi \equiv \lambda G : S. G : S \rightarrow S$, then we may first apply Φ to some operator, like F above, and then “feed” ϕ 's (and x in the end).

Applications to Security

- In this example $\lambda G : S.G$ belongs to $\lambda^1\beta\eta$.
- In $\lambda^2\beta\eta$ we may add a second-order λ and consider $\Theta \equiv \lambda X.\lambda G : S.G$.
- In this way the type X also becomes one of controlled parameters, for example it may be *Nat*, *Bool* or any other type.
- If dependent types are admitted, the type X itself may depend on terms as parameters.
- **Remark.** If we want to use terms (isos, autos) to encrypt, we encrypt **programs**, not texts, and **all remains well-typed**.
- What happens if we add inductive types? For example, *Nat*?

Applications to Security

- The recursion operator from Nat to some type A is of the type $A \rightarrow (Nat \rightarrow A \rightarrow A) \rightarrow Nat \rightarrow A$.
- For example, we may take $A = (B \rightarrow B) \rightarrow (B \rightarrow B)$
- and define the iteration operator on the functional space $B \rightarrow B$:
- $J(n) : f \mapsto f^n$.
- Only $J(1)$ will be an automorphism. It opens a way to “inject” ordinary cryptographic protocols (based on number theory) into Type Theory.
- But more direct way is possible as well.

- Illustration: **ElGamal** cryptosystem (cf. [4, 5]).
- The protocol may use the iterations of a distinguished automorphism $g : A \rightarrow A$, where g^m is $g \circ \dots \circ g$ (m times).
- **Private Key:** $m, m \in N$. **Public Key:** g and g^m .
Encryption. To send a message $a : A$ (in our approach it is not a plain text, but an element of type A , and may have more complex structure) Bob computes g^r and g^{mr} for a random $r \in N$. The ciphertext is $(g^r, g^{mr} a)$.
Decryption. Alice knows m , so if she receives the ciphertext $(g^r, g^{mr} a)$, she computes g^{mr} from g^r , then $(g^{mr})^{-1}$, and then computes a from $g^{mr} a$.

- **Remark.** We do not consider here the cryptosystems like **MOR** based on a more sophisticated group theory [5] but they, too, can be represented in type theory using the results of [6].
- By encoding a finite cyclic group of prime order as a group of automorphism of some type we can implement ElGamal (or any other cryptographic protocol based on finite groups) since the composition and inverse of type automorphisms (represented by finite hereditary permutations [2]) can be computed in linear time.
- However to encode a finite cyclic group is not the only possibility.
- The maximal period of an element in the group $Aut(A)$ may be quite high, and this can be exploited.

ElGamal with Autos

- Let us recall that the longest period in the symmetric group S_n is given by Landau function $\sim n^{\sqrt{n}}$.
- It corresponds to $Aut(a \rightarrow \dots \rightarrow a \rightarrow p)$. This gives an idea of the length of the periods of automorphisms $f \in Aut(A)$.
- Clearly, type-based implementations are going to be less efficient than an equivalent long integer-based ones which would make them less desirable for conventional applications. But that alone can make them more desirable for other uses like **proof-of-work** algorithms.
- Also of note is the fact that the above encryption scheme preserves the structure of $a : A$. Which, for instance, means that Alice needs not typecheck the decrypted a if she trusts Bob to typecheck his.

Other Possibilities

- The type S of the combinator F may have many automorphisms which form a subset of all possible isomorphisms to/from this type.
- Automorphisms do not change the types of parameters (taken in a fixed order). So, if an automorphism θ of S is applied to F , the application $\theta(F)$ to $t_{1 \div n}$ is valid iff the application $Ft_{1 \div n}$ is valid.
- In difference from automorphisms, an action of an isomorphism θ' may make invalid the application $\theta'(F)t_{1 \div n}$.
- This fact may be exploited to detect code transformations performed by third-parties or to execute **zero-knowledge proof** protocols if the distinctions between some type variables remain hidden from external observers.
- **Retractions** may be used to hide relevant parameters and create situations when more automorphisms exist.

Erasures and Finite Hereditary Permutations

- There is a fundamental theorem by Dezani-Ciancaglini that $\beta\eta$ -invertible terms in the untyped λ -calculus are exactly the finite hereditary permutations (f.h.p.):
- and f.h.p. is defined recursively as the term

$$\lambda z. \lambda x_{\sigma(1) \div \sigma(n)}. \text{Ut}_{1 \div n}$$

where $\lambda x_i. t_i$ also are f.h.p.'s.

- Based on this theorem, it is possible to show that $t : A \rightarrow B$ is an isomorphism iff its *erasure* is an f.h.p.
- Similar fact was established by Soloviev in [6] for Luo's LF with dependent product.
- This property has still to be explored, however, in the security context.

Erasures and Finite Hereditary Permutations

- A very interesting point is, however, what is the minimal information to be known needed to restore f with types.
- And what will be the complexity?
- In case of simply typed λ -calculus it is enough to know one of the types A, B and we may restore f types in f with low (linear) complexity.
- The same happens in the second order system.
- **Theorem.** If one of two types A, B is known and it is known, that an f.h.p. is the erasure $e(f)$ of some type isomorphism $f : A \rightarrow B$ then f can be reconstructed.
- **Remark.** The proof goes by structural induction on types and in fact the assumption which type is known has to be alternated.

Erasures and Finite Hereditary Permutations

- In case of dependent types the situation is more complicated (from the complexity point of view).
- This is because partial reconstruction influences the types.
- For example we consider the erasure $\lambda z x' y'. z t_1 t_2$ and the type of z is given: $(x : A)(y : B(x))C$.
- Then the type of t_1 has to be A , by induction we reconstruct the typed form of t'_1 of t_1 before erasure. Only then we can see that the type of t_2 is $B(t'_1(x'))$ where $x' : A'$ (A' isomorphic to A). Then we can proceed.

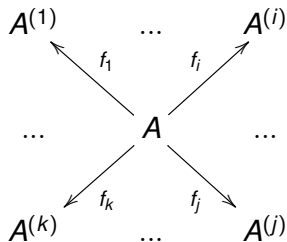
Relative Sizes

- Size of a \sim -equivalence class (recurrent formula).
- Let $A = A_1 \rightarrow \dots \rightarrow A_n \rightarrow p$.
- Some A_i may be isomorphic, some not, let there be “subclasses” (intersections with A_1, \dots, A_n) with k_1, \dots, k_m elements;
 $n = k_1 + \dots + k_m$.
- Let n_i be the full size of the i -th class.
- The size of the \sim -equivalence class of A is

$$\frac{n!}{k_1! \dots k_m!} n_1^{k_1} \dots n_m^{k_m}.$$

Relative Sizes

For each $A^{(i)} \sim A$ we may fix an isomorphism $f_i : A \rightarrow A^{(i)}$



and any other isomorphism $A \rightarrow A^{(i)}$ may be obtained as composition with some automorphism $A \rightarrow A$. As consequence, the number of isomorphisms $A \rightarrow \dots$ is given by







$$|Aut(A)| \cdot \frac{n!}{k_1! \dots k_m!} n_1^{k_1} \dots n_m^{k_m}$$

Conclusion

- Many questions must be solved to make practical the ideas outlined in this paper. The precise communications protocols should be elaborated that would make use of the distinction between iso- and automorphisms, public and private type information.
- The complexity of algorithms (for example, for reconstruction of typed isomorphisms from erasure) needs to be investigated much more precisely.
- Still, we believe that the use of type theory and λ -calculus as a higher-level formal language for data protection (especially software protection) and detection of attacks has good perspectives and must be developed further.

THANKS FOR YOUR ATTENTION!

References

-  Barendregt, H. (1984) *The Lambda Calculus; Its Syntax and Semantics*. North-Holland Plc.
-  Di Cosmo, R. (1995) *Isomorphisms of types: from lambda-calculus to information retrieval and language design*. Birkhauser.
-  Heather, J., Lowe, G., and Schneider, S. (2003) How to prevent type flaw attacks on security protocols. *J. of Computer Security*, 11(2), 217-244.
-  Hoffstein, J., Pipher, J. and Silverman, J.H. (2008) *An introduction to mathematical cryptography*. Springer, New York, 2008.
-  Mahalanobis, A. (2015) The MOR cryptosystem and finite p -groups. *Contemp. Math.*, **633**, 81-95.
-  Soloviev, S. (2018) Automorphisms of Types in Certain Type Theories and Representation of Finite Groups. To appear in *Math. Structures in Computer Science*.