Blank Space: Co-implication and its uses in Algebraic Type Theory

Timothy Hewitt

March 2017

Abstract

Duality is found in many aspects of type theory and programming. If something can be created, it can be destroyed. If something has been acquired, it must be released. Type constructors also come in pairs: Struct and Union, Generic and Interface. Yet Function seems to stand alone, without a corresponding constructor. This paper aims to solve the apparent unknown that is the dual of the function type constructor.

1 Introduction

Co-implication, or the dual of implication, is a relatively unknown and unused connective in logic, and has been completely ignored in type theory. After defining relevant concepts for the following discussion, we shall explore possible definitions and interpretations of co-implication as both a logical connective and a type constructor. Once a workable definition has been reached, we shall observe how it fills in a blank spot in the set of type constructors for algebraic type theory that has previously been ignored.

2 Basic Definitions

There are a few areas of mathematics and computer science that will be discussed, so we will take some time to define the surrounding entities for our eventual derivation of co-implication.

2.1 Inuitionistic Logic

Intuitionistic logic is a counterpoint to the classical logic which is commonly used for mathematical reasoning. It differs by taking a constructive open world viewpoint, instead of the closed world assumed by classical logic. In intuitionistic logic, implication must be given its own axiom instead of being defined in terms of negation and disjunction. In fact, negation is defined in terms of implication and absurdity. The following definitions for the base connectives of intuitionistic logic are from the Stanford Encyclopedia of Philosophy[1]:

- To prove $A \vee B$, we must produce a proof of A or a proof of B.

- To prove $A \wedge B$, we must produce a proof of A and a proof of B.
- To prove $A \rightarrow B$, we must produce an algorithm that transforms any proof of A into a proof of B.

There are also definitions for the quantifiers, but we are only interested in the basic connectives for the topic at hand. One interesting property of this non-classical logic is that the law of excluded middle no longer holds in all cases. This can be interpreted as disavowing the notion that every mathematical problem has a solution, which is not all that extraordinary, since computer science is well aware of undecideable problems.

2.2 Algebraic Type Theory

Algebraic type theory is the system sitting behind proof solvers like Coq, as well as strongly typed functional languages in the ML family. The elements of it exist in most strongly typed languages, though the more mainstream languages often miss one or two critical aspects of the algebra. The definitions we shall use come from Haskell[2]:

- To construct an element of `a | b`, we must construct an element of `a` or an element of `b`.
- To construct an element of `a b`, we must construct an element of `a` and an element of `b`.
- To construct an element of `a -> b`, we must construct a function that consistently maps each of the elements of `a` to an element of `b`.

To use more familiar terms, the first type constructor is a tagged union. It can be instantiated with either of its constituent types, but it also keeps track of which one it contains at runtime. The second is a pair, a struct of two elements. The third is obviously the function type, but expressed differently than C-style type declarations. Also note that the function must be pure (consistently maps) and defined over the entire domain (each of the elements of `a`).

2.3 Curry-Howard Isomorphism

There is a great similarity between the two sets of definitions. This is the crux of the Curry-Howard Isomorphism (CHI): A proof of a proposition in intuitionistic logic can be expressed as a program that constructs an element of the corresponding algebraic type[3]. From the standpoint of algebra, the two systems are isomorphic. This is powerful, because we can frame a question in one system in terms of the other, arrive at an answer, and then translate it back.

This is the extent of definitions we need to start exploring the possibility of the existence of co-implication.

3 Co-Implication in Classical Logic

Before we get into the less familiar territory of intutionistic logic and type theory, it might be helpful to frame the question in terms of the familiar 2-valued boolean algebra we know as classical logic. Thankfully, we already have a definition of what the dual of a connective is, called the DeMorgan Dual. We simply need to negate each operand and negate the overall expression. Applying this to conjunction produces disjunction:

1. $\neg(\neg A \wedge \neg B)$
2. $\neg\neg(A \vee B)$
3. $A \vee B$

And vice-versa:

1. $\neg(\neg A \vee \neg B)$
2. $\neg\neg(A \wedge B)$
3. $A \wedge B$

Therefore, classically, conjunction and disjunction are dual connectives. What, then is the dual of implication?

1. $\neg(\neg A \rightarrow \neg B)$
2. $\neg(\neg\neg A \vee \neg B)$
3. $\neg(A \vee \neg B)$
4. $\neg A \wedge \neg\neg B$
5. $\neg A \wedge B$
6. $A \nleftarrow B$

Officially, this connective is called 'converse nonimplication', and it rarely sees use in classical logic, because it is only true when the left hand side is false and the right hand side is true. The positive here is that we are trying to derive something that is indeed a biadic connective, not negation or identity. Unfortunately, the DeMorgan dual is not very informative for the intuitionistic version, since double negation elimination was used. Double negation elimination does not hold in inuitionistic logic, as it would allow the derivation of the law of excluded middle. Nevertheless, let us move forward into the intuitionistic version.

4 Co-Implication in Intuitionistic Logic

For intuitionistic logic, what could this converse nonimplication mean? To deconstruct the term literally, we need a way to show that the right hand side does not imply the left hand side. Implication is an algorithm that takes proofs of one proposition and turns them into proofs of another proposition. The trick here is to recognize that implication is related to universal quantification. For any proof of A we can construct a proof of B. To show that this is impossible, we simply need to provide a counterexample, a proof of A that cannot be turned into a proof of B. By definition, there is no proof of false, so any proof of B will do if A is false. However, we can get more fine-grained than this in intuitionistic logic, because intutionistic logic defines a partial ordering on propositions, with initial element $\bot$ and final element $\top$.

This partial ordering is generally written as $\vdash$, where the 'stronger' proposition is on the left and the 'weaker' proposition is on the right, though they may be equivalent. It also functions as the usual turnstile, assuming the left hand side, we may conclude the right. Some basic axioms expressed with this notation:

$A \wedge B \vdash A$	$A \wedge B \vdash B$	Product Projection
$A \vdash A \vee B$	$B \vdash A \vee B$	Sum Injection
$A \wedge (A \to B) \vdash B$		Modus Ponens
$A \vdash (B \nleftarrow A) \vee B$		Exclusive Implication

Notice how the co-implicaiton fits into the pattern from the other three connectives. (In intuitionistic sequent calculus, this is made even clearer, though we do not have time to digress into that topic.) The top row states that we can conclude either member of a conjunction from that conjunction. The second states that we can conclude a disjunction from either member of the disjunction. The implicative proposition is defined as the weakest proposition for which modus ponens holds, while the co-implictive proposition must therefore be the strongest proposition for which the law of exclusive implication holds (for any A, we can conclude B or that B is not implied by A). This is different than the law of excluded middle, because co-implication of truth and implication of false are two different forms of negation, thus setting $A = \top$ in exclusive implication is not quite the same as the statement $\top \vdash \neg B \vee B$.

4.1 Co-Implication and Negation

What does it mean, to have two different negations in the same logic? This topic has seen much study, and this paper will not pretend to cover any major part of it. Instead we shall spend a bit of time exploring the meaning of co-implication of truth and how it relates to negation as defined by implication of false.

Traditionally, negation in intuitionistic logic is defined as $\neg A = A \to \bot$. If we assume there is a proof of A, we can construct a proof of false. This is negation by contradiction, and is generally held to be the definition of negation for intuitionistic logic. Double negation under these semantics is to say that it is absurd that A is absurd, which is provable if we can conclude false by assuming A is absurd. However, this does not require constructing a

proof of A, and so is not the same as proving A. We know that it does not imply $\bot$, but that does not mean it can be concluded from $\top$.

Co-implication gives us another way to negate propositions: $\sim A = A \nleftarrow \top$. There is a proof of $\top$ that cannot be turned into a proof of A. This is negation by failure, since there is only one proof of $\top$ (usually named `nil`), by definition. If that proof (which contains no information about A) cannot be turned into a proof of A, then we have failed to prove A. Does this mean A is false? In a closed world, yes. However, in the open world of intuitionistic logic, this simply means we have yet to find a way to prove A. This is extremely useful for reasoning about propositions related to hypotheses that we have not yet proven, and then collapsing the results after the hypothesis is proven or disproven. Thus, adding co-implication to an intuitionistic logic provides us with two operational semantics of negation.

4.2 Negation and Identity

Since there is more than one negation, could there also be more than one identity via the two implicative connectives? Let us consider the possibilities.

Identity via implication is $\top \rightarrow A = A$. It is derivable from the conjunctive identity $\top \wedge A = A$. If we allow that it might have non-trivial operational semantics, what could they be? In this case, an implication is an algorithm for manipulating proofs, and must have a proof to manipulate to produce a result proof. If the source proposition is $\top$, there is only one proof, thus this algorithm must produce the same proof. This is a restriction of the term 'algorithm' in the definition of intuitionistic logic, but a good one to make, as it produces deterministic systems. So by proving that a proof of A can be derived from the proof of $\top$, we know that A is true, and we even have a specific proof hanging around waiting to be produced. However, that proof is not actually expressed until the algorithm is applied. Thus, implicative identity can be seen as a form of temporal indirection, calculation waiting to happen.

Identity via co-implication is $\bot \nleftarrow A = A$. It is similarly derivable from the disjunctive identity $\bot \vee A = A$. In this case there are no semantics of pending calculation, simply the existence of the proof of A. What could this correspond to? If we have some proof of A that cannot be turned into a proof of $\bot$, which has no proofs, we simply have a proof of A, somewhere. Thus it would make sense for co-implication's identity to be a form of spatial indirection. This has, in fact, been in use for a very long time by referencing previous results to form new results. This may be within the same proof, or referencing proofs in other papers. To be able to express such indirection as a first-order logical construct may be a quaint idea for a logician, but it has amazing implications in Algebaric Type Theory.

5 Co-implication in Algebraic Type Theory

Now for the application, since we have been dealing in the theory of logics so far. What happens when we pull the idea of co-implication through the Curry-Howard Isomorphism? In general, the type `a >- b` (where `>-` is the co-implicative type constructor) is a sub-type of `b` that is inhabited by all values of `b` that cannot be mapped to `a`. The trick here is that as long as there is some known value of type `a`, any value of any type can be mapped to that

value. Note that the value must be known. There may be some convoluted type that has values but those values are incalculable, or simply unknown at the time.

The negation by failure construction would indicate that a type cannot be instantiated with the current collection of types and values, since instantiating a type is proving the associated proposition. The spatial indirection of the identity construction seems oddly similar to the concept of pointers in C-like languages. The minimal amount of information needed to go get a value in the computational system. For modern computers, that information is the memory address of the value.

By and large, this co-implicative is only useful when the left type is uninhabited (has no values), though composing it may produce interesting behavior when combined with the tagged union constructor, similarly to the function constructor and structs producing curried multivariate functions.

One other place where the co-implicative type constructor may have interesting uses is when dependent types are introduced. These are types that are quantified over the values that may inhabit them. Ranges, grammars, even comparisons can be expressed as sub-types. In this case, the type proving that no pure function can map from one restricted domain to another is very useful. However, this is a topic that remains to be explored.

6 Conclusion

Thus we have seen that co-implication is indeed a useful logical connective when working in intuitionistic logic by giving us a way to express that a proposition is not implied by another, and by extension, negation by failure. We have also seen that this logical connective has a use in algebraic type theory, in its full form, as a way of asserting that there cannot be a function from one type to another, and a derived form that lets us explicitly discuss spatial indirection in the form of pointers.

To the author's knowledge, no algebraically typed language has implemented this type constructor, and thus the experimental stage of discovery waits for such a language to be written. This is a blank space in type theory and intuitionistic logic, waiting to be filled.

7 Future Work

This is a small (but significant) part of a much larger and longer research project into type theory, intuitionistic logic, and the nature of computer science itself. Co-implication completes a set of eight concepts that seem to underpin all of computer science. The next step for this project is to create a logic programming system based on intuitionistic logic, and then expand that system towards directly implementing the Curry-Howard Isomorphism. The final goal of the project is to have the logic programming system create proofs that are real executable programs, and to encode system types as propositions. The end result of this line of research is a computing environment focused on interrogative interactions instead of imperative ones.

References

[1] Douglas Bridges; Eric Palmgren, *Constructive Mathematics*, Stanford Encyclopedia of Philosophy, Latest revision 2013, Accessed March 2017, `https://plato.stanford.edu/entries/mathematics-constructive/`.

[2] Haskell Wiki, *Types*, Latest Revision October 2015, Accessed March 2017, `https://wiki.haskell.org/Type`.

[3] Pierre-Marie Pédrot, *The Curry-Howard Isomorphism (for dummies)*, Latest Revision February 2015, Accessed March 2017, `https://www.rocq.inria.fr/semdoc/Presentations/20150217_PierreMariePedrot.pdf`.