Propositions as Types

Martin Pfeifhofer & Felix Schett

May 25, 2016

Contents

Abstract

This is a scientific work about the roots of Functional Programming. To be more specific it's about Mathematics & Logic on one side and about Programming & Computation on the other side. These two merged together create a new field of study, which has been later become known as Computational Logic.

1 Introduction

This scientific work is about the origins and the development of the term 'Propositions as Types'. It tells a story of two different sciences, which have been discovered to be linked to each other. Mathematicians and Computer Scientists have worked on different projects, coming out that they are actually equivalent. In this scientific work we wrote mainly about the Definition of 'Effective Computability', which has been made by three different mathematicians and was a Milestone in Computational Logic. After that we go shortly into the specific theme of 'Propositions as Types', also known as Howard-Curry-Correspondence.

2 Content

2.1 Getting Started

The notion Propositions as Types [1] [2] [3] has many names and origins. Computer scientists and mathematicians discovered Propositions as Types in the 1930s and since then its usage and content developed. It describes a correspondence between a programming language and a logic. That means that for any given type in the programming language there is a corresponding proposition in a logic and the other way around. But there is more to become aware of. For each program of a type there is a proof of the corresponding logic and the other way around. So we have 'proofs as programs'. For each way to evaluate a program, there is a corresponding way to simplify a proof and the other way around. We get 'evaluation of programs as simplification of proofs'. So Propositions as Types is an isomorphism between Computer Science and Logic. These properties of Propositions as Types were discovered by computer scientists and mathematicians step by step since the 1930s and are now used in many different programming languages and logics.

Examples of Propositions and their corresponding Types by Haskell Curry:

Implication: A $\supset$ B corresponds to A $\rightarrow$ B
Conjuction: A $\wedge$ B corresponds to A $\times$ B
Disjunction: A $\vee$ B corresponds to A + B

2.2 Effective Computability And The Various Definitions

In the early years of the 20th Century there was no definition of 'Effective Computability' or 'effectively calculable', meaning an algorithm that always terminates and always gives the correct answer. David Hilbert, a German mathematician, published a paper called 'Das Entscheidungsproblem'. It's about an algorithm that gets a statement in formal logic and determines if that statement is either true or false. He wanted to solve the Entscheidungsproblem, depending on the fact that logic is complete, stating that every provable statement is true and every true statement is provable. But before this question could be answered, a formal definition of 'algorithm' was needed. This was made by Alonzo Church, who had a great impact on the development of logics and the notion of propositions as types. Church was one of three scientists who made a definition of Effective Computability in the same decade, together with Kurt Gödel and Alan Turing. But they did not work together, everyone made his own definition independently from the others.

2.2.1 Alonzo Church: Lambda Calculus

Alonzo Church (1903-1995) [4] was an US-American mathematician, logician and philosopher. He first studied, and later teached at the Princeton University. Church once said: "Never had any mathematical conversations with anybody, because there was nobody else in my field." [5] He invented the Lambda Calculus in 1936, first for defining notations for logical formulas in a new presentation of logic. Lambda Calculus contains only 3 constructs: variables, function application and function abstraction. Lambda Calculus provided the first brief

notation for functions, and first class functions (functions as arguments as result from other functions).

Lambda Calculus - Alonzo Church (1932) [6]:

$$\begin{aligned} L, M, N \ &::= \ x \\ &| \ (\lambda x . N) \\ &| \ (L \ M) \end{aligned}$$

Church stated that there are problems which are not solvable and he used Lambda Calculus to proof this statement. After a while Church and his students were convinced, that any effectively calculable-function of numbers can be represented by Lambda Calculus. He also demonstrated in 1937 using Lambda Calculus that the Entscheidungsproblem was not solvable.

2.2.2 Kurt Gödel: General Recursion

Kurt Gödel (1906-1978) [7] was an Austrian-American mathematician and is today known as one of the most important logicians of the 20th Century. He made significant contributions to the predicate logic (completeness and decision problem in arithmetic) and on the relations of intuitionist logic both for classical logic and for modal logic. In 1933 Gödel visited Princeton. He was convinced that Churchs Lambda Calculus wasn't effectively calculable. Church offered Gödel to make his own definition and that he would show afterwards, that it was included in lambda-definability.
About a year later Gödel proposed what is today known as 'General Recursive Functions', as definition for effectively calculable. Church and his students soon noticed that both definitions are equivalent. But instead of believing it, Gödel was doubting his own definition.

2.2.3 Alan Turing: Turing Machine

Alan Turing (1912-1954) [8] was a British logician, mathematician and computer scientist. Today he is known as one of the most influential theorists in the early computer science. In 1937 when Turing was still a student at Cambridge, he came up with his own definition of Effective Computability: the Turing Machine. Turing argumented that any computation done by a human with paper and pencil can also be done by a Turing Machine. Later he used his Turing Machine to show that the 'Entscheidungsproblem' is undecidable. Gödel was finally convinced that all of the three definitions of Effective Computability are right, thanks to Turing.

2.2.4 Gerhard Gentzen: Natural Deduction

Gerhard Gentzen (1909-1945) [9] was a German mathematician and logician. In 1935, at the age of 25, Gentzen introduced two formulations of logic: Natural Deduction [10] and Sequent Calculus [11]. He also introduced the symbol of universal quantification: $\forall$. While working on his formulations of logics he concluded out of his observation, that proof rules should come in pairs. As example in Gentzens Natural Deduction these two were introduction (I) and elimination pairs (E). The introduction rule indicates under what circumstances a given formula with a logical connective can be claimed and the elimination rule shows

how to use that logical connective to solve it.

Gentzen stated:"...The introductions represent, as it were, the "definitions" of the symbols concerned, and the eliminations are no more, in the final analysis, than the consequences of these definitions." [1]

Natural Deduction by Gentzen (1935) [6]:

$$\frac{\begin{array}{c}[A]^{x}\\ \vdots \\ B\end{array}}{A \subset B}\ \supset\text{-I}^{x} \qquad \frac{A \subset B \qquad A}{B}\ \supset\text{-E}$$

$$\frac{A \qquad B}{A \,\&\, B}\ \&\text{-I} \qquad \frac{A \,\&\, B}{A}\ \&\text{-E} \qquad \frac{A \,\&\, B}{B}\ \&\text{-E}$$

Natural Deduction was later shown to be equivalent to Lambda Calculus. While working on these logics and proofs he understood something that he later published under the name of 'Subformula Principle'. It shows that every proof can be normalized to one that is made of a combination of concepts that are included in the final result. The reason why this is important, is the fact that it led to consistency. This is also the reason why logicians have interest in the normalization of proofs. Gentzen introduced Sequent Calculus more or less just to prove the Subformula Principle. Coming out in 1965 Prawitz showed how to prove the Principle, by simplifying Natural Deduction proofs.

2.3 Curry-Howard Correspondence

After having a little insight into the development and definitions of 'Effective Computability' we can go on to the specific topic of Propositions as Types. Haskell Curry (1900-1982) was a US-American logician and mathematician. He studied at Harvard and promoted 1930 in Göttingen, where Hilbert was teaching. [12]What Curry discovered, was the fact, that every type of a function A $\rightarrow$ B could be read as a proposition A $\supset$ B. That means that for a provable proposition there is a corresponding type of a given function. Thus we have propositions as types. Motivated by Curry, William Alvin Howard (*1926) a US-American logician and mathematician, extended this statement. He observed the fact, that there is a similar correspondence between Natural Deduction and simply typed Lambda Calculus. So what we get is that simplification of proofs corresponds to evaluation of programs. By extending Lambda Calculus, he also noticed that the correspondence works out for other logical connectives, such as conjunction and disjunction. He also proposed that the predicate quantifiers ($\forall$,$\exists$) correspond to dependent types. With the introduction of independent types every proof in predicate logic can be represented by a Lambda Calculus term. A lot of systems made by computer scientists and mathematicians are based on this concept. Some examples are Automath, Type Theory and PRL.

3 Conclusion

After deeply researching about Propositions as Types and the specific theme of Effective Computability we came to the conclusion that these scientists have made incredible work on the fields of Logic and Computer Science. They set the foundation of the whole field of Functional Programming, including different logics and their combination with programming languages.

4 Bibliography

References

[1] Philip Wadler. Propositions as types, Dec 2015.

[2] Steven Awodey and Andrej Bauer. Propositions as [types], Nov 2003.

[3] Steven Awodey and Andrej Bauer. Propositions as [types], Jun 2001.

[4] Wikipedia. Alonzo church, April 2016.

[5] Alonzo Church. Alonzo church quote, Mai 2016.

[6] Philip Wadler. Propositions as types - presentation, Sep 2015.

[7] Wikipedia. Kurt gödel, Mai 2016.

[8] Wikipedia. Alan turing, Mai 2016.

[9] Wikipedia. Gerhard gentzen, Mai 2016.

[10] Wikipedia. Natural deduction, Mai 2016.

[11] Wikipedia. Sequent calculus, März 2016.

[12] Wikipedia. Howard-curry-isomorphism, Mai 2016.