

The Curry-Howard Isomorphism

Annika Schmidt

Friedrich-Alexander-Universität Erlangen-Nürnberg

22. September 2020

This work gives a short introduction to the Curry-Howard isomorphism. Therefore some basics for natural deduction and the simply-typed λ -calculus will be given. With these foundations the idea of the Curry-Howard isomorphism will be explained and its application in the proof assistant MMT will be shown.

1 Introduction

Just a look at mathematics and physics is enough to realize that scientific fields cannot be strictly separated. However, it is unknown how many concepts they share. Although some have been discovered, others are still waiting to be found. Indeed, this will be very difficult due to the different terminology and notation. But the effort would be worth considering the benefits of combining the insights about a concept in various scientific fields. Then a faster progress would be possible since different scientists do not have to find a solution for a problem which has already been solved in another subject [3, 4].

John Baez and Mike Stay have formulated the idea that the concept of *object* and *morphism* from category theory exists in physics, topology, logic and computation too. These concepts could just have different names like *proposition* and *proof* in logic. In their paper *Physics, Topology, Logic and Computation: A Rosetta Stone* [3] they show some analogies between these scientific fields. Certainly they are right in at least some points because a correspondence between logic and computation has already been shown in 1969[2, 3].

This correspondence was first discovered by Haskell Curry who detected a connection between positive implicational propositional logic and basic combinators. Later William Howard expanded this idea and showed a correspondence between natural deduction (logic) and simply-typed λ -calculus (computation). Therefore it was named *Curry-Howard isomorphism* and is often explained as *programs are proofs and proofs are programs*. The Curry-Howard isomorphism has still impact on today's research and was even expanded to categorical logic by Joachim Lambek. Also, it is often used in proof assistants like *Coq*, *Haskell* or *MMT* [2, 7–9].

The following sections focus on the Curry-Howard isomorphism. At first some fundamentals about logical reasoning with natural deduction and the simply-typed λ -calculus will be provided. Then the idea of Curry's and Howard's observation will be shortly explained. Finally, an application of the Curry-Howard isomorphism in the proof assistant *MMT* will be demonstrated.

2 Natural deduction and sequent calculus

Natural deduction is a system for formulating logic which was developed by Gerhard Gentzen in 1935. Gentzen's intention was to create a logical system that is similar to human reasoning. Therefore is natural deduction mostly based on assumptions instead of axioms which Gentzen mainly needed for classical logic [6, 9].

A proof in the natural deduction calculus is written in a tree-like structure where the root represents the formula which shall be proved. The leaves portray the assumptions on which the proof shall be based and each node of the tree represents a step of the proof. Between the nodes is a line that indicates which formulas of the proof are used to infer the formula below the line. Next to the line stands the name of the applied rule. The resulting formula of a rule is called the *conclusion* and the zero or more formulas above the line are called *premises*. When all premises are valid the rule can be applied to obtain the conclusion [6, 9].

For each connective in natural deduction there are *introduction* and *elimination* rules. An introduction rule has a formula with the desired logical symbol in its conclusion; an elimination rule takes a premise with the connective and results in a formula without it. In many cases only one introduction rule and one elimination rule is needed for a connective [6, 9].

$$\begin{array}{c}
 \begin{array}{c} [\varphi] \\ \vdots \\ \psi \\ \hline \varphi \rightarrow \psi \end{array} \quad \rightarrow_I \quad \begin{array}{c} \varphi \rightarrow \psi \quad \varphi \\ \hline \psi \end{array} \quad \rightarrow_E \\
 \wedge_I \frac{\varphi \quad \psi}{\varphi \wedge \psi} \quad \wedge_{E1} \frac{\varphi \wedge \psi}{\varphi} \quad \wedge_{E2} \frac{\varphi \wedge \psi}{\psi}
 \end{array}$$

Figure 1: Implication and conjunction rules in natural deduction calculus [9]

An example for the rules can be seen in Figure 1. The characters *I* and *E* stand for *introduction* and *elimination*. The first two rules are for implication (denoted as $\varphi \rightarrow \psi$ where φ and ψ are propositional variables). For \rightarrow_I an assumption has to be made which is marked by the brackets. If ψ can be derived from the assumption φ the implication

$\varphi \rightarrow \psi$ can be inferred and the assumption gets discharged. In \rightarrow_E the premises $\varphi \rightarrow \psi$ and φ are given from which the conclusion ψ can be formed [9].

The remaining inference rules define the conjunction between propositional variables. For \wedge_I the premises φ and ψ must hold to infer $\varphi \wedge \psi$. Both elimination rules \wedge_{E1} and \wedge_{E2} do not differ much. If $\varphi \wedge \psi$ is know then φ (respectively ψ) must hold too [9].

$$\begin{array}{c}
\wedge_{E2} \frac{[\psi \wedge \varphi]}{\varphi} \quad \frac{[\psi \wedge \varphi]}{\psi} \wedge_{E1} \\
\wedge_I \frac{\varphi \quad \psi}{\varphi \wedge \psi} \\
\rightarrow_I \frac{\varphi \wedge \psi}{(\psi \wedge \varphi) \rightarrow (\varphi \wedge \psi)} \quad \frac{\psi \quad \varphi}{\psi \wedge \varphi} \wedge_I \\
\rightarrow_E \frac{(\psi \wedge \varphi) \rightarrow (\varphi \wedge \psi) \quad \psi \wedge \varphi}{\varphi \wedge \psi}
\end{array}$$

Figure 2: Proof (as in resource [9])

With these rules the proof in Figure 2 can be build. When all assumptions are discharged a deduction is called a proof. At the start the assumption $\psi \wedge \varphi$ is used twice to infer φ and ψ . With these premises $\varphi \wedge \psi$ can be concluded using \wedge_I . Then the implication $(\psi \wedge \varphi) \rightarrow (\varphi \wedge \psi)$ is build and the assumption gets discharged. In the other subproof $\psi \wedge \varphi$ is inferred using the two variables ψ and φ . This result is used in \rightarrow_E together with $(\psi \wedge \varphi) \rightarrow (\varphi \wedge \psi)$ to conclude $\varphi \wedge \psi$ [1, 9].

$$\begin{array}{c}
\wedge_I \frac{\psi \quad \varphi}{\psi \wedge \varphi} \quad \frac{\psi \quad \varphi}{\psi \wedge \varphi} \wedge_I \\
\wedge_{E2} \frac{\psi \wedge \varphi}{\varphi} \quad \frac{\psi \wedge \varphi}{\psi} \wedge_{E1} \\
\wedge_I \frac{\varphi \quad \psi}{\varphi \wedge \psi}
\end{array}
\Rightarrow
\frac{\varphi \quad \psi}{\varphi \wedge \psi} \wedge_I$$

Figure 3: Simplified Proofs (as in resource [9])

However, this proof is way more complicated than it should be. To get rid of unnecessary steps the proof can be normalized without changing its validity. Mostly an unnecessary step consists of the introduction rule of the connective directly followed by its elimination rule. In Figure 2 is \rightarrow_I immediately followed by \rightarrow_E . Also the assumption at the beginning can be cut out by replacing it with the subproof on the right. The resulting proof is still not normalized because it contains \wedge_I which is directly followed by one of its two elimination rules. After removing these remaining unnecessary steps the proof only consists of the premises φ and ψ with the conclusion $\varphi \wedge \psi$. The intermediate step and the proof in normal form are represented in Figure 3. The *normalization rules* are displayed in Figure 4. There the proof on the left contains unnecessary steps which are cut out in the proof on the right [2, 9].

Additionally to the natural deduction calculus Gentzen published the sequent calculus in the same year. Both calculi are quite similiar to each other since they share the same rules

$$\begin{array}{ccc}
\begin{array}{c} [\varphi] \\ \vdots \\ \psi \\ \hline \rightarrow_I \frac{\psi}{\varphi \rightarrow \psi} \\ \hline \rightarrow_E \frac{\varphi \rightarrow \psi \quad \varphi}{B} \end{array} & \Rightarrow & \begin{array}{c} \vdots \\ \varphi \\ \vdots \\ \psi \end{array} \\
\\
\begin{array}{c} \vdots \quad \vdots \\ \varphi \quad \psi \\ \hline \wedge_I \frac{\varphi \quad \psi}{\varphi \wedge \psi} \\ \hline \wedge_{E1} \frac{\varphi \wedge \psi}{\varphi} \end{array} & \Rightarrow & \begin{array}{c} \vdots \\ \varphi \end{array}
\end{array}$$

Figure 4: Normalization rules [9]

and therefore the same idea. The main difference is that in sequent calculus the context Γ with all undischarged assumptions is written down in each node. The expression $\Gamma \vdash \varphi$ is called a *sequent* where φ is a proposition. The implication and conjunction rules in sequent calculus are represented in Figure 5 [1, 9].

$$\begin{array}{ccc}
\rightarrow_I \frac{\Gamma, \varphi \vdash \psi}{\Gamma \vdash \varphi \rightarrow \psi} & \rightarrow_E \frac{\Gamma \vdash \varphi \rightarrow \psi \quad \Gamma \vdash \varphi}{\Gamma \vdash \psi} \\
\wedge_I \frac{\Gamma \vdash \varphi \quad \Gamma \vdash \psi}{\Gamma \vdash \varphi \wedge \psi} & \wedge_{E1} \frac{\Gamma \vdash \varphi \wedge \psi}{\Gamma \vdash \varphi} & \wedge_{E2} \frac{\Gamma \vdash \varphi \wedge \psi}{\Gamma \vdash \psi}
\end{array}$$

Figure 5: Implication and conjunction rules in sequent calculus [1, 2]

3 Simply-typed λ -calculus

The λ -calculus is a logical calculus which was invented by Alonzo Church in 1935. Church claimed that every function which could be represented in the λ -calculus is effectively calculable. This was one of the first definitions for the expression *effectively calculable*. However, at the same time other definitions for the term were published by Kurt Gödel and Alan Turing. Later it was proven that all three definitions are equivalent [9].

An expression in λ -calculus consists of variables x, y, \dots which represent terms. With the variables λ -abstractions of the form $\lambda x.y$ can be build. These represent functions that take some input x and return y . Each occurrence of x within the term y is *bound* by λx . Functions can be applied to terms which is written as fx where f is a function and x the term. Also a pair $\langle x, y \rangle$ can be constructed. Then the projections $\pi_1 \langle x, y \rangle$ and $\pi_2 \langle x, y \rangle$ can be formed which select the first (respectively the second) component of the pair. Expression (1) (as in resource [9]) shows an example for a λ -expression where a function which swaps the components of its input pair z gets applied to the pair $\langle y, x \rangle$ [1, 2, 9].

$$(\lambda z. < \pi_2 z, \pi_1 z >) < y, x > \quad (1)$$

Since this seems quite confusing there are *normalization rules* to simplify the term in λ -calculus. The β -reduction can be used when a function of the form $\lambda x.t$ (x is a variable, t is a term) gets applied to some term s . This expression can be rewritten as $t[s/x]$ which means that every free occurrence of x in the term t gets replaced with the term s . $[s/x]$ is called a *substitution*. The other rule can be applied when π_1 or π_2 are used for a pair in the form $< x, y >$. Then only the first (respectively the second) component of the pair is kept in the expression and the rest is omitted. β -reduction is represented in (2) and the π -rules are shown in (3) and (4) [1, 2, 9].

$$(\lambda x.t)s \rightarrow_\beta t[s/x] \quad (2)$$

$$\pi_1 < x, y > \Rightarrow x \quad (3)$$

$$\pi_2 < x, y > \Rightarrow y \quad (4)$$

These reduction rules can now be used to normalize the example (1) (as in resource [9]). First β -reduction is applied to substitute z with $< y, x >$ which is demonstrated in (5). Then $\pi_2 < y, x >$ can be replaced with x and respectively $\pi_1 < y, x >$ with y as in expression (6). The pair $< x, y >$ is exactly the result the function application from (1) would have yielded [1, 2, 9].

$$(\lambda z. < \pi_2 z, \pi_1 z >) < y, x > \rightarrow_\beta < \pi_2 < y, x >, \pi_1 < y, x > > \quad (5)$$

$$< \pi_2 < y, x >, \pi_1 < y, x > > \Rightarrow < x, y > \quad (6)$$

Unfortunately it was discovered that λ -calculus is inconsistent because it allows a predicate to be applied to itself. This creates a non-terminating computation as shown in (7). Therefore λ -calculus is not suited to represent logical formulas. Church fixed this issue by adding types and thus creating the *simply-typed λ -calculus* [9].

$$(\lambda x.xx)(\lambda x.xx) \rightarrow_\beta (\lambda x.xx)(\lambda x.xx) \quad (7)$$

In simply-typed λ -calculus each term x, y, \dots gets assigned some type φ, ψ, \dots which is denoted as $x : \varphi$. The type of a λ -abstraction $\lambda x.y$ is the *function type* $\varphi \rightarrow \psi$ where φ denotes the type of the input x and ψ the type of the output y . When a function gets applied to a term which has the same type as the input type of that function the whole expression corresponds to the output type of the function. A pair $< x, y >$ has the *product type* $\varphi \wedge \psi$. Then the projection π_1 would lead to a term with type φ , respectively π_2 would create a term with type ψ . An example can be seen in (8) which corresponds to the untyped example in (1). There the λ -abstraction has the type $(\psi \wedge \varphi) \rightarrow (\varphi \wedge \psi)$. Since this function gets applied to the pair $< y, x >$ of the type $\psi \wedge \varphi$ everything except

the output type $\varphi \wedge \psi$ can be omitted. With this example it should be noticed that normalizing a term does not change its type [1, 9].

$$(\lambda z. < \pi_2 z, \pi_1 z >) < y, x > : \varphi \wedge \psi \quad (8)$$

4 Curry-Howard isomorphism

In 1958 Haskell Curry examined λ -calculus and positive implicational propositional logic. The latter consists of propositional variables φ, ψ, \dots and the implication rules for sequent calculus defined in Figure 5 as well as the adapted normalization rule from Figure 4. Curry noticed a correspondence between terms and axioms like the one in (9) [2, 9].

$$\Gamma \vdash \varphi \rightarrow (\psi \rightarrow \varphi) \Leftrightarrow \lambda x. \lambda y. x \quad (9)$$

William Howard was inspired by Curry's observation and had some conversations with William Tait about it. Tait discovered a correspondence between the normalization rules in natural deduction and the reduction rules in simply-typed λ -calculus. Howard combined these two observations and added some ideas. He wrote notes which were privately shared in 1969 and are the reason for the name *Curry-Howard isomorphism*. In the notes Howard showed the correspondence between propositions and types. Also, he extended the positive implicational propositional logic with the connectives *not* (\neg), *and* (\wedge) and *or* (\vee) [2, 9].

An example for the correspondence is displayed in Figure 6. Each component of the proof has a corresponding part in the λ -term. The assumption $[\psi \wedge \varphi]$ of the proof (marked red) equals the bound variable z in the λ -term. Obviously the blue resulting formula of the proof is identical to the term's type $\varphi \wedge \psi$. This observation is easier due to the chosen notation. The violet subproof corresponds to the pair $< y, x >$. Additionally each inference rule can be seen as a part of the term. The projections π_1 and π_2 in cyan equal the \wedge -elimination rules. The construction of a pair is indicated by \wedge -introduction. An λ -abstraction (gray) corresponds with \rightarrow_I . In addition, \rightarrow_E equals a function application [9].

$$\begin{array}{c} \wedge_{E_2} \frac{[\psi \wedge \varphi]}{\varphi} \quad \frac{[\psi \wedge \varphi]}{\psi} \wedge_{E_1} \\ \wedge_I \frac{\varphi \wedge \psi}{\varphi \wedge \psi} \\ \rightarrow_I \frac{\varphi \wedge \psi}{(\psi \wedge \varphi) \rightarrow (\varphi \wedge \psi)} \\ \rightarrow_E \frac{(\psi \wedge \varphi) \rightarrow (\varphi \wedge \psi)}{\varphi \wedge \psi} \end{array} \quad \frac{\psi \quad \varphi}{\psi \wedge \varphi} \wedge_I \Leftrightarrow (\lambda z. < \pi_2 z, \pi_1 z >) < y, x > : \varphi \wedge \psi$$

Figure 6: Correspondence between propositional logic and simply-typed λ -calculus [9]

Because of the close correspondence between a formula and the type of a λ -term the Curry-Howard isomorphism is often referred to as *propositions as types*. However, there

$$\begin{array}{c}
\frac{\wedge_I \frac{\psi}{\varphi} \quad \varphi}{\wedge_I \frac{\psi \wedge \varphi}{\varphi}} \quad \frac{\psi \quad \varphi}{\psi \wedge \varphi} \wedge_I \\
\frac{\wedge_{E_2} \frac{\psi \wedge \varphi}{\varphi} \quad \psi}{\wedge_I \frac{\varphi \wedge \psi}{\varphi \wedge \psi}} \quad \frac{\psi \wedge \varphi}{\psi} \wedge_{E_1} \quad \Leftrightarrow \quad \langle \pi_2 \langle y, x \rangle, \pi_1 \langle y, x \rangle \rangle : \varphi \wedge \psi
\end{array}$$

Figure 7: Partially normalized proof and corresponding λ -term [9]

$$\wedge_I \frac{\varphi \quad \psi}{\varphi \wedge \psi} \Leftrightarrow \langle x, y \rangle : \varphi \wedge \psi$$

Figure 8: Fully normalized proof and corresponding λ -term [9]

are some more observations. Whenever some propositions can be proved there exists a λ -term with this type and the other way around. Therefore provability corresponds to the inhabitation of a type. Also, it was shown that the normalization of a proof corresponds to the evaluation of a function with the reduction rules. Therefore the application of normalization rules will not influence the correspondence. Figures 7 and 8 show an example for this phenomenon by normalizing the example given in Figure 6 [2, 9].

There exists an easier notation for these correspondences. Instead of having a separate proof tree and a function both are combined in a single proof tree in sequent calculus which contains a term in simply-typed λ -calculus at each node. There are also typed λ -terms in the context Γ . \rightarrow_I introduces a λ -abstraction whereas \rightarrow_E is equivalent to a function application. \wedge_I constructs a pair which components can be obtained by using \wedge_{E_1} or \wedge_{E_2} to apply a projection on it. The rules to denote the Curry-Howard isomorphism can be seen in Figure 9 and the long example from Figure 6 is displayed in Figure 10 [1, 9].

$$\begin{array}{ccc}
\rightarrow_I \frac{\Gamma, x : \varphi \vdash y : \psi}{\Gamma \vdash \lambda x. y : \varphi \rightarrow \psi} & \rightarrow_E \frac{\Gamma \vdash y : \varphi \rightarrow \psi \quad \Gamma \vdash x : \varphi}{\Gamma \vdash yx : \psi} \\
\wedge_I \frac{\Gamma \vdash x : \varphi \quad \Gamma \vdash y : \psi}{\Gamma \vdash \langle x, y \rangle : \varphi \wedge \psi} & \wedge_{E_1} \frac{\Gamma \vdash z : \varphi \wedge \psi}{\Gamma \vdash \pi_1 z : \varphi} \quad \wedge_{E_2} \frac{\Gamma \vdash z : \varphi \wedge \psi}{\Gamma \vdash \pi_2 z : \psi}
\end{array}$$

Figure 9: Modified rules for Curry-Howard isomorphism [1, 9]

This system can be changed to represent another logic and thus another computational system. Whenever a new logical connective is added the corresponding computational system gets a new type and operators for it. Table 1 shows some examples for these correspondences between logical connectives and type constructs. Falsity \perp equals a term with type f . Like before indicates implication \rightarrow the function type and conjunction \wedge a pair. Negation \neg is just an abbreviation for $\rightarrow f$ so it does not introduce a new type. Disjunction \vee represents a disjoint sum [1, 2, 9].

$$\begin{array}{c}
\wedge_{E_2} \frac{\Gamma, z : \psi \wedge \varphi \vdash z : \psi \wedge \varphi}{\Gamma, z : \psi \wedge \varphi \vdash \pi_2 z : \varphi} \quad \frac{\Gamma, z : \psi \wedge \varphi \vdash z : \psi \wedge \varphi}{\Gamma, z : \psi \wedge \varphi \vdash \pi_1 z : \psi} \wedge_{E_1} \\
\wedge_I \frac{\Gamma, z : \psi \wedge \varphi \vdash \pi_2 z : \varphi \quad \Gamma, z : \psi \wedge \varphi \vdash \pi_1 z : \psi}{\Gamma, z : \psi \wedge \varphi \vdash \langle \pi_2 z, \pi_1 z \rangle : \varphi \wedge \psi} \\
\rightarrow_I \frac{\Gamma, z : \psi \wedge \varphi \vdash \langle \pi_2 z, \pi_1 z \rangle : \varphi \wedge \psi}{\Gamma \vdash \lambda z. \langle \pi_2 z, \pi_1 z \rangle : (\psi \wedge \varphi) \rightarrow (\varphi \wedge \psi)} \quad \frac{\Gamma \vdash y : \psi \quad \Gamma \vdash x : \varphi}{\Gamma \vdash \langle y, x \rangle : \psi \wedge \varphi} \\
\rightarrow_E \frac{\Gamma \vdash \lambda z. \langle \pi_2 z, \pi_1 z \rangle : (\psi \wedge \varphi) \rightarrow (\varphi \wedge \psi) \quad \Gamma \vdash \langle y, x \rangle : \psi \wedge \varphi}{\Gamma \vdash (\lambda z. \langle \pi_2 z, \pi_1 z \rangle) \langle y, x \rangle : \varphi \wedge \psi}
\end{array}$$

Figure 10: Proof in modified notation (as in resource [9])

Symbol	Logic	Type
\perp	falsity	f
\rightarrow	implication	function
\neg	negation	$\rightarrow f$
\wedge	conjunction	pair
\vee	disjunction	disjoint sum

Table 1: Overview of correspondences between logical connectives and types [1, 2, 9]

5 Application in the proof assistant MMT

Although it is nice to know the theory of the Curry-Howard isomorphism, without a practical usage it would not be really helpful. Fortunately, it can be used in proof assistants like *Coq*, *Haskell*, *MMT* and many more. In these systems the Curry-Howard isomorphism is utilized to check whether a proof is valid like a type checker verifies the type [7, 9].

In the following only MMT will be considered. For further information about it take a look at its tutorial [5] which gets regularly updated. Also, this example will be limited to propositional logic to make it as easy as possible. It is based on the lecture *Logic-based knowledge representation for mathematic/technical knowledge* given by M. Kohlhase and F. Rabe [5, 7].

At first propositional logic needs to be defined in MMT (Figure 11). Therefore a *theory* with the name *PL* is started. This theory consists of multiple statements of the form *name : type = definiens # notation definition* and a special delimiter. Only the name and the delimiter are necessary and the rest is optional. First of all the type of a proposition needs to be defined which is called *prop* here. Then the connectives for implication, conjunction and equivalence are defined. Each one has the type of a binary function which inputs and output have the type *prop*. Also, a shorter notation is chosen after the *#*. Additionally, equivalence $p \Leftrightarrow q$ is defined as the conjunction of p implies q and q implies p [5, 7].

Now the natural deduction calculus for propositional logic gets defined in theory *PLND*


```

1 namespace http://kwarc.info/annika
2
3 // defines the logical framework that is used
4 fixmeta http://cds.omdoc.org/urtheories?LF
5
6 // propositional logic
7 theory PL =
8   // define the type prop for propositions
9   prop : type
10
11   // implication and conjunction as binary functions
12   impl : prop → prop → prop # 1 ⇒ 2 prec 20
13   and : prop → prop → prop # 1 ∧ 2 prec 20
14
15   // equivalence can be defined with implication and conjunction
16   equiv : prop → prop → prop = [p,q] (p ⇒ q) ∧ (q ⇒ p) # 1 ⇔ 2 prec 20
17
18

```

Figure 11: MMT - propositional logic

(Figure 12). To be able to use the connectives from above *PL* has to be included. Then *valid* needs to be defined as a function which gets a proposition and returns a *type*. This is necessary to define the natural deduction calculus for the connectives because their rules require valid premises. The rules *ImplI* and *ImplE* are for implication introduction and elimination like in Figure 1. The same applies for *AndI*, *AndEl* and *AndEr* regarding conjunction rules. The equivalence introduction has the premises that from assumption *A* *B* can be inferred and the other way around. The equivalence elimination rules are similar to the implication elimination. Depending on whether *A* or *B* is given additionally to $A \Leftrightarrow B$ as a premise *B* or *A* can be concluded [5, 7].

The equivalence rules in natural deduction got proven which is written after the equality sign. The brackets at the start with the characters in it represent a λ -abstraction where the characters label the input. The type of the input gets inferred by the system. For *EquivI* MMT infers that *A* and *B* have type *prop* and *p* has a type which denotes that from a valid proposition *A* a valid proposition *B* can be inferred. Similarly, *q* has a type that indicates that from a valid proposition *B* some valid proposition *A* can be inferred. After the inputs some function applications are done to prove the equivalence. The λ -term of *EquivI* could be rewritten as in (10) [5, 7].

$$\begin{aligned}
& \lambda A. \lambda B. \lambda p. \lambda q. \text{andI } (\text{implI } p) (\text{implI } q) : \\
& A \rightarrow B \rightarrow (\vdash A \rightarrow \vdash B) \rightarrow (\vdash B \rightarrow \vdash A) \rightarrow \vdash A \Leftrightarrow B
\end{aligned} \tag{10}$$

The Curry-Howard isomorphism states that this expression in simply-typed λ -calculus is corresponding to a proof in natural deduction calculus. Therefore this is a valid way to proof the statement. However, it still has to be checked whether this proof is valid.

```

19 // natural deduction calculus for propositional logic
20 theory PLND =
21   include ?PL
22
23   // unary function needed for proofs
24   valid : prop → type1 # ⊢ 1 prec -5
25
26   // implication rules
27   ImplI : {A,B} (⊢A → ⊢B) → ⊢A⇒B # implI 3
28   ImplE : {A,B} ⊢A⇒B → ⊢A → ⊢B # implE 3 4
29
30   // conjunction rules
31   AndI : {A,B} ⊢A → ⊢B → ⊢A∧B # andI 3 4
32   AndEl : {A,B} ⊢A∧B → ⊢A # andEl 3
33   AndEr : {A,B} ⊢A∧B → ⊢B # andEr 3
34
35   // equivalence rules
36   EquivI : {A,B} (⊢A → ⊢B) → (⊢B → ⊢A) → ⊢A⇔B
37   = [A,B,p,q] andI (implI p) (implI q) # equivI 3 4
38   EquivEl : {A,B} ⊢A⇔B → ⊢A → ⊢B
39   = [A,B,p,q] implE (andEl p) q # equivEl 3 4
40   EquivEr : {A,B} ⊢A⇔B → ⊢B → ⊢A
41   = [A,B,p,q] implE (andEr p) q # equivEr 3 4

```

Figure 12: MMT - natural deduction calculus

For this the system only needs to check the types. In the case of *EquivI* the types of the inputs p and q are apart of the exact names of the placeholders for propositions equal to the required input type of *implI*. Therefore the type of the output of *implI* p is a valid implication $\vdash A \Rightarrow B$ and respectively is the output of *implI* q of the type $\vdash B \Rightarrow A$. Since *andI* only requires two valid propositions as an input and both implications are valid propositions there is no type error. Consequently, the function is a valid proof for *EquivI*. The corresponding proof tree is shown in Figure 13 [5, 7].

$$\begin{array}{c}
\begin{array}{cc}
[A] & [B] \\
\vdots & \vdots \\
\rightarrow_I \frac{B}{A \rightarrow B} & \frac{A}{B \rightarrow A} \rightarrow_I \\
\wedge_I \frac{}{(A \rightarrow B) \wedge (B \rightarrow A)}
\end{array}
\end{array}$$

Figure 13: Proof for *EquivI*

However, it should be noticed that although the proof is valid it does not have to be the right proof. A typed λ -term does not define an exact function but a set of functions with the determined type. So even if a function has the right type it might not be the intended function. Since in this example the proof and the definition are quite similar, this proof should be correct. Furthermore, normalization can be imitated in MMT. Figure 14 shows the MMT examples for the proofs from Figures 2 and 3. The shortest example is just the application of *andI* [2, 9].

```

43 // long example!
44 example1 : {A,B} ⊢A → ⊢B → ⊢A∧B!
45   = [A,B,x,y] implE (implI ([z:⊢B∧A] andI (andEr z) (andEl z))) (andI y x)!
46 // intermediate example!
47 example2 : {A,B} ⊢A → ⊢B → ⊢A∧B!
48   = [A,B,x,y] andI (andEr (andI y x)) (andEl (andI y x))!
49 // short example!
50 example3 : {A,B} ⊢A → ⊢B → ⊢A∧B!
51   = [A,B,x,y] andI x y!
52
53

```

Figure 14: MMT - examples

6 Conclusion

With his observation did Curry not only inspire Howard. After the discovery of the Curry-Howard isomorphism many scientists tried to find corresponding systems. Some of those scientists were Per Martin-Löf and Joachim Lambek. Curry and Howard discovered the correspondence between intuitionistic propositional logic in a natural deduction calculus and the simply-typed λ -calculus. Then Howard introduced *dependent types* which corresponded to the first-order logic quantifiers \forall and \exists . Therefore a correspondence between first-order logic and dependently-typed λ -calculus was shown. Also, correspondences for second-order logic, modal logic, categorical logic and many other logical systems were sought and partially found [2, 8, 9].

But not only correspondences between logic and computation were examined. The scientists were eager to create proof assistants like *Coq*, *Haskell* and *MMT* to put the Curry-Howard isomorphism to a practical use. The ability of these systems to validate proofs is helpful. However, some of these systems are even able to make some proofs themselves which is far more impressive. The further improvement of theorem provers will one day lead to new insights in various scientific fields [7, 9].

Although the Curry-Howard isomorphism is a rather theoretic discovery, it is worth the time to be understood. Even if its only benefit is to realize that separate scientific

fields are not that different at all. And maybe one day the assumption that there exists a correspondence between physics, topology, logic, computation and category theory will be proven correct [3].

References

- [1] J. Gallier. On the Correspondence Between Proofs and λ -Terms. 1993. https://repository.upenn.edu/cgi/viewcontent.cgi?article=1433&context=cis_reports.
- [2] W. H. Howard. The formulae-as-types notion of construction. 1969. <https://www.dcc.fc.up.pt/~acm/howard2.pdf>.
- [3] M. Stay J. C. Baez. Physics, Topology, Logic and Computation: A Rosetta Stone. 2009. <https://arxiv.org/pdf/0903.0340.pdf>.
- [4] S. H. Schneider K. C. Land. Forecasting in the social and natural sciences: an overview and analysis of isomorphisms. 1987. <https://link.springer.com/content/pdf/10.1007/BF00138793.pdf>.
- [5] F. Rabe M. Kohlhase, D. Müller. OMDoc/MMT Tutorial for Mathematicians. 2020. <https://gl.mathhub.info/Tutorials/Mathematicians/blob/master/tutorial/mmt-math-tutorial.pdf>.
- [6] F. J. Pelletier. A History of Natural Deduction and Elementary Logic Textbooks. 1999. <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.33.525&rep=rep1&type=pdf>.
- [7] F. Rabe. A Logic-Independent IDE. 2014. <https://arxiv.org/pdf/1410.8219v1.pdf>.
- [8] L. Schoenbaum. A generalization of the Curry-Howard correspondence. 2016. <https://arxiv.org/pdf/1612.02816.pdf>.
- [9] P. Wadler. Propositions as Types. 2014. <http://homepages.inf.ed.ac.uk/wadler/papers/propositions-as-types/propositions-as-types.pdf>.