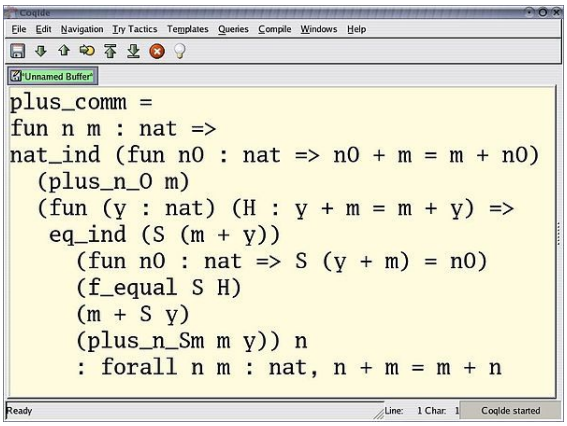
Curry–Howard correspondence



A proof written as a functional program: the proof of commutativity of addition on natural numbers in the proof assistant Coq. nat_ind *stands for mathematical induction,* eq_ind *for substitution of equals and* f_equal *for taking the same function on both sides of the equality. Earlier theorems are referenced showing* m = m + *0 and S (*m + y*)* = m + *S* y.

In programming language theory and proof theory, the **Curry–Howard correspondence** (also known as the **Curry–Howard isomorphism** or **equivalence**, or the **proofs-as-programs** and **propositions-** or **formulae-as-types interpretation**) is the direct relationship between computer programs and mathematical proofs. It is a generalization of a syntactic analogy between systems of formal logic and computational calculi that was first discovered by the American mathematician Haskell Curry and logician William Alvin Howard.[1] It is the link between logic and computation that is usually attributed to Curry and Howard, although the idea is related to the operational interpretation of intuitionistic logic given in various formulations by L. E. J. Brouwer, Arend Heyting and Andrey Kolmogorov (see Brouwer–Heyting–Kolmogorov interpretation)[2] and Stephen Kleene (see Realizability). The relationship has been extended to include category theory as the three-way **Curry–Howard–Lambek correspondence**.

1 Origin, scope, and consequences

The beginnings of the **Curry–Howard correspondence** lie in several observations:

1. In 1934 Curry observes that the types of the combinators could be seen as axiom-schemes for intuitionistic implicational logic[3]
2. In 1958 he observes that a certain kind of proof system, referred to as Hilbert-style deduction systems, coincides on some fragment to the typed fragment of a standard model of computation known as combinatory logic[4]
3. In 1969 Howard observes that another, more "high-level" proof system, referred to as natural deduction, can be directly interpreted in its intuitionistic version as a typed variant of the model of computation known as lambda calculus[5]

In other words, the Curry–Howard correspondence is the observation that two families of seemingly unrelated formalisms—namely, the proof systems on one hand, and the models of computation on the other—are in fact the same kind of mathematical objects.

If one abstracts on the peculiarities of either formalism, the following generalization arises: *a proof is a program, and the formula it proves is the type for the program*. More informally, this can be seen as an analogy that states that the return type of a function (i.e., the type of values returned by a function) is analogous to a logical theorem, subject to hypotheses corresponding to the types of the argument values passed to the function; and that the program to compute that function is analogous to a proof of that theorem. This sets a form of logic programming on a rigorous foundation: *proofs can be represented as programs, and especially as lambda terms*, or *proofs can be* ***run***.

The correspondence has been the starting point of a large spectrum of new research after its discovery, leading in particular to a new class of formal systems designed to act both as a proof system and as a typed functional programming language. This includes Martin-Löf's intuitionistic type theory and Coquand's Calculus of Constructions, two calculi in which proofs are regular objects of the discourse and in which one can state properties of proofs the same way as of any program. This field of research is usually referred to as modern type theory.

Such typed lambda calculi derived from the Curry–Howard paradigm led to software like Coq in which proofs seen as programs can be formalized, checked, and run.

A converse direction is to *use a program to extract a proof*, given its correctness— an area of research closely related to proof-carrying code. This is only feasible if the programming language the program is written for is very richly typed: the development of such type systems has

been partly motivated by the wish to make the Curry–Howard correspondence practically relevant.

The Curry–Howard correspondence also raised new questions regarding the computational content of proof concepts that were not covered by the original works of Curry and Howard. In particular, classical logic has been shown to correspond to the ability to manipulate the continuation of programs and the symmetry of sequent calculus to express the duality between the two evaluation strategies known as call-by-name and call-by-value.

Speculatively, the Curry–Howard correspondence might be expected to lead to a substantial unification between mathematical logic and foundational computer science:

Hilbert-style logic and natural deduction are but two kinds of proof systems among a large family of formalisms. Alternative syntaxes include sequent calculus, proof nets, calculus of structures, etc. If one admits the Curry–Howard correspondence as the general principle that any proof system hides a model of computation, a theory of the underlying untyped computational structure of these kinds of proof system should be possible. Then, a natural question is whether something mathematically interesting can be said about these underlying computational calculi.

Conversely, combinatory logic and simply typed lambda calculus are not the only models of computation, either. Girard's linear logic was developed from the fine analysis of the use of resources in some models of lambda calculus; can we imagine a typed version of Turing's machine that would behave as a proof system? Typed assembly languages are such an instance of "low-level" models of computation that carry types.

Because of the possibility of writing non-terminating programs, Turing-complete models of computation (such as languages with arbitrary recursive functions) must be interpreted with care, as naive application of the correspondence leads to an inconsistent logic. The best way of dealing with arbitrary computation from a logical point of view is still an actively debated research question, but one popular approach is based on using monads to segregate provably terminating from potentially non-terminating code (an approach that also generalizes to much richer models of computation,[6] and is itself related to modal logic by a natural extension of the Curry–Howard isomorphism[ext 1]). A more radical approach, advocated by total functional programming, is to eliminate unrestricted recursion (and forgo Turing completeness, although still retaining high computational complexity), using more controlled corecursion wherever non-terminating behavior is actually desired.

2 General formulation

In its more general formulation, the Curry–Howard correspondence is a correspondence between formal proof calculi and type systems for models of computation. In particular, it splits into two correspondences. One at the level of formulas and types that is independent of which particular proof system or model of computation is considered, and one at the level of proofs and programs which, this time, is specific to the particular choice of proof system and model of computation considered.

At the level of formulas and types, the correspondence says that implication behaves the same as a function type, conjunction as a "product" type (this may be called a tuple, a struct, a list, or some other term depending on the language), disjunction as a sum type (this type may be called a union), the false formula as the empty type and the true formula as the singleton type (whose sole member is the null object). Quantifiers correspond to dependent function space or products (as appropriate). This is summarized in the following table:

At the level of proof systems and models of computations, the correspondence mainly shows the identity of structure, first, between some particular formulations of systems known as Hilbert-style deduction system and combinatory logic, and, secondly, between some particular formulations of systems known as natural deduction and lambda calculus.

Between the natural deduction system and the lambda calculus there are the following correspondences:

3 Corresponding systems

3.1 Hilbert-style deduction systems and combinatory logic

It was at the beginning a simple remark in Curry and Feys's 1958 book on combinatory logic: the simplest types for the basic combinators K and S of combinatory logic surprisingly corresponded to the respective axiom schemes $\alpha \to (\beta \to \alpha)$ and $(\alpha \to (\beta \to \gamma)) \to ((\alpha \to \beta) \to (\alpha \to \gamma))$ used in Hilbert-style deduction systems. For this reason, these schemes are now often called axioms K and S. Examples of programs seen as proofs in a Hilbert-style logic are given below.

If one restricts to the implicational intuitionistic fragment, a simple way to formalize logic in Hilbert's style is as follows. Let Γ be a finite collection of formulas, considered as hypotheses. We say that δ is derivable from Γ, and we write $\Gamma \vdash \delta$, in the following cases:

- δ is an hypothesis, i.e. it is a formula of Γ,
- δ is an instance of an axiom scheme; i.e., under the most common axiom system:
  - δ has the form $\alpha \to (\beta \to \alpha)$, or
  - δ has the form $(\alpha \to (\beta \to \gamma)) \to ((\alpha \to \beta) \to (\alpha \to \gamma))$,

- δ follows by deduction, i.e., for some α, both α → δ and α are already derivable from Γ (this is the rule of modus ponens)

This can be formalized using inference rules, what we do in the left column of the following table.

We can formulate typed combinatory logic using a similar syntax: let Γ be a finite collection of variables, annotated with their types. A term T (also annotated with its type) will depend on these variables [Γ ⊢ T:δ] when:

- T is one of the variables in Γ,
- T is a basic combinator; i.e., under the most common combinator basis:
  - T is K:α → (β → α) [where α and β denote the types of its arguments], or
  - T is S:(α → (β → γ)) → ((α → β) → (α → γ)),
- T is the composition of two subterms which depend on the variables in Γ.

The generation rules defined here are given in the right-column below. Curry's remark simply states that both columns are in one-to-one correspondence. The restriction of the correspondence to intuitionistic logic means that some classical tautologies, such as Peirce's law ((α → β) → α) → α, are excluded from the correspondence.

Seen at a more abstract level, the correspondence can be restated as shown in the following table. Especially, the deduction theorem specific to Hilbert-style logic matches the process of abstraction elimination of combinatory logic.

Thanks to the correspondence, results from combinatory logic can be transferred to Hilbert-style logic and vice versa. For instance, the notion of reduction of terms in combinatory logic can be transferred to Hilbert-style logic and it provides a way to canonically transform proofs into other proofs of the same statement. One can also transfer the notion of normal terms to a notion of normal proofs, expressing that the hypotheses of the axioms never need to be all detached (since otherwise a simplification can happen).

Conversely, the non provability in intuitionistic logic of Peirce's law can be transferred back to combinatory logic: there is no typed term of combinatory logic that is typable with type ((α → β) → α) → α.

Results on the completeness of some sets of combinators or axioms can also be transferred. For instance, the fact that the combinator **X** constitutes a one-point basis of (extensional) combinatory logic implies that the single axiom scheme

$$(((\alpha \to (\beta \to \gamma)) \to ((\alpha \to \beta) \to (\alpha \to \gamma))) \to ((\delta \to (\varepsilon \to \delta)) \to \zeta)) \to \zeta,$$

which is the principal type of **X**, is an adequate replacement to the combination of the axiom schemes

$$\alpha \to (\beta \to \alpha) \text{ and}$$
$$(\alpha \to (\beta \to \gamma)) \to ((\alpha \to \beta) \to (\alpha \to \gamma)).$$

3.2 Natural deduction and lambda calculus

After Curry emphasized the syntactic correspondence between Hilbert-style deduction and combinatory logic, Howard made explicit in 1969 a syntactic analogy between the programs of simply typed lambda calculus and the proofs of natural deduction. Below, the left-hand side formalizes intuitionistic implicational natural deduction as a calculus of sequents (the use of sequents is standard in discussions of the Curry–Howard isomorphism as it allows the deduction rules to be stated more cleanly) with implicit weakening and the right-hand side shows the typing rules of lambda calculus. In the left-hand side, Γ, Γ_1 and Γ_2 denote ordered sequences of formulas while in the right-hand side, they denote sequences of named (i.e., typed) formulas with all names different.

To paraphrase the correspondence, proving Γ ⊢ α means having a program that, given values with the types listed in Γ, manufactures an object of type α. An axiom corresponds to the introduction of a new variable with a new, unconstrained type, the → I rule corresponds to function abstraction and the → E rule corresponds to function application. Observe that the correspondence is not exact if the context Γ is taken to be a set of formulas as, e.g., the λ-terms λx.λy.x and λx.λy.y of type α → α → α would not be distinguished in the correspondence. Examples are given below.

Howard showed that the correspondence extends to other connectives of the logic and other constructions of simply typed lambda calculus. Seen at an abstract level, the correspondence can then be summarized as shown in the following table. Especially, it also shows that the notion of normal forms in lambda calculus matches Prawitz's notion of normal deduction in natural deduction, from what we deduce, among others, that the algorithms for the type inhabitation problem can be turned into algorithms for deciding intuitionistic provability.

Howard's correspondence naturally extends to other extensions of natural deduction and simply typed lambda calculus. Here is a non exhaustive list:

- Girard-Reynolds System F as a common language for both second-order propositional logic and polymorphic lambda calculus,
- higher-order logic and Girard's System Fω
- inductive types as algebraic data type

- necessity □ in modal logic and staged computation[ext 2]
- possibility ◊ in modal logic and monadic types for effects[ext 1]
- The λI calculus corresponds to relevant logic.[7]
- The local truth (∇) modality in Grothendieck topology or the equivalent "lax" modality (∘) of Benton, Bierman, and de Paiva (1998) correspond to CL-logic describing "computation types".[8]

3.3 Classical logic and control operators

At the time of Curry, and also at the time of Howard, the proofs-as-programs correspondence concerned only intuitionistic logic, i.e. a logic in which, in particular, Peirce's law was *not* deducible. The extension of the correspondence to Peirce's law and hence to classical logic became clear from the work of Griffin on typing operators that capture the evaluation context of a given program execution so that this evaluation context can be later on reinstalled. The basic Curry–Howard-style correspondence for classical logic is given below. Note the correspondence between the double-negation translation used to map classical proofs to intuitionistic logic and the continuation-passing-style translation used to map lambda terms involving control to pure lambda terms. More particularly, call-by-name continuation-passing-style translations relates to Kolmogorov's double negation translation and call-by-value continuation-passing-style translations relates to a kind of double-negation translation due to Kuroda.

A finer Curry–Howard correspondence exists for classical logic if one defines classical logic not by adding an axiom such as Peirce's law, but by allowing several conclusions in sequents. In the case of classical natural deduction, there exists a proofs-as-programs correspondence with the typed programs of Parigot's λμ-calculus.

3.4 Sequent calculus

A proofs-as-programs correspondence can be settled for the formalism known as Gentzen's sequent calculus but it is not a correspondence with a well-defined pre-existing model of computation as it was for Hilbert-style and natural deductions.

Sequent calculus is characterized by the presence of left introduction rules, right introduction rule and a cut rule that can be eliminated. The structure of sequent calculus relates to a calculus whose structure is close to the one of some abstract machines. The informal correspondence is as follows:

4 Related proofs-as-programs correspondences

4.1 The role of de Bruijn

N. G. de Bruijn used the lambda notation for representing proofs of the theorem checker Automath, and represented propositions as "categories" of their proofs. It was in the late 1960s at the same period of time Howard wrote his manuscript; de Bruijn was likely unaware of Howard's work, and stated the correspondence independently (Sørensen & Urzyczyn [1998] 2006, pp 98–99). Some researchers tend to use the term Curry–Howard–de Bruijn correspondence in place of Curry–Howard correspondence.

4.2 BHK interpretation

The BHK interpretation interprets intuitionistic proofs as functions but it does not specify the class of functions relevant for the interpretation. If one takes lambda calculus for this class of function, then the BHK interpretation tells the same as Howard's correspondence between natural deduction and lambda calculus.

4.3 Realizability

Kleene's recursive realizability splits proofs of intuitionistic arithmetic into the pair of a recursive function and of a proof of a formula expressing that the recursive function "realizes", i.e. correctly instantiates the disjunctions and existential quantifiers of the initial formula so that the formula gets true.

Kreisel's modified realizability applies to intuitionistic higher-order predicate logic and shows that the simply typed lambda term inductively extracted from the proof realizes the initial formula. In the case of propositional logic, it coincides with Howard's statement: the extracted lambda term is the proof itself (seen as an untyped lambda term) and the realizability statement is a paraphrase of the fact that the extracted lambda term has the type that the formula means (seen as a type).

Gödel's dialectica interpretation realizes (an extension of) intuitionistic arithmetic with computable functions. The connection with lambda calculus is unclear, even in the case of natural deduction.

4.4 Curry–Howard–Lambek correspondence

Joachim Lambek showed in the early 1970s that the proofs of intuitionistic propositional logic and the combinators of typed combinatory logic share a common equational theory which is the one of cartesian closed cat-

egories. The expression Curry–Howard–Lambek correspondence is now used by some people to refer to the three way isomorphism between intuitionistic logic, typed lambda calculus and cartesian closed categories, with objects being interpreted as types or propositions and morphisms as terms or proofs. The correspondence works at the equational level and is not the expression of a syntactic identity of structures as it is the case for each of Curry's and Howard's correspondences: i.e. the structure of a well-defined morphism in a cartesian-closed category is not comparable to the structure of a proof of the corresponding judgment in either Hilbert-style logic or natural deduction. To clarify this distinction, the underlying syntactic structure of cartesian closed categories is rephrased below.

Objects (types) are defined by

- $\top$ is an object
- if α and β are objects then $\alpha \times \beta$ and $\alpha \to \beta$ are objects.

Morphisms (terms) are defined by

- id , $\star$, eval , π_1 and π_2 are morphisms
- if t is a morphism, λt is a morphism
- if t and u are morphisms, (t, u) and $u \circ t$ are morphisms.

Well-defined morphisms (typed terms) are defined by the following typing rules (in which the usual categorical morphism notation $f : \alpha \to \beta$ is replaced with sequent calculus notation $f :- \alpha \vdash \beta$).

Identity:

$$\frac{}{id :- \alpha \vdash \alpha}$$

Composition:

$$\frac{t :- \alpha \vdash \beta \qquad u :- \beta \vdash \gamma}{u \circ t :- \alpha \vdash \gamma}$$

Unit type (terminal object):

$$\frac{}{\star :- \alpha \vdash \top}$$

Cartesian product:

$$\frac{t :- \alpha \vdash \beta \qquad u :- \alpha \vdash \gamma}{(t, u) :- \alpha \vdash \beta \times \gamma}$$

Left and right projection:

$$\frac{}{\pi_1 :- \alpha \times \beta \vdash \alpha} \qquad \frac{}{\pi_2 :- \alpha \times \beta \vdash \beta}$$

Currying:

$$\frac{t :- \alpha \times \beta \vdash \gamma}{\lambda t :- \alpha \vdash \beta \to \gamma}$$

Application:

$$\frac{}{eval :- (\alpha \to \beta) \times \alpha \vdash \beta}$$

Finally, the equations of the category are

- $id \circ t = t$
- $t \circ id = t$
- $(v \circ u) \circ t = v \circ (u \circ t)$
- $\star \circ t = \star$
- $\pi_1 \circ (t, u) = t$
- $\pi_2 \circ (t, u) = u$
- $(\pi_1 \circ t, \pi_2 \circ t) = t$
- $eval \circ (\lambda t \circ \pi_1, \pi_2) = t$
- $\lambda(eval \circ (t \circ \pi_1, \pi_2)) = t$

Now, there exists t such that $t :- \alpha_1 \times \ldots \times \alpha_n \vdash \beta$ iff $\alpha_1, \ldots, \alpha_n \vdash \beta$ is provable in implicational intuitionistic logic,.

5 Examples

Thanks to the Curry–Howard correspondence, a typed expression whose type corresponds to a logical formula is analogous to a proof of that formula. Here are examples.

The identity combinator seen as a proof of α → α in Hilbert-style logic

As a simple example, we construct a proof of the theorem α → α. In lambda calculus, this is the type of the identity function **I** = λ*x*.*x* and in combinatory logic, the identity function is obtained by applying **S** twice to **K**. That is, we have **I** = (**S K**) **K**). As a description of a proof, this says that to prove α → α, we can proceed as follows:

- instantiate the second axiom scheme with the formulas α, β → α and α, so that to obtain a proof of (α → ((β → α) → α)) → ((α → (β → α)) → (α → α)),

- instantiate the first axiom scheme once with α and $\beta \to \alpha$, so that to obtain a proof of $\alpha \to ((\beta \to \alpha) \to \alpha)$,
- instantiate the first axiom scheme a second time with α and β, so that to obtain a proof of $\alpha \to (\beta \to \alpha)$,
- apply modus ponens twice so that to obtain a proof of $\alpha \to \alpha$

In general, the procedure is that whenever the program contains an application of the form $(P\ Q)$, we should first prove theorems corresponding to the types of P and Q. Since P is being applied to Q, the type of P must have the form $\alpha \to \beta$ and the type of Q must have the form α for some α and β. We can then detach the conclusion, β, via the modus ponens rule.

The composition combinator seen as a proof of $(\beta \to \alpha) \to (\gamma \to \beta) \to \gamma \to \alpha$ in Hilbert-style logic

As a more complicated example, let's look at the theorem that corresponds to the **B** function. The type of **B** is $(\beta \to \alpha) \to (\gamma \to \beta) \to \gamma \to \alpha$. **B** is equivalent to (**S** (**K S**) **K**). This is our roadmap for the proof of the theorem $(\beta \to \alpha) \to (\gamma \to \beta) \to \gamma \to \alpha$.

First we need to construct (**K S**). We make the antecedent of the **K** axiom look like the **S** axiom by setting α equal to $(\alpha \to \beta \to \gamma) \to (\alpha \to \beta) \to \alpha \to \gamma$, and β equal to δ (to avoid variable collisions):

K : $\alpha \to \beta \to \alpha$

K$[\alpha = (\alpha \to \beta \to \gamma) \to (\alpha \to \beta) \to \alpha \to \gamma, \beta{=}\delta]$: $((\alpha \to \beta \to \gamma) \to (\alpha \to \beta) \to \alpha \to \gamma) \to \delta \to (\alpha \to \beta \to \gamma) \to (\alpha \to \beta) \to \alpha \to \gamma$

Since the antecedent here is just **S**, we can detach the consequent using Modus Ponens:

K S : $\delta \to (\alpha \to \beta \to \gamma) \to (\alpha \to \beta) \to \alpha \to \gamma$

This is the theorem that corresponds to the type of (**K S**). We now apply **S** to this expression. Taking **S**

S : $(\alpha \to \beta \to \gamma) \to (\alpha \to \beta) \to \alpha \to \gamma$

we put $\alpha = \delta$, $\beta = \alpha \to \beta \to \gamma$, and $\gamma = (\alpha \to \beta) \to \alpha \to \gamma$, yielding

S$[\alpha = \delta, \beta = \alpha \to \beta \to \gamma, \gamma = (\alpha \to \beta) \to \alpha \to \gamma]$: $(\delta \to (\alpha \to \beta \to \gamma) \to (\alpha \to \beta) \to \alpha \to \gamma) \to (\delta \to (\alpha \to \beta \to \gamma)) \to \delta \to (\alpha \to \beta) \to \alpha \to \gamma$

and we then detach the consequent:

S (K S) : $(\delta \to \alpha \to \beta \to \gamma) \to \delta \to (\alpha \to \beta) \to \alpha \to \gamma$

This is the formula for the type of (**S** (**K S**)). A special case of this theorem has $\delta = (\beta \to \gamma)$:

S (K S)$[\delta = \beta \to \gamma]$: $((\beta \to \gamma) \to \alpha \to \beta \to \gamma) \to (\beta \to \gamma) \to (\alpha \to \beta) \to \alpha \to \gamma$

We need to apply this last formula to **K**. Again, we specialize **K**, this time by replacing α with $(\beta \to \gamma)$ and β with α:

K : $\alpha \to \beta \to \alpha$

K$[\alpha = \beta \to \gamma, \beta = \alpha]$: $(\beta \to \gamma) \to \alpha \to (\beta \to \gamma)$

This is the same as the antecedent of the prior formula, so we detach the consequent:

S (K S) K : $(\beta \to \gamma) \to (\alpha \to \beta) \to \alpha \to \gamma$

Switching the names of the variables α and γ gives us

$(\beta \to \alpha) \to (\gamma \to \beta) \to \gamma \to \alpha$

which was what we had to prove.

The normal proof of $(\beta \to \alpha) \to (\gamma \to \beta) \to \gamma \to \alpha$ in natural deduction seen as a λ-term

We give below a proof of $(\beta \to \alpha) \to (\gamma \to \beta) \to \gamma \to \alpha$ in natural deduction and show how it can be interpreted as the λ-expression $\lambda\, a.\ \lambda b.\ \lambda\, g.(a\ (b\ g))$ of type $(\beta \to \alpha) \to (\gamma \to \beta) \to \gamma \to \alpha$.

a:$\beta \to \alpha$, b:$\gamma \to \beta$, g:$\gamma \vdash$ b : $\gamma \to \beta$ a:$\beta \to \alpha$, b:$\gamma \to \beta$, g:$\gamma \vdash$ g : γ ——————

—— a:$\beta \to \alpha$, b:$\gamma \to \beta$, g:$\gamma \vdash$ a : $\beta \to \alpha$ a:$\beta \to \alpha$, b:$\gamma \to \beta$, g:$\gamma \vdash$ b g : β ——————

—————— a:$\beta \to \alpha$, b:$\gamma \to \beta$, g:$\gamma \vdash$ a (b g) : α ——————

—————— a:$\beta \to \alpha$, b:$\gamma \to \beta \vdash$ λ g. a (b g) : $\gamma \to \alpha$ ——————

—————— a:$\beta \to \alpha \vdash$ λ b. λ g. a (b g) : $(\gamma \to \beta)$ -> $\gamma \to \alpha$ ——————

—————— $\vdash$ λ a. λ b. λ g. a (b g) : $(\beta \to \alpha)$ -> $(\gamma \to \beta)$ -> $\gamma \to \alpha$

6 Other applications

Recently, the isomorphism has been proposed as a way to define search space partition in Genetic programming.[9] The method indexes sets of genotypes (the program trees evolved by the GP system) by their Curry–Howard isomorphic proof (referred to as a species).

7 Generalizations

The correspondences listed here go much farther and deeper. For example, cartesian closed categories are generalized by closed monoidal categories. The internal language of these categories is the linear type system (corresponding to linear logic), which generalizes simply-typed lambda calculus as the internal language of cartesian closed categories. What's more, these can be shown to correspond to cobordisms,[10] which play a vital role in string theory.

An extended set of equivalences is also explored in homotopy type theory, which is a very active area of research at this time (2013). Here, type theory is extended by the univalence axiom, ('equivalence is equivalent to equality') which permits homotopy type theory to be used as a foundation for all of mathematics (including set theory and classical logic, providing new ways to discuss the axiom of choice and many other things). That is, the Curry–Howard correspondence that proofs are elements of inhabited types is generalized to the notion homotopic equivalence of proofs (as paths in space, the identity type or equality type of type theory being interpreted as a path).[11]

8 References

[1] The correspondence was first made explicit in Howard 1980. See, for example section 4.6, p.53 Gert Smolka and Jan Schwinghammer (2007-8), Lecture Notes in Semantics

[2] The Brouwer–Heyting–Kolmogorov interpretation is also called the 'proof interpretation': p. 161 of Juliette Kennedy, Roman Kossak, eds. 2011. *Set Theory, Arithmetic, and Foundations of Mathematics: Theorems, Philosophies* ISBN 978-1-107-00804-5

[3] Curry 1934.

[4] Curry & Feys 1958.

[5] Howard 1980.

[6] Moggi, Eugenio (1991), "Notions of Computation and Monads" (PDF), *Information and Computation*, **93** (1): 55–92, doi:10.1016/0890-5401(91)90052-4

[7] Sørenson, Morten; Urzyczyn, Paweł, *Lectures on the Curry-Howard Isomorphism*, CiteSeerX 10.1.1.17.7385

[8] Goldblatt, "7.6 Grothendieck Topology as Intuitionistic Modality", *Mathematical Modal Logic: A Model of its Evolution* (PDF), pp. 76–81. The "lax" modality referred to is from Benton; Bierman; de Paiva (1998), "Computational types from a logical perspective", *Journal of Functional Programming*, **8**: 177–193, doi:10.1017/s0956796898002998

[9] F. Binard and A. Felty, "Genetic programming with polymorphic types and higher-order functions." In *Proceedings of the 10th annual conference on Genetic and evolutionary computation,* pages 1187 1194, 2008.

[10] John c. Baez and Mike Stay, "Physics, Topology, Logic and Computation: A Rosetta Stone", (2009) ArXiv 0903.0340 in *New Structures for Physics*, ed. Bob Coecke, *Lecture Notes in Physics* vol. **813**, Springer, Berlin, 2011, pp. 95-174.

[11] *Homotopy Type Theory: Univalent Foundations of Mathematics.* (2013) The Univalent Foundations Program. Institute for Advanced Study.

8.1 Seminal references

- Curry, Haskell (1934), "Functionality in Combinatory Logic", *Proceedings of the National Academy of Sciences*, **20**, pp. 584–590, doi:10.1073/pnas.20.11.584.
- Curry, Haskell B.; Feys, Robert (1958), Craig, William, ed., *Combinatory Logic Vol. I*, Amsterdam: North-Holland, with 2 sections by William Craig, see paragraph 9E.
- De Bruijn, Nicolaas (1968), *Automath, a language for mathematics*, Department of Mathematics, Eindhoven University of Technology, TH-report 68-WSK-05. Reprinted in revised form, with two pages commentary, in: *Automation and Reasoning, vol 2, Classical papers on computational logic 1967–1970*, Springer Verlag, 1983, pp. 159–200.
- Howard, William A. (September 1980) [original paper manuscript from 1969], "The formulae-as-types notion of construction", in Seldin, Jonathan P.; Hindley, J. Roger, *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, Boston, MA: Academic Press, pp. 479–490, ISBN 978-0-12-349050-6.

8.2 Extensions of the correspondence

[1] Pfenning, Frank; Davies, Rowan (2001), "A Judgmental Reconstruction of Modal Logic" (PDF), *Mathematical Structures in Computer Science*, **11**: 511–540, doi:10.1017/S0960129501003322

[2] Davies, Rowan; Pfenning, Frank (2001), "A Modal Analysis of Staged Computation" (PDF), *Journal of the ACM*, **48** (3): 555–604, doi:10.1145/382780.382785

- Griffin, Timothy G. (1990), "The Formulae-as-Types Notion of Control", *Conf. Record 17th Annual ACM Symp. on Principles of Programming Languages, POPL '90, San Francisco, CA, USA, 17–19 Jan 1990*, pp. 47–57.
- Parigot, Michel (1992), "Lambda-mu-calculus: An algorithmic interpretation of classical natural deduction", *International Conference on Logic Programming and Automated Reasoning: LPAR '92 Proceedings, St. Petersburg, Russia*, Lecture Notes in Computer Science, **624**, Berlin; New York: Springer-Verlag, pp. 190–201, ISBN 978-3-540-55727-2.
- Herbelin, Hugo (1995), "A Lambda-Calculus Structure Isomorphic to Gentzen-Style Sequent Calculus Structure", in Pacholski, Leszek; Tiuryn, Jerzy, *Computer Science Logic, 8th International Workshop, CSL '94, Kazimierz, Poland, September 25–30, 1994, Selected Papers*, Lecture Notes in Computer Science, **933**, Berlin; New York: Springer-Verlag, pp. 61–75, ISBN 978-3-540-60017-6.
- Gabbay, Dov; de Queiroz, Ruy (1992), "Extending the Curry–Howard interpretation to linear, relevant and other resource logics", *Journal of Symbolic Logic*, **57**, pp. 1319–1365, doi:10.2307/2275370. (Full version of the paper presented at *Logic Colloquium '90*, Helsinki. Abstract in *JSL* 56(3):1139–1140, 1991.)
- de Queiroz, Ruy; Gabbay, Dov (1994), "Equality in Labelled Deductive Systems and the Functional Interpretation of Propositional Equality", in Dekker, Paul; Stokhof, Martin, *Proceedings of the Ninth Amsterdam Colloquium*, ILLC/Department of Philosophy, University of Amsterdam, pp. 547–565, ISBN 90-74795-07-2.
- de Queiroz, Ruy; Gabbay, Dov (1995), "The Functional Interpretation of the Existential Quantifier", *Bulletin of the Interest Group in Pure and Applied Logics*, 3(2–3), pp. 243–290. (Full version of a paper presented at *Logic Colloquium '91*, Uppsala. Abstract in *JSL* 58(2):753–754, 1993.)
- de Queiroz, Ruy; Gabbay, Dov (1997), "The Functional Interpretation of Modal Necessity", in de Rijke, Maarten, *Advances in Intensional Logic*, Applied Logic Series, **7**, Springer-Verlag, pp. 61–91, ISBN 978-0-7923-4711-8.
- de Queiroz, Ruy; Gabbay, Dov (1999), "Labelled Natural Deduction", in Ohlbach, Hans-Juergen; Reyle, Uwe, *Logic, Language and Reasoning. Essays in Honor of Dov Gabbay*, Trends in Logic, **7**, Kluwer Acad. Pub., pp. 173–250, ISBN 978-0-7923-5687-5.
- de Oliveira, Anjolina; de Queiroz, Ruy (1999), "A Normalization Procedure for the Equational Fragment of Labelled Natural Deduction", *Logic Journal of the Interest Group in Pure and Applied Logics*, **7** (2), Oxford University Press, pp. 173–215. (Full version of a paper presented at *2nd WoLLIC'95*, Recife. Abstract in *Journal of the Interest Group in Pure and Applied Logics* 4(2):330–332, 1996.)
- Poernomo, Iman; Crossley, John; Wirsing; Martin (2005) [2005], *Adapting Proofs-as-Programs: The Curry–Howard Protocol*, Monographs in Computer Science, Springer, ISBN 978-0-387-23759-6, concerns the adaptation of proofs-as-programs program synthesis to coarse-grain and imperative program development problems, via a method the authors call the Curry–Howard protocol. Includes a discussion of the Curry–Howard correspondence from a Computer Science perspective.
- de Queiroz, Ruy J.G.B.; de Oliveira, Anjolina (2011), "The Functional Interpretation of Direct Computations", *Electronic Notes in Theoretical Computer Science*, **269**, Elsevier, pp. 19–40, doi:10.1016/j.entcs.2011.03.003. (Full version of a paper presented at *LSFA 2010*, Natal, Brazil.)

8.3 Philosophical interpretations

- de Queiroz, Ruy J.G.B. (1994), "Normalisation and language-games", *Dialectica*, **48** (2), pp. 83–123. (Early version presented at *Logic Colloquium '88*, Padova. Abstract in *JSL* 55:425, 1990.)
- de Queiroz, Ruy J.G.B. (2001), "Meaning, function, purpose, usefulness, consequences – interconnected concepts", *Logic Journal of the Interest Group in Pure and Applied Logics*, **9** (5), pp. 693–734. (Early version presented at *Fourteenth International Wittgenstein Symposium (Centenary Celebration)* held in Kirchberg/Wechsel, August 13–20, 1989.)
- de Queiroz, Ruy J.G.B. (2008), "On reduction rules, meaning-as-use and proof-theoretic semantics", *Studia Logica*, **90** (2), pp. 211–247, doi:10.1007/s11225-008-9150-5.

8.4 Synthetic papers

- De Bruijn, Nicolaas Govert (1995), "On the roles of types in mathematics" (PDF), in Groote, Philippe de, *The Curry–Howard isomorphism*, Cahiers du Centre de logique (Université catholique de Louvain), **8**, Academia-Bruyland, pp. 27–54, ISBN 978-2-87209-363-2, the contribution of de Bruijn by himself.
- Geuvers, Herman (1995), "The Calculus of Constructions and Higher Order Logic", in Groote, Philippe de, *The Curry–Howard isomorphism*,

Cahiers du Centre de logique (Université catholique de Louvain), **8**, Academia-Bruyland, pp. 139–191, ISBN 978-2-87209-363-2, contains a synthetic introduction to the Curry–Howard correspondence.

- Gallier, Jean H. (1995), "On the Correspondence between Proofs and Lambda-Terms" (PDF), in Groote, Philippe de, *The Curry–Howard isomorphism*, Cahiers du Centre de logique (Université catholique de Louvain), **8**, Academia-Bruyland, pp. 55–138, ISBN 978-2-87209-363-2, contains a synthetic introduction to the Curry–Howard correspondence.
- Wadler, Philip (2014), "Propositions as Types" (PDF), *Communications of the ACM*, **58**: 75–84, doi:10.1145/2699407

8.5 Books

- edited by Ph. de Groote. (1995), De Groote, Philippe, ed., *The Curry–Howard Isomorphism*, Cahiers du Centre de Logique (Université catholique de Louvain), **8**, Academia-Bruylant, ISBN 978-2-87209-363-2, reproduces the seminal papers of Curry-Feys and Howard, a paper by de Bruijn and a few other papers.
- Sørensen, Morten Heine; Urzyczyn, Paweł (2006) [1998], *Lectures on the Curry–Howard isomorphism*, Studies in Logic and the Foundations of Mathematics, **149**, Elsevier Science, CiteSeerX 10.1.1.17.7385, ISBN 978-0-444-52077-7, notes on proof theory and type theory, that includes a presentation of the Curry–Howard correspondence, with a focus on the formulae-as-types correspondence
- Girard, Jean-Yves (1987–90). *Proof and Types*. Translated by and with appendices by Lafont, Yves and Taylor, Paul. Cambridge University Press (Cambridge Tracts in Theoretical Computer Science, 7), ISBN 0-521-37181-3, notes on proof theory with a presentation of the Curry–Howard correspondence.
- Thompson, Simon (1991). *Type Theory and Functional Programming* Addison–Wesley. ISBN 0-201-41667-0.
- Poernomo, Iman; Crossley, John; Wirsing; Martin (2005) [2005], *Adapting Proofs-as-Programs: The Curry–Howard Protocol*, Monographs in Computer Science, Springer, ISBN 978-0-387-23759-6, concerns the adaptation of proofs-as-programs program synthesis to coarse-grain and imperative program development problems, via a method the authors call the Curry–Howard protocol. Includes a discussion of the Curry–Howard correspondence from a Computer Science perspective.
- F. Binard and A. Felty, "Genetic programming with polymorphic types and higher-order functions." In *Proceedings of the 10th annual conference on Genetic and evolutionary computation,* pages 1187 1194, 2008.
- de Queiroz, Ruy J.G.B.; de Oliveira, Anjolina G.; Gabbay, Dov M. (2011), *The Functional Interpretation of Logical Deduction*, Advances in Logic, **5**, Imperial College Press/World Scientific, ISBN 978-981-4360-95-1.

9 Further reading

- P.T. Johnstone, 2002, *Sketches of an Elephant*, section D4.2 (vol 2) gives a categorical view of "what happens" in the Curry–Howard correspondence.

10 External links

- Howard on Curry-Howard
- The Curry–Howard Correspondence in Haskell
- The Monad Reader 6: Adventures in Classical-Land: Curry–Howard in Haskell, Pierce's law.

11 Text and image sources, contributors, and licenses

11.1 Text

- **Curry–Howard correspondence** *Source:* https://en.wikipedia.org/wiki/Curry%E2%80%93Howard_correspondence?oldid=760416532 *Contributors:* Taral, Toby Bartels, Genneth, Edward, Michael Hardy, Dominus, Chinju, AugPi, Charles Matthews, Doradus, Phil Boswell, Anthony, Iwehrman, Jleedev, Tea2min, Ancheta Wis, Giftlite, DefLog~enwiki, SethTisue, Leibniz, Ben Standeven, Chalst, Nickj, Cmdrjameson, Karlheg, Txa, Anthony Appleyard, Jamiemichelle, Oleg Alexandrov, Linas, Ruud Koot, Dionyziz, Marudubshinki, Rjwilmsi, Vlad Patryshev, Mathbot, Jrtayloriv, Wavelength, Hairy Dude, Michael Slone, Cedar101, Eaefremov, Nnxion, SmackBot, Eskimbot, Mhss, Cybercobra, Jon Awbrey, Igrant, Physis, Iridescent, JRSpriggs, Chris55, CRGreathouse, ShelfSkewed, Gregbard, Goldenowl, Seunghun, Oerjan, Rowandavies, RebelRobot, Four Dog Night, David Eppstein, JaGa, Gwern, N4nojohn, SparsityProblem, McM.bot, SieBot, Laocoön11, Classicalecon, MilesAgain, Hugo Herbelin, Tre2, Addbot, LaaknorBot, Legobot, Yobot, Ptbotgourou, Pcap, EricP, AnomieBOT, Citation bot, Xqbot, Noamz, FrescoBot, Goheeca, OriumX, RedBot, Francis Lima, Burritoburritoburrito, Mattghg, RjwilmsiBot, EmausBot, John of Reading, ZéroBot, Elaz85, ClueBot NG, Helpful Pixie Bot, CitationCleanerBot, BattyBot, Pintoch, Jochen Burghardt, Lambda Fairy, Andyhowlett, François Robere, Monkbot, Bender the Bot and Anonymous: 70

11.2 Images

- **File:Coq_plus_comm_screenshot.jpg** *Source:* https://upload.wikimedia.org/wikipedia/commons/8/8b/Coq_plus_comm_screenshot.jpg *License:* CC-BY-SA-3.0 *Contributors:* snapshot of LGPL software CoqIDE ran in Gnome *Original artist:* Hugo Herbelin
- **File:Lock-green.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/6/65/Lock-green.svg *License:* CC0 *Contributors:* en:File:Free-to-read_lock_75.svg *Original artist:* User:Trappist the monk
- **File:Text_document_with_red_question_mark.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/a/a4/Text_document_with_red_question_mark.svg *License:* Public domain *Contributors:* Created by bdesham with Inkscape; based upon Text-x-generic.svg from the Tango project. *Original artist:* Benjamin D. Esham (bdesham)
- **File:Wikibooks-logo-en-noslogan.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/d/df/Wikibooks-logo-en-noslogan.svg *License:* CC BY-SA 3.0 *Contributors:* Own work *Original artist:* User:Bastique, User:Ramac et al.

11.3 Content license

- Creative Commons Attribution-Share Alike 3.0