Three algorithms

Will Troiani

February 22, 2022

1 Introduction

It is the twenty-first century and formal logic has been irreversibly intertwined with the theory of computation. When pressed on exactly why this intertwinement is actual science and not mere hope one can quickly list off at least three algorithms which are central to the theory of computation which are each born through the study of formal logic, the following is such a list.

1. The cut-elimination process of Sequent Calculus.
2. The construction of a representing formulas for a given general recursive function.
3. The counit of the Curry-Howard-Lambek correspondence.

We explain these in more detail, but first we tantilize the reader: these three examples were not chosen arbitrarily, they all have something crucial in common, we invite the reader to consider what this may be as we describe each of these algorithms in more detail.

Cut-elimination: What distinguishes a logical system to a computational system? An opinion loyal to history would describe how a logical system would concern itself with proofs while a computational system would admit some kind of re-write procedure. For instance, we recognise Turing Machines and the simply typed lambda calculus as computational systems due to their re-write systems respectively being that of the tape head and β-reduction. Hence, at the very least a computational system must be *dynamic*. So then what is the Sequent Calculus? This is a formal language designed for writing proofs, but it also admits a re-write procedure, that described by the cut-elimination Theorem, which states that every formula φ provable using the cut rule (modus ponens) is equivalent to a proof of φ which does *not* make use of the cut-rule. However, the existence of such a cut-free proof is not the whole story, as in fact there is an effective *procedure* which turns a proof π which makes use of the cut rule and iteratively constructs proofs of lower complexity cuts until eventually all the instances of the cut rule are completely eliminated. This cut-reduction procedure has the same expressive power as the simply-typed lambda calculus, and hence the once sharp line between proof systems and systems of computation is blurred.

Representing formulas: Logic is the science which comes before all sciences, *right?* Well, without making a difinitive statement on this, at the very least one must reckon with Gödel's First Incompleteness Theorem before an answer is drawn. A common misquoting of this Theorem is the following: any logical system, sufficiently expressive, is necessarily either incomplete, ie, there exists a formula F which for which there exists no proof F and no proof of $\neg F$, or inconsistent, in that there exists a formula F such that both F and $\neg F$ are proveable. Why is this statement incorrect? It is because the statement failed to include the hypothesis that the set of axioms and the set of deduction rules are *effectively decidable*. In fact, concistent and complete systems sufficiently strong enough to express the basic laws of arithmetic *do* exist! They are just not describable via computable means. The key component to the standard proof of this result is the algorithm which takes a general recursive function and constructs a formula F representing this function. Indeed, it is significant that is procedure is *effectively computable*, if this procedure is allowed to perform uncomputable steps, then the result fails to hold.

Curry-Howard-Lambek: An internal language can be thought of as a concrete syntax which may be used to describe objects and morphisms inside a category, where the structure of the type theory reflects the structure of the category. This is a concrete example of logic as a language for mathematics. The line which is blurred here is between *content* and *form*; be the type theory associated to a category rich enough, then terms in the type theory become more comprehensible than the morphisms or objects they represent. Hence the morphism is percieved via the syntax, a morphism whose existence would be difficult to describe had this syntactic system not been present. Not only do terms of an internal logic describe morphisms, they describe *procedures* for constructing the morphism via the categorical structure. In the particular case of the Curry-Howard-Lambek correspondence, the categorical structure is that of a Cartesian Closed Category, and the internal language is a typed lambda-calculus (with products) taken up to $\alpha\beta\eta$-equivalence.

So what do these three systems have in common? Clearly, they are all results where the most significant component is the providence of some kind of effective procedure, but moreover, *they were all originally proved non-constructively*, that is, none of these results were originally proven in such a way that the implied algorithm can be easily extracted from the proof.

Indeed, Gentzen's originally proved the cut-elimination Theorem [2] by introducing an auxillary deduction rule, the *mix* rule, and proving that this rule can be eliminated. Gödel's First Incompleteness Theorem was proved [**?**] by first coming up with a simpler characterisation of the general recursive functions using his β-function Lemma and then showed that the general recursive functions are representable via this simpler characterisation. The Curry-Howard-Lambek correspondence is the least offender of this pattern, indeed this result crucially uses the algorithm implied by *functional completeness*, a result due to Lambek [**?**], but the standard proof that the two concerned categories are equivalent does not put this algorithm in central focus.

In this note we remedy this by providing constructive proofs of all three results. From the proofs given here, the relevant algorithms are extracted easily and written down concretely for the first time.

2 Curry-Howard-Lambek Correspondence

Definition 2.0.1. A **positive intuitionist calculus** consists of:

- An at most countably infinite collection $\mathcal{T}$ of types $A, B, C, \ldots$. Amongst these is a special type $\top$. There are also two type constructors, conjunction and arrow. More precisely, if A, B are types then so are $A \wedge B, A \Rightarrow B$.
- A binary relation $\vdash$ on T. If A, B are types such that $(A, B) \in \vdash$ then we write $A \vdash B$.

Subject to the following.

- The relation $\vdash$ is subject to the following deduction rules, these are the **Axiom rules**:

$$\frac{}{A \vdash A}\,(\text{ax}) \qquad \frac{}{A \vdash \top}\,0_A \qquad \frac{}{A \wedge B \vdash A}\,(\wedge E)_1 \qquad \frac{}{A \wedge B \vdash B}\,(\wedge E)_2$$

- There are also the **constructor rules:**

$$\frac{A \vdash B \qquad B \vdash C}{A \vdash C}\,(\text{cut}) \qquad \frac{C \vdash A \qquad C \vdash B}{C \vdash \langle A, B \rangle}\,(\wedge I) \qquad \frac{A \wedge B \vdash C}{A \vdash B \Rightarrow C}\,(\Rightarrow I)$$

References

[1] W. Troiani, D. Murfet. *The Gentzen-Mints-Zucker Duality*

[2] G. Gentzen. *Investigations into logical deduction*

[3] Godel K. Gödel, *On Formally Undecidable Propositions of Principia Mathematica*

[4] LambekScott J. Lambek, P. J. Scott

[5] lambek J. Lambek, *Functional Completeness.*