A PROOF THEORETIC FOUNDATION FOR COMPUTATION WITH APPLICATIONS TO CHURCH'S THESIS, ALGORITHM IDENTITY AND COMPUTATIONALIST THEORIES OF MIND

Abstract. This paper uses recent technical results in proof theory to propose that a proof theoretic treatment can be given for general theory of functions and computation: intensional untyped λ-calculus. It is argued that a proof theoretic account for *computational* laws can be given in a similar way to familiar accounts of *logical* laws. We then obtain new responses to questions relating to Church's Thesis, the concept of algorithmic identity and the possibility of a thinking machine.

1. Introduction

There are many versions of the proof theoretic account of logic. Or at least, elements and variations of it appear in much philosophical literature on the nature of logic and logical definition. The origin of the proof theoretic treatment of the logical constants is commonly cited to be Gentzen's remarks in [6] (famously taken up by Prawitz, Hacking and Dummett, among others), where he suggests that certain inference rules may be seen as definitional of their subject connectives.

But Gentzen was not the first to offer an account of logic founded on proof-theory. Frege, at the beginning of [2], defines a truth of logic as one which can be *derived* from some 'primitive' truths. Neither is it the case that Frege was concerned only with logic. The theory developed in [2] and [3] is primarily a foundational theory of *functions*.

With the discovery of the logical paradoxes, Frege's foundational theory of functions was largely abandoned. With the exception of some ideas of Church, which hit similar paradoxes to Frege (see, e.g. [8]), logic and computation have been regarded as separate areas with distinct, although connected, foundations (e.g. by the Curry-Howard isomorphism).

This paper offers a proof theoretic foundation for a theory of functions by appealing to an extension of classical sequent calculus that can reason on computational 'constants'. This suggests a new analysis of the intuitive concept of computation, and the formal nature of this analysis makes is readily applicable to Church's thesis. Furthermore, the theory of functions obtained is intensional – extensionally equivalent functions need not be identified, – so we may apply the approach to 'intensional' issues such as identity criteria for algorithms and the computational analysis of mental content.

2. Proof theory and logic

This paper's version of the proof theoretic account is inspired by Hacking's [7].

In the case of logic we begin, not with proof theory, but with some semantic principles.

(1) The subject matter of logic are consequence relations; logical consequence is a relation on propositions; propositions are either true or false; and consequence relations preserve truth.

These principles are then characterised proof theoretically, by the following structural rules:

(2)
$$\frac{}{A \vdash A}\ \text{(Reflexivity)} \qquad \frac{\Gamma \vdash A, \Delta \quad \Gamma, A \vdash \Delta}{\Gamma \vdash \Delta}\ \text{(Cut)}$$

$$\frac{\Gamma \vdash \Delta}{\Gamma, A \vdash \Delta} \qquad \frac{\Gamma \vdash \Delta}{\Gamma \vdash A, \Delta}\ \text{(Weakening)} \qquad \Gamma, \Delta \text{ are } \textit{sets} \text{ of propositions}$$

The connection between (2) and (1) is closer than one might think. For example, (Cut) and (Reflexivity) are clearly connected to the reflexive-transitive relation of truth preservation. Also, as has been shown in [1], there is a close connection between the remaining structural rules and the condition that there are only two truth values.

The proof theoretic account continues by specifying that a logical connective is one that is definable in accordance with the semantic principles of (1). This corresponds to the proof theoretic condition that the derivation rules for the new connective should preserve the structural rules of (2).

Therefore, on the proof theoretic approach, a connective is *logical* when its inference rules preserve the *admissibility* of the rules of (2); and logic is the theory of what is derivable or definable using logical connectives. Given some putative logical connectives, the hardest admissability result to obtain is generally the admissibility of (Cut) (commonly, and confusingly, called *cut-elimination*).

It is worth noting that this approach yields much from few resources. From the basic semantic principles of (1) we obtain the whole of first order quantified logic, without ever mentioning models, valuations, properties etc..

3. Proof theory and computation

Assuming the proof theoretic answer to the question 'what is logic?' is satisfactory, it turns out we can give a similar account to the concept of computation.

Whereas logic deals primarily with propositions, computation deals primarily with functions. Just as there are laws governing propositions and logical inference, so are there laws governing functions and function application. Examples of such laws are the associativity of function composition and the existence of an identity function.

In fact, computation and logic have much in common. A logical derivation consists of a sequence of derivations each in accordance with a logical inference rule. A computation, similarly consists of a sequence of computational steps, each in accordance a rule of computation. Analogous to the case for logical rules, this paper builds on recent technical results to offer a proof theoretic account of the fundamental rules of computation.

As with logical consequence, a proof theoretic account of computation begins with a semantic principle.

(3) The relata of computational relations are functions; functions take other functions as their values and arguments; and computing on a function preserves its values.

To gain intuition for this principle, consider a simple computation on the function (3×2)+(1+0): 0,1,2, and 3 are (constant) functions; 2×3 is the function '× applied to 2 and 3'; 2×3 computes to 6; 1+0 computes 1; and so the whole thing computes to the function 7. As instantiated in this simple example, each computational step preserves the overall value of the function. In this case the function computes to a constant value, 7, but some computations are far more complex and may not terminate so easily (or at all).

The proposed proof theoretic characterisation of (3) is given by the structural rules of the sequent system developed in [4], where a derivation system is given for *untyped* λ-calculus. Informally, the structural rules are:

(4)
$$\frac{}{x \text{ computes to } x} \quad (\lambda\text{-Reflexivity})$$

$$\frac{x \text{ computes to } y \quad \text{and} \quad g(\dots y \dots) \text{ computes to } z}{g(\dots x \dots) \text{ computes to } z} \quad (\lambda\text{-Cut})$$

In much the same way that rules are given for logical quantifiers, we can present rules for a function abstraction operator λ. These rules maintain the admissibility of (λ-Cut) and can derive all the reduction relations of the (instensional) *untyped* λ-calculus.

The untyped λ-calculus is a theory of computation in at least two senses: it is the most general and abstract theory of functions available; it is itself powerful enough to represent all recursive functions.

4. Some applications of the account

Church's Thesis. We obtain a new argument that every computable function is recursive. The overall structure of the argument is simple, and is similar to the 'squeezing' argument of [9]. Under the proof-theoretic approach to computation, every computable function must be representable, in principle, in the untyped λ-calculus, which itself is a recursively definable formal theory. Thus we can argue that the intuitive concept of an effective computation is grounded in such a way that its extension is identical to that of the recursive functions. Furthermore, given the literature on SAD computers, we can also give a neat characterisation of the intuitive concept of a non-effective 'hypercomputation' and relate it to Church's thesis: a hypercomputation is just like a computation except it has an infinitary proof theoretic foundation.

Identity of computations/algorithms. Given the proof theoretic analysis of what an abstract computation is, we can move towards an account of when two algorithms or computations are the same. A notable feature of the account is that it allows for an *intensional* as well as *extensional* notion of computational identity. Again, the structure of the answer is simple: two algorithms are (intensionally) the same when they are canonically represented abstractly by the same schematic λ-term. The matter is complicated by what counts as a 'canonical' representation in the λ-calculus, this paper will propose that, in fact, the notion of

computational/algorithmic identity is *relative* to a background notion of canonical representation.

Computationalism and the mind. The proposed account also offers a treatment of *computational content*, akin to familiar proof theoretic accounts of logical content. As already noted, the account of computation is intensional rather than merely extensional. So the approach can be applied to computational theories of the mind to help differentiate between between extensionally-equivalent computational states. Thus we can make sense of the idea that two machines can be Turing-equivalent yet distinct in computational complexity and content. This allows us to further a new line of response to famous objections to computationalist theories of the mind such as Searle's Chinese Room argument. The main idea behind the response (e.g. see [10]) is an abstract notion of a program whereby a thinker and the Chinese Room can be said to be running different programs, even if they are otherwise indistinguishable. Conditions are then proposed on what makes a running program 'think', rather than remain merely mechanical as the Chinese Room appears to be.

Connectionism and Symbolic Computation. To provide further support to the application to computationalism above, the paper will appeal to some further technical results in the semantics of λ-reduction (e.g. in [5]). These show that λ-calculus, a calculus of symbolic computation, is complete for a relational semantics. This semantics has many of the key characteristics of a connectionist network and this paper shall suggest, by appeal to these semantic results, that the connectionist and classical (symbolic) notions of computation are actually very close (and perhaps can be unified).

Mathematics. A further, bonus, application which we shall not develop in detail here is that recursive arithemtic, which we noted is representable in terms of λ-reduction, can indeed be given a proof theoretic justification. This has relevance to logicism in the philosophy of mathematics.

References

[1] Arnon Avron. Multiplicative conjunction and an algebraic meaning of contraction and weakening. *Weakening, forthcoming in the Journal of Symbolic Logic*, 1998.

[2] Gottlob Frege. *The Foundations of Arithmetic*. Blackwell, Oxford, 1953. tr. by J. L. Austin.

[3] Gottlob Frege. *The Basic Laws of Arithmetic*. Berkeley: University of California, 1967. M. Furth (trans.).

[4] Michael Gabbay. A proof-theoretic treatment of λ-reduction with cut-elimination: λ-calculus as a logic programming language. *Journal of Symbolic Logic*, June 2011.

[5] Murdoch J. Gabbay and Michael J. Gabbay. A simple class of kripke-style models in which logic and computation have equal standing. In *LPAR 16*, 2010.

[6] Gerhard Gentzen. Untersuchungen über das logische Schliessen. *Mathematische Zeitschrift*, 39:176–210 and 405–431, 1934.

[7] Ian Hacking. What is logic? *The Journal of Philosophy*, 76:285–319, 1979.

[8] S. C. Kleene and J. B. Rosser. The inconsistency of certain formal logics. *Annals of Mathematics*, 36(3), 1935.

[9] Peter Smith. *An Introduction to Godel's Theorems*. Cambridge University Press, 2007.

[10] Mark Sprevak. Chinese rooms and program portability. *Brit. J. Phil. Sci.*, 58:755–776, 2007.