1 Mathematical preliminaries

The idea is to show that every System T definable functional $2^{\mathbb{N}} \to 2$ is continuous (under the usual compact-open topology).

1.1 System T

System T was introduced by Kurt Gödel in 1958 for the purposes of the so-called Dialectica interpretation[2]. It has multiple presentations, but it is essentially a typed lambda calculus with a natural numbers type (denoted N).

We read the sequent $\Gamma \vdash t : T$ as "in the context Γ, the term t (is well-formed and) has type T", where a context is a list of variables and their types. If the context is not relevant I will ommit it.

Specifically we have the following term forming rules:

- Variables: We have an infinite set of variables, usually denoted by x, y, … They form terms $\Gamma, x : A \vdash x : A$.
- Lambda abstraction: For each term $\Gamma, x : A \vdash t : B$ we have a term $\Gamma \vdash \lambda x.t : A \to B$.
- Application: For terms $t : A \to B$ and $u : A$ we have a term $t(u) : B$.
- Zero: $\texttt{zero} : N$ is a term.
- Successor: For each term $t : N$ we have it's successor $\texttt{succ}(t) : N$.
- Induction principle: For terms $t_0 : B$ (base case), $t_s : N \to B \to B$ (successor case), and natural number term $t : N$ we have the term $\texttt{natrec}(t_0, t_s, t) : B$, which is obtained by induction on t.

This can be extended by adding boolean values (and a boolean induction principle) without changing the expressive strength of the system, since we can encode booleans in the natural numbers. This will be useful for talking about the functionals we are interested in.

The computaton rules are fairly standard for a call-by-name lambda calculus:

- For application, if the first term can make a step we make it, otherwise we substitute the second term.
- For the successor, if the term inside can make a step we make it.
- For induction, if the term we are doing induction on can step we make it, otherwise it's either zero or a successor, so we can step into the base or successor cases respectively.
- For induction on booleans the same as above, except we compute to true or false, in which case we apply the true or false case repsectively.

1.2 Topology

The space $2^{\mathbb{N}}$ is called the Cantor space. With the compact-open topology it is homeomorphic to the Cantor set as a subspace of the real numbers. It is useful to think of elements of $2^{\mathbb{N}}$ as infinite sequences of zeroes and ones. Then the open sets are generated by a subbasis of sets of functions that map $n \in \mathbb{N}$ to $b \in 2$, for all possible values of n and b.

Continuity of a functional $F : 2^{\mathbb{N}} \to 2$ now means that the preimage of $b \in 2$ under F is an open set. This means that for every $\alpha \in 2^{\mathbb{N}}$ exists some basic open neighbourhood $U \ni \alpha$, such that F maps it into $F(\alpha)$. But this then means that there must be some $k \in \mathbb{N}$ such that if any β agrees with α in the first k places, then F must map them to the same value.

In fact this can be done uniformly, since $2^{\mathbb{N}}$ is compact.

1.3 Forcing

The basic idea of forcing is to add a new "generic" element to some model, such that it can represent an arbitrary element of some set. In our case we wish to add a constructor $\texttt{generic} : N \to B$ to System T.

Now we need to extend evaluation of the System T, to handle the new constructor. For this purpose we will introduce an evaluation context which we call a condition set. It is a set of pairs of natural numbers and booleans, which will serve as a lookup table for the applications of the generic element.

So for example, if we have the condition set $\{(1, 0)\}$, then $\texttt{generic}(1)$ should evaluate to 0. Now we just need to consider what should happen if the argument to `generic` is not present in the condition set. Intuitively, if we want it to behave as a generic element, we should be able to inspect both branches of evaluation. We can handle this by splitting our condition set by adding the conditions $(n, 0)$ and $(n, 1)$ to each of the branches, and then continue to evaluate both branches. To keep track of branches, we can say that if we have $p \Vdash \texttt{generic}(n)$, and n is not in the condition set, evaluate the expression to a formal sum $p_0 \Vdash 0 + p_1 \Vdash 1$. In general, if p_i is some partition of p, and t evaluates to t_i on those partitions, then $p \Vdash t$ evaluates to $\sum_i p_i \Vdash t_i$.

1.4 The proof

The idea of the proof is as follows. Suppose we have a System T defined functional F. Then we evaluate it on the generic element in an empty context. By some lemmas we can show this has a normal form, which means it evaluates to some formal sum $\sum_i p_i \Vdash b_i$, where b_i are canonical elements of the type B. Now we can inspect the p_i and extract the largest natural number that occurs in any of the condition sets. This gives us the largest argument that the functional F can possibly evaluate, so it is the modulus of continuity for F.

2 Formalization of sheaf forcing

I have not yet found a satisfying solution to the issues at hand. The partitions have a natural tree structure, but if I define them like trees I can't index over the elements of the partition, which becomes a problem.

I expect to solve this to some extent in the coming weeks, but for now I can't write anything here.

2.1 System T

Currently I have implemented System T as an ordinary typed lambda calculus with two inductive types: the natural numbers and the booleans. One possible simplification is to define it as an SK calculus (with those same types), as is done in[1], but I personally prefer the lambda calculus. Currently the lambda calculus uses call-by-name semantics, though using fine-grained call-by-value semantics seem like a significant improvement. However, often the biggest issue in proving theorems is substitutions, and fine-grained call-by-value semantics don't help with that.

2.2 Cantor space

There is actually no definition of the Cantor space or its opens in the project. This is because we have extracted the essence of the continuity the space provides with regards to forcing, and now represent it with condition sets and their partitions. Currently these are defined as lists of number-boolean pairs and trees respectively.

The definition of condition sets is quite adequate, insofar as I have never had to explicitly manipulate them, but the definition of partitions is lacking. The tree structure imposes a tree structure to the computation of normal forms, which is probably undesirable. More importantly though, the tree structure does not expose any way to allow me to talk about for example terms t_i for every p_i in the partition. Further work is needed to verify whether it is possible to also encode the partitions as finite lists, and to find a way to reasonably iterate over them.

There is some concern that this will still produce inadequate results because lists are ordered, but this has not been an issue for condition sets, and I hope it will not be an issue for the partitions either.

3 Goals

In the beginning I had the following checkpoints in mind:

(1) Define the category of sheaves over $2^{\mathbb{N}}$

(2) Define sheaf forcing

(3) Define/interpret Gödel's System T

(4) Show that System T definable $2^{\mathbb{N}} \to \mathbb{N}$ functions are uniformly continuous

(5) Extend 1 and 2 to $\mathbb{N}^{\mathbb{N}}$

(6) Show the proof of 4 doesn't apply to 5

(7) Prove 4 for 5 in the non-uniform case

Trying to define the category is too difficult, but 3 didn't present any issues, though an alternative approach might have been better. As mentioned above, defining sheaf forcing is currently a significant issue, however I have written down the proof/algorithm that computes the modulus of continuity, assuming I can formulate forcing in such a way that allows me to compute normal forms.

Since I've already encountered significant difficulties and have spent a sufficient amount of time on the binary case I will not attempt to complete the three checkpoints.

Since 4 is mostly finished, up to renaming or relabeling of variables, I will set just the checkpoints 2, 3, and 4 as my goal. I sincerely hope to complete this by the defense date.

References

[1] Martín Escardó. "Continuity of Gödel s System T Definable Functionals via Effectful Forcing". In: *Electronic Notes in Theoretical Computer Science* 298 (Nov. 2013), pp. 119–141. ISSN: 1571-0661. DOI: 10.1016/j.entcs.2013.09.010. URL: http://dx.doi.org/10.1016/j.entcs.2013.09.010.

[2] Kurt Gödel. "Über eine bisher noch nicht benützte Erweiterung des finiten Standpunktes". In: *Dialectica*. 1958, pp. 280–287.