Sheaves as Oracle Computations

Andrej Bauer (University of Ljubljana)
Danel Ahman (University of Tartu)

11th International Conference on Formal Structures for Computation and Deduction
Lisbon, Portugal, 20–24 July 2026

1. Thank you for the invitation to talk at the conference. It has been a while since I visited FLoC, and it's good to be here.
2. My co-author Danel Ahman is sitting in the audience.
3. In this talk I would like to speak about ***sheaves***, a mathematical concept of foundational importance in geometry, algebra and logic, which makes rather few appearances in theoretical computer science.
4. Nevertheless, as we shall see, there is a fundamental connection between sheaves and computation that deserves attention and further investigation.

Talk outline

- **Abstract oracles & oracle modalities**
- Sheaves as oracle computations
- Non-propositional oracles
- Related work

1. The talk is divided into four parts.
2. First, we are going to give an account of oracles at an appropriate level of abstraction, namely type-theoretically and without reference to a specific model of computation. We shall see every such abstract oracle induces a modality.
3. Then we express oracle computations as sheaves and investigate the setup a bit.
4. In the third part we shall discuss how and why we want to extend the standard account of sheaves-as-modalities so that it will encompass also the so-called non-propositional oracles (which will be explained in due time).
5. Lastly, I wish do briefly discuss related work.

Definition 1. An ***(abstract) oracle*** $A \triangleleft P$ is specified by:

- a type A of ***queries***, and
- a predicate $P : A \to \mathsf{Prop}$ of ***answers***.

We often refer to an oracle $A \triangleleft P$ just by its predicate P.

1. An oracle is *specified* by two components: a type of queries, think of these as the questions which we can pose to the oracle; and for each query $a : A$ a predicate Pa of answers. We imagine that upon querying a, the oracle provides evidence, or assures us of the truth of Pa.
2. We shall see examples illuminating the definition shortly, but notice now that the oracle is only *specified*, not implemented. In fact, the whole point here is that a non-trivial oracle cannot be implemented!
3. The strange triangle notation $A \triangleleft P$ is often used to denote a *container*, a notion that first arose in modeling datatypes, but has since found many uses. We shall say more about containers a bit later, for now let's just see examples and get used to the notation.

Definition 1. An ***(abstract) oracle*** $A \lhd P$ is specified by:

- a type A of ***queries***, and
- a predicate $P : A \to \mathsf{Prop}$ of ***answers***.

Examples:

- Excluded middle: $\mathsf{Prop} \lhd \lambda p.\, p \vee \neg p$
- Limited principle of omniscience (LPO):
  $2^{\mathbb{N}} \lhd \lambda f.\, (\exists n.\, f\, n = 1) \vee (\forall n.\, f\, n = 0)$
- Traditional oracle $S \subseteq \mathbb{N}$: $\mathbb{N} \lhd \lambda n.\, n \in S \vee n \notin S$
- Uniform continuity principle:
  $2^{2^{\mathbb{N}}} \lhd \lambda f.\, \exists n{:}\mathbb{N}.\, \forall \alpha, \beta{:}2^{\mathbb{N}}.\, (\forall i < n.\, \alpha\, i = \beta\, i) \Rightarrow f\, \alpha = f\, \beta$
- Church's thesis: $\mathbb{N}^{\mathbb{N}} \lhd \lambda f.\, \exists n{:}\mathbb{N}.\, f = \varphi_n$

1. The most obvious example of an oracle is of course excluded middle. We hand the oracle a proposition p, it returns evidence of $p \vee \neg p$, i.e., it decides p.
2. We can devise special cases of excluded middle, such as LPO, as well as traditional oracles of computability theory, which decide membership in a subset of $\mathbb{N}$.
3. We can also consider anti-classical oracles, i.e., oracles which cannot exist in the presence of excluded middle. The uniform continuity principle oracle states that every decidable predicate on Cantor space is uniformly continuous. And Church's thesis states that every total map $\mathbb{N} \to \mathbb{N}$ is computable (here φ is a standard enumeration of partial computable maps).
4. The Church's thesis oracle is a bit unsatisfactory: we hand the oracle a function f, and in return we get a proof of *existence* of its code. It would be nicer if the oracle returned a number n, rather than a proof of its existence, so that we could use the witness n without restrictions. We shall address such *non-propositional* oracles in the third part of the talk.

What foundations are we using?

- An *agnostic* foundation which implements few oracles.
- We formalized our work in Rocq with:
  - function extensionality
  - propositional extensionality
  - uniqueness of proofs
  - equality is propositional
  - definite description (unique choice).
- We may replace Prop with quotient-inductive types.
- Some version of homotopy type theory ought to work, also.

1. Before we go on, let us stop to think about what foundational system we should be working with.
2. The important requirement is that the foundational system should *not* already implement too many oracles. In other words, it ought to be a purely intuitionistic foundation.
3. Suffice it to say that we formalized our work in Rocq's calculus of inductive constructions, using a number of additional principles that make it into a topos-like foundation.
4. I am sure there are alternatives, technicalities of which I am happy to discuss after the talk. Two quick observations: the impredicative propositions may be replaced with quotient-inductive types, and a suitable version of homotopy-type theory should suffice also.

"Proposition s holds relative to oracle $A \triangleleft P$"

- To establish $s : \mathsf{Prop}$ *relative to* $A \triangleleft P$ we may
  - prove s directly, or
  - query the oracle, receive an answer and proceed recursively.
- Define the proposition $\mathcal{O}_P s$ *inductively*:
  - $\mathtt{prf}\, u : \mathcal{O}_P s$ for $u : s$,
  - $\mathtt{ask}\, a\, \kappa : \mathcal{O}_P s$ for $a : A$ and $\kappa : Pa \to \mathcal{O}_P s$.

  Read $\mathcal{O}_P s$ as "s relative to $A \triangleleft P$".
- Impredicative encoding:

  $\mathcal{O}_P s := \forall r{:}\mathsf{Prop}.\, (s \Rightarrow r) \Rightarrow (\forall a{:}A.\, (Pa \Rightarrow r) \Rightarrow r) \Rightarrow r.$

1. Now, given an oracle $A \triangleleft P$, what does it mean for a statement s to hold *relative* to it?
2. Imagine a dialogue with the oracle: we may ask it a question, receive an answer, then proceed with the proof. Eventually we must stop asking questions and prove the statement s with accumulated knowledge.
3. What we are describing is an inductive definition. Writing $\mathcal{O}_P s$ for "s holds relative to $A \triangleleft P$", there are two clauses, corresponding to a direct proof of s, and asking a question. The κ in `ask` is the *continuation*.
4. The same construction can also be done by impredicative encoding. The formula which arises has been known at least since 1980's.

How *not* to reason relative to an oracle

- An oracle *implementation* is a map $f : \forall a{:}A.\, Pa$.
- Naive expectation; "*s relative to $A \lhd P$*" is $(\forall a{:}A.\, Pa) \Rightarrow s$.
- The statement is vacuous when $\neg\forall a{:}A.\, Pa$, but $\text{-}\!\circ_P s$ may still have computational content.
- Mathematical practice, both informal and formalized:
  - *state* $(\forall a{:}A.\, Pa) \Rightarrow s$,
  - but *prove* $\text{-}\!\circ_P s$.

  In fact, many proofs use *exactly one* oracle query.

1. Having seen how to reason relative to an oracle, allow me to address a common mistake.
2. First of all, what does it mean to "have" or "implement" an oracle? Well, given any query $a : A$, it is supposed to give an answer to Pa, so that is a proof, or map, $f : \forall a{:}A.\, Pa$.
3. Now one might expect that "s relative to the oracle P" should be expressed as an implication $(\forall a{:}A.\, Pa) \Rightarrow s$. But this is wrong, because in any model or situation that invalidates $\forall a{:}A.\, Pa$ the implication is vacuous.
4. For example, in the effective topos $\neg\mathsf{LPO}$ is valid because there are no realizers for the halting oracle. Consequently, the computational content of $\mathsf{LPO} \Rightarrow s$ is void. In contrast, a realizer for $\text{-}\!\circ_{\mathsf{LPO}} s$ turns out to be a (code of) an oracle Turing machine which realizers s when given access to the Halting oracle.
5. Unfortunately, such uses of implications are prevalent in formal and formalized mathematics. Fortunately, in most cases the statements claim implications but the proofs actually proceed by establishing the corresponding modal statement. We may classify the affair as a habitual inaccuracy of exposition rather than erroneous mathematical thinking.

$\mathcal{O}_P$: Prop $\longrightarrow$ Prop as a modality

- $\mathcal{O}_P$ is a monad on Prop:
  - monotone: $(s \Rightarrow r) \Rightarrow (\mathcal{O}_P s \Rightarrow \mathcal{O}_P r)$,
  - inflationary: $s \Rightarrow \mathcal{O}_P s$,
  - idempotent: $\mathcal{O}_P(\mathcal{O}_P s) \Leftrightarrow \mathcal{O}_P s$.

  We call such maps ***modalities***.

  (Other names: *Lawvere-Tierney topology*, *local operator*, and *nucleus*.)
- Facts:
  - Modalities form a complete Heyting algebra Mod.
  - Every modality gives rise to a notion of sheaves, and vice versa.

Theorem 2. *$\mathcal{O}_P$ is the least modality j such that $\forall a{:}A.\ j(Pa)$.*

1. Let us look at $\mathcal{O}_P$ as an operator on propositions. It is a monad. We shall call any map j : Prop $\longrightarrow$ Prop satisfying the listed properties a modality. Other names for the same concept are Lawvere-Tierney topology, local operator and nucleus.
2. Modalities form a complete Heyting algebra (a frame), and are one way of describing notions of sheaves. (The other one is in terms of Grothendieck topologies, or coverages.)
3. Say that a modality j forces P when $\forall a{:}A.\ j(Pa)$. Then $\mathcal{O}_P$ is the least modality forcing P. We may interpret this fact as assurance that our definition of "relative to an oracle" is correct.
4. Incidentally, the *open* modality $(\forall a{:}A.\ Pa) \Rightarrow -$ is *above* $\mathcal{O}_P$ in the lattice of modalities Mod.

Large-scale structure

Propositional containers form a category PCont:

- a ***morphism*** $f \lhd g : A \lhd P \longrightarrow B \lhd Q$ is given by $f : A \longrightarrow B$ and $g : \forall a{:}A.\, Q(f\,a) \Rightarrow Pa$.

Theorem 3. $\mathscr{o}$: PCont $\longrightarrow$ Mod *is a functor with a section* $\mathscr{p}$: Mod $\longrightarrow$ PCont.

Proof. Take $\mathscr{p}_j := (\Sigma(s{:}\mathsf{Prop}).\, j\,s) \lhd \lambda(s, \_).\, s$ and verify $\mathscr{o}_{\mathscr{p}_j} = j$. □

The oracle $\mathscr{p}_j$ forces j-dense propositions.

Corollary 4. *Every modality is an oracle modality.*

1. Propositional containers form a category. The definition of morphism shown here is standard in the study of containers. A morphism comprises two maps, one converting queries and the other the answers. Note that they map in opposite directions.
2. It is not hard to check that $\mathscr{o}$ is functorial. Even more, it has a section which shows that every modality arises as an oracle modality: given a modality j, the corresponding oracle is the one that forces j-dense propositions.

Examples

- The $\neg\neg$ modality is induced by both
  - excluded middle oracle: $\mathsf{Prop} \lhd \lambda p.\, p \vee \neg p$, and
  - $\neg\neg$-elimination oracle $p_{\neg\neg}$: $(\Sigma(s{:}\mathsf{Prop}).\, \neg\neg s) \lhd \lambda(s, \_).\, s$.
- Trivial cases:
  - If $A \lhd P$ has $a : A$ such that $\neg Pa$, then $\bigcirc_P s \Leftrightarrow \top$.
  - If $A \lhd P$ has an implementation $f : \forall a{:}A.\, Pa$, then $\bigcirc_P s \Leftrightarrow s$.

1. Several oracles may induce the same modality. For instance, the double negation modality is induced by both the excluded middle oracle and the double-negation-elimination oracle. In a specific situation we may prefer to work with one oracle rather than another.
2. Or to put it differently, we have devised a notion *representation of modalities*. A representation (the oracle) carries additional intensional information (queries and answers) that is not directly visible in the modality itself.
   This situation is analogous to the theory of algebraic operations and monads: a monad may have many presentations in terms of an algebraic theory.
   (Although, not every monad arises in this way, whereas every modality does arise as an oracle modality.)
3. Let us also observe how the two trivial modalities appear. If an oracle can be asked an impossible question, one that does not have an answer, then the induced modality is the top one. And if an oracle can be implemented then the corresponding modality is the identity.

Talk outline

- Abstract oracles & oracle modalities
- **Sheaves as oracle computations**
- Non-propositional oracles
- Related work

1. Let us now proceed to the second part of the talk.
2. So far we thought about what it means to *prove* a logical statement relative to an oracle, but now we ask what it means to *construct* or *compute* relative to an oracle.

Sheaves as oracle computations

Definition 5. For $A \lhd P$, a P-***algebra*** (X, h) is given by:

- a carrier X,
- a structure map $h : \Pi(a{:}A).\, (Pa \to X) \to X$,
- satisfying equations:
  - $ha\kappa = \kappa u$ for all $a : A$, $\kappa : Pa \to X$ and $u : Pa$,
  - $(Pa \Rightarrow x = y) \Rightarrow x = y$ for all $a : A$ and $x, y : X$.

Theorem 6. *P-algebras and $\neg\neg_P$-sheaves are equivalent categories.*

1. Every modality j determines a notion of j-sheaves, which can be described in several different ways. We would like a description that captures the idea that a construction or a computation of an element may be carried out relative to an oracle.
2. Given an oracle $A \lhd P$, the definition of a P-algebra is precisely a notion of "computing with a P-oracle". Think of the structure map h as an oracle implementation (and ask not where it comes from, we are only *describing* a concept, not implementing it): $ha\kappa$ *handles* the query a and a given continuation κ.
3. We impose two equations. The first one says that we may skip a question a if we already know an answer $u : Pa$. The second one says that we may assume answers to be available when proving equalities.
4. This notion of P-algebra coincides with the standard notion of an $\neg\neg_P$-sheaf.
5. You might wonder where such an algebra could come from, especially since in a typical situation an oracle cannot be implemented. The idea is not to try to implement h as the oracle, but to enrich the structure of the carrier X. For example, in the paper we show that $X := (\forall a{:}A.\, Pa) \Rightarrow Y$ is a P-algebra for any Y. Indeed, in this case h just uses the reader monad structure on X to read off the answers to queries.

Sheafification

Definition 7. The ***P*-sheafification** of X is the quotient-inductive type $\mathsf{shf}_P X$, generated by the following constructors and equations:

- $\mathsf{ret} : X \to \mathsf{shf}_P X$,
- $\mathsf{ask} : \Pi(a{:}A).\, (Pa \to \mathsf{shf}_P X) \to \mathsf{shf}_P X$,
- $\mathsf{ask}\, a\, \kappa = \kappa\, u$, for all $a : A$, $\kappa : Pa \to \mathsf{shf}_P X$, and $u : Pa$, and
- if $Pa \Longrightarrow v = w$ then $v = w$, for all $a : A$ and $v, w : \mathsf{shf}_P X$.

Intuition:

- Elements of $\mathsf{shf}_P X$ are "P-computable elements of X".
- Maps $X \to \mathsf{shf}_P Y$ are "P-computable maps $X \to Y$".

1. We have found that oracle modalities capture the notion of "proving relative to an oracle". How about *compupting* or *constructing* an element of type X relative to an oracle $A \lhd P$?
2. The answer is given by *sheafification*, which turns a type X into a type $\mathsf{shf}_P X$, the free P-algebra generated by X. The construction shown here uses a quotient-inductive type. (The constructors ret and ask generate the type, and *at the same* quotienting enforces the equations.) In the paper we review an equivalent construction by Barbara Veit that relies on the impredicativity of Prop.
3. Think of an element of $\mathsf{shf}_P X$ as "an element of X constructed relative to the oracle P", and of a map $X \to \mathsf{shf}_P[Y]$ as a "P-computable map". The intuition is backed up by the fact that in a realizability model these really are the oracle-computable maps.

Example: the intermediate value theorem

Let lem be the excluded middle oracle and $f : [0,1] \to \mathbb{R}$ continuous such that $f\,0 < 0 < f\,1$.

Theorem 8. *We construct* $x : \mathsf{shf}_{\mathsf{lem}}\,[0,1]$ *such that* $f\,x = 0$.

Proof. Use the bisection method: let $[a_0, b_0] := [0,1]$, and given $[a_n, b_n]$ let $m := (a_n + b_n)/2$ and

$$[a_{n+1}, b_{n+1}] \;:=\; \texttt{if}\; f\,m < 0\; \texttt{then}\; [m, b_n]\; \texttt{else}\; [a_n, m],$$

where the oracle is asked to decide $f\,m < 0$. Return $x := \lim_n a_n = \lim_n b_n$. □

Corollary 9. *If* $f\,q < 0 \vee f\,q > 0$ *for all* $q \in \mathbb{Q}$, *then* f *has a zero.*

Proof. We may *handle* the oracle queries in x constructed above because the midpoints appearing in the bisection method are rational. □

1. I would like to take you through one substantial example: the intermediate value theorem. Warning: this slide is not formalized yet.
2. The theorem states that a continuous real-valued map $f : [0,1] \to \mathbb{R}$ has a zero if it is negative at 0 and positive at 1. It cannot be proved constructively. We can constructs a zero relative to the excluded middle oracle. Thus, the number x so constructed is an element of the sheafification $\mathsf{shf}\,\mathsf{lem}[0,1]$, rather than $[0,1]$. The theorem statement hides some details, for example, we cannot literally apply f to x, but must first lift f to a map $\mathsf{shf}\,\mathsf{lem}[0,1] \to \mathsf{shf}\,\mathsf{lem}\mathbb{R}$.
3. The proof proceeds by the usual bisection method, where we use the oracle to decide whether the $f\,m < 0$. The proof is written in a very compressed form here, and would deserve to be spelled out more carefully. Anyhow, I hope the idea is clear.
4. Now suppose additionally that f has no rational zeros, or more precisely, f is negative or positive at rational arguments. Because in the bisection method with started with rational endpoints 0 and 1, at all stages the midpoints is rational, so we can resolve the uses of oracle in x *after* having already constructed it. We could do no such thing if we just assumed excluded middle.

Talk outline

- Abstract oracles & oracle modalities
- Sheaves as oracle computations
- **Non-propositional oracles**
- Related work

Non-propositional oracles

Definition 10. A ***non-propositional oracle*** $A \lhd P$ is a type family $P : A \to \mathsf{Type}$. The associated propositional oracle $\|P\| : A \to \mathsf{Prop}$ is $\|P\|\, a := \exists u{:}Pa.\ \top$.

Examples:

- Church's thesis oracle:
  - non-propositional: $\mathbb{N}^{\mathbb{N}} \lhd \lambda f.\ \Sigma(n{:}\mathbb{N}).\ f = \varphi_n$.
  - propositional: $\mathbb{N}^{\mathbb{N}} \lhd \lambda f.\ \exists n{:}\mathbb{N}.\ f = \varphi_n$.
- Choice in Mathlib: $(\Sigma(X{:}\mathsf{Type}).\ \exists x{:}X.\ \top) \lhd \lambda(X, \_).\ X$.

1. We have so far considered *propositional* oracles, i.e., those that provide evidence of logical statements. But as we already saw earlier, in some situation we would like oracles that provide *elements* of types. For this purpose we may define a *non-propositional* oracle to be a type family, rather than a predicate.
2. To illustrate the difference, consider the earlier example of Church's thesis oracle: the propositional version assures us of the mere *existence* of (the code of) a machine computing f, whereas the non-propositional one provides an actual (code of) a machine doing so. The non-propositional one is of course more convenient, but it also breaks the standard theory of sheaves.
3. An important example of a non-propositional oracle is the *choice oracle* which, given an inhabited set A, gives an element of it. This is how the axiom of choice is formalized in the Mathlib library, for example.

Equifoliate trees

- Let $P : A \to \mathsf{Type}$ be a non-propositional oracle.
- Define $\mathcal{T}_P X$, P-branching trees with leaves X, inductively:
  - $\mathsf{leaf}\, x : \mathcal{T}_P X$ for any $x : X$,
  - $\mathsf{node}\, a\, \kappa : \mathcal{T}_P X$ for any $a : A$, $\kappa : Pa \to \mathcal{T}_P X$.
- A tree $t : \mathcal{T}_P X$ is ***equifoliate*** if all its leaves are $\mathcal{O}_{\|P\|}$-equal.
- Let $\mathcal{E}_P X := \Sigma(t : \mathcal{T}_P X).\ \mathsf{equifoliate}\, t$.

Theorem 11.

1. *The tree monad $\mathcal{T}_P$ restricts to $\mathcal{E}_P$.*
2. *There is a map $\delta_X : \mathcal{E}_P X \to \mathsf{shf}_{\|P\|}\, X$ which is modally surjective*: $\forall u : \mathsf{shf}_{\|P\|}\, X.\ \mathcal{O}_{\|P\|}(\exists t : \mathcal{E}_P X.\ \delta_X t = u)$.

1. If non-propositional oracles cannot be directly incorporated into the standard theory of modalities and sheaves, we may perhaps still get some mileage out of them, for example, by providing a notion of computation trees that avoids quotienting by the sheaf equations. Such trees can then be inspected and manipulated with ease.
2. However, one cannot expect arbitrary trees $\mathcal{T}_P X$ arising from a non-propositional oracle to be useful here. Since a non-propositional oracle may return *many* answers to a query, such trees may carry different elements in their leaves, whereas in the case of propositional oracles the trees are list-like.
3. Be that as it may, we may cut down to *equifoliate trees*, which behave better. See the paper for details and further discussion.

Talk outline

- Abstract oracles & oracle modalities
- Sheaves as oracle computations
- Non-propositional oracles
- **Related work**

Related work

- E. Rijke, M. Shulman, B. Spitters: Modalities in homotopy type theory, Logical Methods in Computer Science, January 8, 2020, Volume 16, Issue 1
- A. Swan: Oracle modalities, June 2024, arXiv:2406.05818
- T. Kihara: Lawvere-Tierney topologies for computability theorists, Transactions of the American Mathematical Society, Series B, Volume 10, Issue 02, January 2023
- M. Baillon, A. Mahboubi, P.-M. Pédrot: In Cantor Space No One Can Hear You Stream, Programming Languages and Systems – 35th European Symposium on Programming (ESOP), April 2026, Turin (Italy)
- D. Ahman, A. Bauer, S. Park: *Setoids of Oracle Computations in Constructive Type Theory*, Continuity, Computability, Constructivity: From Logic to Algorithms, May 2026, Kyoto (Japan)
- C. Park, S. Park: *Computing Noncomputables with Oracle Computation Trees*, Continuity, Computability, Constructivity: From Logic to Algorithms, May 2026, Kyoto (Japan)

1. There is quite a bit of related work. The paper on modalities by Rijke, Shulman and Spitters can be recommended as a comprehensive treatment of modalities in type theory.
2. To see how oracle modalities can be used in sophisticated ways, consult Andrew Swan's work. It was great inspiration for us, and we learned a great deal from Andrew.
3. Takayuki Kihara gave a game-theoretic account of Lawvere-Tierney topologies in realizability toposes. Our notion of oracle modalities, interpreted in a realizability topos, agrees with his, see the paper for details.
4. We would also like to mention independent work on the same topic by Baillon, Mahboubi and Pédrot. They also construct sheafification as a quotient-inductive type, except they only have one equation from which both our equations follow. Their is better and we shall use it in the future.
5. Lastly, Sewon Park with coauthors has explored how to use oracle modalities to extract computational content from classical proofs. In particular, with C. Park he formalized the intermediate value theorem using oracle trees, and computed π as the zero of sin on $[3, 4]$, relying on the fact that sin has no rational zeros on $[3, 4]$.

Ideas for further work

1. Use oracle modalities in reverse constructive mathematics.
2. Use oracle modalities in formalized mathematics.
3. Develop *intensional sheaves* for non-propositional oracles.

1. Thank you for your attention.