Sheaf semantics of termination-insensitive noninterference

Jonathan Sterling
Department of Computer Science, Aarhus University, Denmark

Robert Harper
Computer Science Department, Carnegie Mellon University, United States of America

Abstract

We propose a new *sheaf semantics* for secure information flow over a space of abstract behaviors, based on synthetic domain theory: security classes are open/closed partitions, types are sheaves, and redaction of sensitive information corresponds to restricting a sheaf to a closed subspace. Our security-aware computational model satisfies termination-insensitive noninterference automatically, and therefore constitutes an intrinsic alternative to state of the art extrinsic/relational models of noninterference. Our semantics is the latest application of Sterling and Harper's recent re-interpretation of *phase distinctions* and noninterference in programming languages in terms of Artin gluing and topos-theoretic open/closed modalities. Prior applications include parametricity for ML modules, the proof of normalization for cubical type theory by Sterling and Angiuli, and the cost-aware logical framework of Niu *et al.* In this paper we employ the phase distinction perspective *twice*: first to reconstruct the syntax and semantics of secure information flow as a lattice of phase distinctions between "higher" and "lower" security, and second to verify the computational adequacy of our sheaf semantics with respect to a version of Abadi *et al.*'s *dependency core calculus* to which we have added a construct for declassifying termination channels.

2012 ACM Subject Classification Theory of computation → Abstraction; Theory of computation → Denotational semantics; Theory of computation → Categorical semantics; Theory of computation → Type theory; Security and privacy → Formal methods and theory of security

Keywords and phrases information flow, noninterference, denotational semantics, phase distinction, Artin gluing, modal type theory, topos theory, synthetic domain theory, synthetic Tait computability

Digital Object Identifier 10.4230/LIPIcs.FSCD.2022.15

Related Version *Extended Version*: `https://arxiv.org/abs/2204.09421`

Funding *Jonathan Sterling:* This work was supported by a Villum Investigator grant (no. 25804), Center for Basic Research in Program Verification (CPV), from the VILLUM Foundation. *Robert Harper:* This research was sponsored by the United States Air Force Office of Scientific Research awards FA95502110009 and FA9550-21-1-0385 (Tristan Nguyen, program manager). The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government or any other entity.

Acknowledgements We are grateful for insightful conversations with Aslan Askarov, Stephanie Balzer, Lars Birkedal, Martín Escardó, Marcelo Fiore, Daniel Gratzer, and Tom de Jong. Thanks to Jamie Vicary for funding a visit to Cambridge during which the first author learned some of the tools needed to complete this work satisfactorily. We thank Carlos Tomé Cortiñas, Fabian Ruch, and Sandro Stucki for proof-reading.

1 Introduction

Security-typed languages restrict the ways that classified information can flow from high-security to low-security clients. Abadi *et al.* [1] pioneered the use of *idempotent monads* to deliver this restriction in their ***dependency core calculus*** (DCC), parameterized in a poset of security levels $\mathcal{P}$. Covariantly in security levels $l \in \mathcal{P}$, a family of type operations $\mathsf{T}_l A$ satisfying the rules of an idempotent monad are added to the language; the idea is then that sensitive data can be hidden underneath T_l and unlocked only by a client with a type that can be equipped with a T_l-algebra structure, *i.e.* a ***$\langle l \rangle$-sealed type*** in our terminology.[1] For instance, a high-security client can read a medium-security bit:

$$\begin{aligned}&\mathsf{f} : \mathsf{T_M bool} \to \mathsf{T_H bool}\\ &\mathsf{f}\, u = x \leftarrow u; \mathsf{seal_H}(\mathsf{not}\, x)\end{aligned}$$

There is however no corresponding program of type $\mathsf{T_H bool} \to \mathsf{T_M bool}$, because the type $\mathsf{T_M bool}$ of medium-security booleans is not $\langle \mathsf{H} \rangle$-sealed, *i.e.* it cannot be equipped with the structure of a $\mathsf{T_H}$-algebra. In fact, up to observational equivalence it is possible to state a ***noninterference result*** that fully characterizes such programs:

▶ **Proposition** (Noninterference). *For any closed function* $\cdot \vdash \mathsf{f} : \mathsf{T_H bool} \to \mathsf{T_M bool}$*, there exists a closed* $\cdot \vdash b : \mathsf{T_M bool}$ *such that* $\mathsf{f} \simeq \lambda\_.b$*.*

Intuitively the noninterference result above follows because you cannot "escape" the monad, but to prove such a result rigorously a model construction is needed. Today the state of the art is to employ a ***relational model*** in the sense of Reynolds in which a type is interpreted as a binary relation on some domain, and a term is interpreted by a relation-preserving function. Our contribution is to introduce an ***intrinsic*** and ***non-relational semantics*** of noninterference presenting several advantages that we will argue for, inspired by the recent modal reconstruction of *phase distinctions* by Sterling and Harper [50].

1.1 Termination-insensitivity and the meaning of "observation"

▶ **Notation 1.** We will write $\mathsf{L}A$ for the lifting monad that Abadi *et al.* notate $A_\perp$.

When we speak of noninterference up to observational equivalence, much weight is carried by the choice of what, in fact, counts as an observation. In a functional language with general recursion, it is conventional to say that an observation is given by a computation of unit type — which necessarily either diverges or converges with the unique return value (). Under this notion of observation, noninterference up to observations takes a very strong character:

Termination-sensitive noninterference. For a closed partial function $\cdot \vdash \mathsf{f} : \mathsf{T_H bool} \to \mathsf{L}\,(\mathsf{T_M bool})$, either $\mathsf{f} \simeq \lambda\_.\perp$ or there exists $\cdot \vdash b : \mathsf{T_M bool}$ such that $\mathsf{f} \simeq \lambda\_.b$.

If on the other hand we restrict observations to only terminating computations of type bool, we evince a more relaxed ***termination-insensitive*** version of noninterference that allows leakage through the termination channel but *not* through the "return channel":

Termination-insensitive noninterference. For a closed partial function $\cdot \vdash \mathsf{f} : \mathsf{T_H bool} \to \mathsf{L}\,(\mathsf{T_M bool})$, given any closed u, v on which f terminates, we have $f u \simeq f v$.

[1] We use the term "sealing" for what Abadi *et al.* [1] call "protection"; to avoid confusion, we impose a uniform terminology to encompass both our work and that of *op. cit.* A final notational deviation on our part is that we will distinguish a security level $l \in \mathcal{P}$ from the corresponding syntactical entity $\langle l \rangle$.

1.2 Relational *vs.* intrinsic semantics

To verify the noninterference property for the dependency core calculus, Abadi *et al.* [1] define a ***relational semantics*** that starts from an insecure model of computation (domain theory *qua* dcpos) and restricts it by means of binary relations indexed in security levels that express the indistinguishability of sensitive bits to low-security clients. The indistinguishability relations are required to be preserved by all functions, ensuring the security properties of the model. The relational approach has an extrinsic flavor, being characterized by the *post hoc* imposition of order (noninterference) on an inherently disordered computational model. We contrast the extrinsic relational semantics of *op. cit.* with an *intrinsic* denotational semantics in which the underlying computational model has security concerns "built-in" from the start.

1.3 Our contribution: intrinsic semantics of noninterference

The main contribution of our paper is to develop an *intrinsic semantics* in the sense of Section 1.2, in which termination-insensitive noninterference (Section 1.1) is not bolted on but rather arises directly from the underlying computational model. To summarize our approach, instead of controlling the security properties of ordinary dcpos using a $\mathcal{P}$-indexed logical relation, we take semantics in a category of $\mathcal{P}$-indexed dcpos, *i.e.* sheaves of dcpos on a space $\mathbf{P}$ in which each security level $l \in \mathcal{P}$ corresponds to an open/closed partition. Employing the viewpoint of Sterling and Harper [50], each of these partitions induces a ***phase distinction*** between data visible below security level l (open) and data that is hidden (closed), leading to a novel account of the sealing monad T_l as restriction to a closed subspace.

Our intrinsic semantics has several advantages over the relational approach. Firstly, termination-insensitive noninterference arises directly from our computational model. Secondly, our model of secure information flow contributes to the consolidation and unification of ideas in programming languages by treating general recursion and security typing as instances of two orthogonal and well-established notions, namely ***axiomatic & synthetic domain theory*** and ***phase distinctions***/***Artin gluing*** respectively. Termination-insensitivity then arises from the non-trivial interaction between these orthogonal layers.

In particular, our computational model is an instance of axiomatic domain theory in the sense of Fiore [10], and embeds into a sheaf model of synthetic domain theory [14, 9, 12, 13, 11, 15, 30]. Hence the interpretation of the PCF fragment of DCC is interpreted exactly as in the standard Plotkin semantics of general recursion in categories of partial maps, in contrast to the relational model of Abadi *et al.* Lastly, the view of security levels as phase distinctions per Sterling and Harper [50] advances a uniform perspective on noninterference scenarios that has already proved fruitful for resolving several problems in programming languages:

1. A generalized abstraction theorem for ML modules with strong sums [50].
2. Normalization and decidability of type checking for cubical type theory [49, 48] and multi-modal type theory [17]; guarded canonicity for guarded dependent type theory [18].
3. The design and metatheory of the **calf** logical framework [31] for simultaneously verifying the correctness and complexity of functional programs.

The final benefit of the phase distinction perspective is that logical relations arguments can be re-cast as imposing an *additional* orthogonal phase distinction between *syntax* and *logic/specification*, an insight originally due to Peter Freyd in his analysis of the existence and disjunction properties in terms of Artin gluing [16]. We employ this insight in the present paper to develop a uniform treatment of our denotational semantics and its computational adequacy in terms of phase distinctions.

2 Background: relational semantics of noninterference

To establish noninterference for the dependency core calculus, Abadi *et al.* [1] define a relational model of their monadic language in which each type A is interpreted as a dcpo $|A|$ equipped with a family of admissible binary relations R_l^A indexed in security levels $l \in \mathcal{P}$. In the relational semantics, a term $\Gamma \vdash M : A$ is interpreted as a continuous function $|M| : |\Gamma| \longrightarrow |A|$ such that for all $l \in \mathcal{P}$, if $\gamma \; R_l^\Gamma \; \gamma'$ then $|M|\gamma \; R_l^A \; |M|\gamma'$.

▶ Remark 2. Two elements $u, v \in A$ such that $u \; R_l^A \; v$ have been called *equivalent* in subsequent literature, but this terminology may lead to confusion as there is nothing forcing the relation to be transitive, nor even symmetric nor reflexive.

The essence of the relational model is to impose *relations* between elements that should not be distinguishable by a certain security class; a type like bool or string whose relation is totally discrete, then, allows any security class to distinguish all distinct elements. Non-discrete types enter the picture through the sealing modality T_l:

$$|\mathsf{T}_l A| = |A| \qquad u \; R_k^{\mathsf{T}_l A} \; v \iff \begin{cases} u \; R_k^A \; v & \text{if } l \sqsubseteq k \\ \top & \text{otherwise} \end{cases}$$

Under this interpretation, the denotation of a function $\mathsf{T_H bool} \to \mathsf{T_M bool}$ must be a constant function, as $u \; R_{\mathsf{H}}^{\mathsf{bool}} \; v$ if and only iff $u = v$. By proving computational adequacy for this denotational semantics, one obtains the analogous *syntactic* noninterference result up to observational equivalence.

Generalization and representation of relational semantics. The relations imposed on each type give rise to a form of cohesion in the sense of Lawvere [28], where elements that are related are thought of as "stuck together". Then noninterference arises from the behavior of maps from a relatively codiscrete space into a relatively discrete space, as pointed out by Kavvos [25] in his *tour de force* generalization of the relational account of noninterference in terms of axiomatic cohesion. Another way to understand the relational account is by *representation*, as attempted by Tse and Zdancewic [54] and executed by Bowman and Ahmed [6]: one may embed DCC into a polymorphic lambda calculus in which the security abstraction is implemented by *actual* type abstraction.

Adapting the relational semantics for termination-insensitivity

In the relational semantics of the dependency core calculus, the termination-sensitive version of noninterference is achieved by interpreting the *lift* of a type in the following way:

$$|A_\perp| = |A|_\perp \qquad u \; R_l^{A_\perp} \; v \iff (u, v{\downarrow} \wedge u \; R_l^A \; v) \vee (u = v = \perp)$$

To adapt the relational semantics for termination-insensitivity, Abadi *et al.* change the interpretation of lifts to identify *all* elements with the bottom element:

$$|A_\perp| = |A|_\perp \qquad u \; R_l^{A_\perp} \; v \iff (u, v{\downarrow} \wedge u \; R_l^A \; v) \vee (u = \perp) \vee (v = \perp)$$

That all data is "indistinguishable" from the non-terminating computation means that the indistinguishability relation cannot be both transitive and non-trivial, a somewhat surprising state of affairs that leads to our critique of relational semantics for information flow below and motivates our new perspective based on the analogy between *phase distinctions* in programming languages and *open/closed partitions* in topological spaces [50].

Critique of relational semantics for information flow

From our perspective there are several problems with the relational semantics of Abadi *et al.* [1] that, while not fatal on their own, inspire us to search for an alternative perspective.

Failure of monotonicity. First of all, within the context of the relational semantics it would be appropriate to say that an object A is $\langle l \rangle$-sealed when $A \longrightarrow \mathsf{T}_l A$ is an isomorphism. But in the semantics of Abadi *et al.*, it is not necessarily the case that a $\langle l \rangle$-sealed object is $\langle k \rangle$-sealed when $k \sqsubseteq l$. It is true that objects that are *definable* in the dependency core calculus are better behaved, but in proper denotational semantics one is not concerned with the image of an interpretation function but rather with the entire category.

Failure of transitivity. A more significant and harder to resolve problem is the fact that the indistinguishability relation R_l^A assigned to each type cannot be construed as an equivalence relation — despite the fact that in real life, indistinguishability is indeed reflexive, symmetric, and transitive. As we have pointed out, the adaptation of DCC's relational semantics for termination-insensitivity is evidently incompatible with using (total or partial) equivalence relations to model indistinguishability, as transitivity would ensure that no two elements of $A_\perp$ can be distinguished from another.

Where is the dominance? Conventionally the denotational semantics for a language with general recursion begins by choosing a category of "predomains" and then identifying a notion of *partial map* between them that evinces a *dominance* [10, 42]. It is unclear in what sense the DCC's relational semantics reflects this hard-won arrangement; as we have seen, the adaptation of the relational semantics for termination-insensitivity further increases the distance from ordinary domain-theoretic semantics.

Perspective. Abadi *et al.*'s relational semantics is based on imposing secure information flow properties on an existing insecure model of partial computation, but this is quite distinct from an *intrinsic denotational semantics* for secure information flow — which would necessarily entail new notions of predomain and partial map that are sensitive to security from the start. In this paper we report on such an intrinsic semantics for secure information flow in which termination-insensitive noninterference arises inexorably from the chosen dominance.

3 Central ideas of this paper

In this section, we dive a little deeper into several of the main concepts that substantiate the contributions of this paper. We begin by fixing a poset $\mathcal{P}$ of security levels closed under finite meets, for example $\mathcal{P} = \{\mathsf{L} \sqsubset \mathsf{M} \sqsubset \mathsf{H} \sqsubset \top\}$. The purpose of including a security level even higher than H will become apparent when we explain the meaning of the sealing monad T_l.

▶ **Notation 3.** Given a space $\mathbf{X}$ and an open set $U \in \mathcal{O}_{\mathbf{X}}$, we will write $\mathbf{X}_{/U}$ for the open subspace spanned by U and $\mathbf{X}_{\bullet U}$ for the corresponding complementary closed subspace. We also will write $\mathcal{S}_{\mathbf{X}}$ for the category of sheaves on the space $\mathbf{X}$.

3.1 A space of abstract behaviors and security policies

We begin by transforming the security poset $\mathcal{P}$ into a topological space $\mathbf{P}$ of "abstract behaviors" whose algebra of open sets $\mathcal{O}_{\mathbf{P}}$ can be thought of as a lattice of *security policies* that govern whether a given behavior is permitted.

▶ **Definition 4.** *An* ***abstract behavior*** *is a* filter *on the poset* $\mathcal{P}$*,* i.e. *a monotone subset* $x \subseteq \mathcal{P}$ *such that* $\bigwedge_{i<n} l_i \in x$ *if and only if each* $l_i \in x$*.*

▶ **Definition 5.** *A* ***security policy*** *is a lower set in* $\mathcal{P}$, i.e. *an antitone subset* $U \subseteq \mathcal{P}$*. We will write* $U \Vdash x$ *to mean* U *permits the behavior* x*,* i.e. *the subset* $x \cap U$ *is inhabited.*

An abstract behavior x denotes the set of security levels $l \in \mathcal{P}$ at which it is permitted; a security policy U denotes the set of security levels *above which* some behavior is permitted.

▶ **Construction 6.** We define $\mathbf{P}$ to be the topological space whose points are abstract behaviors, and whose open sets are of the form $\{x \mid U \Vdash x\}$ for some security policy U.[2]

We have a meet-preserving embedding of posets $\langle - \rangle : \mathcal{P} \hookrightarrow \mathcal{O}_{\mathbf{P}}$ that exhibits $\mathcal{O}_{\mathbf{P}}$ as the free completion of $\mathcal{P}$ under joins, or equivalently the free frame on the meet semi-lattice $\mathcal{P}$.

▶ **Intuition 7** (Open and closed subspaces). Each security level $l \in \mathcal{P}$ represents a security policy $\langle l \rangle \in \mathcal{O}_{\mathbf{P}}$ whose corresponding open subspace $\mathbf{P}_{/\langle l \rangle}$ is spanned by the behaviors *permitted* at security levels l and above. Conversely the complementary closed subspace $\mathbf{P}_{\bullet\langle l \rangle} = \mathbf{P} \setminus \mathbf{P}_{/\langle l \rangle}$ is spanned by behaviors that are *forbidden* at security level l and below.

3.2 Sheaves on the space of abstract behaviors

Our intention is to interpret each type of a dependency core calculus as a *sheaf* on the space $\mathbf{P}$ of abstract behaviors. To see why this interpretation is plausible as a basis for secure information flow, we note that a sheaf on $\mathbf{P}$ is the same thing as a presheaf on the poset $\mathcal{P}$, *i.e.* a family of sets $(A_l)_{l \in \mathcal{P}}$ indexed contravariantly in $\mathcal{P}$ in the sense that for $k \sqsubseteq l$ there is a chosen restriction function $A_l \to A_k$ satisfying two laws. Hence a sheaf on $\mathbf{P}$ determines (1) for each security level $l \in \mathcal{P}$ a choice of what data is visible under the security policy $\langle l \rangle$, and (2) a way to *redact* data as it passes under a more restrictive security policy $\langle k \rangle \subseteq \langle l \rangle$.

3.3 Transparency and sealing from open and closed subspaces

For any subspace $\mathbf{Q} \subseteq \mathbf{P}$, a sheaf $A \in \mathcal{S}_{\mathbf{P}}$ can be restricted to $\mathbf{Q}$, and then extended again to $\mathbf{P}$. This composite operation gives rise to an *idempotent monad* on $\mathcal{S}_{\mathbf{P}}$ that has the effect of purging any data from $A \in \mathcal{S}_{\mathbf{P}}$ that cannot be seen from the perspective of $\mathbf{Q}$. The idempotent monads corresponding to the open and closed subspaces induced by a security level $l \in \mathcal{P}$ are named and notated as follows:

1. The ***transparency monad*** $A \mapsto (\langle l \rangle \Rightarrow A)$ replaces A with whatever part of it can be viewed under the policy $\langle l \rangle$. The transparency monad is the function space $A^{\langle l \rangle}$, recalling that an open set of $\mathbf{P}$ is the same as a subterminal sheaf. When the unit is an isomorphism at A, we say that A is $\langle l \rangle$***-transparent***.
2. The ***sealing monad*** $A \mapsto (\langle l \rangle \bullet A)$ removes from A whatever part of it can be viewed under the policy $\langle l \rangle$. The sealing monad can be constructed as the pushout $\langle l \rangle \sqcup_{\langle l \rangle \times A} A$. When the unit is an isomorphism at A, we say that A is $\langle l \rangle$***-sealed***.

The transparency and sealing monads interact in two special ways, which can be made apparent by appealing to the visualization of their behavior that we present in Figure 1.

1. The $\langle l \rangle$-transparent part of a $\langle l \rangle$-sealed sheaf is trivial, *i.e.* we have $(\langle l \rangle \Rightarrow (\langle l \rangle \bullet A)) \cong \{\bigstar\}$.
2. Any sheaf $A \in \mathcal{S}_{\mathbf{P}}$ can be reconstructed as the fiber product $(\langle l \rangle \Rightarrow A) \times_{\langle l \rangle \bullet (\langle l \rangle \Rightarrow A)} \langle l \rangle \bullet A$.

[2] Those familiar with the point-free topology of topoi [23, 55, 2] will recognize that $\mathbf{P}$ is more simply described as the presheaf topos $\widehat{\mathcal{P}}$: viewed as a space, it is the dcpo completion of $\mathcal{P}^{\mathsf{op}}$, and as a frame it is the free cocompletion of $\mathcal{P}$. The definition of $U \Vdash x$ then presents a computation of the *stalk* U_x of the subterminal sheaf $U \in \mathcal{S}_{\mathbf{P}}$ at the behavior $x \in \mathbf{P}$.

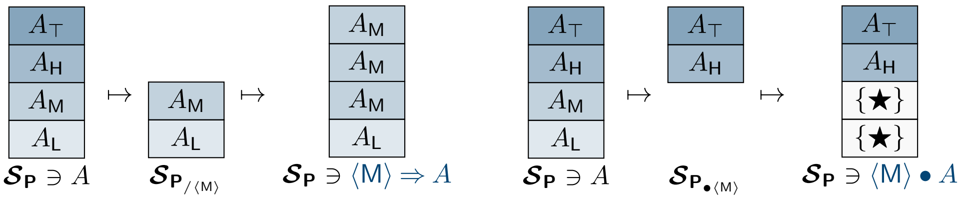

Figure 1 The transparency and sealing monads for $\mathsf{M} \in \mathcal{P}$ on a sheaf $A \in \mathcal{S}_\mathsf{P}$ visualized.

The first property above immediately gives rise to a form of noninterference, which justifies our intent to interpret DCC's sealing monad as $\mathsf{T}_l A = \langle l \rangle \bullet A$.

▶ **Observation 8** (Noninterference)**.** *Any map $\langle l \rangle \bullet A \longrightarrow \mathsf{bool}$ is constant.*

Proof. We may verify that the boolean sheaf bool is $\langle l \rangle$-transparent for all $l \in \mathcal{P}$. ◀

Our sealing monad above is well-known to the type-and-topos–theoretic community as the ***closed modality*** [41, 43, 3] corresponding to the open set $\langle l \rangle \in \mathcal{O}_\mathsf{P}$. In the context of (total) dependent type theory, our sealing monad has excellent properties not shared by those of Abadi *et al.* [1], such as justifying *dependent* elimination rules and commuting with identity types. In contrast to the *classified sets* of Kavvos [25] which cannot form a topos, our account of information flow is compatible with the full internal language of a topos.

3.4 Recursion and termination-insensitivity via sheaves of domains

To incorporate recursion into our sheaf semantics of information flow, in this section we consider *internal dcpos* in $\mathcal{S}_\mathsf{P}$, *i.e.* sheaves of dcpos. Later in the technical development of our paper, we work in the axiomatic setting of synthetic domain theory, but all the necessary intuitions can also be understood concretely in terms of dcpos. Domain theory internal to $\mathcal{S}_\mathsf{P}$ works very similarly to classical domain theory, but it must be developed without appealing to the law of the excluded middle or the axiom of choice as these do not hold in $\mathcal{S}_\mathsf{P}$ except for a particularly degenerate security poset. De Jong and Escardó [8] explain how to set up the basics of domain theory in a suitably constructive manner, which we will not review.

The sheaf-theoretic domain semantics sketched above leads immediately to a new and simplified account of termination-insensitivity. It is instructive to consider whether there is an analogue to Observation 8 for partial continuous functions $\langle l \rangle \bullet A \longrightarrow \mathsf{L}\,\mathsf{bool}$. It is not the case that $\mathsf{L}\,\mathsf{bool}$ is $\langle l \rangle$-transparent for all $l \in \mathcal{P}$, so it would not follow that any continuous map $\langle l \rangle \bullet A \longrightarrow \mathsf{L}\,\mathsf{bool}$ is constant. A partial function always extends to a total function on a restricted domain, however, so we may immediately conclude the following:

▶ **Observation 9** (Termination-insensitive noninterference)**.** *For any continuous map $f : \langle l \rangle \bullet A \longrightarrow \mathsf{L}\,\mathsf{bool}$ and elements $u, v : \langle l \rangle \bullet A$ with fu and fv defined, we have $fu = fv$.*

This is the sense in which termination-insensitive noninterference arises automatically from the combination of domain theory with sheaf semantics for information flow.

4 Refined dependency core calculus

We now embark on the technical development of this paper, beginning with a call-by-push-value (cbpv) style [29] refinement of the dependency core calculus over a poset $\mathcal{P}$ of security levels. We will work informally in the logical framework of locally Cartesian closed categories *à la* Gratzer and Sterling [20]; we will write $\mathfrak{T}$ for the free locally Cartesian closed category generated by all the constants and equations specified herein.

4.1 The basic language

We have value types $A : \mathsf{tp}^{+}$ and computation types $X : \mathsf{tp}^{\ominus}$; because our presentation of cbpv does not include stacks, we will not include a separate syntactic category for computations but instead access them through thunking. The sorts of value and computation types and their adjoint connectives are specified below:

$$\mathsf{tp}^{+}, \mathsf{tp}^{\ominus} : \mathbf{Sort} \qquad \mathsf{tm} : \mathsf{tp}^{+} \to \mathbf{Sort} \qquad \mathsf{U} : \mathsf{tp}^{\ominus} \to \mathsf{tp}^{+} \qquad \mathsf{F} : \mathsf{tp}^{+} \to \mathsf{tp}^{\ominus}$$

We let A, B, C range over tp^{+} and X, Y, Z over $\mathsf{tp}^{\ominus}$. We will often write A instead of $\mathsf{tm}\,A$ when it causes no ambiguity. Free computation types are specified as follows:

$$\begin{array}{ll} \mathsf{ret} : A \to \mathsf{UF}A & \mathsf{bind}\,(\mathsf{ret}\,u)\,f \equiv_{\mathsf{U}X} f\,u \\ \mathsf{bind} : \mathsf{UF}A \to (A \to \mathsf{U}X) \to \mathsf{U}X & \mathsf{bind}\,u\,\mathsf{ret} \equiv_{\mathsf{UF}A} u \\ & \mathsf{bind}\,(\mathsf{bind}\,u\,f)\,g \equiv_{\mathsf{U}X} \mathsf{bind}\,u\,(\lambda x.\mathsf{bind}\,(f\,x)\,g) \end{array}$$

We support general recursion in computation types:

$$\mathsf{fix} : (\mathsf{U}X \to \mathsf{U}X) \to \mathsf{U}X \qquad \mathsf{fix}\,f \equiv f\,(\mathsf{fix}\,f)$$

We close the universe $X : \mathsf{tp}^{\ominus} \vdash \mathsf{tm}\,\mathsf{U}X$ of computation types and thunked computations under all function types $\mathsf{tm}\,A \to \mathsf{tm}\,\mathsf{U}X$ by adding a new computation type constant fn equipped with a universal property like so:

$$\mathsf{fn} : \mathsf{tp}^{+} \to \mathsf{tp}^{\ominus} \to \mathsf{tp}^{\ominus} \qquad \mathsf{fn.tm} : (A \to \mathsf{U}X) \cong \mathsf{U}\,(\mathsf{fn}\,A\,X)$$

We will treat this isomorphism implicitly in our informal notation, writing $\lambda x.u(x)$ for both meta-level and object-level function terms. Finite product types are specified likewise:

$$\begin{array}{ll} \mathsf{prod} : \mathsf{tp}^{+} \to \mathsf{tp}^{+} \to \mathsf{tp}^{+} & \mathsf{unit} : \mathsf{tp}^{+} \\ \mathsf{prod.tm} : A \times B \cong \mathsf{prod}\,A\,B & \mathsf{unit.tm} : \mathbf{1} \cong \mathsf{unit} \end{array}$$

Sum types must be treated specially because we do not intend them to be coproducts in the logical framework: they should have a universal property for types, not for sorts.

$$\begin{array}{ll} \mathsf{sum} : \mathsf{tp}^{+} \to \mathsf{tp}^{+} \to \mathsf{tp}^{+} & \mathsf{case} : \mathsf{sum}\,A\,B \to (A \to C) \to (B \to C) \to C \\ \mathsf{inl} : A \to \mathsf{sum}\,A\,B & \mathsf{case}\,(\mathsf{inl}\,u)\,f\,g \equiv_{C} f\,u \\ \mathsf{inr} : B \to \mathsf{sum}\,A\,B & \mathsf{case}\,(\mathsf{inr}\,v)\,f\,g \equiv_{C} g\,v \\ & \mathsf{case}\,u\,(\lambda x.f\,(\mathsf{inl}\,x))\,(\lambda x.f\,(\mathsf{inr}\,x)) \equiv_{C} f\,u \end{array}$$

4.2 The sealing modality and declassification

For each $l \in \mathcal{P}$, we add an *abstract* proof irrelevant proposition $\langle l \rangle : \mathbf{Prop}$ to the language; this proposition represents the condition that the "client" has a lower security clearance than l. This "redaction" is implemented by isolating the types that are *sealed* at $\langle l \rangle$, *i.e.* those that become singletons in the presence of $\langle l \rangle$:

$$\begin{array}{lll} \langle l \rangle : \mathbf{Prop} & & \mathsf{sealed}_{\langle l \rangle} : \mathsf{tp}^{+} \to \mathbf{Prop} \\ \langle k \rangle \to \langle l \rangle & (k \le l \in \mathcal{P}) & \mathsf{sealed}_{\langle l \rangle}\,A :\equiv \langle l \rangle \to \{x : A \mid \forall y : A. x \equiv_{A} y\} \\ \langle k \rangle \to \langle l \rangle \to \langle k \wedge l \rangle & & \end{array}$$

We will write $\mathsf{tp}^{+}_{\bullet\langle l\rangle} \subseteq \mathsf{tp}^{+}$ for the subtype spanned by value types A for which $\mathsf{sealed}_{\langle l\rangle}\, A$ holds. As in Section 3.3, we will write $\bigstar$ for the unique element of an $\langle l\rangle$-sealed type in the presence of $u : \langle l\rangle$. Next we add the sealing modality itself:

$$\begin{array}{ll}
\mathsf{T}_l : \mathsf{tp}^{+} \to \mathsf{tp}^{+}_{\bullet\langle l\rangle} & \mathsf{unseal}_l : \left\{B : \mathsf{tp}^{+}_{\bullet\langle l\rangle}\right\} \to \mathsf{T}_l\, A \to (A \to B) \to B \\
\mathsf{seal}_l : A \to \mathsf{T}_l\, A & \mathsf{unseal}_l\, (\mathsf{seal}_l\, u)\, f \equiv_B f\, u \\
 & \mathsf{unseal}_l\, u\, (\lambda x. f\, (\mathsf{seal}_l\, x)) \equiv_B f\, u
\end{array}$$

Finally a construct for declassifying the termination channel of a sealed computation:

$$\mathsf{tdcl}_{\langle l\rangle} : \left\{A : \mathsf{tp}^{+}_{\bullet\langle l\rangle}\right\} \to \mathsf{T}_l \mathsf{UF} A \to \mathsf{UF} A \qquad \mathsf{tdcl}_{\langle l\rangle}\, (\mathsf{seal}_l\, (\mathsf{ret}\, u)) \equiv_{\mathsf{UF}A} \mathsf{ret}\, u$$

▶ Remark 10. The $\langle l\rangle$ propositions play a purely book-keeping role, facilitating verification of program equivalences in the same sense as the ghost variables of Owicki and Gries [33].

5 Denotational semantics in synthetic domain theory

We will define our denotational semantics for information flow and termination-insensitive noninterference in a category of domains indexed in $\mathcal{P}$. To give a model of the theory presented in Section 4 means to define a locally Cartesian closed functor $\mathcal{T} \longrightarrow \mathcal{E}$ where $\mathcal{E}$ is locally Cartesian closed. Unfortunately no category of domains can be locally Cartesian closed, but we can *embed* categories of domains in a locally Cartesian closed category by following the methodology of ***synthetic domain theory*** [14, 9, 12, 13, 11, 15, 30].[3]

5.1 A topos for information flow logic

Recall that $\mathcal{P}$ is a poset of security levels closed under finite meets. The presheaf topos $\mathbf{P}$ defined by the identification $\boldsymbol{\mathcal{S}}_{\mathbf{P}} = [\mathcal{P}^{\mathsf{op}}, \mathsf{Set}]$ contains propositions $\mathsf{y}_{\mathcal{P}} l$ corresponding to every security level $l \in \mathcal{P}$, and is closed under both sealing and transparency modalities $\mathsf{y}_{\mathcal{P}} l \Rightarrow E, \mathsf{y}_{\mathcal{P}} l \bullet E$ in the sense of Section 3.3; in more traditional parlance, these are the *open* and *closed* modalities corresponding to the proposition $\mathsf{y}_{\mathcal{P}} l$ [41]. It is possible to give a denotational semantics for a *total* fragment of our language in $\boldsymbol{\mathcal{S}}_{\mathbf{P}}$, but to interpret recursion we need some kind of domain theory. We therefore define a topos model of synthetic domain theory that lies over $\mathbf{P}$ and hence incorporates the information flow modalities seamlessly.

5.2 Synthetic domain theory over the information flow topos

We will now work abstractly with a Grothendieck topos $\mathbf{C}$ equipped with a dominance $\Sigma \in \boldsymbol{\mathcal{S}}_{\mathbf{C}}$, called the ***Sierpiński space***, satisfying several axioms that give rise to a reflective subcategory of objects that behave like predomains. We leave the construction of $\mathbf{C}$ to our extended version, where it is built by adapting the recipe of Fiore and Plotkin [12].

▶ **Definition 11** (Rosolini [42]). *A **dominion** on a category $\mathcal{E}$ is a stable class of monos closed under identity and composition. Given a dominion $\mathcal{M}$ such that $\mathcal{E}$ has finite limits, a **dominance** for $\mathcal{M}$ is a classifier $\top : \mathbf{1}_{\mathcal{E}} \rightarrowtail \Sigma$ for the elements of $\mathcal{M}$ in the sense that every $U \rightarrowtail A \in \mathcal{M}$ gives rise to a* unique *map $\chi_U : A \longrightarrow \Sigma$ such that $U \cong \chi_U^* \top$.*

[3] In particular we focus on the style of synthetic domain theory based on Grothendieck topoi and well-complete objects. There is another very productive strain of synthetic domain theory based on realizability and replete objects that has different properties [22, 34, 53, 35, 36, 37, 38, 39].

If $\mathcal{E}$ is locally cartesian closed, we may form the ***partial element classifier*** monad $\mathsf{L} : \mathcal{E} \longrightarrow \mathcal{E}$ for a dominance Σ, setting $\mathsf{L}E = \sum_{\phi:\Sigma}\phi \Rightarrow E$; given $e \in \mathsf{L}E$, we will write $e{\downarrow} \in \Sigma$ for the termination support $\pi_1\, e$ of e. We are particularly interested in the case where L has a final coalgebra $\bar{\boldsymbol{\omega}} \cong \mathsf{L}\bar{\boldsymbol{\omega}}$ and an initial algebra $\mathsf{L}\boldsymbol{\omega} \cong \boldsymbol{\omega}$. When $\mathcal{E}$ is the category of sets, $\boldsymbol{\omega}$ is just the natural numbers object $\mathbb{N}$ and $\bar{\boldsymbol{\omega}}$ is $\mathbb{N}_\infty$, the natural numbers with an infinite point adjoined. In general, one should think of $\boldsymbol{\omega}$ as the "figure shape" of a formal ω-chain $\boldsymbol{\omega} \longrightarrow E$ that takes into account the data of the dominance; then $\bar{\boldsymbol{\omega}}$ is the figure shape of a formal ω-chain equipped with its supremum, given by evaluation at the infinite point $\infty \in \bar{\boldsymbol{\omega}}$. There is a canonical inclusion $\boldsymbol{\iota} : \boldsymbol{\omega} \rightarrowtail \bar{\boldsymbol{\omega}}$ witnessing the *incidence relation* between a chain equipped with its supremum and the underlying chain.

▶ **Axiom SDT-1.** *Σ has* ***finite joins*** *$\bigvee_{i<n}\phi_i$ that are preserved by the inclusion $\Sigma \subseteq \Omega$. We will write $\bot$ for the empty join and $\phi \vee \psi$ for binary joins.*

▶ **Definition 12** (Complete types)**.** *In the internal language of $\mathcal{E}$, a type E is called* ***complete*** *when it is internally orthogonal to the comparison map $\boldsymbol{\omega} \rightarrowtail \bar{\boldsymbol{\omega}}$. In the internal language, this says that for any formal chain $e : \boldsymbol{\omega} \longrightarrow E$ there exists a* unique *figure $\hat{e} : \bar{\boldsymbol{\omega}} \longrightarrow E$ such that $\hat{e} \circ \boldsymbol{\iota} = e$. In this scenario, we write $\bigsqcup_{i\in\boldsymbol{\omega}} e\, i$ for the evaluation $\hat{e}\,\infty$.*

▶ **Axiom SDT-2.** *The initial lift algebra $\boldsymbol{\omega}$ is the colimit of the following ω-chain of maps:*

$$\emptyset \xrightarrow{\;!\;} \mathsf{L}\emptyset \xrightarrow{\;\mathsf{L}!\;} \mathsf{L}^2\emptyset \xrightarrow{\;\mathsf{L}^2!\;} \ldots$$

▶ **Definition 13.** *A type E is called a* ***predomain*** *when $\mathsf{L}E$ is complete.*

▶ **Axiom SDT-3.** *The dominance Σ is a predomain.*

The category of predomains is complete, cocomplete, closed under lifting, exponentials, and powerdomains, and is a reflective exponential ideal in $\boldsymbol{\mathcal{S}}_{\mathbf{C}}$ — thus better behaved than any classical category of predomains. The predomains with L-algebra structure serve as an appropriate notion of *domain* in which arbitrary fixed points can be interpreted by taking the supremum of formal $\boldsymbol{\omega}$-chains of approximations $f^n\bot$; in addition to "term-level" recursion, we may also interpret recursive types. We impose two additional axioms for information flow:

▶ **Axiom SDT-4.** *The topos $\mathbf{C}$ is equipped with a geometric morphism $\boldsymbol{p}_{\mathbf{C}} : \mathbf{C} \longrightarrow \mathbf{P}$ such that the induced functor $\boldsymbol{p}_{\mathbf{C}}^*\mathsf{y}_{\mathcal{P}} : \mathcal{P} \longrightarrow \boldsymbol{\mathcal{O}}_{\mathbf{C}}$ is fully faithful and is valued in Σ-propositions. We will write $\langle l\rangle$ for each $\boldsymbol{p}_{\mathbf{C}}^*\mathsf{y}_{\mathcal{P}}l$.*

Axiom SDT-4 ensures that our domain theory include computations whose termination behavior depends on the observer's security level. The following Axiom SDT-5 is applied to the semantic noninterference property.

▶ **Axiom SDT-5.** *Any constant object $\mathbf{C}^*[n] \in \boldsymbol{\mathcal{S}}_{\mathbf{C}}$ for $[n]$ a finite set is an $\langle l\rangle$-transparent predomain for any $l \in \mathcal{P}$.*

The category $\boldsymbol{\mathcal{S}}_{\mathbf{C}}$ is closed under as many topos-theoretic universes [51] as there are Grothendieck universes in the ambient set theory. For any such universe $\mathbf{U}_i$, there is a subuniverse $\mathbf{Predom}_i \subseteq \mathbf{U}_i$ spanned by *predomains*; we note that being a predomain is a property and not a structure. The object $\mathbf{Predom}_i$ can exist because being a predomain is a local property that can be expressed in the internal logic. In fact, the predomains can be seen to be not only a reflective subcategory but also a reflective *subfibration* as they are obtained by the internal localization at a class of maps [45]; therefore the reflection can be internalized as a connective $\mathbf{U}_i \longrightarrow \mathbf{Predom}_i$ implemented as a quotient-inductive type [44]. We may define the corresponding universe of domains $\mathbf{Dom}_i$ to be the collection of predomains in $\mathbf{Predom}_i$ equipped with L-algebra structures. We hereafter suppress universe levels.

5.3 The stabilizer of a predomain and its action

In this section, we work internally to the synthetic domain theory of $\boldsymbol{\mathcal{S}}_{\mathsf{C}}$; first we recall the definition of an *action* for a commutative monoid.

▶ **Definition 14.** *Let $(M, 0, +)$ be a monoid object in the category of predomains; an **M-action structure** on a predomain A is given by a function $\|_A : M \times A \longrightarrow A$ satisfying the identities $0 \|_A a = a$ and $m \|_A n \|_A a = (m + n) \|_A a$.*

Write $\Sigma^\vee$ for the additive monoid structure of the Sierpiński domain, with addition given by Σ-join $\phi \vee \psi$ and the unit given by the non-terminating computation $\bot$. Our terminology below is inspired by stabilizer subgroups in algebra.

▶ **Definition 15** (The stabilizer of a predomain)**.** *Given a predomain A, we define the **stabilizer** of A to be the submonoid $\mathsf{Stab}_{\Sigma^\vee} A \subseteq \Sigma^\vee$ spanned by $\phi : \Sigma^\vee$ such that A is ϕ-sealed,* i.e. *the projection map $A \times \phi \longrightarrow \phi$ is an isomorphism.*

▶ Remark 16. We can substantiate the analogy between Definition 15 and stabilizer subgroups in algebra. Up to coherence issues that could be solved using higher categories, any category $\boldsymbol{\mathcal{P}}$ of predomains closed under subterminals and pushouts can be structured with a monoid action over $\Sigma^\vee$; the action $\|_{\boldsymbol{\mathcal{P}}} : \Sigma^\vee \times \boldsymbol{\mathcal{P}} \longrightarrow \boldsymbol{\mathcal{P}}$ takes A to the ϕ-sealed object $\phi \|_{\boldsymbol{\mathcal{P}}} A :\equiv \phi \bullet A$. Up to isomorphism, the identities for a $\Sigma^\vee$-action can be seen to be satisfied. Then we say that the stabilizer of a predomain $A \in \boldsymbol{\mathcal{P}}$ is the submonoid $\mathsf{Stab}_{\Sigma^\vee} A \subseteq \Sigma^\vee$ consisting of propositions ϕ such that $\phi \|_{\boldsymbol{\mathcal{P}}} A \cong A$.

▶ **Lemma 17.** *For any predomain A, we may define a canonical $\mathsf{Stab}_{\Sigma^\vee} A$-action on $\mathsf{L}A$:*

$$\begin{aligned} &\|_{\mathsf{L}A} : \mathsf{Stab}_{\Sigma^\vee} A \times \mathsf{L}A \to \mathsf{L}A \\ \phi \|_{\mathsf{L}A}\, a = (&\phi \vee a{\downarrow}, [\phi \hookrightarrow \bigstar, a{\downarrow} \hookrightarrow a]) \end{aligned}$$

The stabilizer action described in Lemma 17 will be used to implement declassification of termination channels in our denotational semantics.

▶ **Lemma 18.** *The stabilizer action preserves terminating computations in the sense that $\phi \|_{\mathsf{L}A}\, u = u$ for $\phi : \mathsf{Stab}_{\Sigma^\vee} A$ and terminating $u : \mathsf{L}A$.*

Proof. We observe that $\phi \vee \top = \top$, hence for terminating a we have $\phi \|_{\mathsf{L}A}\, a = a$. ◀

5.4 The denotational semantics

We now define an algebra for the theory $\mathfrak{T}$ in $\boldsymbol{\mathcal{S}}_{\mathsf{C}}$; the initial prefix of this algebra is standard:

$$\begin{array}{ll}
[\![\mathsf{tp}^+]\!] = \mathbf{Predom} & [\![\mathsf{prod}]\!]\, A\, B = A \times B \\
[\![\mathsf{tp}^\ominus]\!] = \mathbf{Dom} & [\![\mathsf{prod.tm}]\!] = \langle\text{canonical}\rangle \\
[\![\mathsf{U}]\!]\, X = X & [\![\mathsf{unit}]\!] = \mathbf{1}_{\mathbf{Predom}} \\
[\![\mathsf{F}]\!]\, A = \mathsf{L}A & [\![\mathsf{unit.tm}]\!] = \langle\text{canonical}\rangle \\
[\![\mathsf{ret}]\!]\, a = a & [\![\mathsf{sum}]\!]\, A\, B = A + B \\
[\![\mathsf{bind}]\!]\, m\, f = f^\sharp\, m & [\![\mathsf{inl}]\!]\, a = \mathsf{inl}\, a \\
[\![\mathsf{fix}]\!]\, f = \mathsf{fix}\, f & [\![\mathsf{inr}]\!]\, a = \mathsf{inr}\, a \\
[\![\mathsf{fn}]\!]\, A\, X = A \Rightarrow X & \multirow{2}{*}{$[\![\mathsf{case}]\!]\, u\, f\, g = \begin{cases} f(x) & \text{if } u = \mathsf{inl}\, x \\ g(x) & \text{if } u = \mathsf{inr}\, x \end{cases}$} \\
[\![\mathsf{fn.tm}]\!] = \langle\text{canonical}\rangle &
\end{array}$$

Note that the coproduct $A + B$ above is computed in the category of predomains[4] and need not be preserved by the embedding into $\boldsymbol{\mathcal{S}}_{\mathsf{C}}$. We next add the security levels and the sealing modality, interpreted as the pushout of predomains $\langle l \rangle \bullet A$, again computed in the category of predomains. We define the unsealing operator for $B : [\![\mathsf{tp}^{+}_{\bullet\langle l \rangle}]\!]$ using the universal property of the pushout.

$$[\![\langle l \rangle]\!] = \langle l \rangle = \boldsymbol{p}^{*}_{\mathsf{C}}\mathsf{y}_{\boldsymbol{\mathcal{P}}} l \qquad [\![\mathsf{unseal}_l]\!]\, u\, f =$$
$$[\![\mathsf{T}_l]\!]\, A = \langle l \rangle \bullet A \qquad \begin{cases} f x & \text{if } u = \eta_{\bullet\langle l \rangle}\, x \\ \bigstar & \text{if } u = \bigstar \end{cases}$$
$$[\![\mathsf{seal}_l]\!]\, a = \eta_{\bullet\langle l \rangle}\, a$$

▶ **Observation 19.** *Morphisms $\langle l \rangle \bullet A \longrightarrow B$ are in bijective correspondence with morphisms $A \longrightarrow B$ that restricts to a* weakly constant *function under $\langle l \rangle$.*

We may now interpret the termination declassification operation. Fixing a sealed type $A : [\![\mathsf{tp}^{+}_{\bullet\langle l \rangle}]\!]$, we must define the dotted lift below using the universal property of the pushout and the action of the stabilizer of A on $\mathsf{L}A$, noting that $\langle l \rangle \in \mathsf{Stab}_{\Sigma^{\vee}}\, A$ by assumption:

$$\begin{tikzcd} A \arrow[r, tail, "\eta_A"] \arrow[d, "\eta_{\bullet\langle l \rangle} \circ \eta_A"'] & \mathsf{L}A \\ \langle l \rangle \bullet \mathsf{L}A \arrow[ur, dashed, "{[\![\mathsf{tdcl}_{\langle l \rangle}]\!]}"'] & \end{tikzcd} \qquad \begin{aligned} &[\![\mathsf{tdcl}_{\langle l \rangle}]\!]\, u = \\ &\begin{cases} \langle l \rangle \parallel_{\mathsf{L}A} x & \text{if } u = \eta_{\bullet\langle l \rangle}\, x \\ \langle l \rangle \parallel_{\mathsf{L}A} \bot & \text{if } u = \bigstar \end{cases} \end{aligned}$$

To see that the above is well-defined, we observe that under $\langle l \rangle$ both branches return the (unique) computation whose termination support is $\langle l \rangle$. With this definition, the required computation rule holds by virtue of Lemma 18.

5.5 Noninterference in the denotational semantics

▶ **Definition 20.** *A function $u : A \longrightarrow B$ is called* ***weakly constant*** *[26] if for all $x, y : A$ we have $u\,x = u\,y$. A partial function $u : A \longrightarrow \mathsf{L}B$ is called* ***partially constant*** *if for all $x, y : A$ such that $u\,x\!\downarrow \wedge\, u\,y\!\downarrow$, we have $u\,x = u\,y$.*

For the following, let $l \in \boldsymbol{\mathcal{P}}$ be a security level.

▶ **Lemma 21.** *Let A be a $\langle l \rangle$-sealed predomain and let B be a $\langle l \rangle$-transparent predomain; then (1) any function $A \longrightarrow B$ is weakly constant, and (2) any partial function $A \longrightarrow \mathsf{L}B$ is partially constant.*

The following lemma follows from Axiom SDT-5.

▶ **Lemma 22.** *The predomain $[\![\mathsf{bool}]\!]$ is $\langle l \rangle$-transparent.*

In order for Lemma 21 to have any import as far as the equational theory is concerned, we must establish computational adequacy. This is the topic of Section 6.

---

[4] Any reflective subcategory of a cocomplete category is cocomplete: first compute the colimit in the outer category, and then apply the reflection.

6 Adequacy of the denotational semantics

We must argue that the denotational semantics agrees with the theory as far as convergence and return values is concerned. We do so using a Plotkin-style logical relations argument, phrased in the language of Synthetic Tait Computability [48, 50, 49].

6.1 Synthetic Tait computability of formal approximation

In this section we will work abstractly with a Grothendieck topos $\mathbf{G}$ satisfying several axioms that will make it support a Kripke logical relation for adequacy.

▶ **Notation 23.** For each universe $\mathbf{U} \in \boldsymbol{\mathcal{S}}_{\mathbf{G}}$ there is a type $\mathfrak{T}\text{-}\mathbf{Alg}_{\mathbf{U}}$ of internal $\mathfrak{T}$-algebras whose type components are valued in $\mathbf{U}$. $\mathfrak{T}\text{-}\mathbf{Alg}_{\mathbf{U}}$ is a dependent record containing a field for every constant in the signature by which we generated $\mathfrak{T}$. Assuming enough universes, functors $\mathfrak{T} \longrightarrow \boldsymbol{\mathcal{S}}_{\mathbf{G}}/E$ correspond up to isomorphism to morphisms $E \longrightarrow \mathfrak{T}\text{-}\mathbf{Alg}_{\mathbf{U}}$. This is the relationship between the internal language and the *functorial semantics* à la Lawvere [27].

▶ **Axiom STC-1.** *There are two disjoint propositions* $\textsc{t}, \textsc{c} \in \boldsymbol{\mathcal{O}}_{\mathbf{G}}$ *such that* $\textsc{t} \wedge \textsc{c} = \bot$*. We will refer to these as the* ***syntactic*** *and* ***computational phases*** *respectively. We will write* $\textsc{b} = \textsc{t} \vee \textsc{c}$ *for the disjoint union of the two phases.*

▶ **Axiom STC-2.** *Within the syntactic phase, there exists a* $\mathfrak{T}$*-algebra* $\mathcal{A}^{\textsc{t}} : \mathfrak{T}\text{-}\mathbf{Alg}_{\mathbf{U}_{/\textsc{t}}}$ *such that the corresponding functor* $\mathfrak{T} \longrightarrow \boldsymbol{\mathcal{S}}_{\mathbf{G}}/\textsc{t}$ *is fully faithful.*

▶ **Axiom STC-3.** *Within the computational phase, the axioms of* $\mathfrak{P}$*-indexed synthetic domain theory (Axioms SDT-1–SDT-5) are satisfied.*

As a consequence of Axiom STC-3, we have a *computational* $\mathfrak{T}$-algebra $\mathcal{A}^{\textsc{c}} : \mathfrak{T}\text{-}\mathbf{Alg}_{\mathbf{U}_{/\textsc{c}}}$ given by the constructions of Section 5.4. Gluing together the two models $\mathcal{A}^{\textsc{t}}, \mathcal{A}^{\textsc{c}}$ we see that $\mathbf{G}_{/\textsc{b}}$ supports a model $\mathcal{A}^{\textsc{b}} = \left[\textsc{t} \hookrightarrow \mathcal{A}^{\textsc{t}}, \textsc{c} \hookrightarrow \mathcal{A}^{\textsc{c}}\right]$ of $\mathfrak{T}$. The final Axiom STC-4 above is needed in the approximation structure of $\mathsf{tdcl}_{\langle l \rangle}$.

▶ **Axiom STC-4.** *For each* $l \in \mathfrak{P}$ *we have* $\mathcal{A}^{\textsc{c}}.\langle l \rangle \leq \textsc{b} \bullet \mathcal{A}^{\textsc{t}}.\langle l \rangle$*.*

▶ **Theorem 24.** *There exists a topos* $\mathbf{G}$ *satisfying Axioms STC-1–STC-4 containing open subtopoi* $\mathbf{G}_{/\textsc{t}} = \widehat{\mathfrak{T}}$ *and* $\mathbf{G}_{/\textsc{c}} = \mathbf{C}$ *such that the complementary closed subtopos is* $\mathbf{G}_{\bullet\textsc{b}} = \mathbf{P}$*.*

Proof. We may construct a topos using a variant of the Artin gluing construction of Sterling and Harper [50], which we detail in our extended version. ◀

By Axioms STC-1 and STC-2, any such topos $\mathbf{G}$ supports a model of the ***synthetic Tait computability*** of Sterling and Harper [50, 48]. In the internal language of $\boldsymbol{\mathcal{S}}_{\mathbf{G}}$, the phase $\textsc{b}$ induces a pair of complementary transparency/open and sealing/closed modalities that can be used to synthetically construct formal approximation relations in the sense of Plotkin between computational objects and syntactical objects. Viewing an object $E \in \boldsymbol{\mathcal{S}}_{\mathbf{G}}$ as a family $x : \textsc{c} \Rightarrow E, x' : \textsc{t} \Rightarrow E \vdash \{E \mid \textsc{c} \hookrightarrow x, \textsc{t} \hookrightarrow x'\}$ of $\textsc{b}$-sealed types over the $\textsc{b}$-transparent type $(\textsc{b} \Rightarrow E) \cong ((\textsc{c} \Rightarrow E) \times (\textsc{t} \Rightarrow E))$, we may think of E as a *proof-relevant* formal approximation relation between its computational and syntactic parts, which we might term a "formal approximation structure".

▶ **Notation 25** (Extension types)**.** We recall ***extension types*** from Riehl and Shulman [40]. Given a proposition $\phi : \Omega$ and a partial element $e : \phi \Rightarrow E$, we will write $\{E \mid \phi \hookrightarrow e\}$ for the collection of elements of E that restrict to e under ϕ, *i.e.* the subobject $\{x : E \mid \phi \Rightarrow (x = e)\} \rightarrowtail E$. Note that $\{E \mid \phi \hookrightarrow e\}$ is always ϕ-sealed, since it becomes the singleton type $\{e\}$ under ϕ.

Each universe $\mathbf{U}$ of $\boldsymbol{\mathcal{S}}_{\mathsf{G}}$ satisfies a remarkable *strictification* property with respect to any proposition $\phi : \Omega$ that allows one to construct codes for dependent sums of families of ϕ-sealed types over a ϕ-transparent type in such a way that they restrict *exactly* to the ϕ-transparent part under ϕ. This refinement of dependent sums is called a ***strict glue type***:[5]

$$\frac{\textsc{strict glue types} \qquad A : \phi \Rightarrow \mathbf{U} \qquad B : ((z : \phi) \Rightarrow A\, z) \to \mathbf{U} \qquad \forall x.\ \mathsf{isSealed}_\phi\,(B, x)}{(x : A) \ltimes_\phi B\, x : \{\mathbf{U} \mid z : \phi \hookrightarrow A\, z\} \qquad \mathsf{glue}_\phi : \{((x : (z : \phi) \Rightarrow A\, z) \times B\, x) \cong (x : A) \ltimes_\phi B\, x \mid \phi \hookrightarrow \pi_1\}}$$

▶ **Notation 26** (Strict glue types). We impose two notations assuming A, B as above. Given $a : (z : \phi) \Rightarrow A\, z$ and $b : B\, a$, we write $\mathsf{glue}\,[b \mid \phi \hookrightarrow a]$ for $\mathsf{glue}_\phi(a, b)$. Given $g : (x : A) \ltimes_\phi B\, x$, we write $\mathsf{unglue}_\phi g : B\, g$ for the element $\pi_2\left(\mathsf{glue}_\phi^{-1}\, g\right)$.

▶ **Notation 27.** Let E be a type in $\boldsymbol{\mathcal{S}}_{\mathsf{G}}$ and fix elements $e : \textsc{c} \Rightarrow E$ and $e' : \textsc{t} \Rightarrow E$ of the computational and syntactical parts of E respectively; we will write $e \lhd_E e'$, pronounced "e formally approximates e'", for the extension type $\{E \mid \textsc{c} \hookrightarrow e, \textsc{t} \hookrightarrow e'\}$.

This is the connection between synthetic Tait computability and analytic logical relations; the open parts of an object correspond to the *subjects* of a logical relation and the closed parts of an object correspond to the evidence of that relation.

▶ **Definition 28** (Formal approximation relations). *A type E is called a **formal approximation relation** when for any* $\textsc{b}$*-point $e : \textsc{b} \Rightarrow E$, the extension type $\{E \mid \textsc{b} \hookrightarrow e\}$ is a proposition,* i.e. *any two elements of $e \lhd_E e$ are equal.*

We will write $\mathsf{Rel}_{\mathbf{U}} \subseteq \mathbf{U}$ for the subuniverse of formal approximation relations.

▶ **Definition 29** (Admissible formal approximation relations). *Let E be a formal approximation relation such that $\textsc{c} \Rightarrow E$ is a predomain equipped with an L-algebra structure. We say that E is **admissible** at $x : \textsc{t} \Rightarrow E$ when the subobject $\{E \mid \textsc{t} \hookrightarrow x\} \subseteq \textsc{c} \Rightarrow E$ is admissible in the sense of synthetic domain theory,* i.e. *contains $\perp$ and is closed under formal suprema of formal $\boldsymbol{\omega}$-chains. We say that E is admissible when it is admissible at every such x.*

▶ **Lemma 30** (Scott induction). *Let X be a formal approximation relation such that $\textsc{c} \Rightarrow X$ is a domain. Let $f : X \to X$ be an endofunction on X and let $x : \textsc{t} \Rightarrow X$ be a syntactical fixed point of f in the sense that $\textsc{t} \Rightarrow (x = f\, x)$; if X is admissible at x, then we have $\mathsf{fix}\, f \lhd_X x$.*

Our goal can be rephrased now in the internal language; choosing a universe $\mathbf{V} \supset \mathbf{U}$, we wish to define a suitable $\mathbf{V}$-valued algebra $\mathcal{A} \in \boldsymbol{\mathfrak{T}}\text{-}\mathbf{Alg}_{\mathbf{V}}$ that restricts under $\textsc{b}$ to $\mathcal{A}^{\textsc{b}}$, *i.e.* an element $\mathcal{A} \in \left\{\boldsymbol{\mathfrak{T}}\text{-}\mathbf{Alg}_{\mathbf{V}} \;\middle|\; \textsc{b} \hookrightarrow \mathcal{A}^{\textsc{b}}\right\}$. This can be done quite elegantly in the internal language of $\boldsymbol{\mathcal{S}}_{\mathsf{G}}$, *i.e.* the *synthetic Tait computability of formal approximation structures.* The high-level structure of our model construction is summarized as follows:

> We interpret value types as ***formal approximation structures*** over a syntactic value type and a predomain; we interpret computation types as ***admissible formal approximation relations*** between a syntactic computation type and a domain.

[5] In presheaves, the universes of Hofmann and Streicher [21, 51] satisfy this property directly; for sheaves, there is an alternative transfinite construction of universes enjoying this property [19]. Our presentation in terms of transparency and sealing is an equivalent reformulation of the strictness property identified by several authors in the context of the semantics of homotopy type theory [24, 52, 46, 7, 32, 5, 47, 4].

To make this precise, we will define $\mathcal{A}.\mathsf{tp}^+ \in \{\mathbf{V} \mid \textsc{b} \hookrightarrow \mathcal{A}^{\textsc{b}}.\mathsf{tp}^+\}$ as the collection of types that restrict to an element of $\mathcal{A}^{\textsc{t}}.\mathsf{tp}^+$ in the syntactic phase and to an element of $\mathcal{A}^{\textsc{c}}.\mathsf{tp}^+ = \mathbf{Predom}$ in the computational phase. This is achieved using strict gluing:

$$\mathcal{A}.\mathsf{tp}^+ = (A : \mathcal{A}^{\textsc{b}}.\mathsf{tp}^+) \ltimes_{\textsc{b}} \{\mathbf{U} \mid \textsc{b} \hookrightarrow \mathcal{A}^{\textsc{b}}.\mathsf{tm}\, A\} \qquad \mathcal{A}.\mathsf{tm} = \mathsf{unglue}_{\textsc{b}}$$

The above is well-defined because $\mathcal{A}^{\textsc{b}}.\mathsf{tp}^+$ is $\textsc{b}$-transparent and $\{\mathbf{U} \mid \textsc{b} \hookrightarrow \mathcal{A}^{\textsc{b}}.\mathsf{tm}\, A\}$ is $\textsc{b}$-sealed. We also have $\textsc{t} \Rightarrow \mathcal{A}.\mathsf{tp}^+ = \mathcal{A}^{\textsc{t}}.\mathsf{tp}^+$ and $\textsc{c} \Rightarrow \mathcal{A}.\mathsf{tp}^+ = \mathbf{Predom}$. Next we define the formal approximation structure of computation types:

$$\mathcal{A}.\mathsf{tp}^{\ominus} = (X : \mathcal{A}^{\textsc{b}}.\mathsf{tp}^{\ominus}) \ltimes_{\textsc{b}} \{X' : \{\mathsf{Rel}_{\mathbf{U}} \mid \textsc{b} \hookrightarrow \mathcal{A}^{\textsc{b}}.\mathsf{tm}\,(\mathcal{A}^{\textsc{b}}.\mathsf{U}\, X)\} \mid X' \text{ is admissible}\}$$

To see that the above is well-defined, we must check that the family component of the gluing is pointwise $\textsc{b}$-sealed, which follows because the property of being admissible is $\textsc{b}$-sealed. To see that this is the case, we observe that it is obviously $\textsc{t}$-sealed and also (less obviously) $\textsc{c}$-sealed: under $\textsc{c}$, X' restricts to the "total" predicate on X which is always admissible. To define the thunking connective, we simply forget that a given admissible approximation relation was admissible: $\mathcal{A}.\mathsf{U}\, X = \mathsf{glue}\,[\mathsf{unglue}_{\textsc{b}} X \mid \textsc{b} \hookrightarrow \mathcal{A}^{\textsc{b}}.\mathsf{U}\, X]$. To interpret free computation types, we proceed in two steps; first we define the formal approximation relation as an element of $\mathsf{Rel}_{\mathbf{U}}$ and then we glue it onto syntax and semantics.

$$\begin{aligned}&[\mathsf{F}]\, A = (u : \mathcal{A}^{\textsc{b}}.(\mathsf{UF})\, A) \ltimes_{\textsc{b}} (\textsc{c} \Rightarrow u{\downarrow}) \Rightarrow \textsc{b} \bullet \exists a : A.\textsc{b} \Rightarrow u = \mathcal{A}^{\textsc{b}}.\mathsf{ret}\, a\\ &\mathcal{A}.\mathsf{F}\, A = \mathsf{glue}\,[[\mathsf{F}]\, A \mid \textsc{b} \hookrightarrow \mathcal{A}^{\textsc{b}}.\mathsf{F}]\end{aligned}$$

In simpler language, we have $u \lhd_{[\mathsf{F}]\, A}\, v$ if and only if v terminates syntactically whenever u terminates such that the value of u formally approximates the value of v. This is the standard clause for lifting in an adequacy proof, phrased in synthetic Tait computability. ; the use of the sealing modality is an artifact of synthetic Tait computability, ensuring that the relation is pointwise $\textsc{b}$-sealed. The $\mathsf{ret}, \mathsf{bind}$ operations are easily shown to preserve the formal approximation relations. The construction of formal approximation structures for product and function spaces is likewise trivial. Using Scott induction (Lemma 30) we can show that fixed points also lie in the formal approximation relations; we elide the details. Next we deal with the information flow constructs, starting by interpreting each security policy $\mathcal{A}.\langle l \rangle$ as $\mathcal{A}^{\textsc{b}}.\langle l \rangle$. The sealing modality is interpreted below:

$$\begin{aligned}&[\mathsf{T}_l]\, A = (u : \mathcal{A}^{\textsc{b}}.\mathsf{T}_l A) \ltimes_{\textsc{b}} \textsc{b} \bullet \mathcal{A}.\langle l \rangle \bullet \{a : A \mid \textsc{b} \Rightarrow u = \mathcal{A}^{\textsc{b}}.\mathsf{seal}_l a\}\\ &\mathcal{A}.\mathsf{T}_l A = \mathsf{glue}\,[[\mathsf{T}_l]\, A \mid \textsc{b} \hookrightarrow \mathcal{A}^{\textsc{b}}.\mathsf{T}_l]\end{aligned}$$

▶ **Theorem 31** (Fundamental theorem of logical relations)**.** *The preceding constructions arrange into an algebra* $\mathcal{A} \in \{\mathfrak{T}\text{-}\mathbf{Alg}_{\mathbf{V}} \mid \textsc{b} \hookrightarrow \mathcal{A}^{\textsc{b}}\}$.

6.2 Adequacy and syntactic noninterference results

The following definitions and results in this section are global rather than internal. We may immediately read off from the logical relation of Section 6.1 a few important properties relating value terms and their denotations. The results of this section depend heavily on the assumption that the functor $\mathfrak{T} \hookrightarrow \boldsymbol{\mathcal{S}}_{\mathbf{G}}/\textsc{t}$ is fully faithful (Axiom STC-2).

▶ **Theorem 32** (Value adequacy)**.** *For any closed values* $u, v : \mathbf{1}_{\mathfrak{T}} \longrightarrow \mathsf{bool}$*, we have* $[\![u]\!] = [\![v]\!]$ *if and only if* $u \equiv_{\mathsf{bool}} v$*; moreover we have either* $u \equiv_{\mathsf{bool}} \mathsf{tt}$ *or* $u \equiv_{\mathsf{bool}} \mathsf{ff}$.

Let $u : \mathbf{1}_{\mathfrak{T}} \longrightarrow \mathsf{UF}A$ be a closed computation.

▶ **Definition 33** (Convergence and divergence)**.** *We say that u* ***converges*** *when there exists $a : \mathbf{1}_{\mathcal{T}} \longrightarrow A$ such that $u = \mathsf{ret}\, a$. Conversely, we say that u* ***diverges*** *when there does not exist such an a. We will write $u\Downarrow$ to mean that u converges, and $u\Uparrow$ to mean that u diverges.*

▶ **Theorem 34** (Computational adequacy)**.** *The computation u converges iff $[\![u]\!]\downarrow = \top$.*

▶ **Theorem 35** (Termination-insensitive noninterference)**.** *Let A be a syntactic type such that $\mathsf{sealed}_{\langle l \rangle}\, A$ holds; fix a term $c : A \longrightarrow \mathsf{UF}\, \mathsf{bool}$. Then for all $x, y : \mathbf{1}_{\mathcal{T}} \longrightarrow A$ such that $c\, x\Downarrow$ and $c\, y\Downarrow$, we have $c\, x \equiv_{\mathsf{UF}\,\mathsf{bool}} c\, y$.*

We give an example of a program whose termination behavior hinges on a classified bit to demonstrate that our noninterference result is non-trivial.

▶ **Example 36.** There exists an $\langle l \rangle$-sealed type A and a term $c : A \longrightarrow \mathsf{UF}\, \mathsf{unit}$ such that for some $x, y : \mathbf{1}_{\mathcal{T}} \longrightarrow A$ we have $c\, x\Downarrow$ and yet $c\, y\Uparrow$.

Proof. Choose $A := \mathsf{T}_l\, \mathsf{bool}$ and consider the following terms:

$$\begin{array}{l} \top := \mathsf{ret}\,() \quad \bot := \mathsf{fix}\,(\lambda z. z) \quad x := \mathsf{seal}_l\, \mathsf{tt} \quad y := \mathsf{seal}_l\, \mathsf{ff} \\ c := \lambda u.\, \mathsf{tdcl}_{\langle l \rangle}\, (\mathsf{unseal}_l\, u\, (\lambda b.\, \mathsf{seal}_l\, (\mathsf{if}\, b\, \top\, \bot))) \end{array}$$

We then have $c\, x \equiv_{\mathsf{UF}\,\mathsf{unit}} \top$ and therefore $c\, x\Downarrow$. On the other hand, we have $c\, y \equiv_{\mathsf{UF}\,\mathsf{unit}} \mathsf{tdcl}_{\langle l \rangle}\, (\mathsf{seal}_l\, \bot)$; executing the denotational semantics, we have $[\![c\, y]\!]\downarrow = \langle l \rangle$. From the full and faithfulness assumption of Axiom SDT-4, we know that $\langle l \rangle$ is not globally equal to $\top$; hence we conclude from Theorem 34 that $c\, y\Uparrow$. ◀

References

1 Martín Abadi, Anindya Banerjee, Nevin Heintze, and Jon G. Riecke. A core calculus of dependency. In *Proceedings of the 26th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '99, pages 147–160, San Antonio, Texas, USA, 1999. Association for Computing Machinery. `doi:10.1145/292540.292555`.

2 Mathieu Anel and André Joyal. Topo-logie. In Mathieu Anel and Gabriel Catren, editors, *New Spaces in Mathematics: Formal and Conceptual Reflections*, volume 1, chapter 4, pages 155–257. Cambridge University Press, 2021. `doi:10.1017/9781108854429.007`.

3 Michael Artin, Alexander Grothendieck, and Jean-Louis Verdier. *Théorie des topos et cohomologie étale des schémas*, volume 269, 270, 305 of *Lecture Notes in Mathematics*. Springer-Verlag, Berlin, 1972. Séminaire de Géométrie Algébrique du Bois-Marie 1963–1964 (SGA 4), Dirigé par M. Artin, A. Grothendieck, et J.-L. Verdier. Avec la collaboration de N. Bourbaki, P. Deligne et B. Saint-Donat.

4 Steve Awodey. A Quillen model structure on the category of cartesian cubical sets. Unpublished notes, 2021. URL: `https://github.com/awodey/math/blob/e8c715cc5cb6a966e736656bbe54d0483f9650fc/QMS/qms.pdf`.

5 Lars Birkedal, Aleš Bizjak, Ranald Clouston, Hans Bugge Grathwohl, Bas Spitters, and Andrea Vezzosi. Guarded Cubical Type Theory: Path Equality for Guarded Recursion. In Jean-Marc Talbot and Laurent Regnier, editors, *25th EACSL Annual Conference on Computer Science Logic (CSL 2016)*, volume 62 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 23:1–23:17, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. `doi:10.4230/LIPIcs.CSL.2016.23`.

6 William J. Bowman and Amal Ahmed. Noninterference for free. In Kathleen Fisher and John H. Reppy, editors, *Proceedings of the 20th ACM SIGPLAN International Conference on Functional Programming, ICFP 2015, Vancouver, BC, Canada, September 1-3, 2015*, pages 101–113. Association for Computing Machinery, 2015. `doi:10.1145/2784731.2784733`.

7 Cyril Cohen, Thierry Coquand, Simon Huber, and Anders Mörtberg. Cubical Type Theory: a constructive interpretation of the univalence axiom. *IfCoLog Journal of Logics and their Applications*, 4(10):3127–3169, November 2017. `arXiv:1611.02108`.

8 Tom de Jong and Martín Hötzel Escardó. Domain Theory in Constructive and Predicative Univalent Foundations. In Christel Baier and Jean Goubault-Larrecq, editors, *29th EACSL Annual Conference on Computer Science Logic (CSL 2021)*, volume 183 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 28:1–28:18, Dagstuhl, Germany, 2021. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. URL: `https://drops.dagstuhl.de/opus/volltexte/2021/13462`, `doi:10.4230/LIPIcs.CSL.2021.28`.

9 M. Fiore, G. Plotkin, and J. Power. Complete cuboidal sets in axiomatic domain theory. In *Logic in Computer Science, Symposium on*, page 268, Los Alamitos, CA, USA, July 1997. IEEE Computer Society. `doi:10.1109/LICS.1997.614954`.

10 Marcelo Fiore. *Axiomatic Domain Theory in Categories of Partial Maps*. PhD thesis, University of Edinburgh, November 1994. URL: `https://era.ed.ac.uk/handle/1842/406`.

11 Marcelo P. Fiore. An enrichment theorem for an axiomatisation of categories of domains and continuous functions. *Mathematical Structures in Computer Science*, 7(5):591–618, October 1997. `doi:10.1017/S0960129597002429`.

12 Marcelo P. Fiore and Gordon D. Plotkin. An extension of models of axiomatic domain theory to models of synthetic domain theory. In Dirk van Dalen and Marc Bezem, editors, *Computer Science Logic, 10th International Workshop, CSL '96, Annual Conference of the EACSL, Utrecht, The Netherlands, September 21-27, 1996, Selected Papers*, volume 1258 of *Lecture Notes in Computer Science*, pages 129–149. Springer, 1996. `doi:10.1007/3-540-63172-0\_36`.

13 Marcelo P. Fiore and Giuseppe Rosolini. The category of cpos from a synthetic viewpoint. In Stephen D. Brookes and Michael W. Mislove, editors, *Thirteenth Annual Conference on Mathematical Foundations of Progamming Semantics, MFPS 1997, Carnegie Mellon University, Pittsburgh, PA, USA, March 23-26, 1997*, volume 6 of *Electronic Notes in Theoretical Computer Science*, pages 133–150. Elsevier, 1997. `doi:10.1016/S1571-0661(05)80165-3`.

14 Marcelo P. Fiore and Giuseppe Rosolini. Two models of synthetic domain theory. *Journal of Pure and Applied Algebra*, 116(1):151–162, 1997. `doi:10.1016/S0022-4049(96)00164-8`.

15 Marcelo P. Fiore and Giuseppe Rosolini. Domains in H. *Theoretical Computer Science*, 264(2):171–193, August 2001. `doi:10.1016/S0304-3975(00)00221-8`.

16 Peter Freyd. On proving that **1** is an indecomposable projective in various free categories. Unpublished manuscript, 1978.

17 Daniel Gratzer. Normalization for multimodal type theory. To appear, *Symposium on Logic in Computer Science Logic (LICS) '22*, 2021. `arXiv:2106.01414`.

18 Daniel Gratzer and Lars Birkedal. A stratified approach to Löb induction. To appear, *International Conference on Formal Structures for Computation and Deduction (FSCD) '22*, 2022. URL: `https://jozefg.github.io/papers/a-stratified-approach-to-lob-induction.pdf`.

19 Daniel Gratzer, Michael Shulman, and Jonathan Sterling. Strict universes for Grothendieck topoi. Unpublished manuscript, February 2022. `arXiv:2202.12012`, `doi:10.48550/arXiv.2202.12012`.

20 Daniel Gratzer and Jonathan Sterling. Syntactic categories for dependent type theory: sketching and adequacy. Unpublished manuscript, 2020. `arXiv:2012.10783`.

21 Martin Hofmann and Thomas Streicher. Lifting Grothendieck universes. Unpublished note, 1997. URL: `https://www2.mathematik.tu-darmstadt.de/~streicher/NOTES/lift.pdf`.

22 J. M. E. Hyland. First steps in synthetic domain theory. In Aurelio Carboni, Maria Cristina Pedicchio, and Guiseppe Rosolini, editors, *Category Theory*, pages 131–156, Berlin, Heidelberg, 1991. Springer Berlin Heidelberg.

23 Peter T. Johnstone. *Sketches of an Elephant: A Topos Theory Compendium: Volumes 1 and 2*. Number 43 in Oxford Logical Guides. Oxford Science Publications, 2002.

24 Chris Kapulkin and Peter LeFanu Lumsdaine. The simplicial model of Univalent Foundations (after Voevodsky). *Journal of the European Mathematical Society*, 23:2071–2126, March 2021. arXiv:1211.2851, doi:10.4171/JEMS/1050.

25 G. A. Kavvos. Modalities, cohesion, and information flow. *Proceedings of the ACM on Programming Languages*, 3(POPL), January 2019. doi:10.1145/3290333.

26 Nicolai Kraus, Martín Escardó, Thierry Coquand, and Thorsten Altenkirch. Notions of anonymous existence in Martin-Löf type theory. *Logical Methods in Computer Science*, 13(1):1–36, March 2017. doi:10.23638/LMCS-13(1:15)2017.

27 F. William Lawvere. Functorial Semantics of Algebraic Theories. *Reprints in Theory and Applications of Categories*, 4:1–121, 2004. URL: http://tac.mta.ca/tac/reprints/articles/5/tr5.pdf.

28 F. William Lawvere. Axiomatic cohesion. *Theory and Applications of Categories*, 19:41–49, June 2007. URL: http://www.tac.mta.ca/tac/volumes/19/3/19-03.pdf.

29 Paul Blain Levy. *Call-By-Push-Value: A Functional/Imperative Synthesis (Semantics Structures in Computation, V. 2)*. Kluwer Academic Publishers, Norwell, MA, USA, 2004.

30 Cristina Matache, Sean Moss, and Sam Staton. Recursion and Sequentiality in Categories of Sheaves. In Naoki Kobayashi, editor, *6th International Conference on Formal Structures for Computation and Deduction (FSCD 2021)*, volume 195 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 25:1–25:22, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. URL: https://drops.dagstuhl.de/opus/volltexte/2021/14263, doi:10.4230/LIPIcs.FSCD.2021.25.

31 Yue Niu, Jonathan Sterling, Harrison Grodin, and Robert Harper. A cost-aware logical framework. *Proceedings of the ACM on Programming Languages*, 6(POPL), January 2022. arXiv:2107.04663, doi:10.1145/3498670.

32 Ian Orton and Andrew M. Pitts. Axioms for modelling cubical type theory in a topos. In Jean-Marc Talbot and Laurent Regnier, editors, *25th EACSL Annual Conference on Computer Science Logic (CSL 2016)*, volume 62 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 24:1–24:19, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.CSL.2016.24.

33 Susan Owicki and David Gries. An axiomatic proof technique for parallel programs I. *Acta Informatica*, 6(4):319–340, December 1976. doi:10.1007/BF00268134.

34 Wesley Phoa. *Domain Theory in Realizability Toposes.* PhD thesis, University of Edinburgh, July 1991.

35 Bernhard Reus. *Program Verification in Synthetic Domain Theory.* PhD thesis, Ludwig-Maximilians-Universität München, München, November 1995.

36 Bernhard Reus. Synthetic domain theory in type theory: Another logic of computable functions. In Gerhard Goos, Juris Hartmanis, Jan van Leeuwen, Joakim von Wright, Jim Grundy, and John Harrison, editors, *Theorem Proving in Higher Order Logics*, pages 363–380, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg. doi:10.1007/BFb0105416.

37 Bernhard Reus. Formalizing synthetic domain theory. *Journal of Automated Reasoning*, 23(3):411–444, 1999. doi:10.1023/A:1006258506401.

38 Bernhard Reus and Thomas Streicher. Naïve synthetic domain theory — a logical approach. Unpublished manuscript, September 1993.

39 Bernhard Reus and Thomas Streicher. General synthetic domain theory — a logical approach. *Mathematical Structures in Computer Science*, 9(2):177–223, 1999. doi:10.1017/S096012959900273X.

40 Emily Riehl and Michael Shulman. A type theory for synthetic ∞-categories. *Higher Structures*, 1:147–224, 2017. URL: https://journals.mq.edu.au/index.php/higher_structures/article/view/36, arXiv:1705.07442.

41 Egbert Rijke, Michael Shulman, and Bas Spitters. Modalities in homotopy type theory. *Logical Methods in Computer Science*, Volume 16, Issue 1, January 2020. URL: https://lmcs.episciences.org/6015, arXiv:1706.07526, doi:10.23638/LMCS-16(1:2)2020.

42 Guiseppe Rosolini. *Continuity and effectiveness in topoi*. PhD thesis, University of Oxford, 1986.

43 Patrick Schultz and David I. Spivak. *Temporal Type Theory*, volume 29 of *Progress in Computer Science and Applied Logic*. Birkhäuser Basel, 2019. arXiv:1710.10258, doi:10.1007/978-3-030-00704-1.

44 Michael Shulman. Localization as an inductive definition, December 2011. URL: https://homotopytypetheory.org/2011/12/06/inductive-localization/.

45 Michael Shulman. Reflective subfibrations, factorization systems, and stable units, December 2011. URL: https://golem.ph.utexas.edu/category/2011/12/reflective_subfibrations_facto.html.

46 Michael Shulman. The univalence axiom for elegant reedy presheaves. *Homology, Homotopy and Applications*, 17:81–106, 2015. arXiv:1307.6248, doi:10.4310/HHA.2015.v17.n2.a6.

47 Michael Shulman. All $(\infty, 1)$-toposes have strict univalent universes. Unpublished manuscript, April 2019. arXiv:1904.07004.

48 Jonathan Sterling. *First Steps in Synthetic Tait Computability: The Objective Metatheory of Cubical Type Theory*. PhD thesis, Carnegie Mellon University, 2021. CMU technical report CMU-CS-21-142. doi:10.5281/zenodo.5709838.

49 Jonathan Sterling and Carlo Angiuli. Normalization for cubical type theory. In *2021 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–15, Los Alamitos, CA, USA, July 2021. IEEE Computer Society. arXiv:2101.11479, doi:10.1109/LICS52264.2021.9470719.

50 Jonathan Sterling and Robert Harper. Logical relations as types: Proof-relevant parametricity for program modules. *Journal of the ACM*, 68(6), October 2021. arXiv:2010.08599, doi:10.1145/3474834.

51 Thomas Streicher. Universes in toposes. In Laura Crosilla and Peter Schuster, editors, *From Sets and Types to Topology and Analysis: Towards practical foundations for constructive mathematics*, volume 48 of *Oxford Logical Guides*, pages 78–90. Oxford University Press, Oxford, 2005. doi:10.1093/acprof:oso/9780198566519.001.0001.

52 Thomas Streicher. A model of type theory in simplicial sets: A brief introduction to voevodsky's homotopy type theory. *Journal of Applied Logic*, 12(1):45–49, 2014. doi:10.1016/j.jal.2013.04.001.

53 Paul Taylor. The fixed point property in synthetic domain theory. In *[1991] Proceedings Sixth Annual IEEE Symposium on Logic in Computer Science*, pages 152–160, 1991. doi:10.1109/LICS.1991.151640.

54 Stephen Tse and Steve Zdancewic. Translating dependency into parametricity. In *Proceedings of the Ninth ACM SIGPLAN International Conference on Functional Programming*, pages 115–125, Snow Bird, UT, USA, 2004. Association for Computing Machinery. doi:10.1145/1016850.1016868.

55 Steven Vickers. Locales and toposes as spaces. In Marco Aiello, Ian Pratt-Hartmann, and Johan Van Benthem, editors, *Handbook of Spatial Logics*, pages 429–496. Springer Netherlands, Dordrecht, 2007. doi:10.1007/978-1-4020-5587-4_8.