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 reinterpretation 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


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.
Sheaf semantics of termination-insensitive noninterference

of security levels \( \mathcal{P} \). Covariantly in security levels \( l \in \mathcal{P} \), a family of type operations \( T_l A \) 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, \textit{i.e.} a \( (l) \)-\textit{sealed type} in our terminology.\footnote{We use the term “sealing” for what Abadi \textit{et al.} [1] call “protection”; to avoid confusion, we impose a uniform terminology to encompass both our work and that of \textit{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 \( (l) \).} For instance, a high-security client can read a medium-security bit:

\[
\begin{align*}
f &: T_H \text{bool} \to T_M \text{bool} \\
    f u &= x \leftarrow u; \text{seal}_H(\neg x)
\end{align*}
\]

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

\[\text{Proposition (Noninterference). For any closed function } \vdash f : T_H \text{bool} \to T_M \text{bool}, \text{ there exists a closed } \vdash b : T_M \text{bool} \text{ such that } f \simeq \lambda_. b.\]

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 \textit{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 \textit{intrinsic} and \textit{non-relational semantics} of noninterference presenting several advantages that we will argue for, inspired by the recent modal reconstruction of \textit{phase distinctions} by Sterling and Harper [50].

1.1 Termination-insensitivity and the meaning of “observation”

\[\text{Notation 1. We will write } L A \text{ for the lifting monad that Abadi \textit{et al.} notate } A \perp.\]

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:

\[\text{Termination-sensitive noninterference. For a closed partial function } \vdash f : T_H \text{bool} \to L \langle T_M \text{bool} \rangle, \text{ either } f \simeq \lambda_. \perp \text{ or there exists } \vdash b : T_M \text{bool} \text{ such that } f \simeq \lambda_. b.\]

If on the other hand we restrict observations to only terminating computations of type \( \text{bool} \), we evince a more relaxed \textit{termination-insensitive} version of noninterference that allows leakage through the termination channel but \textit{not} through the “return channel”:

\[\text{Termination-insensitive noninterference. For a closed partial function } \vdash f : T_H \text{bool} \to L \langle T_M \text{bool} \rangle, \text{ given any closed } u,v \text{ on which } f \text{ terminates, we have } fu \simeq fv.\]
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 \( P \)-indexed logical relation, we take semantics in a category of \( P \)-indexed dcpos, i.e. sheaves of dcpos on a space \( P \) in which each security level \( l \in P \) 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 $R^A_l$ indexed in security levels $l \in \mathcal{P}$. In the relational semantics, a term $\Gamma \vdash M : A$ is interpreted as a continuous function $|M| : |\Gamma| \rightarrow |A|$ such that for all $l \in \mathcal{P}$, if $\gamma R^\gamma_l \gamma'$ then $|M| \gamma R^A_l |M| \gamma'$.

**Remark 2.** Two elements $u,v \in A$ such that $u R^A_l v$ 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 $\text{bool}$ or $\text{string}$ 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 $T_l$:

$$|T_l A| = |A| \quad u R^A_{T_k} v \iff \begin{cases} u R^A_{T_k} v & \text{if } l \subseteq k \\ \perp & \text{otherwise} \end{cases}$$

Under this interpretation, the denotation of a function $T_h \text{bool} \rightarrow T_m \text{bool}$ must be a constant function, as $u R^\text{bool}_{T_h} v$ if and only if $u = v$. 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:

$$|A_\perp| = |A|_\perp \quad u R^A_{T_l} v \iff (u, v \downarrow \wedge u R^A_\perp v) \lor (u = v = \perp)$$

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

$$|A_\perp| = |A|_\perp \quad u R^A_{T_l} v \iff (u, v \downarrow \wedge u R^A_\perp v) \lor (u = \perp) \lor (v = \perp)$$

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 \( (l) \)-sealed when \( A \simeq T_l A \) is an isomorphism. But in the semantics of Abadi et al., it is not necessarily the case that a \( (l) \)-sealed object is \( (k) \)-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 \( \mathcal{P} = \{ L \sqsubseteq M \sqsubseteq H \sqsubseteq \top \} \). The purpose of including a security level even higher than \( H \) will become apparent when we explain the meaning of the sealing monad \( T_l \).

\[ \text{Notation 3.} \text{ Given a space } X \text{ and an open set } U \in \mathcal{O}_X, \text{ we will write } X_{/U} \text{ for the open subspace spanned by } U \text{ and } X_{\cdot U} \text{ for the corresponding complementary closed subspace. We also will write } \mathcal{S}_X \text{ for the category of sheaves on the space } X. \]

3.1 A space of abstract behaviors and security policies

We begin by transforming the security poset \( \mathcal{P} \) into a topological space \( \mathcal{P} \) of “abstract behaviors” whose algebra of open sets \( \mathcal{O}_P \) can be thought of as a lattice of security policies that govern whether a given behavior is permitted.

\[ \text{Definition 4.} \text{ An abstract behavior is a filter on the poset } \mathcal{P}, \text{ i.e. a monotone subset } x \subseteq \mathcal{P} \text{ such that } \bigwedge_{i < n} l_i \in x \text{ if and only if each } l_i \in x. \]
15:6 Sheaf semantics of termination-insensitive noninterference

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.

Definition 6. We define \( \mathcal{P} \) 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 \( U \).

Intuition 7 (Open and closed subspaces). Each security level \( l \in \mathcal{P} \) represents a security policy \( \langle l \rangle \) in \( \mathcal{O}_\mathcal{P} \) whose corresponding open subspace \( \mathcal{P}_{\langle l \rangle} \) is spanned by the behaviors permitted at security levels \( l \) and above. Conversely the complementary closed subspace \( \mathcal{P}_{\langle l \rangle}^c = \mathcal{P} \setminus \mathcal{P}_{\langle l \rangle} \) 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 \( \mathcal{P} \) of abstract behaviors. To see why this interpretation is plausible as a basis for secure information flow, we note that a sheaf on \( \mathcal{P} \) 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 \( k \subseteq l \) there is a chosen restriction function \( A_l \to A_k \) satisfying two laws. Hence a sheaf on \( \mathcal{P} \) 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 \( \mathcal{Q} \subseteq \mathcal{P} \), a sheaf \( A \in \mathcal{S}_\mathcal{P} \) can be restricted to \( \mathcal{Q} \), and then extended again to \( \mathcal{P} \). This composite operation gives rise to an idempotent monad on \( \mathcal{S}_\mathcal{P} \) that has the effect of purging any data from \( A \in \mathcal{S}_\mathcal{P} \) that cannot be seen from the perspective of \( \mathcal{Q} \). 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 \( A \mapsto \langle \langle l \rangle \Rightarrow A \rangle \) replaces \( A \) with whatever part of it can be viewed under the policy \( \langle l \rangle \). The transparency monad is the function space \( A^{\langle \langle l \rangle \rangle} \), recalling that an open set of \( \mathcal{P} \) 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 \( A \mapsto \langle \langle l \rangle \bullet A \rangle \) 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 \( \langle \langle l \rangle \cup A \rangle \). 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 \( \langle \langle l \rangle \Rightarrow \langle l \rangle \bullet A \rangle \cong \{ \star \} \).

2. Any sheaf \( A \in \mathcal{S}_\mathcal{P} \) can be reconstructed as the fiber product \( \langle \langle l \rangle \Rightarrow A \rangle \times \langle \langle l \rangle \bullet A \rangle \).

Those familiar with the point-free topology of topoi \([23, 55, 2]\) will recognize that \( \mathcal{P} \) is more simply described as the presheaf topos \( \mathcal{P}^\text{sp} \) viewed as a space, it is the dcpo completion of \( \mathcal{P}^\text{op} \), 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 \( U \in \mathcal{S}_\mathcal{P} \) at the behavior \( x \in \mathcal{P} \).
The first property above immediately gives rise to a form of noninterference, which justifies our intent to interpret DCC’s sealing monad as $T_lA = \langle l \rangle \cdot A$.

**Observation 8 (Noninterference).** Any map $\langle l \rangle \cdot A \to \text{bool}$ is constant.

**Proof.** We may verify that the boolean sheaf $\text{bool}$ 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 $\langle l \rangle \in \mathcal{O}_P$. 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 $\mathcal{S}_P$, 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 $\mathcal{S}_P$ 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 $\mathcal{S}_P$ 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 $\langle l \rangle \cdot A \to L\text{bool}$. It is not the case that $L\text{bool}$ is $\langle l \rangle$-transparent for all $l \in \mathcal{P}$, so it would not follow that any continuous map $\langle l \rangle \cdot A \to L\text{bool}$ 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 $f : \langle l \rangle \cdot A \to L\text{bool}$ and elements $u, v : \langle l \rangle \cdot A$ 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...
Sheaf semantics of termination-insensitive noninterference

levels. We will work informally in the logical framework of locally Cartesian closed categories à la Gratzer and Sterling [20]; we will write \( \mathcal{F} \) 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 \( A : \text{tp}^+ \) and computation types \( X : \text{tp}^\ominus \); 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:

\[
\text{tp}^+, \text{tp}^\ominus : \text{Sort} \quad \text{tm} : \text{tp}^+ \to \text{Sort} \quad U : \text{tp}^\ominus \to \text{tp}^+ \quad F : \text{tp}^+ \to \text{tp}^\ominus
\]

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

\[
\begin{align*}
\text{ret} : A &\to UF_A \\
\text{bind} : UF_A &\to (A \to UX) \to UX \\
\text{bind} u &\equiv_{UFA} u \\
\text{bind} (u f) g &\equiv_{UX} \text{bind} u (\lambda x. \text{bind} (f x) g)
\end{align*}
\]

We support general recursion in computation types:

\[
\text{fix} : (UX \to UX) \to UX \quad \text{fix} f \equiv f (\text{fix} f)
\]

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

\[
\text{fn} : \text{tp}^+ \to \text{tp}^\ominus \to \text{tp}^\ominus \\
\text{fn.tm} : (A \to UX) \equiv U (\text{fn} A X)
\]

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:

\[
\begin{align*}
\text{prod} : \text{tp}^+ \to \text{tp}^+ \to \text{tp}^+ \\
\text{prod.tm} : A \times B &\equiv \text{prod} A B \\
\text{unit} : \text{tp}^+ \\
\text{unit.tm} : 1 &\equiv \text{unit}
\end{align*}
\]

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.

\[
\begin{align*}
\text{sum} : \text{tp}^+ \to \text{tp}^+ \to \text{tp}^+ \\
\text{inl} : A &\to \text{sum} AB \\
\text{inr} : B &\to \text{sum} AB \\
\text{case} : \text{sum} AB &\to (A \to C) \to (B \to C) \to C \\
\text{case} (\text{inl} u) f g &\equiv_C f u \\
\text{case} (\text{inr} v) f g &\equiv_C g v \\
\text{case} u (\lambda x. f (\text{inl} x)) &\equiv_C (\lambda x. f (\text{inr} x)) f u
\end{align*}
\]

4.2 The sealing modality and declassification

For each \( l \in \mathcal{P} \), we add an abstract proof irrelevant proposition \( \langle l \rangle : \text{Prop} \) 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 \( \langle l \rangle \):

\[
\begin{align*}
\langle l \rangle : \text{Prop} \\
\text{sealed}_{\langle l \rangle} : \text{tp}^+ &\to \text{Prop} \\
\text{sealed}_{\langle l \rangle} A := \langle l \rangle &\to \{ x : A \mid \forall y : A.x \equiv_A y \}
\end{align*}
\]
We will write \( tp_{\bullet}^+ \subseteq tp^+ \) for the subtype spanned by value types \( A \) for which \( \text{sealed}_{(l)} A \) holds. As in Section 3.3, we will write \( \star \) for the unique element of an \( (l) \)-sealed type in the presence of \( u : (l) \). Next we add the sealing modality itself:

\[
T_l : tp^+ \rightarrow tp_{\bullet(l)}^+ \quad \text{unseal}_l : \{ B : tp_{\bullet(l)}^+ \} \rightarrow T_l A \rightarrow (A \rightarrow B) \rightarrow B
\]

\[
\text{seal}_l : A \rightarrow T_l A \quad \text{unseal}_l (\text{seal}_l u) \equiv_B f u
\]

\[
\text{unseal}_l u (\lambda x. f (\text{seal}_l x)) \equiv_B f u
\]

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

\[
\text{tdcl}_{(l)} : \{ A : tp_{\bullet(l)}^+ \} \rightarrow T_l \text{UF} A \rightarrow \text{UF} A \quad \text{tdcl}_{(l)} (\text{seal}_l \text{ret} u) \equiv_{\text{UF} A} \text{ret} u
\]

\( \triangleright \) Remark 10. The \( (l) \) 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 \( P \). To give a model of the theory presented in Section 4 means to define a locally Cartesian closed functor \( T : E \rightarrow \mathcal{C} \) where \( E \) 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].

5.1 A topos for information flow logic

Recall that \( P \) is a poset of security levels closed under finite meets. The presheaf topos \( P = [P^{op}, \text{Set}] \) contains propositions \( y_{P l} \) corresponding to every security level \( l \in P \), and is closed under both sealing and transparency modalities \( y_{P l} \Rightarrow E, y_{P l} \bullet E \) in the sense of Section 3.3; in more traditional parlance, these are the open and closed modalities corresponding to the proposition \( y_{P l} \) [41]. It is possible to give a denotational semantics for a total fragment of our language in \( S_P \), but to interpret recursion we need some kind of domain theory. We therefore define a topos model of synthetic domain theory that lies over \( P \) 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 \( C \) equipped with a dominance \( \Sigma \in SC \), 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 \( C \) to our extended version, where it is built by adapting the recipe of Fiore and Plotkin [12].

\( \triangleright \) 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 \( \top : 1 \rightarrow \Sigma \) for the elements of \( \mathcal{M} \) in the sense that every \( U : A \in \mathcal{M} \) gives rise to a unique map \( \chi_U : A \rightarrow \Sigma \) such that \( U \cong \chi_U^* \top \).

\( \triangleright \) In particular we focus on the style of synthetic domain theory based on Grothendieck toposi 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 \( \mathbb{L} : \mathcal{E} \to \mathcal{E} \) for a dominance \( \Sigma \), setting \( LE = \sum_\phi \Sigma \phi \Rightarrow E; \) given \( e \in LE \), we will write \( e \downarrow \in \Sigma \) for the termination support \( \pi_1 e \) of \( e \). We are particularly interested in the case where \( \mathbb{L} \) has a final coalgebra \( \hat{\omega} \cong \mathbb{L} \omega \) and an initial algebra \( \mathbb{L} \omega \cong \omega \). When \( \mathcal{E} \) is the category of sets, \( \omega \) is just the natural numbers object \( \mathbb{N} \) and \( \hat{\omega} \cong \mathbb{N}_\infty \), the natural numbers with an infinite point adjoined. In general, one should think of \( \omega \) as the “figure shape” of a formal \( \omega \)-chain \( \omega \to E \) that takes into account the data of the dominance; then \( \hat{\omega} \) is the figure shape of a formal \( \omega \)-chain equipped with its supremum, given by evaluation at the infinite point \( \infty \in \hat{\omega} \).

There is a canonical inclusion \( \iota : \omega \to \hat{\omega} \) witnessing the **incidence relation** between a chain equipped with its supremum and the underlying chain.

**Axiom SDT-1.** \( \Sigma \) has **finite joins** \( \bigvee_{i<n} \phi_i \) that are preserved by the inclusion \( \Sigma \subseteq \Omega \). We will write \( \perp \) for the empty join and \( \phi \lor \psi \) 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 \( \omega \Rightarrow \hat{\omega} \). In the internal language, this says that for any formal chain \( e : \omega \to E \) there exists a unique figure \( \hat{e} : \omega \to E \) such that \( \hat{e} \circ \iota = e \). In this scenario, we write \( \bigsqcup_{i \in \omega} e_i \) for the evaluation \( \hat{e} \).

**Axiom SDT-2.** The initial lift algebra \( \omega \) is the colimit of the following \( \omega \)-chain of maps:

\[
\begin{array}{cccccc}
\emptyset & \xrightarrow{!} & \mathbb{L} \emptyset & \xrightarrow{L!} & \mathbb{L}^2 \emptyset & \xrightarrow{L^2!} & \ldots
\end{array}
\]

**Definition 13.** A type \( E \) is called a **predomain** when \( LE \) is complete.

**Axiom SDT-3.** The dominance \( \Sigma \) is a predomain.

The category of predomains is complete, cocomplete, closed under lifting, exponentials, and powerdomains, and is a reflective exponential ideal in \( \mathcal{S}_C \) — thus better behaved than any classical category of predomains. The predomains with \( \mathbb{L} \)-algebra structure serve as an appropriate notion of domain in which arbitrary fixed points can be interpreted by taking the suprema of formal \( \omega \)-chains of approximations \( f^n \perp \); 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 \( \mathcal{C} \) is equipped with a geometric morphism \( p_C : \mathcal{C} \to \mathcal{P} \) such that the induced functor \( p_C^* \mathbb{P}_C : \mathbb{P} \to \mathcal{O}_C \) is fully faithful and is valued in \( \Sigma \)-propositions. We will write \( \langle l \rangle \) for each \( p_C^* l \).

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 \( \mathbb{C}^* \in \mathcal{S}_C \) for \( [n] \) a finite set is an \( \langle l \rangle \)-transparent predomain for any \( l \in \mathbb{P} \).

The category \( \mathcal{S}_C \) is closed under as many topos-theoretic universes \([51]\) as there are Grothendieck universes in the ambient set theory. For any such universe \( \mathbb{U}_i \), there is a subuniverse \( \mathbb{Predom}_i \subseteq \mathbb{U}_i \) spanned by predomains; we note that being a predomain is a property and not a structure. The object \( \mathbb{Predom}_i \) 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 \( \mathbb{U}_i \to \mathbb{Predom}_i \) implemented as a quotient-inductive type \([44]\). We may define the corresponding universe of domains \( \mathbb{Dom}_i \) to be the collection of predomains in \( \mathbb{Predom}_i \), equipped with \( \mathbb{L} \)-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 $\mathcal{S}_C$; 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 $\|_A : M \times A \to A$ 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 $\Sigma$-join $\phi \lor \psi$ 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 $\text{Stab}_{\Sigma^\vee} A \subseteq \Sigma^\vee$ spanned by $\phi : \Sigma^\vee$ such that $A$ is $\phi$-sealed, i.e., the projection map $A \times \phi \to \phi$ 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 $\mathcal{P}$ of predomains closed under subterminals and pushouts can be structured with a monoid $\mathcal{P}$ in algebra. 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 $A \in \mathcal{P}$ is the submonoid $\text{Stab}_{\Sigma^\vee} A \subseteq \Sigma^\vee$ consisting of propositions $\phi$ such that $\phi \|_\mathcal{P} A \cong A$.

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

$$\|_{L A} \colon \text{Stab}_{\Sigma^\vee} A \times L A \to L A$$

$$\phi \|_{L A} a = (\phi \lor a\downarrow, [\phi \mapsto \star, a\downarrow \mapsto a])$$

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 $\phi \|_{L A} u = u$ for $\phi : \text{Stab}_{\Sigma^\vee} A$ and terminating $u : L A$.

**Proof.** We observe that $\phi \lor T = T$, hence for terminating $a$ we have $\phi \|_{L A} a = a$. ◀

5.4 The denotational semantics

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

| $[\text{tp}^+]$ | Predom | $[\text{prod}] \ AB = A \times B$ |
| $[\text{tp}^\circ]$ | Dom | $[\text{prod}.tm] = (\text{canonical})$ |
| $[\cup] \ X = X$ | | $[\text{unit}] = 1_{\text{Predom}}$ |
| $[\mathcal{F}] \ A = L A$ | | $[\text{unit}.tm] = (\text{canonical})$ |
| $[\text{ret}] \ a = a$ | | $[\text{sum}] \ AB = A + B$ |
| $[\text{bind}] \ m \ f = f \circ m$ | | $[\text{inl}] \ a = \text{inl} \ a$ |
| $[\text{fix}] \ f = \text{fix} \ f$ | | $[\text{inr}] \ a = \text{inr} \ a$ |
| $[\text{fn}] \ A \ X = A \Rightarrow X$ | | $[\text{case}] \ u \ f \ g = \begin{cases} f(x) & \text{if } u = \text{inl} \ x \\ g(x) & \text{if } u = \text{inr} \ x \end{cases}$ |
| $[\text{fn}.tm] = (\text{canonical})$ | | |
Sheaf semantics of termination-insensitive noninterference

Note that the coproduct $A + B$ above is computed in the category of predomains and need not be preserved by the embedding into $\mathcal{S}_C$. We next add the security levels and the sealing modality, interpreted as the pushout of predomains $(l) \bullet A$, again computed in the category of predomains. We define the unsealing operator for $B : \llbracket \text{tp}_*(l) \rrbracket$ using the universal property of the pushout.

\[
\begin{align*}
[\llbracket (l) \rrbracket] &= \llbracket (l) \rrbracket = p_C^y p_l \\
[\llbracket T_l \rrbracket] A &= \llbracket (l) \rrbracket \bullet A \\
[\llbracket \text{seal} \rrbracket] a &= \eta_{(l)} a \\
[\llbracket \text{unseal} \rrbracket] u f &= f x \quad \text{if } u = \eta_{(l)} x \\
&= \star \quad \text{if } u = \star
\end{align*}
\]

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

We may now interpret the termination declassification operation. Fixing a sealed type $A : \llbracket \text{tp}_*(l) \rrbracket$, we must define the dotted lift below using the universal property of the pushout and the action of the stabilizer of $A$ on $L_A$, noting that $(l) \in \text{Stab}_\Sigma A$ by assumption:

\[
\begin{align*}
\eta_A A \xrightarrow{\eta_A} L_A \\
\eta_{(l)} \circ \eta_A \downarrow \llbracket \text{tdcl} \rrbracket \\
(l) \bullet L_A
\end{align*}
\]

\[
\begin{align*}
\llbracket \text{tdcl} \rrbracket u &= \llbracket (l) \rrbracket |_{L_A} x \quad \text{if } u = \eta_{(l)} x \\
&= \llbracket (l) \rrbracket |_{L_A} \perp \quad \text{if } u = \star
\end{align*}
\]

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

### 5.5 Noninterference in the denotational semantics

**Definition 20.** A function $u : A \to B$ is called weakly constant [26] if for all $x, y : A$ we have $u x = u y$. A partial function $u : A \to LB$ is called partially constant if for all $x, y : A$ such that $u x \downarrow \land u y \downarrow$, we have $u x = u y$.

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

**Lemma 21.** Let $A$ be a $(l)$-sealed predomain and let $B$ be a $(l)$-transparent predomain; then (1) any function $A \to B$ is weakly constant, and (2) any partial function $A \to LB$ is partially constant.

The following lemma follows from Axiom SDT-5.

**Lemma 22.** The predomain $\llbracket \text{bool} \rrbracket$ is $(l)$-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.
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 $G$ satisfying several axioms that will make it support a Kripke logical relation for adequacy.

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

- **Axiom STC-1.** There are two disjoint propositions $T, C \in O_G$ such that $T \land C = \bot$. We will refer to these as the *syntactic* and *computational phases* respectively. We will write $B = T \lor C$ for the disjoint union of the two phases.

- **Axiom STC-2.** Within the syntactic phase, there exists a $\mathcal{T}$-algebra $\mathcal{A}^c : \mathcal{T}$-$\text{Alg}_U$, such that the corresponding functor $\mathcal{T} \to S_G/T$ is fully faithful.

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

As a consequence of Axiom STC-3, we have a computational $\mathcal{T}$-algebra $\mathcal{A}^c : \mathcal{T}$-$\text{Alg}_U$, given by the constructions of Section 5.4. Glueing together the two models $\mathcal{A}^T, \mathcal{A}^C$ we see that $G_{/T}$ supports a model $\mathcal{A}^b = \{ T \mapsto \mathcal{A}^T, C \mapsto \mathcal{A}^C \}$ of $\mathcal{T}$. The final Axiom STC-4 above is needed in the approximation structure of $\text{tdcl}(l)$.

- **Axiom STC-4.** For each $l \in \mathcal{P}$ we have $\mathcal{A}^T(l) \leq B \bullet \mathcal{A}^C(l)$.

- **Theorem 24.** There exists a topos $G$ satisfying Axioms STC-1–STC-4 containing open subtopoi $G_{/T}$ and $G_{/C} = C$ such that the complementary closed subtopos is $G_{/b} = \mathcal{P}$.

**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 $G$ supports a model of the *synthetic Tait computability* of Sterling and Harper [50, 48]. In the internal language of $S_G$, the phase $B$ 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 $E \in S_G$ as a family $x : C \vdash E, x' : T \vdash E \vdash \{ E \mid C \mapsto x, T \mapsto x' \}$ of $b$-sealed types over the $b$-transparent type $(B \Rightarrow E) \cong ((C \Rightarrow E) \times (T \Rightarrow E))$, 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 $\{ E \mid \phi \mapsto e \}$ for the collection of elements of $E$ that restrict to $e$ under $\phi$, i.e. the subobject $\{ x : E \mid \phi \mapsto (x = e) \} \rightarrowtail E$. Note that $\{ E \mid \phi \mapsto e \}$ is always $\phi$-sealed, since it becomes the singleton type $\{ e \}$ under $\phi$. 
Each universe \( U \) of \( \mathcal{S}_G \) satisfies a remarkable strictification property with respect to any proposition \( \phi : \Omega \) that allows one to construct codes for dependent sums of families of \( \phi \)-sealed types over a \( \phi \)-transparent type in such a way that they restrict exactly to the \( \phi \)-transparent part under \( \phi \). This refinement of dependent sums is called a strict glue type:\(^5\)

\[
\text{strict glue types} \quad A : \phi \Rightarrow U \quad B : ((z : \phi) \Rightarrow A z) \Rightarrow U \quad \forall x. \text{isSealed}_\phi(B, x)
\]

\[\text{glue}_\phi : \{((x : (z : \phi) \Rightarrow A z) \times B x) \cong (x : A) \times B x \mid \phi \Rightarrow \pi_1\}\]

**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 \( \text{glue}[b \mid \phi \Leftrightarrow a] \) for \( \text{glue}_\phi(a, b) \). Given \( g : (x : A) \times B x \), we write \( \text{unglue}_\phi g : B x \) for the element \( \pi_2(\text{glue}_\phi^{-1} g) \).

**Notation 27.** Let \( E \) be a type in \( \mathcal{S}_G \) and fix elements \( e : c \Rightarrow E \) and \( e' : \tau \Rightarrow E \) of the computational and syntactical parts of \( E \) respectively; we will write \( e \ll_E e' \), pronounced “\( e \) formally approximates \( e' \)”, for the extension type \( \{E \mid c \Rightarrow e, \tau \Rightarrow e'\} \).

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 \( B \)-point \( e : B \Rightarrow E \), the extension type \( \{E \mid B \Rightarrow e\} \) is a proposition, i.e. any two elements of \( e \ll_E e \) are equal.

We will write \( \text{Rel}_U \subseteq U \) for the subuniverse of formal approximation relations.

**Definition 29** (Admissible formal approximation relations). Let \( E \) be a formal approximation relation such that \( c \Rightarrow E \) is a predomain equipped with an \( L \) algebra structure. We say that \( E \) is admissible at \( x : \tau \Rightarrow E \) when the subobject \( \{E \mid \tau \Rightarrow x\} \subseteq C \Rightarrow E \) is admissible in the sense of synthetic domain theory, i.e. contains \( \bot \) and is closed under formal suprema of formal \( \omega \)-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 \( c \Rightarrow X \) is a domain. Let \( f : X \Rightarrow X \) be an endofunction on \( X \) and let \( x : \tau \Rightarrow X \) be a syntactical fixed point of \( f \) in the sense that \( \tau \Rightarrow (x = f x) \); if \( X \) is admissible at \( x \), then we have \( f x \ll_X x \).

Our goal can be rephrased now in the internal language; choosing a universe \( V \supseteq U \), we wish to define a suitable \( V \)-valued algebra \( A \in \mathcal{T}\text{-Alg}_V \) that restricts under \( B \) to \( A^B \), i.e. an element \( A \in \{\mathcal{T}\text{-Alg}_V \mid B \Rightarrow A^B\} \). This can be done quite elegantly in the internal language of \( \mathcal{S}_G \), 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 \( A.t\text{p}^+ \in \{ V \mid b \leftrightarrow A^\text{p}.tm^+ \} \) as the collection of types that restrict to an element of \( A^\text{p}.tm^+ \) in the syntactic phase and to an element of \( A^\text{c}.tm^+ = \text{Predom} \) in the computational phase. This is achieved using strict gluing:

\[
A.t\text{p}^+ = (A : A^\text{p}.tp^+) \times_b \{ U \mid b \leftrightarrow A^\text{p}.tm A \} \quad A.t\text{m} = \text{unglue}_b
\]

The above is well-defined because \( A^\text{p}.tp^+ \) is \( b \)-transparent and \( \{ U \mid b \leftrightarrow A^\text{p}.tm A \} \) is \( b \)-sealed. We also have \( T \Rightarrow A.t\text{p}^+ = A^\text{p}.tp^+ \) and \( c \Rightarrow A.t\text{p}^+ = \text{Predom} \). Next we define the formal approximation structure of computation types:

\[
A.t\text{p}^\approx = (X : A^\text{p}.tp^\approx) \times_b \{ X' : \{ \text{Rel}_U \mid b \leftrightarrow A^\text{p}.tm (A^\text{p}.U X) \} \mid X' \text{ is admissible} \}
\]

To see that the above is well-defined, we must check that the family component of the gluing is pointwise \( b \)-sealed, which follows because the property of being admissible is \( b \)-sealed. To see that this is the case, we observe that it is obviously \( t \)-sealed and also (less obviously) \( c \)-sealed: under \( c \), \( X' \) 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: \( A.U X = \text{glue} [\text{unglue}_b X \mid b \leftrightarrow A^\text{p}.U X] \).

To interpret free computation types, we proceed in two steps; first we define 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 \( A.l \) as \( A^\text{p}.l \). The sealing modality is interpreted below:

\[
[T]_l A = (u : A^\text{p}.T_l A) \times_b b \cdot A.l \cdot \{ a : A \mid b \Rightarrow u = A^\text{p}.\text{seal}_l a \}
\]

\[
A.T_l A = \text{glue} [[T]_l A \mid b \leftrightarrow A^\text{p}.T_l]
\]

\[\blacktriangleright \text{Theorem } 31\text{ (Fundamental theorem of logical relations). The preceding constructions arrange into an algebra } A \in \{ \mathcal{J} \text{-Alg}_V \mid b \leftrightarrow A^\text{p} \}.\]

\section{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 \( \mathcal{J} \rightarrow \mathcal{S}_G/T \) is fully faithful (Axiom STC-2).

\[\blacktriangleright \text{Theorem } 32\text{ (Value adequacy). For any closed values } u, v : 1_\mathcal{J} \rightarrow \text{bool}, \text{ we have } \llbracket u \rrbracket = \llbracket v \rrbracket \text{ if and only if } u \equiv_{\text{bool}} v; \text{ moreover we have either } u \equiv_{\text{bool}} \text{tt or } u \equiv_{\text{bool}} \text{ff.}\]

Let \( u : 1_\mathcal{J} \rightarrow \text{UFA} \) be a closed computation.
Definition 33 (Convergence and divergence). We say that $u$ converges when there exists $a : 1_T \to A$ such that $u = \text{ret}_a$. 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 $\llbracket u \rrbracket \downarrow = \top$.

Theorem 35 (Termination-insensitive noninterference). Let $A$ be a syntactic type such that $\text{sealed}_\langle l \rangle A$ holds; fix a term $c : A \to \text{UF bool}$. Then for all $x,y : 1_T \to A$ such that $cx \downarrow$ and $cy \uparrow$, we have $cx \equiv_{\text{UF bool}} cy$.

Example 36. There exists an $\langle l \rangle$-sealed type $A$ and a term $c : A \to \text{UF unit}$ such that for some $x,y : 1_T \to A$ we have $cx \downarrow$ and yet $cy \uparrow$.

Proof. Choose $A := T_l \text{bool}$ and consider the following terms:

\[ \top := \text{ret}() \quad \bot := \text{fix}(\lambda z.z) \quad x := \text{seal}_l \text{tt} \quad y := \text{seal}_l \text{ff} \]

\[ c := \lambda u. \text{tdcl}_\langle l \rangle (\text{unseal}_l u (\lambda b. \text{seal}_l (\text{if } b \top \bot))) \]

We then have $cx \equiv_{\text{UF unit}} \top$ and therefore $cx \downarrow$. On the other hand, we have $cy \equiv_{\text{UF unit}} \text{tdcl}_\langle l \rangle (\text{seal}_l \bot)$; executing the denotational semantics, we have $\llbracket cy \rrbracket \downarrow = \langle l \rangle$. 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 $cy \uparrow$.

References


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


