Extensional and Intensional Semantic Universes: A Denotational Model of Dependent Types

Valentin Blot, Jim Laird

Extensional and Intensional Semantic Universes: A Denotational Model of Dependent Types

Valentin Blot
LRI, Université Paris Sud, CNRS
Université Paris-Saclay, France

Jim Laird
Department of Computer Science
University of Bath, UK

Abstract

We describe a dependent type theory, and a denotational model for it, that incorporates both intensional and extensional semantic universes. In the former, terms and types are interpreted as strategies on certain graph games, which are concrete data structures of a generalized form, and in the latter as stable functions on event domains.

The concrete data structures themselves form an event domain, with which we may interpret an (extensional) universe type of (intensional) types. A dependent game corresponds to a stable function into this domain; we use its trace to define dependent product and sum constructions as it captures precisely how unfolding moves combine with the dependency to shape the possible interaction in the game. Since each strategy computes a stable function on CDS states, we can lift typing judgements from the intensional to the extensional setting, giving an expressive type theory with recursively defined type families and type operators.

We define an operational semantics for intensional terms, giving a functional programming language based on our type theory, and prove that our semantics for it is computationally adequate. By extending it with a simple non-local control operator on intensional terms, we can precisely characterize behaviour in the intensional model. We demonstrate this by proving full abstraction and full completeness results.

1 Introduction

Intuitionistic dependent type theory has been proposed as a foundation for constructive mathematics [25], within which proofs correspond to functional programs with dependent types which precisely specify their properties [5]. It is the basis for proof assistants and dependently typed programming languages such as Coq, Agda and Idris which exploit this correspondence. Its denotational semantics is therefore of interest — both for underpinning these logical foundations (cf. Martin-Löf's meaning-explanations for typing judgments [24]) — and in formulating and analysing new type systems — witness, for example, the role of the groupoid model [17] in the development of homotopy type theory. Most attention has been on models which are extensional in character (in particular, validating the principle of function extensionality) [1].

Game semantics is also a foundational theory, describing the meaning of proofs and programs *intensionally* — i.e. how, rather than what, they compute — in terms of a dialogue between two players. With related models such as concrete data structures [10] it has been used to give interpretations of many programming languages and logical systems. These models are distinguished by desirable properties, notably, *full abstraction* [3, 18] and *full completeness* [1], but also connections to resource-sensitive computation and linear logic, direct representations of effectful computation, and the possibility of extracting computational content. Extending game semantics to dependent type theory is therefore a natural objective. It is also challenging, with little progress in this direction until recently [4]. Arguably, one source of difficulty is that the intensional representation of terms as strategies, which progressively reveal themselves by interaction, does not extend to types. This may reflect an intuition that types are static specifications and programs are more dynamic computational objects, but raises the question — how can one depend on the other? A related problem is: how can we interpret types themselves as terms of some special type (i.e. a *universe*) — a principle from which dependent type theory derives much of its expressive power — if the meanings of types and terms are defined in different ways?

We present solutions to these two problems, in the form of a new type theory, and a denotational semantics and categorical model for it. They are based on two "semantic universes" — intensional and extensional — of terms and types (or, more precisely, type and term formation judgments). Each of these has its own dependent type theory, and one can lift judgments from the intensional world to the extensional one — a form of cumulativity sending a program to the function it computes — while the extensional universe contains a type of intensional types, so that type-families and type-operators can be represented as terms at this type.

The main technical challenge consists in the intensional interpretation of dependent types. As we explain in the next section, unfolding play in a dependent game constrains and enables future moves both explicitly and implicitly. We can capture this precisely by representing it as a *stable* function; the *trace* of the function gives an exact characterization of the information contained in the dependency, forming a bridge between the intensional and extensional worlds.

We show that our model captures key properties of dependently typed programs by proving that it is computationally adequate with respect to a simple operational semantics. Adding a non-local control operator on intensional terms demonstrates that our type system can accommodate effectful computation, as well as leading to a *full abstraction* result for our model, and *full completeness* for its finite, total fragment.

2 Overview: Games and Dependent Types

In this section we will give an overview of the paper, and related work, via some examples. We leave the detail and formal results for later sections.

[1]Note that here (and throughout), the notions of intensionality and extensionality refer to models of type theories rather than the theories themselves, where they have a different (albeit related) meaning.

LICS '18, July 9–12, 2018, Oxford, United Kingdom
2018. ACM ISBN 978-1-4503-5583-4/18/07.
https://doi.org/10.1145/3209108.3209206

Our first task is to develop an interpretation of dependent types and terms as games and strategies — to be precise, the core Martin-Löf dependent type theory presented in Section 3. Vákár, Abramsky and Jagadeesan [4] have recently presented a model of a similar theory, built on a simply-typed (AJM-style) game semantics by a realizability-like interpretation. This gives a different solution to one of our core problems (how to constrain the rules of a game as it is played): we aim for a more direct construction of dependent games, and one which accommodates dependent sums directly and can be readily extended with a universe type. We outline our key ideas here in the setting of graph games [19], in which positions are given directly, and moves by a *relation* between them.

Definition 2.1. A game A is a directed, acyclic, bipartite graph with a single specified source node $*_A$.

In other words, the set of nodes of A may be partitioned into sets P^+_A (containing $*_A$) and P^-_A of "Opponent" and "Player" *positions*, such that the edge relation $\vdash_A$ is contained in $(P^+_A \times P^-_A) \cup (P^-_A \times P^+_A)$ (and is thus partitioned into Opponent and Player *moves*). A is *stable* if any two paths from $*_A$ to the same node $u \in P^+_A$ branch at a Player move. For example, the graph game of the Booleans, $\mathbb{B}$, consists of the Player position $\{?\}$, Opponent positions $\{*, \mathtt{tt}, \mathtt{ff}\}$ and moves $* \vdash_{\mathbb{B}} ?$ and $? \vdash_{\mathbb{B}} \mathtt{tt}$ and $? \vdash_{\mathbb{B}} \mathtt{ff}$. The game N of "lazy" natural numbers consists of the Player positions $\{?_n \mid n \in \omega\}$, the Opponent positions $\{*\} \cup \{n_=, n_> \mid n \in \omega\}$, and the moves $* \vdash_{\mathsf{N}} ?_0$, $?_n \vdash_{\mathsf{N}} n_=, ?_n \vdash n_>$, and $n_> \vdash_{\mathsf{N}} ?_{n+1}$ (i.e. once Opponent knows that a number is at least n he asks whether it is equal to n, and is told "yes" or "no, it is greater").

A *strategy* for Player on a stable game A is given by a set of positions $\sigma \subseteq P_A$ containing $*_A$ and satisfying:

If $v \in \sigma$ then there is a path from $*_A$ to v in σ.
If $u \in \sigma^-$ then there exists a unique $v \in \sigma$ such that $u \vdash v$.

A strategy is (hereditarily) *total* if it is finite and for any $u \in \sigma^+$, if $u \vdash v$ then $v \in \sigma$. We write $D(A)$ for the set of strategies on A, and $D_*(A)$ for the finite strategies.

For example, $D(\mathsf{N})$ consists of the total strategies $\lceil n_= \rceil \triangleq \lceil n_\leq \rceil \cup \{?_n, n_=\}$ for $n < \omega$ (corresponding to the natural numbers) together with non-total approximants $\lceil n_\leq \rceil \triangleq \{*\} \cup \bigcup_{i<n}\{?_i, i_<\}$ for each $n \leq \omega$.

One can give a semantics of intuitionistic simple type theory (a Cartesian closed category) [19], based on Cartesian product and function-space constructions on stable games. These yield important clues about the dependent sum and product, of which they must occur as special cases. The Cartesian product of graph games is their coalesced sum (i.e. the disjoint sum of graphs, with the $*$-nodes identified) with unit $\{*\}$. So for each $n \leq \omega$ we have a game $\mathsf{vec}_{\mathbb{B}}(n)$ — the n-ary product of copies of the Boolean game, in which Player positions are $\{(?, i) \mid i < n\}$ (requests for the ith Boolean value) and Opponent positions are $*$, together with the responses $\bigcup_{i<n}\{(\mathtt{tt}, i), (\mathtt{ff}, i)\}$.

In $A \Rightarrow B$, Player can choose to make his own move in B, or query Opponent about how her strategy in A responds to a new, accessible Player position.

Definition 2.2. Say that a (Player) position v is *accessible* from $x \in D_*(A)$ if $u \vdash_A v$ for some $u \in x$ but $v \notin x$. For games A and B, positions in $A \Rightarrow B$ are pairs $(x, v) \in \mathcal{P}_*(P_A) \times P_B$ such that:

- $(x, v) \in D_*(A) \times P^-_B$ (Player positions)
- or $(x, v) \in D_*(A) \times P^+_B$ or $v \in P^-_B$ and $x = x' \cup \{w\}$, where w is accessible from $x' \in D_*(A)$ (Opponent positions).

The move relation is: $(x, u) \vdash_{A \Rightarrow B} (y, v)$ iff $x = y$ and $u \vdash_B v$, or $u = v$ and $y = x \cup \{w\}$ for some $w \notin x$. ($*_{A\Rightarrow B} \triangleq (\{*_A\}, *_B)$).

For example, in $\mathsf{N} \Rightarrow \mathsf{vec}_{\mathbb{B}}(\omega)$, Opponent opens by requesting the ith Boolean value $(\{*\}, (?, i))$ for some $i \in \omega$; Player and Opponent then exchange moves on the left until reaching a partial or total n for which Player can return $(n, (\mathtt{tt}, i))$ or $(n, (\mathtt{ff}, i))$.

Suppose we want to define a dependent product $\Pi(n : \mathsf{N}).\mathsf{vec}_{\mathbb{B}}(n)$: a strategy on this game should represent a functional program which takes a natural number argument n and returns an n-tuple of Booleans. This is similar to $\mathsf{N} \Rightarrow \mathsf{vec}_B(\omega)$, except that by asking Proponent for the ith Boolean, Opponent already gives the information that the argument is greater than i. Thus to derive $\Pi(n : \mathsf{N}).\mathsf{vec}_{\mathbb{B}}(n)$ from $\mathsf{N} \Rightarrow \mathsf{vec}_{\mathbb{B}}(\omega)$ we *remove* the positions:

- $\{\lceil n_= \rceil, (?, i)) \mid n \leq i\}$ (which are not consistent with the dependency), and
- $\bigcup_{n \leq i}\{(\lceil n_\leq \rceil, (?, i)), (\lceil n_\leq \rceil \cup \{?_n\}, (?, i))\}$, which are implicit in the dependency and thus redundant .

and *add* the Opponent moves $* \vdash (\lceil (n+1)_\leq \rceil, (?, n))$ for $n \in \omega$.

How do we move from this *ad hoc* definition to a general construction for the dependent product? First, we need to represent a game B with a dependency on A as a function taking each strategy x on A to a graph game $B(x)$ (a *parametrization* over A). We may then specify the positions of the game $\Pi(A, B)$ —

- pairs (x, v) with $x \in D_*(A)$ and $v \in P^-_{B(x)}$ (Player positions),
- pairs (x, v) with $x \in D_*(A)$ and $v \in P^+_{B(x)}$ or $x = x' \cup \{u\}$, where u is accesible from $x' \in D_*(A)$, and $v \in P^-_{B(x)}$ (Opponent positions).

To define the *moves* of $\Pi(A, B)$, we need to formalize the idea that by making a move in B, Opponent can also change the position in A by implicitly giving information about it. This requires us to constrain parametrizations so that for any position $u \in B(x)$ there is unique $\subseteq$-least $y \subseteq x$ such that $u \in B(y)$. To be more precise, we require a parametrization over A to be a *stable function* from $(D(A), \subseteq)$ into the cpo $(\mathcal{G}, \trianglelefteq)$ of stable graph games, where $A \trianglelefteq B$ if $P_A \subseteq P_B$ and $\vdash_A = \vdash_B \upharpoonright (P_A \times P_A)$. In fact $(D(A), \subseteq)$ and $(\mathcal{G}, \trianglelefteq)$ are *dI-domains* (bounded complete, algebraic, distributive cpos in which every compact element dominates finitely many elements) [6]: a function between dI-domains is stable if it is continuous and preserves bounded meets.

We may thus define the moves of $\Pi(A, B)$ to be:
$(x, u) \vdash_{\Pi(A,B)} (y, v)$ iff $u = v$ and $y = x \cup \{w\}$ for some $w \notin x$, or $u \vdash v$ and y is minimal in $\{z \supseteq x \mid v \in B(z)\}$.

What about the dependent sum? In $\Sigma(x : A).B$, Opponent typically requires some information about Player's strategy on A before he can make moves in B. We construct the dependent sum by adding this necessary information to positions in B. For example, for any games C and D consider the dependent sum over the conditional parametrization which returns C if its argument is $\mathtt{tt}$ and D if it is $\mathtt{ff}$. In $\Sigma(x : \mathbb{B}, \mathtt{If}\, x\, \mathtt{then}\, C\, \mathtt{else}\, D)$, Opponent needs to have requested and received a Boolean value on the left, in order to know whether to make a move in C or D on the right. The set of positions of $\Sigma(x; \mathtt{bool}).\mathtt{If}\, x\, \mathtt{then}\, C\, \mathtt{else}\, D)$ is thus $P_{\mathbb{B}} + \{\{*, ?, \mathtt{tt}\}\} \times C$ and $\{\{*, ?, \mathtt{ff}\}\} \times D$ and the (additional) moves are $\mathtt{tt}.l \vdash (\{*, ?, \mathtt{tt}\}, u).r$ if $*_C \vdash u$ and $\mathtt{ff}.l \vdash (\{*, ?, \mathtt{ff}\}, v).r$ if $*_D \vdash v$. This is the familiar "lifted sum" of games.

However, there is a problem in generalizing this definition. Positions in the right component of the dependent product may depend on a minimal *set* of positions on the left (and right) — e.g. $\Sigma(x : \mathbb{B}\times\mathbb{B}).\mathtt{If}\,(\mathtt{fst}(x)\,\mathtt{and}\,\mathtt{snd}(x))\,\mathtt{then}\,C\,\mathtt{else}\,D$. So we are unable to form a dependent sum of graph games, in general. In [4], the analogous problem is resolved by using a formal categorical construction which "completes" a model of Π-types with Σ-types. This can be applied to our model but there is a more direct solution — we can extend the notion of game to allow each move to depend on multiple Opponent positions. These are presented in Section 5, as a generalized form of the concrete data structures introduced by Kahn and Plotkin and developed by Berry and Curien [10, 20].

2.1 Intensional Types as Extensional Terms

Having defined a game with dependency on A to be a stable function between dI-domains the second problem we address is: how to construct such functions systematically? Since the category of dI-domains and stable functions is Cartesian closed [9], we may define the examples from the previous section (informally) as λ-terms with basic operations, such as the conditional used in the previous section to define the lifted sum $B \oplus C$ as $\Sigma(x : \mathbb{B}).\mathtt{If}\,x\,\mathtt{then}\,B\,\mathtt{else}\,C$.

Our model is total when restricted to finite types (as in [4]). Including recursion gives an expressive partial type theory: by a standard result of domain theory, every stable endofunction has a least fixed point which we may use to give recursive definitions of dependent and non-dependent games (refining games models of recursively typed metalanguages such as FPC [26]) — e.g.

- the singleton game (Cartesian unit) as $\bot \triangleq \mu x.x$.
- the lazy natural numbers as $\mu x.\bot \oplus x$ — i.e. $\mathsf{N} \triangleq \mu x.\Sigma(y : \mathtt{bool}).\mathtt{If}\,y\,\mathtt{then}\,\bot \oplus x$.
- the dependent game $\mathsf{vec}_{\mathbb{B}} : \mathsf{N} \to \mathcal{G}$, as $\mu f.\lambda x.\mathtt{If}\,\mathtt{fst}(x)\,\mathtt{then}\,\bot\,\mathtt{else}\,(\mathbb{B} \times f(\mathtt{snd}(x)))$.

We can make these definitions formal by deriving them in a typing system in which types denote dI-domains and terms denote stable functions, with a universe type to denote the dI-domain of concrete data structures. This type system will itself require dependent types (since type families may have arguments with dependent types — e.g. $\lambda x : \mathsf{nat}.\lambda y : \mathsf{vec}_{\mathbb{B}}(\mathsf{suc}(x)).\mathtt{If}\,\mathtt{fst}(x)\,\mathtt{then}\,S\,\mathtt{else}\,T$). In other words, we need a semantics of partial dependent type theory in which terms denote stable functions. We can obtain such a model by recasting the Scott domain and information system semantics of partial type theory [27, 28] in the setting of stable domain theory.

In fact, our intensional semantics may be seen as a refinement of this stable model. We adopt a unified presentation which demonstrates this by defining both in terms of *event structures* [32], using the fact that every dI-domain is isomorphic to the partial order of states of an event structure (event domain), while concrete data structures may be themselves be characterised as certain polarised event structures (related to the *confusion-free* event structures [31]).

We define a notion of categorical model with which to structure our semantics based on Dybjer's interpretation of dependent type theory in a *category with families* (CwF) [15], first defining a CwF of event structures and stable functions in Section 4, and then showing that it may be refined to a CwF of concrete data structures in Section 5. In Section 6 we add new typing rules which relate them, allowing recursively defined type families. We interpret these via a morphism of CwFs [13] from the intensional to the extensional models (based on the functor which sends a sequential algorithm to the stable function it computes), together with further structure on the extensional CwF picking out the universe of intensional types in each context.

In the final part of the paper we study the relationship between syntax and semantics, primarily viewing our type system as a basis for a prototypical programming language obtained by giving a simple operational semantics. First, in Section 7, we show that our denotational interpretation of this language is computationally adequate by a new adaptation of admssible logical relations [30] to dependent types. In Section 8, we recast the work of Cartwright, Curien and Felleisen [11] from the simply-typed to the dependently typed case by adding a simple, non-local control operator (`catch`) on intensional terms. This captures evaluation order, allowing us to distinguish operationally any programs which have different denotations (full abstraction), as well as to define all elements in the finite, *total* fragment as terms (full completeness).

3 A Martin Löf Partial Type Theory

We will first describe models of intuitionistic dependent type theory, extended with Booleans and fixed points (on terms) — which one might call "dependent PCF". We fix a set of *universes* and annotate judgements and type constructors with the universe in which they hold — we shall first consider a setting with a single universe in which this annotation appears otiose: in Section 6 we add rules relating the universes of intensional and extensional judgements, allowing dependent and non-dependent types such as lazy natural numbers and vectors of length n to be defined.

The *pseudo-expressions* (types and terms) are given by the following grammar:

$$\begin{aligned} t, T \quad ::= \quad & x \in \mathrm{Var} \mid \mu x.t \mid \Pi_{\mathcal{U}}(x : T).T \mid \lambda x.t \mid t\,t \\ & \mid \Sigma_{\mathcal{U}}(x : T).T \mid \langle t, t\rangle \mid \mathtt{fst}(t) \mid \mathtt{snd}(t) \\ & \mid \mathtt{bool} \mid \mathtt{tt} \mid \mathtt{ff} \mid \mathtt{If}\,t\,\mathtt{then}\,t\,\mathtt{else}\,t \end{aligned}$$

where Var is a set of variables. Upper and lower case lettering is used (informally) to suggest whether an expression denotes a term or a type. We define an equational theory on pseudo-expressions as the least congruence containing the following axioms.

$$\mu x.t = t\,[\mu x.t/x] \quad (\lambda x.s)\,t = s\,[t/x] \quad \lambda x.\,(t\,x) = t\ (x \notin FV(t))$$
$$\langle \mathtt{fst}\,(t), \mathtt{snd}\,(t)\rangle = t \qquad \mathtt{fst}\,\langle t_1, t_2\rangle = t_1 \qquad \mathtt{snd}\,\langle t_1, t_2\rangle = t_2$$
$$\mathtt{If}\,\mathtt{tt}\,\mathtt{then}\,t_1\,\mathtt{else}\,t_2 = t_1 \qquad \mathtt{If}\,\mathtt{ff}\,\mathtt{then}\,t_1\,\mathtt{else}\,t_2 = t_2$$

Pseudo-contexts are finite sequence of pairs of variables and pseudo expressions $x_1 : e_1, \ldots, x_n : e_n$ such that $x_i \neq x_j$ for $i \neq j$. *Judgements* take one of the following forms:

- $\Gamma \vdash_{\mathcal{U}} T$ type where Γ is a pseudo-context, T is a pseudo-expression, and $\mathcal{U}$ is a universe — "T is a type in context Γ, in universe U".
- $\Gamma \vdash_{\mathcal{U}} t : T$, where Γ is a pseudo-context and t and T are pseudo-expressions — "t is a term of type T, in context Γ, in universe $\mathcal{U}$".

A context Γ is well-formed in $\mathcal{U}$ if we may derive $\Gamma \vdash_{\mathcal{U}} \mathtt{bool}$ type. A term derivable without using fixed points is said to be total.

4 An Event-Structure Model

Definition 4.1. An *event structure* is a triple $(|E|, \mathrm{Con}_E, \vdash_E)$, where:

- $|E|$ is a set of *events*,
- $\mathrm{Con}_E \subseteq \mathcal{P}_{fin}(E)$ is a set (closed under $\subseteq$) of sets of events,

$$\frac{}{\vdash_{\mathcal{U}} \mathtt{bool}\,\mathrm{type}} \qquad \frac{\Gamma \vdash_{\mathcal{U}} S\,\mathrm{type} \quad \Gamma, \Gamma' \vdash_{\mathcal{U}} T\,\mathrm{type}}{\Gamma, x : S, \Gamma' \vdash_{\mathcal{U}} T\,\mathrm{type}} \qquad \frac{\Gamma \vdash_{\mathcal{U}} S\,\mathrm{type} \quad \Gamma, x : S \vdash_{\mathcal{U}} T\,\mathrm{type}}{\Gamma \vdash_{\mathcal{U}} \Pi_{\mathcal{U}}\,(x : S)\,.T\,\mathrm{type}} \qquad \frac{\Gamma \vdash_{\mathcal{U}} S\,\mathrm{type} \quad \Gamma, x : S \vdash_{\mathcal{U}} T\,\mathrm{type}}{\Gamma \vdash_{\mathcal{U}} \Sigma_{\mathcal{U}}\,(x : S)\,.T\,\mathrm{type}}$$

Table 1. Derivation Rules for Types

$$\frac{\Gamma \vdash_{\mathcal{U}} T\,\mathrm{type}}{\Gamma, x : T \vdash_{\mathcal{U}} x : T}\,x{\notin}FV(\Gamma) \qquad \frac{\Gamma \vdash_{\mathcal{U}} S\,\mathrm{type} \quad \Gamma, \Gamma' \vdash_{\mathcal{U}} t : T}{\Gamma, x : S, \Gamma' \vdash_{\mathcal{U}} t : T}\,x{\notin}FV(\Gamma,\Gamma') \qquad \frac{\Gamma \vdash_{\mathcal{U}} t : T_1 \quad \Gamma \vdash_{\mathcal{U}} T_2\,\mathrm{type}}{\Gamma \vdash_{\mathcal{U}} t : T_2}\,T_1{=}T_2 \qquad \frac{\Gamma, x : S \vdash_{\mathcal{U}} t : T}{\Gamma \vdash_{\mathcal{U}} \lambda x : S.t : \Pi_{\mathcal{U}}\,(x : S)\,.T}$$

$$\frac{\Gamma \vdash_{\mathcal{U}} s : S \quad \Gamma \vdash_{\mathcal{U}} t : T\,[s/x] \quad \Gamma, x : S \vdash_{\mathcal{U}} T\,\mathrm{type}}{\Gamma \vdash_{\mathcal{U}} \langle s, t\rangle : \Sigma_{\mathcal{U}}\,(x : S)\,.T} \qquad \frac{\Gamma \vdash_{\mathcal{U}} t : \Sigma_{\mathcal{U}}\,(x : S)\,.T}{\Gamma \vdash_{\mathcal{U}} \mathtt{fst}\,(t) : S} \qquad \frac{\Gamma \vdash_{\mathcal{U}} t : \Sigma_{\mathcal{U}}\,(x : S)\,.T}{\Gamma \vdash_{\mathcal{U}} \mathtt{snd}\,(t) : T\,[\mathtt{fst}\,(t)\,/x]} \qquad \frac{\Gamma \vdash_{\mathcal{U}} t : \Pi_{\mathcal{U}}\,(x : S)\,.T \quad \Gamma \vdash_{\mathcal{U}} s : S}{\Gamma \vdash_{\mathcal{U}} t\,s : T\,[s/x]}$$

$$\frac{}{\vdash_{\mathcal{U}} \mathtt{tt} : \mathtt{bool}} \qquad \frac{}{\vdash_{\mathcal{U}} \mathtt{ff} : \mathtt{bool}} \qquad \frac{\Gamma \vdash_{\mathcal{U}} s : \mathtt{bool} \quad \Gamma, x : \mathtt{bool} \vdash_{\mathcal{U}} T\,\mathrm{type} \quad \Gamma \vdash_{\mathcal{U}} t_1 : T\,[\mathtt{tt}/x] \quad \Gamma \vdash_{\mathcal{U}} t_2 : T\,[\mathtt{ff}/x]}{\Gamma \vdash_{\mathcal{U}} \mathtt{If}\,s\,\mathtt{then}\,t_1\,\mathtt{else}\,t_2 : T\,[s/x]} \qquad \frac{\Gamma, x : T \vdash_{\mathcal{U}} t : T}{\Gamma \vdash_{\mathcal{U}} \mu x.t : T}$$

Table 2. Derivation Rules for Terms

- $\vdash_E \subseteq \mathrm{Con}_E \times |E|$ is a relation (*enabling*) such that if $X \vdash_E e$ then $X \cup \{e\} \in \mathrm{Con}_E$.

A proof of an event e is a sequence $e_0, \ldots, e_n$ of events such that $e_n = e$ and for each $i \leq n$ there exists $X \subseteq \{e_0, \ldots, e_{i-1}\}$ such that $X \vdash_E e_i$.

For example, the event structure of Booleans has two events: t and f, the consistent sets are $\emptyset$, $\{t\}$ and $\{f\}$, and the enabling relation is defined by $\emptyset \vdash t$ and $\emptyset \vdash f$.

Definition 4.2. A *state* of an event structure E is a set of events $x \subseteq E$ which satisfies:

- Consistency: If $X \subseteq_{fin} x$ then $X \in \mathrm{Con}_E$.
- Safety : If $e \in x$ then there is a proof of e in x.

E is *stable* if for each state x of E and each event $e \in x$, e has a unique enabling in x.

A stable event structures is the concrete representations of a dI-domain in the following sense:

Proposition 4.3 ([6, 32])**.** *If E is a stable event structure then $D(E)$ ordered with $\subseteq$ is a dI-domain. Moreover, every dI-domain is order-isomorphic to $(D(E), \subseteq)$ for some event structure E.*

Thus we may form a category in which objects are event structures and morphisms from E to E' are stable functions from $D(E)$ to $D(E')$.

4.1 Dependent Event Structures

We may form a dI-domain $\mathbf{Ev}$ of all stable event structures[2].

Definition 4.4. For stable event structures E, E', define $E \lhd E'$ if $|E| \subseteq |E'|$ and:

- If $e \in E$ and $X \vdash_{E'} e$ then $X \subseteq E$ and $X \vdash_E e$.
- If $X \subseteq_{fin} |E|$ then $X \in \mathrm{Con}_E \Leftrightarrow X \in \mathrm{Con}_{E'}$.

It is straightforward to show that this is a directed and bounded complete partial order in which the compact elements are the finite stable event structures, and thus:

Proposition 4.5. $\mathbf{Ev} = (|\mathbf{Ev}|, \lhd)$ *is a dI-domain, where* $|\mathbf{Ev}|$ *is the set of all stable event structures.*

Definition 4.6 ($\mathbf{Ev}$-parametrization)**.** An $\mathbf{Ev}$-parametrization over a stable event structure E is a stable function $F : D(E) \to \mathbf{Ev}$.

[2]To avoid Russell's paradox, we shall assume that the collection of all events forms a set, which contains various other sets we need and is closed under disjoint union, product, finite powerset, etc.

In particular, for $E, E' \in \mathbf{Ev}$ there is a constant parametrization on E mapping every $x \in D(E)$ to E'.

Definition 4.7. Given an $\mathbf{Ev}$-parametrization F over E, define the *dependent product* $\Pi_{\mathcal{E}}(E, F)$ as follows:

- $|\Pi_{\mathcal{E}}(E, F)| = \left\{(x, e) \in D(E) \times \bigcup_{x \in D(E)} F(x) \,\middle|\, e \in F(x)\right\}$
- $\{(x_i, e_i)\}_{i \in I} \in \mathrm{Con}_{\Pi_{\mathcal{E}}(E,F)}$ if and only if:
  - $\bigcup_{i \in I} x_i \in \mathrm{Con}_E$ implies $\{e_i\}_{i \in I} \in \mathrm{Con}_{F(\bigcup_{i \in I} x_i)}$
  - $e_i = e_j$ and $x_i \neq x_j$ implies $x_i \not\uparrow x_j$,
- $\{(x_i, e_i)\}_{i \in I} \vdash_{\Pi_{\mathcal{E}}(E,F)} (x, e)$ if and only if:
  - x min s.t. $\bigcup_{i \in I} x_i \subseteq x$ and $e \in F(x)$
  - $\{e_i\}_{i \in I} \vdash_{F(x)} e$.

We give an alternative characterization of states of the dependent product as stable functions:

Definition 4.8. Given an $\mathbf{Ev}$-parametrization F over E, we define a *dependent stable function* on F as a stable function f from $D(E)$ to $\bigcup_{x \in D(E)} D(F(x))$ such that $\forall x \in D(E), f(x) \in D(F(x))$. These form a dI-domain with the the usual stable ordering:

$$f \leq g \qquad \text{iff} \qquad \forall x, y \in D(E), x \subseteq y \Rightarrow f(x) = f(y) \cap g(x)$$

Proposition 4.9. *Given an* $\mathbf{Ev}$*-parametrization F over E, there is an order-isomorphism between the set of dependent stable functions on F and $D(\Pi_{\mathcal{E}}(E, F))$. This isomorphism is given by:*

- *for f dependent stable function on F:*

$$\mathsf{tr}(f) = \{(x, e) \mid x \text{ min } s.t.\ e \in f(x)\}$$

- *for $\sigma \in D(\Pi_{\mathcal{E}}(E, F))$:*

$$\mathsf{fun}(\sigma)(x) = \{e \mid \exists y \subseteq x, (y, e) \in \sigma\}$$

In the following, we will use the two characterizations indifferently and write $x : \Pi_{\mathcal{E}}(E, F)$ for a state of $D(\Pi_{\mathcal{E}}(E, F))$ as well as for a dependent stable function on F. In the particular case of constant parametrization $F : x \in D(E) \mapsto E'$, $D(\Pi_{\mathcal{E}}(E, F))$ is thus isomorphic to the usual domain of stable functions from $D(E)$ to $D(E')$, and we may write $E \Rightarrow_{\mathcal{E}} E'$ instead of $\Pi_{\mathcal{E}}(E, F)$.

Definition 4.10. Given a $\mathbf{Ev}$-parametrization F over E, define the *dependent sum* $\Sigma_{\mathcal{E}}(E, F)$ as follows:

- $|\Sigma_{\mathcal{E}}(E, F)| = |E| \uplus \{(x, e) \mid x \text{ min s.t. } e \in F(x)\}$
- $\{d_i\}_{i \in I} \cup \left\{(x_j, e_j)\right\}_{j \in J} \in \mathrm{Con}_{\Sigma_{\mathcal{E}}(E,F)}$ if and only if:

$$\{d_i\}_{i \in I} \cup \bigcup_{j \in J} x_j \in \mathrm{Con}_E \quad \text{and} \quad \left\{e_j\right\}_{j \in J} \in \mathrm{Con}_{F\left(\bigcup_{j \in J} x_j\right)}$$

- $\{d_i\}_{i \in I} \vdash_{\Sigma_{\mathcal{E}}(E,F)} d$ if and only if $\{d_i\}_{i \in I} \vdash_E d$

- $x \cup \{(x_j, e_j)\}_{j\in J} \vdash_{\Sigma_{\mathcal{E}}(E,F)} (x, e)$ if and only if:
$$\bigcup_{j\in J} x_j \subseteq x \quad \text{and} \quad \{e_j\}_{j\in J} \vdash_{F(x)} e$$

Events in the right component have the form (x, e) so we can distinguish (x, e) from (x', e) when $x \not\uparrow x'$.

We give an alternative characterization of the states of the dependent sum as pairs of states:

Definition 4.11. Given an **Ev**-parametrization F over E, a *dependent pair* on F is a pair $(x, x') \in D(E) \times \bigcup_{x\in D(E)} D(F(x))$ such that $x' \in D(F(x))$.

Proposition 4.12. *Given an* **Ev**-*parametrization* F *over* E*, there is an order-isomorphism between the set of dependent pairs on* F *(ordered pointwise) and* $D(\Sigma_{\mathcal{E}}(E,F))$*. This isomorphism is given by:*

- *If* (x, x') *is a dependent pair on* F *then the following is a state on* $\Sigma_{\mathcal{E}}(E,F)$

$$x \cup \{(y, e') \mid e' \in x' \text{ and } y \min \text{ s.t. } y \subseteq x \text{ and } e' \in |F(y)|\}$$

- *If* $\{d_i\}_{i\in I} \cup \{(x_i, e_i)\}_{j\in J} \in D(\Sigma_{\mathcal{E}}(E,F))$ *then the following is a dependent pair on* F*:*

$$\left(\{d_i\}_{i\in I}, \{e_j\}_{j\in J}\right)$$

In the following, we will use the two characterizations indifferently and write $x : \Sigma_{\mathcal{E}}(E,F)$ for a state of $D(\Sigma_{\mathcal{E}}(E,F))$ as well as for a dependent pair on F. In the particular case of constant parametrization $F : x \in D(E) \mapsto E'$, $D(\Sigma_{\mathcal{E}}(E,F))$ is thus isomorphic to the usual product of the domains $D(E)$ and $D(E')$ with the pairwise ordering, so we may write $E \times_{\mathcal{E}} E'$ instead of $\Sigma_{\mathcal{E}}(E,F)$.

We can precompose parametrizations with stable functions and stable dependent functions with stable functions:

Proposition 4.13. *If* F *is an* **Ev**-*parametrization over* E' *and* $f : E \Rightarrow E'$*, then* $F \circ f$ *is an* **Ev**-*parametrization over* E*. Moreover, if* $f' : \Pi_{\mathcal{E}}(E', F)$*, then* $f' \circ f : \Pi_{\mathcal{E}}(E, F \circ f)$*.*

The following notions provide support for dependent types in contexts:

Definition 4.14. Let F be an **Ev**-parametrization over E and G be an **Ev**-parametrization over $\Sigma_{\mathcal{E}}(E,F)$. Then for every $x : E$, $y \mapsto G(x,y)$ is a parametrization over $F(x)$ so we can define the following **Ev**-parametrizations over E:

- $\Sigma_{\mathcal{E}}(F,G) : x \mapsto \Sigma_{\mathcal{E}}(F(x), y \mapsto G(x,y))$
- $\Pi_{\mathcal{E}}(F,G) : x \mapsto \Pi_{\mathcal{E}}(F(x), y \mapsto G(x,y))$

4.2 Category with Families Interpretation

We may give a formal interpretation of dependent PCF based on the "categories with families" interpretation of Dybjer [15].

Definition 4.15. Let Fam be the category of set-indexed families of sets. A *category with families* (CwF) is given by:

- A base category $\mathcal{C}_{\mathcal{U}}$
- A functor $\mathcal{F}_{\mathcal{U}} : \mathcal{C}_{\mathcal{U}}^{op} \to \mathsf{Fam}$. For Γ an object of $\mathcal{C}_{\mathcal{U}}$ we write $\mathcal{F}_{\mathcal{U}}(\Gamma) = (\mathsf{Tm}_{\mathcal{U}}(\Gamma \vdash T))_{T\in\mathsf{Ty}_{\mathcal{U}}(\Gamma)}$ and for $\gamma \in \mathcal{C}_{\mathcal{U}}(\Gamma', \Gamma)$, $T \in \mathsf{Ty}_{\mathcal{U}}(\Gamma)$ and $t \in \mathsf{Tm}_{\mathcal{U}}(\Gamma \vdash T)$ we write $\mathcal{F}_{\mathcal{U}}(\gamma)(T) = T[\gamma]$ and $\mathcal{F}_{\mathcal{U}}(\gamma)(t) = t[\gamma]$.
- A terminal object $[]$ of $\mathcal{C}_{\mathcal{U}}$ and for Γ object of $\mathcal{C}_{\mathcal{U}}$ and $T \in \mathsf{Ty}_{\mathcal{U}}(\Gamma)$ an object $\Gamma \cdot T$ of $\mathcal{C}_{\mathcal{U}}$ together with a morphism $p \in \mathcal{C}_{\mathcal{U}}(\Gamma \cdot T, \Gamma)$ and an element $q \in \mathsf{Tm}_{\mathcal{U}}(\Gamma \cdot T \vdash T[p])$ satisfying:
$$\forall \gamma \in \mathcal{C}_{\mathcal{U}}(\Gamma', \Gamma), \forall T \in \mathsf{Ty}_{\mathcal{U}}(\Gamma), \forall a \in \mathsf{Tm}_{\mathcal{U}}(\Gamma' \vdash T[\gamma])$$
$$\exists! \delta \in \mathcal{C}_{\mathcal{U}}(\Gamma', \Gamma \cdot T), p \circ \delta = \gamma \text{ and } q[\delta] = a$$

For a given universe $\mathcal{U}$, objects of $\mathcal{C}_{\mathcal{U}}$ represent contexts over $\mathcal{U}$ and morphisms represent substitutions between them. Types in context Γ in $\mathcal{U}$ are represented as elements of the set $\mathsf{Ty}_{\mathcal{U}}(\Gamma)$ and terms of type T in context Γ as elements of the set $\mathsf{Tm}_{\mathcal{U}}(\Gamma \vdash T)$. The terminal object $[]$ of $\mathcal{C}_{\mathcal{U}}$ represents the empty context and $\Gamma \cdot T$ the extension of context Γ with type T. Substitution p interprets weakening through projection and term q interprets the axiom rule.

The action of $\mathcal{F}_{\mathcal{U}}$ on morphisms describes the action of substitution on types and terms — it sends a morphism $\gamma \in \mathcal{C}_{\mathcal{U}}(\Gamma', \Gamma)$ in $\mathcal{C}_{\mathcal{U}}$ to a pair consisting of a reindexing function sending $T \in \mathsf{Ty}_{\mathcal{U}}(\Gamma)$ to $T[\gamma] \in \mathsf{Ty}_{\mathcal{U}}(\Gamma')$ and a family of maps sending $t \in \mathsf{Tm}_{\mathcal{U}}(\Gamma \vdash T)$ to $t[\gamma] \in \mathsf{Tm}_{\mathcal{U}}(\Gamma' \vdash T[\gamma])$.

CwF of stable event structures. The base category of the extensional CwF is the category of stable event structures and stable functions. For each event structure E, $\mathcal{F}_{\mathcal{E}}(E)$ is given by:

- $\mathsf{Ty}_{\mathcal{E}}(E)$ is the set of **Ev**-parametrizations over E,
- for each $F \in \mathsf{Ty}_{\mathcal{E}}(E)$, $\mathsf{Tm}_{\mathcal{E}}(E \vdash F) \triangleq D(\Pi_{\mathcal{E}}(E,F))$.

For $\gamma : E' \Rightarrow E$, $\mathcal{F}_{\mathcal{E}}(\gamma)$ sends

- $F \in \mathsf{Ty}_{\mathcal{E}}(E)$ to $F[\gamma] \triangleq F \circ \gamma \in \mathsf{Ty}(E')$
- $f \in \mathsf{Tm}_{\mathcal{E}}(E \vdash_{\mathcal{E}} F)$ to $f[\gamma] \triangleq f \circ \gamma \in \mathsf{Tm}_{\mathcal{E}}(E' \vdash_{\mathcal{E}} F[\gamma])$.

The empty context is the empty event structure, and context extension sends an event-structure E and parametrization F over E to the event structure $\Sigma_{\mathcal{E}}(E,F)$, with projections $p : \Sigma_{\mathcal{E}}(E,F) \Rightarrow E$ and $q : \Pi_{\mathcal{E}}(\Sigma(E,F), F \circ p)$ given by the set-theoretic projections, using the characterizations of Definitions 4.8 and 4.11.

Π and Σ types. To interpret dependent type theory, our categories with families require Π and Σ types — i.e. for each context Γ (object in our base category) and types $T \in \mathsf{Ty}_{\mathcal{U}}(\Gamma)$ and $U \in \mathsf{Ty}_{\mathcal{U}}(\Gamma \cdot T)$, we have types $\Pi_{\mathcal{U}}(T,U)$ and $\Sigma_{\mathcal{U}}(T,U)$ in $\mathsf{Ty}(\Gamma)$, with operations taking:

- $t \in \mathsf{Tm}_{\mathcal{U}}(\Gamma \cdot T \vdash U)$ to $\lambda(t) \in \mathsf{Tm}_{\mathcal{U}}(\Gamma \vdash \Pi_{\mathcal{U}}(T,U))$
- $t \in \mathsf{Tm}_{\mathcal{U}}(\Gamma \vdash \Pi_{\mathcal{U}}(T,U))$ and $u \in \mathsf{Tm}_{\mathcal{U}}(\Gamma \vdash T))$ to $\mathsf{App}(t,u) \in \mathsf{Tm}_{\mathcal{U}}(\Gamma \vdash U[\langle \mathsf{id}_\Gamma, u\rangle])$
- $t \in \mathsf{Tm}_{\mathcal{U}}(\Gamma \vdash T)$ and $u \in \mathsf{Tm}_{\mathcal{U}}(\Gamma \vdash U[\langle \mathsf{id}_\Gamma, t\rangle])$ to $\mathsf{Pair}(t,u) \in \mathsf{Tm}_{\mathcal{U}}(\Gamma \vdash \Sigma_{\mathcal{U}}(T,U))$
- $t \in \mathsf{Tm}_{\mathcal{U}}(\Gamma \vdash \Sigma(T,U))$ to $\mathsf{Fst}(t) \in \mathsf{Tm}_{\mathcal{U}}(\Gamma \vdash T)$ and $\mathsf{Snd}(t) \in \mathsf{Tm}_{\mathcal{U}}(\Gamma \vdash U[\langle \mathsf{id}_\Gamma, \mathsf{Fst}(t)\rangle])$

satisfying equations specifying β- and η-conversion, as well as the action of substitutions on these constructs, see e.g. [13]. In the CwF of stable event structures, $\Pi_{\mathcal{E}}$ and $\Sigma_{\mathcal{E}}$ are those of Definition 4.14, λ is currying, App, Pair, Fst and Snd are the corresponding set-theoretic operations.

Semantics of Dependent PCF A CwF has *fixed points* if there is an operation taking $t \in \mathsf{Tm}_{\mathcal{U}}(\Gamma \cdot A \vdash A[p])$ (where $A \in \mathsf{Ty}_{\mathcal{U}}(\Gamma)$) to $\mathsf{Rec}(t) \in \mathsf{Tm}_{\mathcal{U}}(\Gamma \vdash A)$ such that:

$$\mathsf{Rec}(t) = t[\langle \mathsf{id}_\Gamma, \mathsf{Rec}(t)\rangle] \qquad \mathsf{Rec}(t)[\gamma] = \mathsf{Rec}(t[\langle \gamma \circ p, q\rangle])$$

The extensional CwF has fixed points: if $f : \Pi_{\mathcal{E}}(\Sigma_{\mathcal{E}}(E,F), F \circ \pi_1)$, $\mathsf{Rec}(f) : \Pi_{\mathcal{E}}(E,F)$ maps x to $\bigvee_{n\in\mathbb{N}} f_x^n(\emptyset)$ where $f_x : F(x) \Rightarrow F(x)$ is $y \mapsto f(x,y)$.

To interpret dependent PCF in a CwF we also require interpretations of the Booleans as a type $\mathbb{B} \in \mathsf{Ty}_{\mathcal{U}}([])$ with elements $\mathtt{tt}, \mathtt{ff} \in \mathsf{Tm}_{\mathcal{U}}(\vdash \mathbb{B})$ and the *conditional*, as an operation taking a type $T \in \mathsf{Ty}_{\mathcal{U}}(\Gamma \cdot \mathbb{B}_\Gamma)$ and terms $t_1 \in \mathsf{Tm}_{\mathcal{U}}(\Gamma \vdash T[\langle \mathtt{id}_\Gamma, \mathtt{tt}_\Gamma \rangle])$ and $t_2 \in \mathsf{Tm}_{\mathcal{U}}(\Gamma \vdash T[\langle \mathtt{id}_\Gamma, \mathtt{ff}_\Gamma \rangle])$ to $\mathtt{If}(t_1, t_2) \in \mathsf{Tm}_{\mathcal{U}}(\Gamma \cdot \mathbb{B}_\Gamma \vdash T)$, which satisfies:

$$\mathtt{If}(t_1,t_2)[\langle \mathtt{id}_\Gamma, \mathtt{tt}_\Gamma\rangle] = t_1 \qquad \mathtt{If}(t_1,t_2)[\langle \mathtt{id}_\Gamma, \mathtt{ff}_\Gamma\rangle] = t_2$$
$$\mathtt{If}(t_1,t_2)[\gamma] = \mathtt{If}(t_1[p \circ \gamma], t_2[p\circ\gamma])[\langle \mathtt{id}_\Gamma, q[\gamma]\rangle]$$

where $\mathbb{B}_\Gamma \triangleq \mathbb{B}[1_\Gamma]$, $\mathtt{tt}_\Gamma \triangleq \mathtt{tt}[1_\Gamma]$ and $\mathtt{ff}_\Gamma \triangleq \mathtt{ff}[1_\Gamma]$ if $1_\Gamma : \Gamma \to []$.

In our CwF of stable event structures, $\mathbb{B}$ is the constant parametrization which returns the event structure of Booleans, and $\mathtt{tt}$ and $\mathtt{ff}$ the corresponding states. For $f_1 : \Pi_{\mathcal{E}}(E, x \mapsto F(x, \{t\}))$ and $f_2 : \Pi_{\mathcal{E}}(E, x \mapsto F(x, \{f\}))$, $\mathtt{If}(f_1, f_2)$ maps $(x, \emptyset)$ to $\emptyset$, $(x, \{t\})$ to $f_1(x)$ and $(x, \{f\})$ to $f_2(x)$.

Given a CwF $(\mathcal{C}_{\mathcal{U}}, \mathcal{F}_{\mathcal{U}})$ with Π and Σ types, Booleans, conditionals and fixpoints, we interpret (by induction on derivation):

- Each well-formed context Γ as a (unique) object $[\![\Gamma]\!]_{\mathcal{U}}$ of $\mathcal{C}_{\mathcal{U}}$ (in our extensional universe, an event structure).
- Each type in context $\Gamma \vdash_{\mathcal{U}} T$ type as a (unique) element $[\![T]\!]^\Gamma_{\mathcal{U}}$ of $\mathsf{Ty}_{\mathcal{U}}([\![\Gamma]\!]_{\mathcal{U}})$ (an **Ev**-parametrization over $[\![\Gamma]\!]_{\mathcal{E}}$).
- Each term in context $\Gamma \vdash_{\mathcal{U}} t : T$ as an element $[\![t : T]\!]^\Gamma_{\mathcal{U}}$ of $\mathsf{Tm}_{\mathcal{U}}\left([\![\Gamma]\!]_{\mathcal{U}} \vdash [\![T]\!]^\Gamma_{\mathcal{U}}\right)$ (a state in $D\left(\Pi_{\mathcal{E}}\left([\![\Gamma]\!]_{\mathcal{E}}, [\![T]\!]^\Gamma_{\mathcal{E}}\right)\right)$).

5 Concrete Data Structures

Our extensional model of dependent type theory refines the domain theoretic semantics [29] in the sense that it excludes elements such as parallel-or. However, it is not sequential, retaining parallel elements such as "Gustave's function" [8]. Nor is there a direct characterization of the total elements of the model. We now formally describe the intensional model sketched in Section 2, which does have these properties. Types and terms will denote *concrete data structures* and *sequential algorithms*, which generalize graph games and strategies. By presenting them within the framework of event structures, we relate them to the extensional model.

Definition 5.1. A concrete data structure (CDS) A is an event structure $(|A|, \mathrm{Con}_A, \vdash_A)$, together with a polarization $|A| = |A|^- \uplus |A|^+$ of its events such that:

- if $X \vdash_A e \in |A|^-$ then $X \subseteq |A|^+$,
- if $X \vdash_A e \in |A|^+$ then X^- is a singleton,
- $X \in \mathrm{Con}_A$ if and only if for all $e_1, e_2 \in X^+$ and $X_1, X_2 \subseteq X$, if $X_1 \vdash_A e_1$, $X_2 \vdash_A e_2$ and $X_1^- = X_2^-$ then $e_1 = e_2$ (so the consistency relation is implicit in the polarization and enabling relation).

We may subsume the graph games defined in Section 2 into this definition by observing that the former correspond precisely to the *filliform* CDSs, in which enabling sets consist of at most one event (i.e. edges are enablings, and we remove the opening position $*$).

In the original terminology of CDSs [21], negative events are called *cells* and positive events are called *decisions* and correspond to the *filling* of a cell with a *value*[3], and states consist only of filled cells. We recast the latter as *positive* states of our event structure CDSs, while those with exactly one unfilled cell are deemed *negative* — i.e. a state x of a stable CDS is:

[3]Note that our definition is more general in that it allows the filling relation to depend on other events, which is necessary for the modelling of dependent sums.

- *positive* if for any $e \in x^-$ there exists $X \subseteq x$ and $e' \in x^+$ such that $X, e \vdash e'$.
- *negative* if there exists a unique $e \in x^-$ such that there is no $X \subseteq x$ and $e' \in x^+$ satisfying $X, e \vdash e'$.
- *total* if for any $X \subseteq_{fin} x$, $X \vdash e \in |A|^-$ implies $e \in x$.

We write $D(A)^+$ (resp. $D(A)^-$) for the set of positive (resp. negative) states of A. The positive states of a filiform CDS correspond exactly to the strategies on the corresponding graph game.

The CDS for Booleans is as defined in Section 2, its only negative state is $\{?\}$ and its positive states are $\emptyset$, $\{?, t\}$ and $\{?, f\}$.

5.1 Dependent Concrete Data Structures

We now define **CDS**-parametrizations similarly to parametrizations over event structures. Since CDSs' events are polarized, we refine the order on stable event structures into an order on stable CDSs: $A \blacktriangleleft B$ if $A \lhd B$, $|A|^+ = |B|^+ \cap |A|$ and $|A|^- = |B|^- \cap |A|$. Once again we can form the large dI-domain of all stable CDSs:

Proposition 5.2. $\mathbf{CDS} = (|\mathbf{CDS}|, \blacktriangleleft)$ *is a dI-domain, where* $|\mathbf{CDS}|$ *is the set of all stable CDSs.*

Moreover, the set of positive states of a stable CDS is a dI-domain, so we may define a **CDS**-parametrization over a stable CDS A to be a stable map from $D(A)^+$ to **CDS**. If P is a **CDS**-parametrization over A then dependent products and sums are defined as follows:

Definition 5.3. The *dependent product* $\Pi_{\mathcal{I}}(A, P)$ is defined by:

- $|\Pi_{\mathcal{I}}(A,P)|^+ = \left\{ (x, e) \in D(A) \times \bigcup_{x \in D(A)^+} \,\middle|\, \begin{array}{l}\text{x,e positive} \\ \text{or x,e negative}\end{array} \right\}$
- $|\Pi_{\mathcal{I}}(A,P)|^- = \left\{ (x, e) \in D(A) \times \bigcup_{x \in D(A)^+} \,\middle|\, \begin{array}{l}\text{x positive} \\ \text{and e negative}\end{array} \right\}$
- If:
  - $\{(x_i, e_i)\}_{i \in I} \in |\Pi_{\mathcal{I}}(A,P)|$ with x_i positive
  - $x \in D(A)^+$ min s.t. $\bigcup_{i \in I} x_i \subseteq x$ and $e \in P(x)$
  - $\{e_i\}_{i \in I} \vdash_{P(x)} e$

  then $\{(x_i, e_i)\}_{i\in I} \vdash_{\Pi_{\mathcal{I}}(A,P)} (x, e)$
- if $(x, e), (x \cup \{f\}, e) \in |\Pi_{\mathcal{I}}(A,P)|$
  then $\{(x, e)\} \vdash_{\Pi_{\mathcal{I}}(A,P)} (x \cup \{f\}, e)$

A sequential algorithm on P is a positive state of $\Pi_{\mathcal{I}}(A,P)$, that is, an element of $D(\Pi_{\mathcal{I}}(A,P))^+$.

We write $a : \Pi_{\mathcal{I}}(A,P)$ when a is a sequential algorithm on P. If P is constant with value B we write $A \Rightarrow_{\mathcal{I}} B$ instead of $\Pi_{\mathcal{I}}(A,P)$. In the terminology of [14], a state of $\Pi_{\mathcal{I}}(A,P)$ is an observable sequential algorithm. It can be shown that if $a : A \Rightarrow_{\mathcal{I}} B$ then $\mathrm{fun}(a)$ (see Proposition 4.9) is a stable function from $D(A)$ to $D(B)$. Moreover, the restriction $\mathrm{fun}^+(a)$ of $\mathrm{fun}(a)$ to $D(A)^+$ takes values in $D(B)^+$, it is therefore a stable function from $D(A)^+$ to $D(B)^+$.

Definition 5.4. The *dependent sum* $\Sigma_{\mathcal{I}}(A,P)$ is defined by:

- $|\Sigma_{\mathcal{I}}(A,P)| = |A| \uplus \left\{(x, e) \,\middle|\, x \in D(A)^+ \text{ min s.t. } e \in P(x)\right\}$
- polarities in $|A|$ are inherited, polarity of (x, e) is that of e
- $\{d_i\}_{i\in I} \vdash_{\Sigma_{\mathcal{I}}(E,F)} d$ if and only if $\{d_i\}_{i \in I} \vdash_E d$
- $x \cup \left\{(x_j, e_j)\right\}_{j \in J} \vdash_{\Sigma_{\mathcal{I}}(E,F)} (x, e)$ if and only if:

$$\bigcup_{j \in J} x_j \subseteq x \quad \text{and} \quad \left\{e_j\right\}_{j \in J} \vdash_{F(x)} e$$

We handle precomposition of parametrizations and sequential algorithms using the fact that if $a : A \Rightarrow_{\mathcal{I}} A'$ then $\mathrm{fun}^+(a)$ is a stable function from $D(A)^+$ to $D(A')^+$:

$$\frac{\Gamma \vdash_{\mathcal{I}} T \text{ type}}{\Gamma \vdash_{\mathcal{E}} T \text{ type}} \qquad \frac{\Gamma \vdash_{\mathcal{I}} t : T}{\Gamma \vdash_{\mathcal{E}} t : T}$$

$$\frac{}{\vdash_{\mathcal{E}} \mathcal{I} \text{ type}} \qquad \frac{}{\vdash_{\mathcal{E}} \mathsf{bool} : \mathcal{I}} \qquad \frac{\Gamma \vdash_{\mathcal{I}} \mathsf{bool}\,\mathsf{type} \quad \Gamma \vdash_{\mathcal{E}} T : \mathcal{I}}{\Gamma \vdash_{\mathcal{I}} T \text{ type}}$$

$$\frac{\Gamma \vdash_{\mathcal{E}} S : \mathcal{I} \quad \Gamma, x : S \vdash_{\mathcal{E}} T : \mathcal{I}}{\Gamma \vdash_{\mathcal{E}} \Pi_{\mathcal{I}}\,(x : S)\,.T : \mathcal{I}} \qquad \frac{\Gamma \vdash_{\mathcal{E}} S : \mathcal{I} \quad \Gamma, x : S \vdash_{\mathcal{E}} T : \mathcal{I}}{\Gamma \vdash_{\mathcal{E}} \Sigma_{\mathcal{I}}\,(x : S)\,.T : \mathcal{I}}$$

Table 3. Rules for the Semantic Universe of Intensional Types

Proposition 5.5. *If P is a* **CDS**-*parametrization over A' and $a : A \Rightarrow_{\mathcal{I}} A'$, then $P \circ \mathsf{fun}^+(a)$ is a* **CDS**-*parametrization over A. Moreover, if $a' : \Pi_{\mathcal{I}}(A', P)$, then $\mathsf{tr}(\mathsf{fun}(a') \circ \mathsf{fun}(a)) : \Pi_{\mathcal{I}}(E, F \circ f)$.*

The Intensional Category with Families. The intensional category with families is given by the pair $(\mathcal{C}_{\mathcal{I}}, \mathcal{I})$, where $\mathcal{C}_{\mathcal{I}}$ is the category of stable CDS and sequential algorithms, and $\mathcal{F}_{\mathcal{I}} : \mathcal{C}_{\mathcal{I}}^{op} \to \mathsf{Fam}$ is defined on objects by:

- $\mathsf{Ty}_{\mathcal{I}}(A)$ is the set of **CDS**-parameterizations over A.
- for each $P \in \mathsf{Ty}_{\mathcal{I}}(A)$, $\mathsf{Tm}_{\mathcal{I}}(A \vdash P) \triangleq D(\Pi_{\mathcal{I}}(A, P))^+$

For $\gamma : A' \Rightarrow_{\mathcal{I}} A$, $\mathcal{F}_{\mathcal{I}}(\gamma)$ sends

- $P \in \mathsf{Ty}_{\mathcal{I}}(A)$ to $P[\gamma] \triangleq P \circ \mathsf{fun}^+(\gamma)$,
- $a \in \mathsf{Tm}_{\mathcal{I}}(A \vdash_{\mathcal{I}} P)$ to $a[\gamma] \triangleq \mathsf{tr}(\mathsf{fun}(a) \circ \mathsf{fun}(\gamma)) \in \mathsf{Tm}_{\mathcal{I}}(A' \vdash_{\mathcal{I}} P[\gamma])$

The empty context is the empty CDS and context extension sends a stable CDS A and parametrization P over A to the stable CDS $\Sigma_{\mathcal{I}}(A, P)$. With an equivalent of Definition 4.11 we can build set-theoretic projections. Since their traces are sequential algorithms on the appropriate CDSs, we use them for building the intensional CwF. Booleans are interpreted with the CDS given above and the operations for Booleans and fixed points are obtained from the set-theoretic ones, the traces of which are sequential algorithms. The intensional CwF is therefore a model of dependent PCF.

6 A Universal Type of Intensional Types

Having described interpretations of dependent PCF in two different semantic universes we now add typing rules for relating these interpretations, allowing a rich variety of recursive type families to be expressed in our system — which might therefore be called "dependent FPC". In particular, we may type each of the examples which were given in Section 2 — the lifted sum, lazy natural numbers and dependent type of vectors of length n.

Letting the set of universes be $\{\mathcal{E}, \mathcal{I}\}$, we extend dependent PCF with the rules of Table 3, giving two universes (à la Russell), of intensional and extensional types and terms. The first pair of rules "lift" type and term judgements from the intensional to the extensional universe (a form of *cumulativity*). The next three rules state that there is a type of intensional types in the extensional universe, of which `bool` is an element, and that any term of this type in an intensional context is an intensional type. Finally, we add extensional term formation rules for the intensional type operators.[4]

We may compare our type-theory with Barendregt's λ-cube by taking $\mathcal{I}$ to be the sort of terms and $\mathcal{E}$ to be the sort of types [7]. It lacks polymorphism (terms depending on types) but can express both the *type-families* of λP or LF (types depending on terms), and the *type-operators* of λ_ω or Haskell (types depending on types). However, it does not combine these in the same way as the λ-cube due to the distinction between intensional and extensional typing judgments.

6.1 Interpretation of Dependent FPC

Having described extensional and intensional interpretations of dependent PCF in the CwFs $(\mathcal{C}_{\mathcal{E}}, \mathcal{F}_{\mathcal{E}})$ and $(\mathcal{C}_{\mathcal{I}}, \mathcal{F}_{\mathcal{I}})$ we now relate them, leading to interpretations of the rules of Table 3. To lift judgments from the intensional to the extensional universe, we require a map between CwFs.

Definition 6.1 ([13]). A (weak) morphism between CwFs $(\mathcal{C}_1, \mathcal{F}_1)$ and $(\mathcal{C}_2, \mathcal{F}_2)$ is given by a functor between the base categories $G : \mathcal{C}_1 \to \mathcal{C}_2$, together with a natural transformation $\phi : \mathcal{F}_1 \to \mathcal{F}_2 \circ G^{op}$ which preserves empty context and context formation — i.e. for any object C of $\mathcal{C}_1$ and type $T \in \mathsf{Ty}_1(C)$, $G(C \cdot T) \cong G(C) \cdot \Phi_C(T)$. (Where $\Phi_C : \mathsf{Ty}_1(C) \to \mathsf{Ty}_2(G(C))$ is the reindexing component of $\phi_C : \mathcal{F}_1(C) \to \mathcal{F}_2 \circ G(C)$.)

We define a functor $G : \mathcal{C}_{\mathcal{I}} \to \mathcal{C}_{\mathcal{E}}$ which acts on objects by sending a CDS A to the event structure A^+ over its set of positive events, with $X, X' \vdash e$ if $X \vdash f$ and $X', f \vdash e$ for some (negative) f and $X \in \mathrm{Con}_{A^+}$ if for any $Y \subseteq |A|$, $Y^+ = X$ implies $Y \in \mathrm{Con}_A$.

If $x \in D(A)^+$ then $x^+ \in D(A^+)$ and this defines an order-isomorphism $\psi_A : D(A^+) \cong D(A)^+$ in the category of dI-domains. Thus we define the action of G on sequential algorithm $a : A \Rightarrow_{\mathcal{I}} B$ as $G(a) : D(A^+) \to D(B^+) \triangleq \psi_B^{-1} \circ \mathsf{fun}^+(a) \circ \psi_A$.[5]

The natural transformation $\phi : \mathcal{F}_{\mathcal{I}} \to \mathcal{F}_{\mathcal{E}} \circ G^{op}$ is induced by the stable function $S : \mathbf{CDS} \to \mathbf{Ev}$ mapping A to A^+ (the restriction of G to the posetal categories **CDS** and **Ev**), — for each object $A \in \mathcal{C}_{\mathcal{I}}$, the morphism $\phi_A : \mathcal{F}_{\mathcal{I}}(A) \to \mathcal{F}_{\mathcal{E}}(A^+)$ consists of:

- A reindexing function $\Phi_A : \mathsf{Ty}_{\mathcal{I}}(A) \to \mathsf{Ty}_{\mathcal{E}}(A^+)$, which sends a **CDS**-parametrization $P : D(A)^+ \to \mathbf{CDS}$ over A to the **Ev**-parametrization $S \circ P \circ \psi_A : D(A^+) \to \mathbf{Ev}$,
- A morphism $s_{A,P} : \mathsf{Tm}_{\mathcal{I}}(A \vdash P) \to \mathsf{Tm}_{\mathcal{E}}(A^+ \vdash_{\mathcal{E}} \Phi_A(P))$ for each CDS A and **CDS**-parametrization $P \in \mathsf{Ty}_{\mathcal{I}}(A)$ that sends a dependent sequential algorithm $a : \Pi_{\mathcal{I}}(A, P)$ to the dependent stable function:
$$x \mapsto \psi^{-1}_{P \circ \psi_A(x)} \circ \mathsf{fun}^+(a) \circ \psi_A(x) : \Pi_{\mathcal{E}}(A^+, S \circ P)$$

For coherence we require that our morphism of CwFs preserves the interpretations of the Booleans. G evidently does so.

Intensional Types as Extensional Terms. We must also interpret the rules which allow extensional terms of the distinguished type $\mathcal{I}$ to be used as intensional types. By Proposition 4.3, there is an event structure $E_{\mathcal{I}}$ for which $D(E_{\mathcal{I}})$ is order-isomorphic to the dI-domain **CDS** of concrete data structures, so that stable functions into $D(E_{\mathcal{I}})$ correspond to CDS parametrizations. (Concretely, the events of $E_{\mathcal{I}}$ are CDSs which are down-closures of a single event (positive or negative), with the causal order [32] being $\blacktriangleleft$ and a set of events consistent if they are bounded above in $\blacktriangleleft$.)

More generally, if $V : \mathsf{Fam} \to \mathsf{Set}$ is the projection of Fam onto indexing sets and reindexing functions, we require that the functor $V \circ \mathcal{F}_{\mathcal{I}} : \mathcal{C}_{\mathcal{I}} \to \mathsf{Set}$ (which projects the intensional category with families onto just its typing information) is *representable* by $E_{\mathcal{I}}$, in a sense which we now explain.

[4]Observe that adding the rules from Table 3 renders all of the intensional type formation rules from Table 1 admissible.

[5]Note that the event structures in the image of this functor are *confusion free* [31], although we can't recover every dependent CDS from its confusion-free event structure.

Let Fam_* be the category of *pointed-set*-indexed families — i.e. each indexing set J contains a distinguished object $*_J$ and reindexing functions satisfy $f\left(*_J\right) = *_K$. This comes with functors $U : \mathsf{Fam}_* \to \mathsf{Fam}$ (which forgets the pointed structure) and $W : \mathsf{Fam}_* \to \mathsf{Set}$ which sends each family $\left\{A_j \,\middle|\, j \in J\right\}$ to A_{*_J}.

Definition 6.2. A category with pointed families is given by a base category C with a functor $\mathcal{F}_* : C^{op} \to \mathsf{Fam}_*$, corresponding to a category with families over C together with a type $*_\Gamma$ in $\mathsf{Ty}_*(\Gamma)$ for each context Γ such that for each $\gamma : \Delta \to \Gamma$, $\mathcal{F}_*(\gamma)(*_\Gamma) = *_\Delta$.

We extend $(C_\mathcal{E}, \mathcal{F}_\mathcal{E})$ to a category with pointed families by setting each $*_E$ to be the constant parametrization over E which returns $E_\mathcal{I}$. To interpret dependent FPC, we require that forgetting the pointed structure returns the extensional category with families (i.e. $U \circ \mathcal{F}_{\mathcal{E}*} = \mathcal{F}_\mathcal{E}$, which is evident) and that the following diagram commutes up to natural isomorphism.

$$\begin{tikzcd}
C_\mathcal{I}^{op} \arrow[r, "\mathcal{F}_\mathcal{I}"] \arrow[d, "G^{op}"] & \mathsf{Fam} \arrow[dr, "V"] & \\
 & & \mathsf{Set} \\
C_\mathcal{E}^{op} \arrow[r, "\mathcal{F}_{\mathcal{E}*}"] & \mathsf{Fam}_* \arrow[ur, "W"'] &
\end{tikzcd}$$

This holds since $W \circ \mathcal{F}_{\mathcal{E}*} : C_\mathcal{E}^{op} \to \mathsf{Set}$ is the Yoneda embedding of $E_\mathcal{I}$.

Finally, for any CDS A, the functions from $A^+ \Rightarrow_\mathcal{E} E_\mathcal{I}$ into $E_\mathcal{I}$ sending a CDS-parametrization P to $\Pi_\mathcal{I}(A, P)$ and $\Sigma_\mathcal{I}(A, P)$ are stable and therefore correspond to maps from $\mathsf{Tm}_\mathcal{E}(C \cdot A \vdash E_\mathcal{I})$ to $\mathsf{Tm}_\mathcal{E}(C \vdash E_\mathcal{I})$, giving interpretations of the dependent product and sum as term-formation rules in $C_\mathcal{E}$ which are consistent with their interpretations in $C_\mathcal{I}$ as type-formers.

Proposition 6.3 (Soundness). *For terms $\Gamma \vdash_\mathcal{U} s : T$ and $\Gamma \vdash_\mathcal{U} t : T$, if $s = t$ then $[\![s : T]\!]_\mathcal{U}^\Gamma = [\![t : T]\!]_\mathcal{U}^\Gamma$.*

7 Dependently Typed Programs

We now define a programming language based on our typing system, by giving an operational semantics for evaluating intensional terms, based on head-reduction in the λ-calculus with pairing and conditionals. (We omit fixed points on terms, as $\mu x.s$ is macro-expressible as $(\lambda y.s\,[(y\,y)/x])\ \lambda y.s\,[(y\,y)/x]$ in the presence of recursive types.) Formally, we give a (small-step) reduction relation on (pseudo)terms, consisting of pairs of the form $E[s] \longrightarrow E[t]$ where $E[\_]$ is an evaluation context, given by the grammar:
$E[\_] ::= [\_] \mid E[\_]\ t \mid \mathtt{If}\,E[\_]\ \mathtt{then}\,t\,\mathtt{else}\,t \mid \mathtt{fst}\,(E[\_]) \mid \mathtt{snd}\,(E[\_])$
and $s \longrightarrow t$ is a reduction rule, given by the schema:

$$(\lambda x.s)\ t \longrightarrow s[t/x] \qquad \mathtt{fst}\,\langle s, t\rangle \longrightarrow s \qquad \mathtt{snd}\,\langle s, t\rangle \longrightarrow t$$
$$\mathtt{If}\,\mathtt{tt}\,\mathtt{then}\,t_1\,\mathtt{else}\,t_2 \longrightarrow t_1 \qquad \mathtt{If}\,\mathtt{ff}\,\mathtt{then}\,t_1\,\mathtt{else}\,t_2 \longrightarrow t_2$$

Thus we define observational approximation for terms $\Gamma \vdash_\mathcal{I} s, t : T$ — $s \lesssim_T^\Gamma t$ if and only if for any compatible, closing context $C[\_] : \mathtt{bool}$, $C[s] \Downarrow \mathtt{tt}$ implies $C[t] \Downarrow \mathtt{tt}$.

It follows from soundness of the equational theory (Proposition 6.3) that for $\Gamma \vdash_\mathcal{I} s, t : T$, if $s \to^* t$ then $[\![s : T]\!]_\mathcal{I}^\Gamma = [\![t : T]\!]_\mathcal{I}^\Gamma$.

7.1 Adequacy

To show that our semantics is adequate — that is, for every program $\Gamma \vdash_\mathcal{I} t : \mathtt{bool}$, $t \Downarrow$ if and only if $[\![t]\!]_\mathcal{I}^\Gamma \neq \emptyset$ — we adapt Pitts' theory of admissible relations [30] to dependent type theory, defining a system of dependent admissible logical relations. This takes advantage of our two semantic universes by incorporating the relational structure within our interpretation of extensional types and terms as event structures and stable functions. Specifically, we define a new interpretation of the extensional type $\mathcal{I}$ of intensional types in which states are dependent pairings of a CDS (representing an intensional type) with a relation between pseudo expressions and states of the CDS.

Definition 7.1. Let $\mathbf{R}$ be the dI-domain consisting of pairs $(B, \mathcal{R}_B)$ of a CDS B and a relation $\mathcal{R}_B$ between the set of pseudo-expressions and the set of finite, positive states of B, with $(B, \mathcal{R}_B) \leq (C, \mathcal{R}_C)$ if $B \blacktriangleleft C$ and $\mathcal{R}_B \subseteq \mathcal{R}_C$.

An $\mathbf{R}$-parametrization over a CDS A (a stable function F from $D(A)^+$ to $\mathbf{R}$) therefore corresponds to a pair consisting of a $\mathbf{CDS}$-parametrization P over A and a family of relations $\mathcal{R}(x)$ between pseudo-expressions and finite states of $P(x)$ for each $x \in D(A)^+$. We now give operations on $\mathbf{R}$-parametrizations corresponding to the dependent sum and product. Given $S = (A, \mathcal{R}_A) \in \mathbf{R}$:

- $\Pi_R(S, F) = \left(\Pi_\mathcal{I}(A, P), \mathcal{R}_{\Pi_\mathcal{I}(A,P)}\right)$, where $(s, a) \in \mathcal{R}_{\Pi_\mathcal{I}(A,P)}$ iff $(t, x) \in \mathcal{R}_A$ implies $\left(s\,t, \mathsf{fun}^+(a)(x)\right) \in \mathcal{R}(x)$
- $\Sigma_R(S, F) = \left(\Sigma_\mathcal{I}(A, P), \mathcal{R}_{\Sigma_\mathcal{I}(A,P)}\right)$, where $(t, x) \in \mathcal{R}_{\Sigma_\mathcal{I}(A,P)}$ iff $(\mathtt{fst}(t), \pi_1(x)) \in \mathcal{R}_A$ and $(\mathtt{snd}(t), \pi_2(x)) \in \mathcal{R}(\pi_1(x))$.

Note that in each case, the first component is given by the corresponding construction on CDS parametrizations.

By Proposition 4.3, $\mathbf{R}$ is order-isomorphic to the domain of states of an event-structure E_R, which we can use in place of the former $E_\mathcal{I}$ — i.e. we obtain a pointed category with families modelling the extensional universe in which the extensional type $\mathcal{I}$ denotes E_R, with the above operations.

The interpretation of an extensional term T of type $\mathcal{I}$ in an intensional context Γ then corresponds to an $\mathbf{R}$-parametrization over $[\![\Gamma]\!]_\mathcal{I}$ consisting of the $\mathbf{CDS}$-parametrization $[\![T]\!]_\mathcal{I}^\Gamma$ over $[\![\Gamma]\!]_\mathcal{I}$ (i.e. an intensional type) together with the relation $\mathcal{R}_T(x)$ between pseudo-expressions and positive finite states of $[\![T]\!]_\mathcal{I}(x)$, parametrized by $x \in D\left([\![\Gamma]\!]_\mathcal{I}\right)^+$ (a logical relation on $[\![T]\!]_\mathcal{I}^\Gamma$).

Finally, we fix the denotation of $[\![\mathtt{bool} : I]\!]$ to be the constant $\mathbf{R}$-parametrization with value $(\mathbb{B}, \mathcal{R}_\mathbb{B}) \in \mathbf{R}$, where $\mathbb{B}$ is the CDS of Booleans and $(t, x) \in \mathcal{R}_\mathbb{B}$ iff $x \neq \emptyset$ implies $t \Downarrow$. We may now use our logical relations to give a proof of adequacy along standard lines.

Lemma 7.2. *For all types $\Gamma \vdash_\mathcal{I} T$ type and $x \in D\left([\![\Gamma]\!]_\mathcal{I}\right)^+$:*

- *$(t, \emptyset) \in \mathcal{R}_T(x)$ for all pseudo expressions t,*
- *If $s \longrightarrow t$ and $(t, y) \in \mathcal{R}_T$, then $(s, y) \in \mathcal{R}_T$.*

For each intensional context $\Gamma = x_1 : T_1, \ldots, x_n : T_n$, we define a relation $\mathcal{R}_\Gamma$ between n-tuples of pseudo-expressions (which act as substitutions on terms over Γ) and finite positive states of $[\![\Gamma]\!]_\mathcal{I}$, as follows:

- $\mathcal{R}_{[]} = (\langle\rangle, \emptyset)$
- $(\langle \vec{r}, s\rangle, x) \in \mathcal{R}_{\Gamma, y:T}$ if $(\vec{r}, \pi_1(x)) \in \mathcal{R}_\Gamma$
  $(s, \pi_2(x)) \in \mathcal{R}_T(\pi_1(x))$.

Proposition 7.3. *For any term $\Gamma \vdash_\mathcal{I} t : T$, if $(\vec{r}, x) \in \mathcal{R}_\Gamma$, and $y \subseteq_{fin} [\![t : T]\!]_\mathcal{I}^\Gamma(x)$ positive state of $[\![T]\!]_\mathcal{I}^\Gamma$ then $(t[\vec{r}], y) \in \mathcal{R}_T(x)$.*

Proof. We prove by structural induction:

- If $t \in \{\mathtt{tt}, \mathtt{ff}\}$, then this is evident.

- If $t \equiv \mathtt{If}\, s\, \mathtt{then}\, t_1\, \mathtt{else}\, t_2$. If $y = \emptyset$ then $(t[\vec{r}], y) \in \mathcal{R}_T(x)$ by Lemma 7.2; otherwise $[\![s : \mathtt{bool}]\!]^{\Gamma}_{\mathcal{I}}(x) \neq \emptyset$ and so $s[\vec{r}] \Downarrow$ by inductive hypothesis. Suppose w.l.o.g. that $s[\vec{r}] \Downarrow \mathtt{tt}$, then $t[\vec{r}] \longrightarrow^* t_1[\vec{r}]$, and so by the inductive hypothesis on t_1, Lemma 7.2 and soundness of the reduction relation, $(t[\vec{r}], y) \in \mathcal{R}_T(x)$.
- If $t \equiv \lambda z.t' : \Pi_{\mathcal{I}}(z : S).T$ then $t[\vec{r}] \equiv \lambda z.t'[\vec{r}]$ and so for any $(s, w) \in \mathcal{R}_S(x)$, $t[\vec{r}]\, s$ reduces to $t'[\vec{r}][s/z]$ and so by induction hypothesis applied to t', and Lemma 7.2, $(t[\vec{r}], y) \in \mathcal{R}_T(x)$.
- If $t \equiv t'\, s$, so that $\Gamma \vdash t' : \Pi_{\mathcal{I}}(x : S).T'$ and $\Gamma \vdash s : S$ for some S and some T' such that $T'[s/x] = T$, then there exist positive states $y' \subseteq_{fin} [\![t']\!](x)$ and $z \subseteq_{fin} [\![s]\!](x)$ such that $\mathrm{fun}^+(y')(z) = y$. By inductive hypothesis, $(t'[\vec{r}], y') \in \mathcal{R}_{\Pi_{\mathcal{I}}(x:S).T'}(x)$ and $(s, z) \in \mathcal{R}_S(x)$, and hence $(t, y) \in \mathcal{R}_T(x)$.
- The cases for pairing or projection are similar. □

For any closed term $t : \mathtt{bool}$, $\left(t, [\![t : \mathtt{bool}]\!]_{\mathcal{I}}\right) \in \mathcal{R}_{\mathbb{B}}$ and hence:

Theorem 7.4 (Adequacy). *For any program $\vdash_{\mathcal{I}} t : \mathtt{bool}$: $t \Downarrow \mathtt{tt}$ if and only if $[\![t : \mathtt{bool}]\!]_{\mathcal{I}} = [\![\mathtt{tt} : \mathtt{bool}]\!]_{\mathcal{I}}$.*

8 Observable Sequentiality

Since observational equivalence for our programming language is conservative over its simply-typed sublanguage (finitary PCF) it can have no effectively presentable and fully abstract semantics [23]. Our model contains the sequential algorithms model of finitary PCF and thus has distinct denotations for observationally equivalent terms such as left-strict-or and right-strict-or [11]. By adding a simple non-local control operator (catch) to PCF to capture this kind of intensional information in the sequential algorithms model, Cartwright, Curien and Felleisen established full abstraction for the resulting "observably sequential PCF" (SPCF). We will extend these results to our partial and total models of dependent FPC.

We extend the intensional terms by adding a Boolean version of catch — a strictness test with the following typing rule:

$$\frac{\Gamma, k : \mathtt{bool} \vdash_I t : \mathtt{bool}}{\Gamma \vdash_I \mathtt{catch}(k).t : \mathtt{bool}}$$

Some caution is required when extending dependent type theory with control operators — adding full call/cc in the presence of dependent sum types leads to an inconsistent theory [16]. However, the extension of the total fragment of our type theory with (simply-typed) catch is consistent, as we show by giving an interpretation of terms as total states. Note also that we are using the distinction between intensional and extensional typing judgments to allow side-effecting terms without side-effecting types.

Operationally, catch is characterized by the reduction rules:

$$\mathtt{catch}(k).E[k] \longrightarrow \mathtt{tt} \qquad \mathtt{catch}(k).v \longrightarrow \mathtt{ff} \quad (v \in \{\mathtt{tt}, \mathtt{ff}\})$$

with which we extend the operational semantics of Section 7. We also add $\mathtt{catch}(k).E[\_]$ to the grammar of evaluation contexts.

Denotationally, $\mathtt{catch}(k).t$ is interpreted as the composition of $[\![\lambda k.t]\!]^{\Gamma}_{\mathcal{I}}$ with the sequential algorithm from $(\mathbb{B} \Rightarrow \mathbb{B})$ to $\mathbb{B}$ with maximal events $(\{(\emptyset, ?), (\{?\}, ?)\}, \mathtt{tt})$ and $(\{(\emptyset, ?), (\emptyset, v)\}, \mathtt{ff})$ for $v \in \{\mathtt{tt}, \mathtt{ff}\}$.

This is sound with respect to the operational rules [11], and our proof of adequacy (Theorem 7.4) extends to include catch:

Proposition 8.1. *$t \Downarrow \mathtt{tt}$ if and only if $[\![t : \mathtt{bool}]\!]_{\mathcal{I}} = [\![\mathtt{tt} : \mathtt{bool}]\!]_{\mathcal{I}}$.*

Equationally, we extend the raw theory of intensional terms with the above reduction rules as axioms. To give a complete characterization of equivalence at finite types we add:

$$t = \mathtt{If}\,\mathtt{catch}(k).t\,\mathtt{then}\,(\mathtt{If}\,k\,\mathtt{then}\,t[\mathtt{tt}/k]\,\mathtt{else}\,t[\mathtt{ff}/k])\ \mathtt{else}\,t[\mathtt{tt}/k]$$

together with the typed rule:

$$\frac{\Gamma \vdash_{\mathcal{I}} t : \mathtt{If}\, s\, \mathtt{then}\, T\, \mathtt{else}\, T}{\Gamma \vdash_{\mathcal{I}} \mathtt{If}\, s\, \mathtt{then}\, t\, \mathtt{else}\, t = t : \mathtt{If}\, s\, \mathtt{then}\, T\, \mathtt{else}\, T}$$

In general $\mathtt{If}\, s\, \mathtt{then}\, t\, \mathtt{else}\, t = t$ does not hold in our model (e.g. $[\![\mathtt{If} \perp \mathtt{then}\,\mathtt{tt}\,\mathtt{else}\,\mathtt{tt} : \mathtt{bool}]\!]_{\mathcal{I}} \neq [\![\mathtt{tt} : \mathtt{bool}]\!]_{\mathcal{I}}$). Yet, the typed version above is sound since the parametrization over bool denoted by $\mathtt{If}\, x\, \mathtt{then}\, T\, \mathtt{else}\, T$ returns the empty CDS unless its argument is a Boolean value.

8.1 Definability, Completeness and Full Abstraction

We establish a series of results, showing that our total semantics is fully complete and equationally complete, and our partial semantics is fully abstract. First, we prove a lemma on which each of our results depends — that every finite type (i.e. one in which the only instances of fixed points are the empty type $\perp \triangleq \mu x.x$) is a *definable retract* of $\mathtt{bool}^n$ (i.e. $\mathrm{vec}_{\mathbb{B}}(n)$)) for some n, in the following sense:

Definition 8.2. For $\Gamma \vdash_{\mathcal{I}} S, T$: type, $\Gamma \vdash_{\mathcal{I}} S \trianglelefteq T$ is a *definable retraction* if there are terms $\Gamma \vdash_{\mathcal{I}} \mathtt{in} : S \rightarrow T$ and $\Gamma \vdash_{\mathcal{I}} \mathtt{out} : T \rightarrow S$ such that $\Gamma, x : S \vdash_{\mathcal{I}} \mathtt{out}(\mathtt{in}\, x) = x : S$.

The axioms introduced above in the equational theory play a key role in the proof of following lemma:

Lemma 8.3. *There are definable retractions:*

1. $\vdash_{\mathcal{I}} \mathtt{bool} \rightarrow_{\mathcal{I}} \mathtt{bool} \trianglelefteq \mathtt{bool}^3$
2. $\vdash_{\mathcal{I}} \mathtt{bool}^m \rightarrow_{\mathcal{I}} \mathtt{bool}^n \trianglelefteq \mathtt{bool}^{n.3^m}$
3. *If* $\Gamma \vdash_{\mathcal{I}} s : \mathtt{bool}, T$ type *then* $\Gamma \vdash_{\mathcal{I}} \mathtt{If}\, s\, \mathtt{then}\, T\, \mathtt{else}\, T \trianglelefteq T$.

Proof. (1) with $\mathtt{in} \triangleq \lambda f.\langle \mathtt{catch}(k).f\,k, \langle f\,\mathtt{tt}, f\,\mathtt{ff}\rangle\rangle$ and $\mathtt{out} \triangleq \lambda x.\lambda y.\mathtt{If}\,\mathtt{fst}(x)\ \mathtt{then}\ (\mathtt{If}\,y\,\mathtt{then}\,\mathtt{fst}(\mathtt{snd}(x))\,\mathtt{else}\,\mathtt{snd}(\mathtt{snd}(x)))$.
$\mathtt{else}\,\mathtt{fst}(\mathtt{snd}(x))$
(2) by induction on lexicographically ordered (m, n) using (1).
(3) with $\mathtt{in} \triangleq \mathtt{out} \triangleq \lambda x.\mathtt{If}\, s\, \mathtt{then}\, x\, \mathtt{else}\, x$. □

We may thus prove by structural induction:

Proposition 8.4. *For any finite $\Gamma \vdash_{\mathcal{I}} T$ type there exists n such that $\Gamma \vdash_{\mathcal{I}} T \trianglelefteq \mathtt{bool}^n$. We write $\mathtt{in}_T$ and $\mathtt{out}_T$ for the associated terms.*

Theorem 8.5 (Full Completeness). *For finite $\Gamma \vdash T$ type, and (total) $x \in [\![T]\!]^{\Gamma}_{\mathcal{I}}$ there exists a (total) term $\Gamma \vdash_{\mathcal{I}} t_x : T$ such that $[\![t_x : T]\!]^{\Gamma}_{\mathcal{I}} = x$.*

Proof. $[\![\mathtt{in}_T]\!]^{\Gamma}_{\mathcal{I}}(x) \in [\![\mathtt{bool}^n]\!]^{\Gamma}_{\mathcal{I}}$ is a finite tuple of Boolean states and therefore definable as a term $s : \mathtt{bool}^n$. Define $t_x \triangleq \mathtt{out}_T\, s$, so $[\![t_x]\!]^{\Gamma}_{\mathcal{I}} = [\![\mathtt{out}_T\, s]\!]^{\Gamma}_{\mathcal{I}} = [\![\mathtt{out}_T]\!]^{\Gamma}_{\mathcal{I}}\left([\![\mathtt{in}_T]\!]^{\Gamma}_{\mathcal{I}}(x)\right) = x$. □

Theorem 8.6 (Equational Completeness). *For any total terms $\Gamma \vdash_{\mathcal{I}} t, t' : T$, if $[\![t : T]\!]^{\Gamma}_{\mathcal{I}} = [\![t' : T]\!]^{\Gamma}_{\mathcal{I}}$ then $\Gamma \vdash_{\mathcal{I}} t = t' : T$.*

Proof. It is sufficient to prove this for closed terms. Suppose that $[\![t : T]\!]_{\mathcal{I}} = [\![t' : T]\!]_{\mathcal{I}}$. By Proposition 8.4, $T \trianglelefteq \mathtt{bool}^n$ for some n. For each $i \leq n$, $\pi_i\left([\![\mathtt{in}_T\, t : \mathtt{bool}^n]\!]_{\mathcal{I}}\right) = \pi_i\left([\![\mathtt{in}_T\, t' : \mathtt{bool}^n]\!]_{\mathcal{I}}\right)$, so by adequacy $\pi_i(\mathtt{in}_T(t)) \Downarrow v$ iff $\pi_i(\mathtt{in}_T(t)) \Downarrow v$ and $\mathtt{in}_T(t) = \mathtt{in}_T(t')$, so $t = \mathtt{out}_T(\mathtt{in}_T(t)) = \mathtt{out}_T(\mathtt{in}_T(t')) = t'$. □

Lemma 8.7. *Every type $\Gamma \vdash_{\mathcal{I}} T$ type denotes the least upper bound of a $\blacktriangleleft$-chain of CDSs interpreting finite types $(T_i)_{i\in\omega}$ in which each embedding-projection pair from $\blacktriangleleft$ are definable as terms.*

Proof. Let $\sqsubseteq$ be the least precongruence on extensional terms such that $\bot \sqsubseteq t$ for all t, so that $s \sqsubseteq t$ implies $[\![s]\!]^{\Gamma}_{\mathcal{E}} \subseteq [\![t]\!]^{\Gamma}_{\mathcal{E}}$. We show by induction on β-normal forms that if $\Gamma \vdash_{\mathcal{I}} S, T$ type and $S \sqsubseteq T$ then the embedding-projection pair from S to T is definable.

For any type T, we obtain a type T'_i such that $T = T'_i$ by (recursively) expanding each of its fixed points i times, and thus an approximating chain of types T_i such that $T_i \sqsubseteq T'_i = T$ for each i by replacing every fixed point in T'_i by $\bot$. ◻

As in [26], this combines with definability at finite types to prove:

Theorem 8.8 (Finite Definability). *For any finite $x \in [\![T]\!]^{\Gamma}_{\mathcal{I}}$ there exists a term $\Gamma \vdash_{\mathcal{I}} t_x : T$ such that $[\![t_x : T]\!]^{\Gamma}_{\mathcal{I}} = x$.*

By standard arguments from adequacy and finite definability:

Theorem 8.9 (Full Abstraction). *For any terms $\Gamma \vdash_{\mathcal{I}} t_1, t_2 : T$:*

$$t_1 \lesssim^{\Gamma}_{T} t_2 \iff [\![t_1 : T]\!]^{\Gamma}_{\mathcal{I}} \subseteq [\![t_2 : T]\!]^{\Gamma}_{\mathcal{I}}$$

9 Conclusions and Further Directions

We have described an intensional semantics of dependent types, and an expressive type system for defining them. There are some notable omissions, partly due to lack of space. Our full completeness result shows that, in principle, we may identify the total elements of the intensional model by an intrinsic property (unlike the domain-theoretic semantics) but going beyond the restriction to finite types requires consideration of "winning conditions" for infinitary positions; a next step is to give total interpretations of inductive families of types based on the game semantics of inductive types [12].

We have not included identity types, which may receive different interpretations in our model based on identifying functional programs up to intensional or extensional equivalence. By giving an internal encoding of a strategy as a tuple of Booleans, definable retractions give a way to express "extensional equality" of sequential algorithms without requiring function extensionality.

A final requirement, if we are to give a semantics of an expressive logical system such as the calculus of constructions, is a treatment of higher rank polymorphism. It is not yet clear how the existing games models for System F style polymorphism [2, 22] may be adapted to the positional style of games described here.

Our model includes some side effects (partiality, local control) and thus gives some indications of how a more general approach might combine computational effects with dependent types. A related facet of the model is its relationship to linear logic and type theory — graph games and concrete data structures enjoy a decomposition into a model of linear logic; extending this to dependent types may be illuminating on both fronts.

Acknowledgments

We would like to thank Pierre Clairambault and Thomas Streicher for valuable discussions, and the anonymous referees for their comments. This research was supported by the Labex DigiCosme (project ANR11LABEX0045DIGICOSME) operated by ANR as part of the program "Investissements d'Avenir" Idex ParisSaclay (ANR11IDEX000302), and by UK EPSRC grant EP/K037633/1.

References

[1] Samson Abramsky and Radha Jagadeesan. 1994. Games and Full Completeness for Multiplicative Linear Logic. *Journal of Symbolic Logic* 59, 2 (1994), 543–574.
[2] Samson Abramsky and Radha Jagadeesan. 2005. A game semantics for generic polymorphism. *Annals of Pure and Applied Logic* 133, 1-3 (2005), 3–37.
[3] Samson Abramsky, Radha Jagadeesan, and Pasquale Malacaria. 2000. Full Abstraction for PCF. *Information and Computation* 163, 2 (2000), 409–470.
[4] Samson Abramsky, Radha Jagadeesan, and Matthijs Vákár. 2015. Games for Dependent Types. In *42nd International Colloquium on Automata, Languages, and Programming*. Springer, 31–43.
[5] Thorsten Altenkirch, Conor McBride, and James McKinna. 2005. Why dependent types matter. (2005). http://www.e-pig.org/downloads/ydtm.pdf.
[6] Roberto Amadio and Pierre-Louis Curien. 1998. *Domains and Lambda-Calculi*. Cambridge Tracts in Theoretical Computer Science, Vol. 46. Cambridge University Press.
[7] Henk Barendregt. 1992. Lambda Calculi with Types. In *Handbook of Logic in Computer Science*. Vol. 2. Oxford University Press, 117–309.
[8] Gérard Berry. 1976. Bottom-Up Computation of Recursive Programs. *Informatique Théorique et Applications* 10, 1 (1976), 47–82.
[9] Gérard Berry. 1978. Stable Models of Typed lambda-Calculi. In *5th Colloquium on Automata, Languages and Programming*. Springer, 72–89.
[10] Gérard Berry and Pierre-Louis Curien. 1982. Sequential Algorithms on Concrete Data Structures. *Theoretical Computer Science* 20, 3 (1982), 265–321.
[11] Robert Cartwright, Pierre-Louis Curien, and Matthias Felleisen. 1994. Fully Abstract Semantics for Observably Sequential Languages. *Information and Computation* 111, 2 (1994), 297–401.
[12] Pierre Clairambault. 2009. Least and Greatest Fixpoints in Game Semantics. In *12th International Conference on Foundations Of Software Science And Computational Structures (Lecture Notes in Computer Science)*. Springer, 16–31.
[13] Pierre Clairambault and Peter Dybjer. 2014. The biequivalence of locally cartesian closed categories and Martin-Löf type theories. *Mathematical Structures in Computer Science* 24, 6 (2014).
[14] Pierre-Louis Curien. 1992. Observable Algorithms on Concrete Data Structures. In *7th Annual Symposium on Logic in Computer Science*. IEEE Computer Society, 432–443.
[15] Peter Dybjer. 1995. Internal Type Theory. In *International Workshop on Types for Proofs and Programs, Selected Papers*. Springer, 120–134.
[16] Hugo Herbelin. 2005. On the Degeneracy of Sigma-Types in Presence of Computational Classical Logic. In *7th International Conference on Typed Lambda Calculi and Applications (Lecture Notes in Mathematics)*. Springer, 209–220.
[17] Martin Hofmann and Thomas Streicher. 1998. The Groupoid Interpretation of Type Theory. In *Twenty-Five Years of Constructive Type Theory, Proceedings of a Congress held in Venice, October 1995 (Oxford Logic Guides)*. Oxford University Press, 83–111.
[18] Martin Hyland and Luke Ong. 2000. On Full Abstraction for PCF: I, II, and III. *Information and Computation* 163, 2 (2000), 285–408.
[19] Martin Hyland and Andrea Schalk. 2002. Games on Graphs and Sequentially Realizable Functionals. In *17th IEEE Symposium on Logic in Computer Science*. IEEE Computer Society, 257–264.
[20] Gilles Kahn and Gordon D. Plotkin. 1993. Concrete Domains. *Theoretical Computer Science* 121, 1&2 (1993), 187–277.
[21] Gilles Kahn and Gordon D. Plotkin. 1993. Concrete Domains. *Theoretical Computer Science* 121, 1&2 (1993), 187–277.
[22] James Laird. 2013. Game semantics for a polymorphic programming language. *Journal of the ACM* 60, 4 (2013), 29.
[23] Ralph Loader. 2001. Finitary PCF is not decidable. *Theoretical Computer Science* 266, 1-2 (2001), 341–364.
[24] Per Martin-Löf. 1982. Constructive mathematics and computer programming. In *Studies in Logic and the Foundations of Mathematics*. Vol. 104. Elsevier, 153–175.
[25] Per Martin-Löf. 1984. *Intuitionistic Type Theory*. Bibliopolis.
[26] Guy McCusker. 1998. *Games and full abstraction for a functional metalanguage with recursive types*. Springer.
[27] Erik Palmgren. 1993. An Information System Interpretation of Martin-Löf's Partial Type Theory with Universes. *Information and Computation* 106, 1 (1993), 26–60.
[28] Erik Palmgren and Viggo Stoltenberg-Hansen. 1990. Domain Interpretations of Martin-Löf's Partial Type Theory. *Annals of Pure and Applied Logic* 48, 2 (1990), 135–196.
[29] Erik Palmgren and Viggo Stoltenberg-Hansen. 1990. Domain Interpretations of Martin-Löf's Partial Type Theory. *Annals of Pure and Applied Logic* 48, 2 (1990), 135–196.
[30] Andrew M. Pitts. 1996. Relational Properties of Domains. *Information and Computation* 127, 2 (1996), 66–90.
[31] Daniele Varacca, Hagen Völzer, and Glynn Winskel. 2004. Probabilistic Event Structures and Domains. In *15th International Conference on Concurrency Theory*. Springer, 481–496.
[32] Glynn Winskel. 1980. *Events in computation*. Ph.D. Dissertation. University of Edinburgh.