Type Refinements for the Working Class

Jon Sterling and Darin Morrison

There are two conflicting views which bedevil any discussion of the nature of type theory. First, there is the notion of type theory as an extension or generalization of universal algebra to support interdependency of sorts and operations, possibly subject to an arbitrary equational theory [2, 4]; we will call this *formal type theory*. Typing, in such a setting, is a mere matter of grammar and is nearly always decidable. In hindsight, we may observe that this is the sort of type theory which Martin-Löf first proposed in 1972 [12], even if we will admit that this was not the intention at the time. A model for such a type theory is usually given by interpreting the types or sorts as presheaves or sheaves over contexts of hypotheses, and as such, a proof theoretic interpretation of the hypothetical judgment is possible.

Secondly, there is the view of type theory as semi-formal theory of constructions for the Brouwer-Heyting-Kolmogorov interpretation of intuitionistic mathematical language, which we will call *behavioral* or *semantic type theory*. The most widely known development of this program is Martin-Löf's 1979 "extensional" type theory [13, 14], but we must give priority to Dana Scott for inventing this line of research in 1970 with his prophetic report, *Constructive Validity* [15]. Since the 1980s, behavioral type theory has been developed much further in the Nuprl family [3] of proof assistants, including MetaPRL [8] and JonPRL [16].

Martin-Löf's key innovation was the commitment to pervasive functionality (extensionality) as part of the *definitions* of the judgments and the types, in contrast to the state of affairs in formal type theory where functionality is a metatheorem which must be shown to obtain, based on the (somewhat arbitrary) equational theory which has been imposed. Furthermore, models for behavioral type theory interpret the types as partial equivalence relations (PERs) on only closed terms, and the meaning of the hypothetical judgment is defined separately and uniformly in the logical relations style.

Our position is that these views of type theory are not in conflict, but rather describe two distinct layers in a single, harmonious system. From this perspective, formal type theories can do little more than negotiate matters of grammar, and therefore may serve as a linguistic framework for mathematical expression, being responsible for the management of variable binding and substitution, among other things. On the other hand, mathematical objects find *meaning* in their extension within the behavioral type theory, and are subject to an extensional equality.

The types of the semantic theory can then be said to *refine* the types of the syntactic

theory [6, 7, 5], both by placing restrictions on membership and by coarsening equivalence. Thus far, most developments of behavioral type theory have been built on a *unityped* syntactic framework, and so the relation to type refinements has been difficult to see.

In this paper, we contribute a full theory of behavioral refinements over multi-sorted nominal abstract binding trees with symbolic parameters, a simple but interesting formal type theory [5, 17]; this hybrid system allows the deployment of a Nuprl-style type theory over any signature of sorts and operators.

Owing to the *nominal* aspect of our abstract binding tree (abt) framework, this development constitutes a possible alternative to Allen's semantics for unguessable atoms in Nuprl [1]; Allen proposed a "supervaluation" semantics, that explained sequents involving atoms (symbols) by quantifying over the possible ways to interpret the class of atoms. Our semantics seem to be a more direct and concrete way to approach the problem, which is licensed in part by the nominal character of the abt framework.

Colors In this paper, we hint at the *modes* of judgments and assertions [5] using colors, marking inputs with `blue` and outputs with `red`. As a rule of thumb, inputs are things which are supplied when checking the correctness of a judgment, and outputs are things which are synthesized in the process.

1 Abstract Binding Trees and Symbols

See [17] for the development of abstract binding trees with symbols.

give a brief description of the framework, and present its rules.

2 Behavioral Refinements

Fixing a signature $\Sigma \equiv \langle \mathcal{S}, \mathcal{O} \rangle$ in the abt framework, we will define the notion of behavioral refinement by propounding several judgments and their semantical explanations. Let us first define the copresheaf of τ-sorted expressions, $\mathbf{E}_\tau$ as follows

$$\mathbf{E}_\tau(\Upsilon \parallel \Gamma) \triangleq \{ M \mid \cdot \triangleright \Upsilon \parallel \Gamma \vdash M : \tau \}$$

We will also write $\mathbf{E}_\tau$ for the copresheaf $\mathbf{E}_\tau(- \parallel \cdot)$ on $\mathbb{I} \downarrow \mathcal{S}$.

2.1 Parametric Refinement

The first judgment that will concern us is called *parametric refinement*, $\Upsilon \parallel \phi \sqsubset \tau$, which means that ϕ refines the sort τ under the symbolic parameters Υ. We define this judgment through a meaning explanation in the style of Martin-Löf as follows:

Definition 2.1. To know $\Upsilon \parallel \phi \sqsubset \tau$ (presupposing Υ *sctx* and τ *sort*) is to know, for any $\Upsilon \xhookrightarrow{\rho} \Upsilon'$, what it means for any $M, N \in \mathbf{E}_\tau(\Upsilon')$ to be equated by ϕ (written $\phi\{\rho\}(M, N)$), such that $\phi\{\rho\}(-,-)$ is a partial equivalence relation (PER) on $\mathbf{E}_\tau(\Upsilon')$. Morever, that for any $\Upsilon' \xhookrightarrow{\rho'} \Upsilon''$, from $\phi\{\rho\}(M, N)$ we may conclude $\phi\{\rho' \circ \rho\}(M \cdot \rho', N \cdot \rho')$.

Then, we say that ϕ *globally refines* τ (written $\phi \sqsubset \tau$) when we have $\cdot \parallel \phi \sqsubset \tau$. Furthermore, when $\Upsilon \parallel \phi \sqsubset \tau$, for any renaming $\Upsilon \xhookrightarrow{\rho} \Upsilon'$ we can clearly define a new refinement $\Upsilon' \parallel \rho \cdot \rho \sqsubset \tau$.

Remark 2.2. At this point, it should be noted that this is very similar to the semantical explanation of typehood given in [13], except that we have generalized it to a multi-sorted setting, and that we have fibred the entire apparatus over collections Υ of symbols; furthermore, whereas Martin-Löf defines types in terms of evaluation to canonical form, we have remained agnostic on this point as far as refinements are concerned. We will come back to the notions of *type* and *computation* in section 4.

In fact, the complexity of the above meaning explanation is an artifact of the pointwise style in which we have expressed it. Considered internally to the copresheaf topos $\mathbf{Set}^{\mathbb{I} \downarrow \mathcal{S}}$, a refinement is merely a section of the object of $\mathbf{E}_\tau$-PERs $\mathfrak{per}(\mathbf{E}_\tau)$, defined internally as a subobject of $\Omega^{\mathbf{E}_\tau \times \mathbf{E}_\tau}$:

$$\mathfrak{per}(X) \triangleq \{\phi : \Omega^{X \times X} \mid \mathtt{symmetric}(\phi) \wedge \mathtt{transitive}(\phi)\}$$

So, $\Upsilon \parallel \phi \sqsubset \tau$ obtains just when we have $\phi \in \mathfrak{per}(\mathbf{E}_\tau)(\Upsilon)$. The benefit of explaining such judgments as objects in the topos is that we do not need to deal with the complexities of quantifying over renamings, since this is implicit in the definition of the exponential of presheaves, where $\mathfrak{y}(\Upsilon) \triangleq \Upsilon \hookrightarrow -$ is the co-yoneda embedding:

$$\begin{aligned} B^A(\Upsilon) &\triangleq \mathbf{Set}^{\mathbb{I} \downarrow \mathcal{S}}[\mathfrak{y}(\Upsilon) \times A, B] \\ &\cong \int_{\Upsilon'} (\Upsilon \hookrightarrow \Upsilon') \implies B(\Upsilon')^{A(\Upsilon')} \end{aligned}$$

It should be noted that PERs can be understood as 0-semigroupoids. In other words, these are 1-groupoids without identity morphisms and without coherence laws regarding composition and inverse operations. This distinction is especially relevant in formal type theory where groupoids "one dimension down" are not sets but setoids (i.e., groupoids sans coherence). Thus, a PER can formally be thought of as a kind of partial setoid that facilitates constructions on subsets in addition to quotients.

Say something about internal (semi)-groupoids?

2.1.1 Order and Equality of Parametric Refinements

We will write $\Upsilon \parallel \phi \subseteq \psi \sqsubset \tau$ to mean that ϕ is a *subrefinement* of ψ.

Definition 2.3. To know $\Upsilon \parallel \phi \subseteq \psi \sqsubset \tau$ (presupposing $\Upsilon \parallel \phi \sqsubset \tau$ and $\Upsilon \parallel \psi \sqsubset \tau$), is to know, for any renamings $\Upsilon \xhookrightarrow{\rho} \Upsilon' \xhookrightarrow{\rho'} \Upsilon''$, that from $\phi\{\rho\}(M, N)$ you can conclude $\psi\{\rho' \circ \rho\}(M \cdot \rho', N \cdot \rho')$ for any $M, N \in \mathbf{E}_\tau(\Upsilon')$.

Two refinements are equal when they denote the same PER.That is, we have $\Upsilon \parallel \phi = \psi \sqsubset \tau$ just when both $\Upsilon \parallel \phi \subseteq \psi \sqsubset \tau$ and $\Upsilon \parallel \psi \subseteq \phi \sqsubset \tau$.

2.2 Parametric Equality

The primary judgment concerning refinements is the parametric equality, $\Upsilon \parallel M = N \in \phi$, which presupposes $\Upsilon' \parallel \phi \sqsubset \tau$ for some Υ' such that we have $\Upsilon' \xhookrightarrow{\rho} \Upsilon$, and $M, N \in \mathbf{E}_\tau(\Upsilon)$. The meaning of this judgment is that M and N are identified by ϕ at Υ:

$$\frac{\phi\{\rho\}(M, N)}{\Upsilon \parallel M = N \in \phi}$$

2.3 Functional Refinement

Next, we define functional refinement $\Upsilon \parallel \Psi \vDash \phi = \psi \sqsubset \tau$ in terms of parametric refinement, simultaneously with parametric context refinement $\Upsilon \parallel \Psi \sqsubset^\star \Gamma$ and functional equality $\Upsilon \parallel \Psi \vDash M = N \in \phi$. We will write $\Upsilon \parallel \Psi \vDash \phi \sqsubset \tau$ as a shorthand for $\Upsilon \parallel \Psi \vDash \phi = \phi \sqsubset \tau$.

Parametric context refinement $\Upsilon \parallel \Psi \sqsubset^\star \Gamma$ shall be defined as follows, presupposing Υ *sctx* and Γ *vctx*:

$$\frac{}{\Upsilon \parallel \cdot \sqsubset^\star \cdot}\ \sqsubset^\star_{\mathtt{nil}} \qquad \frac{\Upsilon \parallel \Psi \sqsubset^\star \Gamma \quad \Upsilon \parallel \Psi \vDash \phi \sqsubset \tau}{\Upsilon \parallel \Psi, x : \phi \sqsubset^\star \Gamma, x : \tau}\ \sqsubset^\star_{\mathtt{snoc}}$$

For a context refinement $\Upsilon \parallel \Psi \sqsubset^\star \Gamma$ and a renaming $\Upsilon \xhookrightarrow{\rho} \Upsilon'$, we can define a new context refinement $\Upsilon' \parallel \Psi \cdot \rho \sqsubset^\star \Gamma$ by applying ρ pointwise at each of the sort refinements in the context.

Functional refinement $\Upsilon \parallel \Psi \vDash \phi = \psi \sqsubset \tau$ presupposes $\Upsilon \parallel \Psi \sqsubset^\star \Gamma$ and τ *sort*, and is explained by induction on the evidence for its first presupposition.

Case $\dfrac{}{\Upsilon \parallel \cdot \sqsubset^\star \cdot}\ \sqsubset^\star_{\mathtt{nil}}$.

$$\frac{\Upsilon \parallel \phi = \psi \sqsubset \tau}{\Upsilon \parallel \cdot \vDash \phi = \psi \sqsubset \tau}\ \vDash^{\sqsubset}_{\mathtt{nil}}$$

Case $\dfrac{\Upsilon \parallel \Psi \sqsubset^\star \Gamma \quad \Upsilon \parallel \Psi \vDash \chi \sqsubset \sigma}{\Upsilon \parallel \Psi, x : \chi \sqsubset^\star \Gamma, x : \sigma}\ \sqsubset^\star_{\mathtt{snoc}}$.

To know $\Upsilon \parallel \Psi, x : \chi \vDash \phi = \psi \sqsubset \tau$ is to know, for any renaming $\Upsilon \xhookrightarrow{\rho} \Upsilon'$ and closed terms $M_0, M_1 \in \mathbf{E}_\sigma(\Upsilon')$, that from $\Upsilon' \parallel \Psi \cdot \rho \vDash M_0 = M_1 \in \chi \cdot \rho$, you can conclude $\Upsilon' \parallel \Psi \cdot \rho \vDash [M_0 \,/\, x]\,\phi \cdot \rho = [M_1 \,/\, x]\,\psi \cdot \rho \sqsubset \tau$. In other words:

$$\frac{\begin{array}{l}\forall \Upsilon \xhookrightarrow{\rho} \Upsilon'.\ \forall M_0, M_1 \in \mathbf{E}_\sigma(\Upsilon'). \\ (\Upsilon' \parallel \Psi \cdot \rho \vDash M_0 = M_1 \in \chi \cdot \rho) \\ \quad \Longrightarrow\ (\Upsilon' \parallel \Psi \cdot \rho \vDash [M_0 \,/\, x]\,\phi \cdot \rho = [M_1 \,/\, x]\,\psi \cdot \rho \sqsubset \tau)\end{array}}{\Upsilon \parallel \Psi, x : \chi \vDash \phi = \psi \sqsubset \tau}\ \vDash^{\sqsubset}_{\mathsf{snoc}}$$

2.4 Functional Equality

Functional equality $\Upsilon \parallel \Psi \vDash N_0 = N_1 \in \phi$ presupposes $\Upsilon \parallel \Psi \sqsubset^\star \Gamma$, $\Upsilon \parallel \Psi \vDash \phi \sqsubset \tau$, and $N_0, N_1 \in \mathbf{E}_\tau(\Upsilon \parallel \Gamma)$; this judgment is defined by induction on the evidence for the first presupposition $\Upsilon \parallel \Psi \sqsubset^\star \Gamma$:

Case $\dfrac{}{\Upsilon \parallel \cdot \sqsubset^\star \cdot}\ \sqsubset^\star_{\mathtt{nil}}$.

$$\frac{\Upsilon \parallel N_0 = N_1 \in \phi}{\Upsilon \parallel \cdot \vDash N_0 = N_1 \in \phi}$$

Case $\dfrac{\Upsilon \parallel \Psi \sqsubset^\star \Gamma \quad \Upsilon \parallel \Psi \vDash \chi \sqsubset \sigma}{\Upsilon \parallel \Psi, x : \chi \sqsubset^\star \Gamma, x : \sigma}\ \sqsubset^\star_{\mathsf{snoc}}$.

To know $\Upsilon \parallel \Psi, x : \chi \vDash N_0 = N_1 \in \phi$ is to know, for any renaming $\Upsilon \xhookrightarrow{\rho} \Upsilon'$ and closed terms $M_0, M_1 \in \mathbf{E}_\sigma(\Upsilon')$, that from $\Upsilon' \parallel \Psi \cdot \rho \vDash M_0 = M_1 \in \chi \cdot \rho$, you can conclude $\Upsilon' \parallel \Psi \cdot \rho \vDash [M_0 \,/\, x]\,N_0 \cdot \rho = [M_1 \,/\, x]\,N_1 \cdot \rho \in [M_0 \,/\, x]\,\phi \cdot \rho$. In other words:

$$\frac{\begin{array}{l}\forall \Upsilon \xhookrightarrow{\rho} \Upsilon'.\ \forall M_0, M_1 \in \mathbf{E}_\sigma(\Upsilon'). \\ (\Upsilon' \parallel \Psi \cdot \rho \vDash M_0 = M_1 \in \chi \cdot \rho) \\ \quad \Longrightarrow\ (\Upsilon' \parallel \Psi \cdot \rho \vDash [M_0 \,/\, x]\,N_0 \cdot \rho = [M_1 \,/\, x]\,N_1 \cdot \rho \in [M_0 \,/\, x]\,\phi \cdot \rho)\end{array}}{\Upsilon \parallel \Psi, x : \psi \vDash N_0 = N_1 \in \phi}$$

3 Constructions on Refinements

Refinements can be arranged in a partial order via the $\Upsilon \parallel \phi \subseteq \psi \sqsubset \tau$ judgment, and admit a lattice structure at any sort τ.

We have the bottom refinement $\bot_\tau \sqsubset \tau$, which simply contains no elements. Next, the top refinement $\top_\tau \sqsubset \tau$ identifies all terms $M, N \in \mathbf{E}_\tau(\Upsilon)$. The join (union) $\Upsilon \parallel \phi \cup \psi \sqsubset \tau$ of two refinements $\Upsilon \parallel \phi \sqsubset \tau$ and $\Upsilon \parallel \psi \sqsubset \tau$ is defined as follows:

$$\frac{\Upsilon \parallel M = N \in \phi}{\Upsilon \parallel M = N \in \phi \cup \psi} \qquad \frac{\Upsilon \parallel M = N \in \psi}{\Upsilon \parallel M = N \in \phi \cup \psi}$$

Finally the meet (intersection) $\Upsilon \parallel \phi \cap \psi \sqsubset \tau$ of two refinements $\Upsilon \parallel \phi \sqsubset \tau$ and $\Upsilon \parallel \psi \sqsubset \tau$ is defined dually:

$$\frac{\Upsilon \parallel M = N \in \phi \quad \Upsilon \parallel M = N \in \psi}{\Upsilon \parallel M = N \in \phi \cap \psi}$$

It is trivial to verify that these constructions on refinements give rise to a lattice.

3.1 Families of Refinements

Given a family of refinements (i.e. a functional refinement) $\Upsilon \parallel x : \phi \vDash \psi \sqsubset \tau$ such that $\Upsilon \parallel \phi \sqsubset \sigma$, including the family intersection, the dependent intersection, the family union, and the set comprehension.

Family intersection The intersection $\Upsilon \parallel \bigcap_{(x:\phi)} \psi \sqsubset \tau$ of a family of refinements is defined by the following rule, for any $\Upsilon \xhookrightarrow{\rho} \Upsilon'$:

$$\frac{\Upsilon' \parallel x : \phi \vDash M = N \in [M\,/\,x]\,\psi}{\Upsilon' \parallel M = N \in \bigcap_{(x:\phi)} \psi}$$

The intersection of a family of refinements is a kind of uniform universal quantification, in the sense that the terms themselves bear no trace of the quantification, but are guaranteed in the metalogic to be related in all fibres of ψ.

In practice, the family intersection is perhaps the most-used type constructor in Nuprl developments, as it facilitates the simultaneous specification and implementation of programs in type theory, without causing details of the program's verification to leak into the extracted algorithm. Additionally, the family intersection can be used to implement coinductive types.

Dependent intersection The dependent intersection $\Upsilon \parallel (x : \phi) \cap \psi \sqsubset \sigma$ contains the members M of ϕ which are also members of $[M\,/\,x]\,\psi$:

$$\frac{\Upsilon' \parallel M = N \in \phi \quad \Upsilon' \parallel M = N \in [M\,/\,x]\,\psi}{\Upsilon' \parallel M = N \in (x : \phi) \cap \psi}$$

The dependent intersection, first introduced by Kopylov in [10], are a generalization of the binary intersection to the dependent case, not to be confused with the intersection of a family of sets. The dependent intersection has many applications, including the compositional representation of *dependent records* or modules in Type Theory.

Family union The union $\Upsilon \parallel \bigcup_{(x:\phi)} \psi \sqsubset \tau$ of a family of refinements is defined as by the following rule:

$$\frac{\Upsilon' \parallel L_0 = L_1 \in \phi \quad \Upsilon' \parallel M = N \in [L_0 \,/\, x]\,\psi}{\Upsilon' \parallel M = N \in \bigcup_{(x:\phi)} \psi}$$

The union of a family of refinements can be used to implement inductive types, among other things.

Set comprehension We can also use a family of refinements to define something analogous to set comprehension, $\Upsilon' \parallel \{\, x : \phi \mid \psi \,\} \sqsubset \sigma$, defined as follows:

$$\frac{\Upsilon' \parallel M = N \in \phi \quad \Upsilon' \parallel L_0 = L_1 \in [M \,/\, x]\,\psi}{\Upsilon' \parallel M = N \in \{\, x : \phi \mid \psi \,\}}$$

The set comprehension $\{\, x : \phi \mid \psi \,\}$ contains just those elements of ϕ such that their fibres of ψ are inhabited; this can be seen as an "implicit" version of existential quantification, and can be used to simultaneously specify and implement program without the evidence for ψ intruding into the extracted algorithm.

4 Multi-Sorted Computational Type Theory

Martin-Löf's type theory is a theory of *value types*: to define a type, you specify how to form its equal canonical inhabitants, or values. Then, the inhabitants of the types are explained uniformly by extending this relation over an operational semantics such that two terms are equal members of a type just when they compute to equal values of that type.

More generally, one may wish to consider types which include computations in addition to values; an example of such a type would be $\psi_\perp$ when ψ is a type, which would include all the members of ψ as well as a divergent computation. Another example is the type $\top_\tau$ which identifies all closed terms of sort τ.

In order to ensure that such types are sensible, we must formulate a notion of computational equivalence, and then say that a refinement is a *type* just when it respects computational equivalence; then, we are able to define types as PERs on *programs* in addition to values.

4.1 Lazy Computation Systems

In this section we generalize Howe's notion of lazy computation system [9] to the multi-sorted, symbol-parameterized setting. An *lazy computation language* is an abt signature $\Sigma \equiv \langle\, \mathcal{S}, \mathcal{O} \,\rangle$ along with a distinguished arity-indexed family of copresheaves $\mathcal{K}_a : \mathbf{Set}^{\mathbb{I}\downarrow\mathcal{S}}$ $(a \in \mathcal{A})$ of *canonical* operators such that $\mathcal{K}_a \subseteq \mathcal{O}_a$ for each arity a.

For a lazy computation language $\mathbb{L} \equiv \langle \mathcal{S}, \mathcal{O}, \mathcal{K} \rangle$, let us define sort-indexed (resp. valence-indexed) copresheaves on $\mathbb{I} \downarrow \mathcal{S}$ of closed values (resp. bound terms) as follows:

$$\mathbf{V}_\tau(\Upsilon) \triangleq \left\{ M \equiv \vartheta(\vec{E}) \mid M \in \mathbf{E}_\tau(\Upsilon) \wedge \exists a.\mathcal{K}_a(\Upsilon) \ni \vartheta \right\}$$
$$\mathbf{B}_\nu(\Upsilon) \triangleq \{ E \mid \cdot \triangleright \Upsilon \parallel \cdot \vdash E : \nu \}$$

Then, a *lazy computation system* (lcs) is a lazy computation language $\mathbb{L} \equiv \langle \Sigma, \mathcal{K} \rangle$ along with an $\mathbb{I} \downarrow \mathcal{S}$-indexed $\equiv_\alpha$-functional evaluation relation $\Upsilon \parallel M \Downarrow_\tau^n N$ presupposing $n \in \mathbb{N}$, $M \in \mathbf{E}_\tau(\Upsilon)$ and $N \in \mathbf{V}_\tau(\Upsilon)$, expressing that M evaluates to N in n steps. We will write $\Upsilon \parallel M \Downarrow_\tau N$ to mean that there exists an n such that $\Upsilon \parallel M \Downarrow_\tau^n N$.

Remark 4.1. In any topos $\mathcal{E}$, for an object X we can define the object of relations on X as the exponential $\wp(X) \triangleq \Omega^X$. Then, the data of such a relation is contained in a global section $R : \mathcal{E}[\mathbb{1}, \wp(X)]$.

Fix a sort-indexed family of binary relations on closed expressions $R_\tau : \mathbf{Set}^{\mathbb{I}\downarrow\mathcal{S}}\left[\mathbb{1}, \wp\left(\mathbf{E}_\tau^2\right)\right]$; we will also write the relation as a judgment scheme $\Upsilon \parallel M \; R_\tau \; N$. Now, we can always extend R_τ to a new family of relations $R_\nu : \mathbf{Set}^{\mathbb{I}\downarrow\mathcal{S}}\left[\mathbb{1}, \wp\left(\mathbf{B}_\nu^2\right)\right]$ for any valence $\nu \equiv \{\vec{\sigma}\}[\vec{\tau}].\tau$, defined pointwise as follows:

$$\frac{\begin{array}{c}\forall \vec{w} \# |\Upsilon| \cup \vec{u} \cup \vec{v}. \quad \forall \Upsilon, \vec{w} : \vec{\sigma} \xhookrightarrow{\rho} \Upsilon'. \quad \forall \vec{X} : \mathbf{E}^{[\vec{\tau}]}(\Upsilon'). \\ \Upsilon' \parallel \left[\vec{X}/\vec{x}\right] M \cdot (\rho \circ \vec{u} \mapsto \vec{w}) \; R_\tau \; \left[\vec{X}/\vec{y}\right] M \cdot (\rho \circ \vec{u} \mapsto \vec{w})\end{array}}{\Upsilon \parallel \mathbb{\lambda}\{\vec{u}\}[\vec{x}].\, M \; R_{\{\vec{\sigma}\}[\vec{\tau}].\tau} \; \mathbb{\lambda}\{\vec{v}\}[\vec{y}].\, N}$$

Notation. *For a family of copresheaves $X_\tau : \mathbf{Set}^{\mathbb{I}\downarrow\mathcal{S}}$ $(\tau \in \mathcal{S})$, we will use $X^{[\vec{\tau}]}$ as a shorthand for the following iterated product, as in [17]:*

$$X^{[\vec{\tau}]} \triangleq \prod_{\tau \in \vec{\tau}} X_\tau$$

As a matter of convenience, we'll also define this judgment over vectors of bound terms and valences:

$$\frac{\forall (E, F, \nu) \in (\vec{E}, \vec{F}, \vec{\nu}).\, \Upsilon \parallel E \; R_\nu \; F}{\Upsilon \parallel \vec{E} \; R_{\vec{\nu}} \; \vec{F}}$$

Now, we can extend R_τ to a new relation $[R_\tau]$ on closed expressions which respects a single "layer" of computation. We will say $\Upsilon \parallel M \; [R_\tau] \; N$ when, supposing $\Upsilon \parallel M \Downarrow_\tau \vartheta(\vec{E})$ such that $\Upsilon \Vdash_{\mathcal{K}} \vartheta : (\vec{\nu})\tau$, for any $\Upsilon \xhookrightarrow{\rho} \Upsilon'$, we have $\Upsilon' \parallel N \cdot \rho \Downarrow_\tau \vartheta \cdot \rho(\vec{F} \cdot \rho)$ and $\Upsilon' \parallel \vec{E} \cdot \rho \; R_{\vec{\nu}} \; \vec{F} \cdot \rho$.

Definition 4.2 (Computational Approximation)**.** Because $[-_\tau]$ is monotonic, it has a greatest fixed point, which we will call *computational approximation*, $\preccurlyeq_\tau : \mathbf{Set}^{\mathbb{I}\downarrow\mathcal{S}}\left[\mathbb{1}, \wp\left(\mathbf{E}_\tau^2\right)\right]$, written pointwise as $\Upsilon \parallel M \preccurlyeq_\tau N$.

Definition 4.3 (Computational Equivalence)**.** Two terms are said to be *computationally equivalent* when they approximate each other:

$$\frac{\Upsilon \parallel M \preccurlyeq_\tau N \quad \Upsilon \parallel N \preccurlyeq_\tau M}{\Upsilon \parallel M \sim_\tau N}$$

4.2 When is a refinement a type?

As we remarked in section 2.1, the notion of *refinement* that we described is very much akin to a multi-sorted generalization of Martin-Löf's explanation of types in [13], except that we have not required an operational semantics to be present prior to the definition of refinements.

Following Martin-Löf, a type is a refinement which behaves well with respect to computation, in some specified sense; contrary to Martin-Löf, however, we wish to consider a full range of *computational types*, including those which are not necessarily defined by their values; a general apparatus for computation-respecting refinements will allow us to define, for instance, a type for partial functions.

Definition 4.4 (Typehood)**.** For any refinement $\Upsilon \parallel \phi \sqsubset \tau$, we say $\Upsilon \parallel \phi \ \mathit{type}_\tau$ just when for any $\Upsilon \xhookrightarrow{\rho} \Upsilon'$, and for all $L, M, N \in \mathbf{E}_\tau(\Upsilon')$, from $\Upsilon' \parallel L \sim_\tau M$ and $\Upsilon' \parallel M = N \in \phi$ we can conclude $\Upsilon' \parallel L = N \in \phi$.

Stated internally as an object in the topos, the meaning of parametric typehood for ϕ is perhaps a bit more clear:

$$- \parallel \phi \ \mathit{type}_\tau \triangleq \prod_{L,M,N \in \mathbf{E}_\tau} \phi(L,N)^{L \sim_\tau M \times \phi(M,N)}$$

Let us write $\mathfrak{type}_\tau$ for the object of types,

$$\mathfrak{type}_\tau \triangleq \left\{ \phi \in \mathfrak{per}(\mathbf{E}_\tau) \mid - \parallel \phi \ \mathit{type}_\tau \right\}$$

Now, let us return briefly to Martin-Löf's notion of type, which we call a *value type*. The minimal data of such a type's definition consists in a partial equivalence relation on values, $\phi : \mathbf{Set}^{\mathbb{N}\downarrow\mathcal{S}}\,[\mathfrak{N}(\Upsilon), \mathfrak{per}(\mathbf{V}_\tau)]$. It is easy to turn such a relation into a computational type $\Upsilon \parallel \mathfrak{c}(\phi) \ \mathit{type}_\tau$, as follows:

$$\begin{array}{ccc} \mathfrak{N}(\Upsilon) \times \mathbf{V}_\tau^2 & \xrightarrow{\lambda(\phi)} & \Omega \\ {\scriptstyle \langle 1, i \rangle} \downarrow & \nearrow_{\lambda(\mathfrak{c}(\phi))} & \\ \mathfrak{N}(\Upsilon) \times \mathbf{E}_\tau^2 & & \end{array}$$

where $\mathfrak{c}(\phi)$ is the computational extension of ϕ in the following sense:

$$\mathfrak{c}(\phi)(M, N) \triangleq \forall M', N' \in \mathbf{V}_\tau.\ M \sim_\tau M' \wedge N \sim_\tau N' \implies \phi(M', N')$$

Theorem 4.5. *When viewed as an internal functor* $\mathfrak{per}(\mathbf{V}_\tau) \to \mathfrak{type}_\tau$, $\mathfrak{c}$ *is the left adjoint to the canonical inclusion* $\iota : \mathfrak{type}_\tau \hookrightarrow \mathfrak{per}(\mathbf{V}_\tau)$.

Proof. We will exhibit a counit $\epsilon : \mathfrak{c} \circ \iota \to 1_{\mathfrak{per}(\mathbf{V}_\tau)}$ and a unit $\eta : 1_{\mathfrak{type}_\tau} \to \iota \circ \mathfrak{c}$ for the adjunction $\mathfrak{c} \dashv \iota$.

The counit witnesses the fact that for any refinement $\phi : \mathfrak{type}_\tau$ and terms $M, N \in \mathbf{E}_\tau$, if $\mathfrak{c}(\iota(\phi))(M, N)$, then $\phi(M, N)$; this is evidently the case, because the premise only obtains if M, N are computationally equal to values, and ϕ respects computational equivalence by virtue of being a type. The unit witnesses the fact that for any value PER $\psi : \mathfrak{per}(\mathbf{V}_\tau)$ and values $M, N \in \mathbf{V}_\tau$, if $\psi(M, N)$, then $\iota(\mathfrak{c}(\psi))(M, N)$; this is clearly true, since $\mathfrak{c}(\psi)$ can do nothing but coarsen ψ's relation on values.

□

Definition 4.6 (Value Closure)**.** For any computational type $\Upsilon \parallel \phi$ *type*$_\tau$, let $\underline{\phi} \triangleq \mathfrak{c}(\iota(\phi))$ be called ϕ's *value closure*, which we will use shortly in order to give a formal definition to the notion of value types.

Definition 4.7 (Value Types)**.** We can give a simple characterization of Martin-Löf's *value types* in terms of our more general computational types. For any type $\Upsilon \parallel \phi$ *type*$_\tau$, we will say $\Upsilon \parallel \phi$ *vtype*$_\tau$ in case ϕ is equal to its value closure, i.e. $\Upsilon \parallel \phi = \underline{\phi} \sqsubset \tau$.

References

[1] S. Allen. An abstract semantics for atoms in nuprl. Technical report, 2006.

[2] J. Cartmell. Generalised algebraic theories and contextual categories. *Annals of Pure and Applied Logic*, 32:209 – 243, 1986.

[3] R. L. Constable, S. F. Allen, H. M. Bromley, W. R. Cleaveland, J. F. Cremer, R. W. Harper, D. J. Howe, T. B. Knoblock, N. P. Mendler, P. Panangaden, J. T. Sasaki, and S. F. Smith. *Implementing Mathematics with the Nuprl Proof Development System*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1986.

[4] P. Dybjer. Internal type theory. In S. Berardi and M. Coppo, editors, *Types for Proofs and Programs*, volume 1158 of *Lecture Notes in Computer Science*, pages 120–134. Springer Berlin Heidelberg, 1996.

[5] R. Harper. *Practical Foundations for Programming Languages*. Cambridge University Press, New York, NY, USA, 2016.

[6] R. Harper and R. Davies. Refining Objects (Preliminary Summary). `https://www.cs.cmu.edu/~rwh/papers/lc60/lc60.pdf`, 2014.

[7] R. Harper and W. Duff. Type Refinements in an Open World. `https://www.cs.cmu.edu/~rwh/papers/trow/summary.pdf`, 2015.

[8] J. Hickey, A. Nogin, R. L. Constable, B. E. Aydemir, E. Barzilay, Y. Bryukhov, R. Eaton, A. Granicz, A. Kopylov, C. Kreitz, V. N. Krupski, L. Lorigo, S. Schmitt, C. Witty, and X. Yu. MetaPRL – a modular logical environment. In D. Basin and B. Wolff, editors, *Theorem Proving in Higher Order Logics*, volume 2758 of *Lecture Notes in Computer Science*, pages 287–303. Springer Berlin Heidelberg, 2003.

[9] D. J. Howe. Equality in lazy computation systems. In *Proceedings of Fourth IEEE Symposium on Logic in Computer Science*, pages 198–203.

[10] A. Kopylov. Dependent intersection: A new way of defining records in type theory. Technical report, Ithaca, NY, 2000.

[11] S. Mac Lane. *Categories for the Working Mathematician*. Number 5 in Graduate Texts in Mathematics. Springer-Verlag, 1971.

[12] P. Martin-Löf. An intuitionistic theory of types, 1972.

[13] P. Martin-Löf. Constructive mathematics and computer programming. In L. J. Cohen, J. Łoś, H. Pfeiffer, and K.-P. Podewski, editors, *Logic, Methodology and Philosophy of Science VI, Proceedings of the Sixth International Congress of Logic, Methodology and Philosophy of*

Science, Hannover 1979, volume 104 of *Studies in Logic and the Foundations of Mathematics*, pages 153–175. North-Holland, 1982.

[14] P. Martin-Löf and G. Sambin. *Intuitionistic type theory*. Studies in proof theory. Bibliopolis, Napoli, 1984.

[15] D. Scott. Constructive validity. In M. Laudet, D. Lacombe, L. Nolin, and M. Schtzenberger, editors, *Symposium on Automatic Demonstration*, volume 125 of *Lecture Notes in Mathematics*, pages 237–275. Springer Berlin Heidelberg, 1970.

[16] J. Sterling, D. Gratzer, and V. Rahli. JonPRL. `http://www.jonprl.org/`, 2015.

[17] J. Sterling and D. Morrison. Syntax and Semantics of Abstract Binding Trees. 2015.