Type-Theory In Color

Jean-Philippe Bernardy Guilhem Moulin
Chalmers University of Technology and University of Gothenburg
{bernardy,mouling}@chalmers.se

Abstract

Dependent type-theory aims to become the standard way to formalize mathematics at the same time as displacing traditional platforms for high-assurance programming. However, current implementations of type theory are still lacking, in the sense that some obvious truths require explicit proofs, making type-theory awkward to use for many applications, both in formalization and programming. In particular, notions of erasure are poorly supported.

In this paper we propose an extension of type-theory with colored terms, color erasure and interpretation of colored types as predicates. The result is a more powerful type-theory: some definitions and proofs may be omitted as they become trivial, it becomes easier to program with precise types, and some parametricity results can be internalized.

Categories and Subject Descriptors F.4.1 [*Mathematical logic*]: Lambda calculus and related systems

Keywords type-theory, parametricity, erasure

1. Introduction

Intelligent use of color in a written argument can go a long way into conveying an idea. But how convincing can it really be? Consider the case of the computer scientist Philip Wadler, who is fond of using color in his papers. On multiple occasions, Wadler (2003, 2007, 2012) presents a programming language and its type-system, and shows that, by erasing the appropriate parts of the type-system, a logic appears. This is done by a straightforward but clever use of colors. Typically, in the presentation of a typing rule, the program parts are written in blue. The corresponding logic rule appears if one erases that color. As Wadler suggests, one can see the erasure simply by putting on blue glasses.

$$\frac{f : A \to B \qquad u : A}{f\ u : B} \qquad\qquad \frac{A \to B \qquad A}{B}$$

Typing rule for application — After erasure: modus ponens

The relationship between the programming language and the logic is deep: for every aspect of the language, there is a "blue part" that can be erased away to obtain the corresponding logical concept. For example, a computation step on the programming side yields a cut-elimination step on the logic side. That is, Wadler does not play mindlessly with colors, he is consistent; he follows in fact a precise (however unwritten) logic of colors[1]. In fact, his "proofs by putting on glasses" are extremely compelling.

In this paper, we set-out to formalize this kind of reasoning with colors as an extension of dependent type-theory. The result is a more powerful type-theory: some definitions and proofs may be omitted as they become trivial; it becomes easier to program with precise types; and some propositions that were impossible to prove now become provable.

In Sec. 2 we demonstrate how one can program and reason with colors via a number of examples, and introduce the fundamental concepts of type-theory with color at the same time. In Sec. 3, we describe CCCC (the main technical contribution of this paper), a core calculus of constructions with colors, and prove meta-theoretical properties (subject-reduction, normalization). In Sec. 4, we discuss some possible extensions of CCCC. Related work is discussed in Sec. 5, and we conclude in Sec. 6.

2. Programming And Reasoning With Colors

In this section we explain how we envision a full-featured type-theory with color (TTC) would be designed, in the form of a short tutorial. We assume familiarity with a proof assistant based on type-theory such as AGDA or COQ (Norell 2007; The Coq development team 2012).

2.1 Colored Lists

We start with an example similar in structure, but significantly simpler than those presented by Wadler. In the standard inductive definition of lists, the structure of the list does not depend on the elements it contains. Hence, it makes sense to color the elements: erasing that color yields a meaningful definition. In fact, the result is structurally equal to the usual unary representation of natural numbers:

$$\mathbb{N} = \lfloor \mathsf{List}\ a \rfloor_i$$

(In this section we assume a color i that we render in blue. Even though we strongly recommend reading the colored version, we index an i-tainted typing with i, so readers can make sense of what follows even if it is printed in black and white.)

$$\begin{array}{l} data\ \mathsf{List}\ (a :_i \star) : \star \\ \quad \mathsf{stop} : \mathsf{List}\ a \\ \quad \mathsf{more} : (x :_i a) \to \mathsf{List}\ a \to \mathsf{List}\ a \\ data\ \mathsf{List} \qquad\quad : \star \\ \quad \mathsf{stop} : \mathsf{List} \\ \quad \mathsf{more} : \qquad\quad \mathsf{List} \quad \to \mathsf{List} \end{array}$$

As in Wadler's examples, the relationship between colored objects and their erasure carries over everywhere. For example, erasing color from a given list yields its length — and the typing relation is preserved.

[1] A meta-level logic, not to be confused with the object logics studied by Wadler.

$$\begin{array}{l}\mathsf{more}\ 'b'\ (\mathsf{more}\ 'l'\ (\mathsf{more}\ 'u'\ \mathsf{stop})) : \mathsf{List}\ \mathsf{Char}\\ \mathsf{more}\quad(\mathsf{more}\quad(\mathsf{more}\quad\mathsf{stop})) : \mathsf{List}\end{array}$$

Concatenation yields addition; assuming $a :_i \star$:

$$\begin{array}{llllll}(+) : \mathsf{List}\ a & \to \mathsf{List}\ a & \to \mathsf{List}\ a\\ \mathsf{stop} & + & x_s & = & x_s\\ \mathsf{more}\ x\ x_s & + & y_s & = & \mathsf{more}\ x\ (x_s + y_s)\\ (+) : \mathsf{List} & \to \mathsf{List} & \to \mathsf{List}\\ \mathsf{stop} & + & x_s & = & x_s\\ \mathsf{more}\quad x_s & + & y_s & = & \mathsf{more}\quad (x_s + y_s)\end{array}$$

This structural relation is a benefit of abiding to color discipline: colorless parts shall never refer to tainted ones, and in return one gets some equalities for free. For example, the length of the concatenation is the addition of the lengths. This proposition requires a proof in AGDA or COQ, but thanks to colors, it holds by definition. Writing $\lfloor t \rfloor_i$ for the i-erasure of t:

$$\lfloor x_s + y_s \rfloor_i = \lfloor x_s \rfloor_i + \lfloor y_s \rfloor_i$$

In fact, TTC does not have a special purpose operator for erasure: the context determines whether variables refer to complete objects or to their erasures. For example in the following signature, the annotation $\dot{\iota}$ indicates that type of the first argument of $<$ is any type which yields $\mathbb{N}$ after erasing i. (We will say that x_s is oblivious to i in the definition of $<$.)

$$(<) : (x_s :_{\dot{\iota}} \mathbb{N}) \to (y : \mathbb{N}) \to \mathsf{Bool}$$

Hence, if $x_s : \mathsf{List}\ a$ then $x_s < 5$ is a valid expression[2]; and it tests whether x_s has less than 5 elements. The expression $3 < 5$ is also type-correct, because the erasure is idempotent. In general, it has no effect on terms which do not mention the erased color ($\lfloor \mathbb{N} \rfloor_i = \mathbb{N}$).

We note also already that substitution behaves specially on oblivious arguments. Consider again the expression $x_s < 5$. In it, one can substitute for x_s a concrete list containing information. The remarkable feature is that in the resulting term, the *erased* list will stand for x_s. For example:

$$\begin{array}{l}(x_s < 5)[\mathsf{more}\ 'b'\ (\mathsf{more}\ 'l'\ (\mathsf{more}\ 'u'\ \mathsf{stop}))/x_s]\\ \quad = \mathsf{more}\quad(\mathsf{more}\quad(\mathsf{more}\quad\mathsf{stop})) < 5\\ \quad = 3 < 5\end{array}$$

2.2 Types as predicates

By using colored types, one effectively specifies structural invariants. For example, the type of the above concatenation operation constrains the length of its result. In our TTC it is possible to reveal these invariants explicitly by viewing types as predicates, and terms as proofs that the predicates are satisfied by the i-erasure. This is done by modulating the typing judgment. For example, under the modality i, the type of lists is seen as a predicate over $\mathbb{N}$. To indicate that the judgment is modulated, the typing operator (colon) is indexed with i.

$$\mathsf{List}\ (a :_i \star)\ :_i\ (x_s :_{\dot{\iota}} \mathbb{N}) \to \star$$

Any typing can be so modulated. For example, a list $x_s : \mathsf{List}\ a$ becomes a proof that $\lfloor x_s \rfloor_i$ satisfies the $\mathsf{List}\ a$ seen as a predicate.

$$x_s :_i \mathsf{List}\ a \bullet_{\dot{\iota}} x_s$$

(To avoid confusion we write predicate test using the $\bullet_{\dot{\iota}}$ operator. Formally it is just the application corresponding to i-oblivious abstraction.) Likewise, the concatenation returns a list whose length is the sum of the lengths of its inputs.

$$(+) :_i (x_s : \mathsf{List}\ a) \to (y_s : \mathsf{List}\ a) \to \mathsf{List}\ a \bullet_{\dot{\iota}} (x_s + y_s)$$

[2] We take the liberty to use decimal notation for unary naturals.

(To make sense of the above typings, recall that the second argument to List is a natural after erasing i; in general a variable $x_s :_i \mathsf{List}\ a \bullet_{\dot{\iota}} n$ stands for a list of size n.)

Additionally, we remark that even though every type becomes a predicate, the computations (or data) that it represents do not essentially change. Taking our list example, the union of the types $\mathsf{List}\ a \bullet_{\dot{\iota}} n$ for any $n : \mathbb{N}$ is isomorphic to $\mathsf{List}\ a$ seen as a type. Hence, assuming one has existential quantification, the type $A : \star$ remains available as a type under the modality i, as $\exists x.A\,x : \star$. Hence, one can continue to use types as types, even under a colored typing, referring implicitly to the above existential construction. We will take advantage of this shortcut in Sec. 2.4 for concision.

2.3 Colored Pairs

Another (dual) way to introduce colors is via pairs. A colored pair type, whose general form is written $(x : A) \times_i B$, is similar to the usual type $\Sigma(x : A)\ B$ (in particular x may occur in B). The difference is that B is tainted with the color i; and A is oblivious to i. Given $a : A$ and $b :_i B[a/x]$ one can construct an inhabitant of the pair type. As usual, colors must match: $a,_i b : (x : A) \times_i B$ is valid only if b is tainted and a is oblivious to i. Erasure extracts the first component of a pair, and interpreting a pair as a predicate yields its second component. The following example illustrates how colored pairs can be used to prove some parametricity properties. Assume the following (i-oblivious) context.

$$\begin{array}{l}f : (a : \star) \to a \to a\\ b : \star\\ y : b\end{array}$$

We then can define

$$\begin{array}{l}t : (x : b) \times_i (x \equiv y)\\ t = f\ ((x : b) \times_i (x \equiv y))\ (y,_i \mathsf{refl})\end{array}$$

By definition of erasure:

$$\lfloor t \rfloor_i = f\ b\ y \qquad (1)$$

The above equation can also be intuited by looking at t under i-glasses:

$$f\ ((\quad b) \times_i (\lfloor x \rfloor_i \equiv y))\ (y\quad)$$

In an i-modulated typing, one implicitly refers to the colored component of pairs, and therefore we have: $t :_i \lfloor t \rfloor_i \equiv y$. By (1) we obtain

$$t :_i f\ b\ y \equiv y$$

This result is normally obtained by a logical relation argument *outside* the theory, while it is internalized here — albeit via a judgment with an extra color. (A fully formal version of this example is presented in Sec. 4.) It may be worth stressing that, if one were to use a regular pair type, then, because f is abstract, the first component of t would *not* compute. In contrast, erasure is defined even on neutral terms.

2.4 Multiple Colors

We propose to support arbitrarily many colors. This feature is important for compositionality: it ensures that one can always mark a binding as tainted without corrupting interactions with the rest of the program. Indeed the other parts of the programs will use other, orthogonal colors. For example, the $\mathsf{List}\ a$ type described above is sufficient, there is no need to define a version without color. If one needs to access the elements of the list in a function, one simply taints its typing with the color. For example, a summation function may be given the type $\mathsf{sum} :_i \mathsf{List}\,\mathbb{N} \to \mathbb{N}$. The taint will be transitively inherited by all functions using sum (and they can use other colors at will).

In fact, it is advisable to use colors even more effectively, and instead define sum by erasure. Assume the following definition of concat, where the nested lists use a different color j for the type a of elements, which we render in red. (In the type of concat, a occurs in an i-tainted context, so it is tainted both with i and j. We have $a :_{ij} \star$, and we render it with the combination of blue and red: magenta).

$$\begin{aligned}&\mathsf{concat} :_i \mathsf{List}\,(\mathsf{List}\,a) \to \mathsf{List}\,a\\&\mathsf{concat}\;\mathsf{stop} &&= \mathsf{stop}\\&\mathsf{concat}\;(\mathsf{more}\,x\,x_s) &&= x + \mathsf{concat}\,x_s\end{aligned}$$

Then one can obtain sum by erasing j from concat: (sum $=$ $\lfloor \mathsf{concat} \rfloor_j$). This would be impossible with a single color: attempting to erase the elements of the inner lists would erase the whole function.

Another use for multiple colors is to nest pairs. One cannot nest pairs which differentiate on the same given color, because this would break the rule that either side of an i-colored pair must respectively be oblivious to i or be tainted with it. However one can nest pairs which use different colors. The general form of j-colored pairs nested inside an i-colored one is the following:

$$(x : (w : A) \times_j B[w]) \times_i ((z : C[x]) \times_j D[x, z])$$

C can only refer to the j-oblivious part of x. C may not refer to the j-tainted part of w, since it does not carry that color itself. Looking at the above type successively with i and j glasses[3]:

$$\begin{array}{ll}(\quad (w : A) \times_j B[w]) \times_i ((z : C[x]) \times_j D[x, z])\\(x : (\quad A) \times_j B[w]) \times_i ((\quad C[x]) \times_j D[x, z])\end{array}$$

Using nested pairs is syntactically inconvenient, hence in the rest of the section we use a record-like syntax. Using record syntax, the above pair would be written

$$\{w :_{\cancel{ij}} A;\; y :_{\cancel{i}} B[w];\; z :_{\cancel{j}} C[w];\; D((w,_j y)\;,\; z)\}$$

The following example illustrates how multiple colors can be used to program with relations. Assume a definition of streams $\mathsf{Stream} : \star \to \star$. Stream is a functor as witnessed by $\mathsf{map} : (a : \star) \to (b : \star) \to (a \to b) \to \mathsf{Stream}\,a \to \mathsf{Stream}\,b$, with standard definitions. Assume furthermore the following abstract context:

$$\begin{aligned}l &: (a : \star) \to \mathsf{Stream}\,a \to \mathbb{N} \to a\\a &: \star\\b &: \star\\f &: a \to b\\x_s &: \mathsf{Stream}\,a\\p &: \mathbb{N}\end{aligned}$$

We define:

$$\begin{aligned}&r : \star\\&r = \{x :_{\cancel{i}} a;\; y :_{\cancel{j}} b;\; f\,x \equiv y\}\\&u : a \to r\\&u = \lambda z.\{x = z;\; y = f\,z;\; \mathsf{refl}\}\\&t : r\\&t = l\,r\,(\mathsf{map}\,u\,x_s)\,p\\&\quad = l\,\{x :_{\cancel{i}} a;\; y :_{\cancel{j}} b;\; f\,x \equiv y\}\,(\mathsf{map}\,(\lambda z.\{x = z;\; y = f\,z;\; \mathsf{refl}\})\,x_s)\,p\end{aligned}$$

[3] The analogy to perceptual colors still holds here: magenta both appears blue under blue glasses and red under red glasses. (i.e. it fades into the background in both cases.) The analogy breaks only when one uses too many colors: most humans can only perceive three primary colors, while we allow an unbounded number of colors in TTC.

Erasing colors from t yields:

$$\begin{aligned}\lfloor t \rfloor_i &= l\,a\,(\mathsf{map}\,\mathsf{id}_a\,x_s)\,p\\\lfloor t \rfloor_j &= l\,b\,(\mathsf{map}\,f\,x_s)\,p\end{aligned}$$

Indeed, looking at t respectively under i-colored and j-colored glasses:

$$\begin{array}{lllll}l\,\{ & a & \} & (\mathsf{map}\,(\lambda z.\{ & z & \})\,x_s)\,p\\l\,\{ & b & \} & (\mathsf{map}\,(\lambda z.\{ & f\,z & \})\,x_s)\,p\end{array}$$

$$\begin{aligned}&t : r\\&t :_i r\,\lfloor t \rfloor_i\\&t :_{i,j} r\,\lfloor t \rfloor_i\,\lfloor t \rfloor_j\\&t :_{i,j} f\,\lfloor t \rfloor_i \equiv \lfloor t \rfloor_j\\&t :_{i,j} f(l\,a\,(\mathsf{map}\,\mathsf{id}_a\,x_s)\,p) \equiv l\,b\,(\mathsf{map}\,f\,x_s)\,p\end{aligned}$$

Together with $\mathsf{map}\,\mathsf{id}_a\,x_s \equiv x_s$, we obtain the expected law: $f(l\,a\,x_s\,p) \equiv l\,b\,(\mathsf{map}\,f\,x_s)\,p$

2.5 Conclusion

A motto of programming with dependent types is to use more and more types to express one's intentions more and more precisely. However, there is a drawback to precise types: hard work to convince a type-checker that programs inhabit them. We observe that the use of colors is a way to specify invariants in types which does not complicate user code. For instance we have seen that it is just as easy to program with colored lists as with regular ones, and length invariants are captured. Furthermore, the system can automatically discover equations which would require a proof without the use of colors.

One cannot "go wrong" by using more colors in a library. In the worst case, colors can simply be ignored by the users of the library. In the best case, they serve to specify invariants concisely, facilitate reasoning, and provide a variant of the library to the user for each possible erasure combination.

We have implemented a prototype of TTC as an extension of the AGDA system. The prototype, in its current version at the time of writing, features colored bindings and abstraction over colors, but is still lacking erasure, oblivious bindings and colored pairs. The prototype, together with a short tutorial for it, can be obtained online: `http://www.cse.chalmers.se/~mouling/Parametricity/TCC.html`.

3. CCCC: A Core Calculus of Colored Constructions

In this section we present CCCC (the Core Calculus of Colored Constructions) which is the formal core of the TTC we envision. Technically, CCCC is an extension of CC (the plain Calculus of Constructions (Coquand and Huet 1986)) with the notion of color informally introduced in the previous section. Even though we use CC as a base, we do not rely on its specifics. Along the lines presented here, it is conceivable to construct a variant of any type-system with colors, including intuitionistic type-theory (Martin-Löf 1984).

The rest of the section describes the main features of CCCC in pedagogical order. A summary is shown in appendix for reference. We emphasize that we describe only on the core features of TTC. Some features used in the previous section will not be included here, even though it is conceivable to integrate them with limited effort.

Even though we continue to render some expressions in color as a visual aid, the formal system does not rely on them in any way. This section can be read in black and white without any loss in precision.

3.1 CC as a PTS

We use CC as a base, so we recall briefly its definition, using a pure type system (Barendregt 1992) presentation. The typing rules are as follows:

$$\textsc{Conv}\ \frac{\Gamma \vdash a : A \qquad A =_\beta A'}{\Gamma \vdash a : A'} \qquad \textsc{Axiom}\ \frac{\vdash \Gamma}{\Gamma \vdash \star : \Box} \qquad \textsc{Var}\ \frac{\vdash \Gamma \qquad x : A \in \Gamma}{\Gamma \vdash x : A}$$

$$\textsc{Prod}\ \frac{\Gamma, x : A \vdash B : s}{\Gamma \vdash (x : A) \to B : s} \qquad \textsc{Abs}\ \frac{\Gamma, x : A \vdash b : B}{\Gamma \vdash \lambda x : A.b : (x : A) \to B}$$

$$\textsc{App}\ \frac{\Gamma \vdash F : (x : A) \to B \qquad \Gamma \vdash u : A}{\Gamma \vdash F \bullet u : B[u/x]}$$

To limit clutter we omit the well-formedness conditions of types A and B in the rule Abs. The product $(x : A) \to B$ may be also written $A \to B$ when x does not occur free in B, and we generally omit the application operator $\bullet$. The metasyntactic variable s ranges over the sorts $\star$ and $\Box$. The context-lookup relation $(x : A \in \Gamma)$ is straightforward, and the context-formation rules are:

$$\textsc{Empty}\ \frac{}{\vdash -} \qquad \textsc{Bind}\ \frac{\vdash \Gamma \qquad \Gamma \vdash A : s}{\vdash \Gamma, x : A}$$

Traditional presentations of PTSs, including that of Barendregt (1992), use another formulation, which integrates the context-lookup, context-formation and typing rules. Instead of the Var rule, one has the following Weakening and Start rules, and axioms can only be used in the empty context.

$$\textsc{Start}\ \frac{\Gamma \vdash A : s}{\Gamma, x : A \vdash x : A} \qquad \textsc{Weakening}\ \frac{\Gamma \vdash A : B \qquad \Gamma \vdash C : s}{\Gamma, x : C \vdash A : B}$$

While the presentation of Barendregt economizes a couple derivation rules, it has the disadvantage to conflate separate concepts in a single definition. Consequently, is it harder to extend, and modern presentations tend to use separate context-lookup and context-formation relations.

3.2 Colors, Taints and Modalities

We assume an infinite supply of color names; the metasyntactic variables i and j stand for them in the remainder. We call a set of such color names a *taint* and use θ or ι to range over taints. A color may be introduced in the context by its name. After such a mention, terms may contain i-tainted parts, but also non i-tainted parts. In contrast, before the mention of i, terms are i-oblivious: they cannot depend on i in any way.

Our typing judgment $\Gamma \vdash A :_\theta B$ is indexed by a taint θ, which must be a subset of the colors present in Γ. The presence of a given color i in θ indicates how i can be used in A and B.

- $i \notin \theta$ indicates that A is not tainted with i. A term A typed in such a taint may still mention the color i. For example $\lambda(x :_i T) \to a$ is allowed. However, the usage of i-tainted variables is forbidden in the target (a) of the term. For example $\lambda(x :_i T) \to x$ is forbidden.
- $i \in \theta$ indicates that the term A is tainted with i. In such a judgment, using i-tainted variables is allowed in the targets of terms and types. *Remark*: It does not make sense to erase i from a judgment using this taint; conceptually the whole typing is tainted, so it would be entirely removed.

For each taint θ we have two sorts $\star_\theta$ and $\Box_\theta$, with the axiom $\star_\theta : \Box_\theta$. The conversion rule merely preserves taints.

$$\textsc{Conv}\ \frac{\Gamma \vdash a :_\theta A \qquad A =_\beta A'}{\Gamma \vdash a :_\theta A'} \qquad \textsc{Axiom}\ \frac{\vdash \Gamma}{\Gamma \vdash \star_\theta :_\theta \Box_\theta}$$

A variable binding $x :_\psi A$ does not only carry a taint, but also a *modality*. (ψ and φ range over modalities.) A modality is composed of two disjoint sets of colors (say $\psi = (\theta, \iota)$ with $\theta \cap \iota = \varnothing$) that constrain what kind of a term u can be substituted for x. The first set θ is the taint of u. The second set ι is an "anti-taint": a set of colors which u must be oblivious to. We often use the compact notation $i_1 \dots i_n, \cancel{j_1} \dots \cancel{j_m}$ for the modality $(\{i_1 \dots i_n\}, \{j_1 \dots j_n\})$. Similarly we will write $\cancel{j} \in \psi$ to mean that j is found in the second set. Using this notation, the following two contexts are equivalent (one can substitute one for another without changing the provability of a judgment):

$$\Gamma, x : A, i, \Delta$$
$$\Gamma, i, x :_{\cancel{i}} A, \Delta$$

That is, declaring a variable before i, or declaring it i-oblivious explicitly are equivalent. The product and abstraction rules can change the modality of the type quantified over; the application rule behaves correspondingly.

$$\textsc{Prod}\ \frac{\Gamma, x :_\psi A \vdash B :_\theta s}{\Gamma \vdash (x :_\psi A) \to B :_\theta s} \qquad \textsc{Abs}\ \frac{\Gamma, x :_\psi A \vdash b :_\theta B}{\Gamma \vdash (\lambda x :_\psi A.b) :_\theta (x :_\psi A) \to B}$$

$$\textsc{App}\ \frac{\Gamma \vdash F :_\theta (x :_\psi A) \to B \qquad \Gamma \vdash u :_\psi A}{\Gamma \vdash F \bullet_\psi u :_\theta B[u/x]}$$

(where the metasyntactic variable s now ranges over the sorts $\star_\iota$ or $\Box_\iota$, for any ι.) To be able to merely forward modalities, we need to extend the judgment to support arbitrary modalities φ, not just taints θ, as we do at the beginning of the next section.

We generally omit the modality annotation on applications, because they are easily inferred from the context. One can embed any derivation from CC into CCCC simply by using the empty taint everywhere.

3.3 Obliviousness and variable lookup

We extend the typing judgment to support any modality ψ (not just a taint) as follows:

Definition 1 (Oblivious judgment)**.**

$$\textit{If } \psi = (\theta, \iota) \textit{ then } \Gamma \vdash A :_\psi B \triangleq \lfloor \Gamma \rfloor_\iota \vdash A :_\theta B$$

This captures the intuition that A and B are oblivious to every color in ι. Indeed, the erasure removes all mentions of i from the context Γ for every $i \in \iota$ (a complete definition and justification of erasure is given in the following section). In particular, if $x : A \in \Gamma$, then $\Gamma \vdash x :_{\cancel{i}} \lfloor A \rfloor_i$; in words, referencing a variable from an i-oblivious judgment yields only a witness of the i-erased type.

Hence, variable lookup requires equality of taints, not merely inclusion:

$$\textsc{Var}\ \frac{\vdash \Gamma \qquad x :_\theta A \in \Gamma}{\Gamma \vdash x :_\theta A}$$

There are two ways to access an i-oblivious variable x. First, it is accessible in an i-oblivious judgment, as can be seen by expanding Def. 1:

$$\Gamma, x :_{\cancel{i}} A, \Delta \vdash x :_{\cancel{i}} A \triangleq \lfloor \Gamma \rfloor_i, x : A, \lfloor \Delta \rfloor_i \vdash x : A$$

Second, it can be accessed from an i-aware judgment, as formalized in the context-lookup rules:

$$\textsc{Start}\ \frac{}{x :_\theta A \in \Gamma, x :_{(\theta,\iota)} A} \qquad \textsc{Col. Wk}\ \frac{x :_\theta A \in \Gamma \qquad i \notin \theta}{x :_\theta A \in \Gamma, i}$$

$$\textsc{Wk}\ \frac{x :_\theta A \in \Gamma}{x :_\theta A \in \Gamma, y :_\psi B}$$

The Col. Wk rule ensures that x is accessible from an i-aware context (but not an i-tainted one), even if it is declared before the introduction the color i. The Start rule plays a similar role: x can be explicitly oblivious to any set of colors, it does not change its accessibility.

Consider as an example the following definition, a variant of Leibniz equality.

$$x \equiv^i_a y \ \triangleq\ (P :_i (z :_\dot{\iota} a) \to \star_i) \to P\, x \to P\, y$$

One can verify that it is a well-colored type as follows. Let $\Delta = a : \star, x : a, y : a$:

$$\textsc{Prod}\ \frac{\textsc{App}\ \dfrac{\vdots \quad \textsc{Def. 1}\ \dfrac{\Delta \vdash y : a}{\Delta, i, \ldots \vdash y :_\dot{\iota} a}}{\Delta, i, P :_i (z :_\dot{\iota} a) \to \star_i, q :_i P\, x \vdash P\, y : \star_i}}{\Delta, i \vdash x \equiv^i_a y :_i \star_i}$$

With $\mathsf{refl}^i \triangleq \lambda(P :_i \star).\lambda(q :_i P).q$, one can also derive $a : \star, x : a, i \vdash \mathsf{refl}^i :_i x \equiv^i_a x$.

In order to use $\equiv$ as an equality, we need to access oblivious variables from non-oblivious contexts, as we explain in this paragraph. The key difference between $\equiv$ as defined above and the usual Leibniz equality here is that we use propositions P of type $(x :_\dot{\iota} a) \to \star_i$; that is, the parameter of P is i-oblivious.

Thanks to the context lookup rules, a variable $x :_\dot{\iota} \mathsf{Bool}$ may be used even in an i-aware context, so one can construct a proposition Q^i of the right type which returns truth if the Boolean is true and falsity otherwise. Then one can obtain falsity from $\mathsf{true} \equiv_{\mathsf{Bool}} \mathsf{false}$ by substituting Q^i for P. Assuming a definition $\mathsf{Test} :_i \mathsf{Bool} \to \star_i$ which does the adequate case analysis on booleans, one has:

$$\textsc{Def.}\ \frac{\textsc{Abs}\ \dfrac{\textsc{App}\ \dfrac{\vdots \quad \textsc{Var}\ \dfrac{x : \mathsf{Bool} \in i, x :_\dot{\iota} \mathsf{Bool}}{i, x :_\dot{\iota} \mathsf{Bool} \vdash x : \mathsf{Bool}}}{i, x :_\dot{\iota} \mathsf{Bool} \vdash \mathsf{Test}\, x :_i \star_i}}{i \vdash \lambda(x :_\dot{\iota} \mathsf{Bool}).\mathsf{Test}\, x :_i (x :_\dot{\iota} \mathsf{Bool}) \to \star_i}}{i \vdash Q^i :_i (x :_\dot{\iota} \mathsf{Bool}) \to \star_i}$$

3.4 Erasure

Color erasure is defined by structural induction on terms. The effect on each modality is the following. Applying i-erasure on an i-tainted binding removes it. The i-erasure of a non i-tainted binding is the binding of the erasure. Erasing i from an i-oblivious binding has no effect besides removing the mention of $\dot{\iota}$.

Erasure of product, abstraction and application follows directly from the behavior on bindings. Erasure preserves all variable occurrences, as well as sorts. In the following table we sum up all cases. The erasure of terms which do not mention a modality are show in the first column. For terms which mention a modality, we show the various cases in various columns. The first column shows the case where the color occurs nowhere in the modality. The second one shows the case where i occurs as a taint. The third one shows the case where the i occurs as an anti-taint.

	$i \notin \psi$	$i \in \psi$	$\psi = \varphi, \dot{\iota}$
$\lfloor x \rfloor_i$	$= x$		
$\lfloor s \rfloor_i$	$= s$		
$\lfloor (x :_\psi A) \to B \rfloor_i$	$= (x :_\psi \lfloor A \rfloor_i) \to \lfloor B \rfloor_i$	$\lfloor B \rfloor_i$	$(x :_\varphi A) \to \lfloor B \rfloor_i$
$\lfloor \lambda x :_\psi A.b \rfloor_i$	$= \lambda x :_\psi \lfloor A \rfloor_i.\lfloor b \rfloor_i$	$\lfloor b \rfloor_i$	$\lambda x :_\varphi A.\lfloor b \rfloor_i$
$\lfloor F \bullet_\psi a \rfloor_i$	$= (\lfloor F \rfloor_i) \bullet_\psi \lfloor a \rfloor_i$	$\lfloor F \rfloor_i$	$(\lfloor F \rfloor_i) \bullet_\varphi a$
$\lfloor \Gamma, x :_\psi A \rfloor_i$	$= \lfloor \Gamma \rfloor_i, x :_\psi \lfloor A \rfloor_i$	$\lfloor \Gamma \rfloor_i$	$\lfloor \Gamma \rfloor_i, x :_\varphi A$
$\lfloor \Gamma, j \rfloor_i$	$= \lfloor \Gamma \rfloor_i, j$		
$\lfloor \Gamma, i \rfloor_i$	$= \Gamma$		

Lemma 1 (Erasure preserves typing)**.** *If* $\Gamma \vdash A :_\theta B$ *and* $i \notin \theta$ *then* $\lfloor \Gamma \rfloor_i \vdash \lfloor A \rfloor_i :_\theta \lfloor B \rfloor_i$.

Proof. By induction on the derivation. The proof relies on the color-discipline enforced by the typing rules. □

This lemma means that erasure makes sense as a meta-level definition. The precondition is important: erasing i makes no sense on an i-tainted term; conceptually the whole term would be erased in that case. This justifies for example the case for sorts in the definition: the precondition guarantees that erasure will not be applied to a sort $\star_i$. In the system, we use erasure only in situations where this precondition is satisfied.

Erasure is used in the definition of substitution (whose full definition is given in Sec. 6): when substituting in an oblivious argument, or in the type of an oblivious parameter, one needs to erase the substitutee. For example:

$$(f \bullet_\dot{\iota} u)[t/x] = f[t/x] \bullet_\dot{\iota} u[\lfloor t \rfloor_i / x]$$

Note that if x is i-tainted, the type-system prevents any occurrence of x in u, ensuring that the precondition of Lem. 1 is respected.

We have not yet defined how to reduce terms in CCCC, but it is worth mentioning already that erasure preserves computation. A proof is given later in Lem. 3.

$$\text{if } A \longrightarrow^\star B, \quad \text{then} \quad \lfloor A \rfloor_i \longrightarrow^\star \lfloor B \rfloor_i, \text{ for any color } i.$$

3.5 Types as Predicates

By modulating a judgment with a color i, a type B becomes a predicate over $\lfloor B \rfloor_i$ [4]. The erasure of a term A of type B satisfies the type B seen as a predicate.

Theorem 1 (Parametricity of closed terms)**.** *if* $\vdash A :_\theta B$*, then*

$$\vdash [\![A]\!] :_{\theta,i} [\![B]\!] \bullet_\dot{\iota} \lfloor A \rfloor_i$$

Proof. The proof uses the standard techniques of logical relations, extended to dependent types by Bernardy et al. (2010), which we also refer the reader to for the definition of the construction of the parametric interpretation $[\![\cdot]\!]$. □

We wish however not to be limited to closed terms, and want parametricity even on open terms. The presence of colors allows[5] to add the following rule, which internalizes the reinterpretation of terms as predicates and proofs.

$$\textsc{Param}\ \frac{\Gamma \vdash A :_\theta B \qquad i \notin \theta}{\Gamma \vdash A :_{\theta,i} B \bullet_{\theta,\dot{\iota}} \lfloor A \rfloor_i}$$

[4] With the exception of terms which are sorted in under that color. For example, $\star_i :_i \square_i$ and not $\star_i :_i \star_i \to \square_i$. This feature prevents an "infinite descent" into deeper and deeper predicates.

[5] The issues that one faces when attempting to internalize parametricity in a theory without colors are detailed by Bernardy and Moulin (2012).

This rule is a generalization of Th. 1 and it allows to deduce, within the calculus, theorems which could only be obtained meta-theoretically without it. However, in this paper, the treatment of logical relations differs from the usual one: a type is not *interpreted* as a predicate (via a transformation of terms $[\![\cdot]\!]$), but is directly *used* as such in an i-modulated judgment.

In order to use T as a predicate we extend reduction rules as follows, where we recall that s ranges over sorts, while t ranges over terms.

$$s_\theta \bullet_\varphi t \longrightarrow (z :_\varphi t) \to s_{\theta\cup\iota} \quad (1)$$
$$\text{where } \varphi = (\theta, \iota)$$
$$((x :_\psi A) \to B) \bullet_\varphi t \longrightarrow (x :_\psi A) \to (B \bullet_\varphi t) \quad (2)$$
$$\text{if } \exists i \text{ such that } i \in \psi \text{ and } \dot\iota \in \varphi$$
$$((x :_\psi A) \to B) \bullet_\varphi t \longrightarrow (x :_\psi A) \to (B \bullet_\varphi (t \bullet_\psi x)) \quad (3)$$
$$\text{otherwise}$$
$$(\lambda x :_\psi A.b) \bullet_\psi t \longrightarrow b[t/x] \quad (4)$$

- (1): Since a type becomes a predicate, a type of types (a sort) becomes a type of predicates. The target sort of the predicate type is adjusted, in order to obtain a type (and not again a predicate — see footnote 4).
- (2) and (3): A function t satisfies the predicate of a function type if an argument x (which implicitly satisfies the predicate of the domain A) is mapped by the function t to a value satisfying the codomain B. In the case of (2), the modality φ mandates erasure of the domain A, therefore x is not given as an argument to t.
- (4): The β-reduction is trivially amended to account for colors. The rules (1,2,3) do not interfere with β, because they concern other syntactic forms.

The cases (1,2,3) agree with the standard interpretation of types as predicates. Bernardy and Moulin (2012); Bernardy et al. (2012) give a detailed account of logical relations in the presence of dependent types. As an example, one can check that reduction behaves as expected for the list concatenation type:

$$\begin{aligned}&((x_s : \mathsf{List}\, a) \to (y_s : \mathsf{List}\, a) \to \mathsf{List}\, a) \bullet_{\dot\iota} c\\ \longrightarrow\ &(x_s : \mathsf{List}\, a) \to ((y_s : \mathsf{List}\, a) \to \mathsf{List}\, a) \bullet_{\dot\iota} (c\, x_s)\\ \longrightarrow\ &(x_s : \mathsf{List}\, a) \to (y_s : \mathsf{List}\, a) \to \mathsf{List}\, a \bullet_{\dot\iota} (c\, x_s\, y_s)\end{aligned}$$

3.6 Example

Assume the colors i and j as well as the context

$$\begin{aligned}\mathsf{List}_i &: (a :_i \star_i) \to \star\\ \mathsf{List}_j &: (b :_j \star_j) \to \star\\ \mathsf{fold} &: (a :_i \star_i) \to (b : \star) \to\\ &\quad (a \to b \to b) \to b \to \mathsf{List}_i\, a \to b\end{aligned}$$

One can define a variant of the ubiquitous map function as follows:

$$\begin{aligned}map &: (a :_i \star_i) \to (b :_j \star_j) \to\\ &\quad (f :_j (x :_i a) \to b) \to \mathsf{List}_i\, a \to \mathsf{List}_j\, b\\ map &= \lambda f.\mathsf{fold}\, a\, (\mathsf{List}_j\, b)\, (\lambda x\, x_s.\mathsf{more}\, (f\, x)\, x_s)\, \mathsf{stop}\end{aligned}$$

This version of map is versatile. After erasing j, the f argument disappears and map becomes a function returning the length of its input. After erasing i, f becomes a constant of type b, the input "list" becomes a natural number (n), and map returns a list containing n copies of f.

3.7 Analysis

In this section we state and prove a number of standard meta-theoretical results for our calculus.

Lemma 2 (Substitution)**.** *For any term A, u, and v and variables $x \neq y$ such that x is not free in v,*

$$A[u/x][v/y] = A[v/y][u[v/y]/x]$$

Proof. By structural induction on the raw term A. CCCC does not support abstraction over colors, therefore we can ignore the case where x or y are color variables, and the proof below follows exactly the structure of the usual proof of substitution lemma for PTSs. We show only the variable and abstraction cases; other cases are similar.

Variable z As usual, we have the three following cases:

- $z = x$:

  $$x[u/x][v/y] = u[v/y] = x[u[v/y]/x] = x[v/y][u[v/y]/x]$$
- $z = y$: $y[u/x][v/y] = v = y[v/y][u[v/y]/x]$
- otherwise: $z[u/x][v/y] = z = z[v/y][u[v/y]/x]$

Abstraction $\lambda z :_\psi A.b$

$$\begin{aligned}&(\lambda z :_\psi A.b)[u/x][v/y]\\ =\ &(\lambda z :_\psi A[u\{\psi\}/x].b[u/x])[v/y]\\ =\ &\lambda z :_\psi A[u\{\psi\}/x][v\{\psi\}/y].b[u/x][v/y]\\ &\text{by IH}\\ =\ &\lambda z :_\psi A[v\{\psi\}/y][u\{\psi\}[v\{\psi\}/y]/x].\\ &\qquad b[v/y][u[v/y]/x]\\ =\ &(\lambda z :_\psi A.b)[v/y][u[v/y]/x]\end{aligned}$$

□

We proceed to show the confluence of the reduction relation. To do this, we use the Tait/Martin-Löf technique of parallel reduction.

Definition 2 (Parallel nested reduction)**.**

$$\textsc{Refl}\ \frac{}{A \triangleright A} \qquad \beta\ \frac{b \triangleright b' \qquad a \triangleright a'}{(\lambda z :_\psi A.b) \bullet_\psi a \triangleright b'[a'/z]}$$

$$\textsc{AppSort}\ \frac{t \triangleright t' \qquad \varphi = (\theta, \iota)}{s_\theta \bullet_\varphi t \triangleright (z :_\varphi t') \to s_{\theta\cup\iota}}$$

$$\textsc{AppAll}_1\ \frac{\begin{array}{c}\exists i \text{ such that } i \in \psi \text{ and } \dot\iota \in \varphi\\ A \triangleright A' \qquad B \triangleright B' \qquad t \triangleright t\end{array}}{((z :_\psi A) \to B) \bullet_\varphi t \triangleright (z :_\psi A') \to (B' \bullet_\varphi t')}$$

$$\textsc{AppAll}_2\ \frac{\begin{array}{c}\nexists i \text{ such that } i \in \psi \text{ and } \dot\iota \in \varphi\\ A \triangleright A' \qquad B \triangleright B' \qquad t \triangleright t'\end{array}}{((z :_\psi A) \to B) \bullet_\varphi t \triangleright (z :_\psi A') \to (B' \bullet_\varphi (t' z))}$$

$$\textsc{App-Cong}\ \frac{F \triangleright F' \qquad a \triangleright a'}{F \bullet_\psi a \triangleright F' \bullet_\psi a'}$$

$$\textsc{Abs-Cong}\ \frac{A \triangleright A' \qquad b \triangleright b'}{\lambda z :_\psi A.b \triangleright \lambda z :_\psi A'.b'}$$

$$\textsc{All-Cong}\ \frac{A \triangleright A' \qquad B \triangleright B'}{(z :_\psi A) \to B \triangleright (z :_\psi A') \to B'}$$

Since one needs to erase the substitutee when substituting under an oblivious binding (see Def. 10), we use the fact that erasure preserves parallel reduction.

Lemma 3. *For each A, A' such that $A \triangleright A'$, we have $\lfloor A \rfloor_i \triangleright \lfloor A' \rfloor_i$ for all i.*

Proof. By induction on the derivation $A \triangleright A'$. □

We can now prove that substitution preserves parallel reduction.

Lemma 4. *For each A,A' and u,u' such that $A \triangleright A'$ and $u \triangleright u'$, we have $A[u/x] \triangleright A'[u'/x]$.*

Proof. By induction on the derivation $A \triangleright A'$. The proof is almost completely standard, except for the use of Lem. 3. In addition of the β case which uses it, we show the REFL case for reference. Other cases are similar or standard.

REFL: $A \triangleright A$. We get $A[u/x] \triangleright A[u'/x]$ by structural induction on A.

β: $(\lambda z :_\psi A.b) \bullet_\psi a \triangleright b'[a'/z]$.

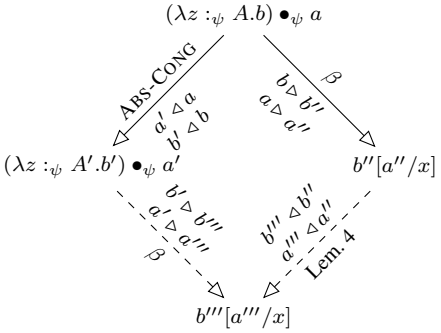
$$\begin{array}{ll} & ((\lambda z :_\psi A.b) \bullet_\psi a)[u/x] \\ = & (\lambda z :_\psi A[u\{\psi\}/x].b[u/x]) \bullet_\psi a[u\{\psi\}/x] \\ & \text{by IH and Lem. 3} \\ \triangleright & b[u'/x][a'[u'\{\psi\}/x]/z] \\ = & b'[u'/x][a'[u'/x]/z] \\ & \text{by Lem. 2} \\ = & b'[a'/z][u'/x] \end{array}$$

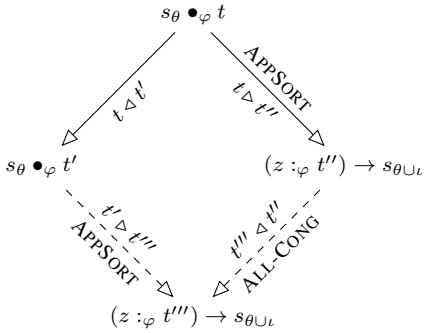
□

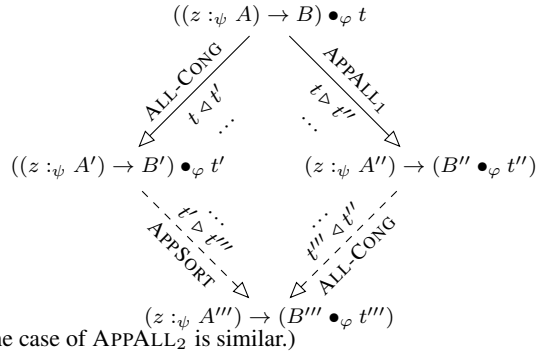
Theorem 2 (Diamond)**.** *The rewriting system ($\triangleright$) has the diamond property. That is, for each A, B, B' such that $B \triangleleft A \triangleright B'$, there exists C such that $B \triangleright C \triangleleft B'$*

Proof. By induction on the derivations.

- If one of the derivations ends with REFL, one has either $A = B$, or $A = B'$. We pick $C = B'$ in the former case and $C = B$ in the latter.
- If one of the derivations ends with APP-CONG, the other one has to end with APP-CONG, β, APPSORT, APPALL$_1$, or with APPALL$_2$. The first case is straightforward. In the other cases, the diverging reductions meet as shown below:







(The case of APPALL$_2$ is similar.)

- If both derivations end with the same rule β, APPSORT, APPALL$_1$, or with APPALL$_2$, the result is a straightforward use of the induction hypothesis (using Lem. 4 in the case of β).
- If one of the derivations ends with ABS-CONG or ALL-CONG, the other one has to end with the same rule, and the result is a straightforward use of the induction hypothesis. □

Corollary 1 (Confluence)**.** *CCCC has the confluence property.*

Proof. A direct consequence of Th. 2 and $\longrightarrow^\star = \triangleright^\star$. □

Lemma 5 (Thinning)**.** *Let Γ and Δ be legal contexts such that $\Gamma \subseteq \Delta$. Then $\Gamma \vdash A :_\theta B \Longrightarrow \Delta \vdash A :_\theta B$.*

Proof. As in (Barendregt 1992, lem. 5.2.12). □

The generation lemma for PTS can be extended to colored bindings. The only difficulty is the following. In (Barendregt 1992), the generation lemma includes a case for applications. This case is difficult to extend to our calculus, since application can be done not only on lambda abstractions, but also on types, when they are used as predicates. Fortunately, that case is not used in the subject reduction lemma, and therefore we can omit it from our version of the generation lemma.

Lemma 6 (Generation)**.** *The statement is similar to that of (Barendregt 1992, lem. 5.2.13). Points 1. to 4. (constant, variable, product, abstraction) are adapted in a straightforward manner to colored binding. Point 5. (application) is removed. The following two points are added.*

- *If $\Gamma \vdash (s_{\theta,i} \bullet_{\theta,\lightning} t) :_{\theta,i} C$, then*
$$\Gamma \vdash t :_{\theta,\lightning} s_\theta \text{ and } C \equiv_\beta s'_\theta \text{ with } (s, s') \in \mathcal{A}$$
- *If $\Gamma \vdash ((x :_\varphi A) \to B) \bullet_{\theta,\lightning} t :_{\theta,i} C$, then*
$$\Gamma \vdash t :_{\theta,\lightning} \lfloor (x :_\varphi A) \to B \rfloor_i \text{ and } C \equiv_\beta s'_\theta \text{ with } \Gamma \vdash (x :_\varphi A) \to B :_\theta s_\theta$$

Proof. As in (Barendregt 1992): we follow the derivations until $s_{\theta,i}$ (resp. $(x :_\varphi A) \to B$) is introduced. It can only be done by the PARAM rule, and the conclusion follows from a use of the Thinning Lemma. (An example of such derivations can be found in Fig. 1.) □

Theorem 3 (Subject reduction)**.** *If $A \longrightarrow A'$ and $\Gamma \vdash A : T$, then $\Gamma \vdash A' : T$.*

Proof. Most of the technicalities of the proof of subject reduction for PTSs (Barendregt 1992) concern β-reduction, and are not changed by the addition of colors.

Hence we discuss here only the handling of reduction rules (1) to (3). We treat first the case of sorts (1):

$$s_\theta \bullet_\varphi t \longrightarrow (z :_\varphi t) \to s_{\theta \cup \iota} \qquad (1)$$
$$\text{where } \varphi = (\theta, \iota)$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdash \Gamma}{\Gamma \vdash \star : \Box}\,\textsc{Ax}}{\Gamma \vdash \star :_i \Box \bullet_{\dot\star} \star}\,\textsc{Param}}{\Gamma \vdash \star :_i (x :_{\dot\star} \star) \to \Box_i}\,\textsc{Conv} \qquad \Gamma \vdash t :_{\dot\star} \star}{\Gamma \vdash \star \bullet_{\dot\star} t :_i \Box_i}\,\textsc{App} \quad \Longrightarrow \quad \dfrac{\dfrac{\vdash \Gamma \qquad \Gamma \vdash t :_{\dot\star} \star}{\vdash \Gamma, x :_{\dot\star} t}\,\textsc{Bind}}{\Gamma, x :_{\dot\star} t \vdash \star_i :_i \Box_i}\,\textsc{Ax}}{\Gamma \vdash (x :_{\dot\star} t) \to \star_i :_i \Box_i}\,\textsc{Prod}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\Gamma, x : A \vdash B : \star}{\Gamma \vdash (x : A) \to B : \star}\,\textsc{Prod}}{\Gamma \vdash (x : A) \to B :_i \star \bullet_{\dot\star} \lfloor (x : A) \to B \rfloor_i}\,\textsc{Param}}{\Gamma \vdash (x : A) \to B :_i \lfloor (x : A) \to B \rfloor_i \to \star_i}\,\textsc{Conv} \qquad \Gamma \vdash t :_{\dot\star} \lfloor (x : A) \to B \rfloor_i}{\Gamma \vdash ((x : A) \to B) \bullet_{\dot\star} t :_i \star_i}\,\textsc{App} \quad \Longrightarrow$$

$$\dfrac{\dfrac{\Gamma, x : A \vdash B : \star}{\Gamma, x : A \vdash B :_i \lfloor B \rfloor_i \to \star_i}\,\textsc{Param} \qquad \dfrac{\Gamma \vdash t :_{\dot\star} \lfloor (x : A) \to B \rfloor_i}{\Gamma, x : A \vdash t\, x :_{\dot\star} B}\,\textsc{App}}{\Gamma, x : A \vdash B \bullet_{\dot\star} (t\, x) :_i \star_i}\,\textsc{App}}{\Gamma \vdash (x : A) \to B \bullet_{\dot\star} (t\, x) :_i \star_i}\,\textsc{Prod}$$

Figure 1. Proof-reduction templates corresponding to the term-reductions (1) and (3). The sorts and taint annotations are specialized to the simplest case to reduce clutter; generalization to arbitrary taints and sorts is straightforward.

In this case, the Generation Lemma indicates that the derivation tree must end with an APP rule and contain a chain PARAM-AX on the left-hand side of the derivation. A template for such a tree is shown in Fig. 1. One can then construct a typing derivation for the reduct, which ends with the PROD rule, and does not mention PARAM nor APP. The template for typing the reduct is also shown in Fig. 1.

Second we treat the case of products (2) and (3).

$$((x :_\psi A) \to B) \bullet_\varphi t \longrightarrow (x :_\psi A) \to (B \bullet_\varphi t) \qquad (2)$$
$$\text{if } \exists i \text{ such that } i \in \psi \text{ and } \dot\star \in \varphi$$
$$((x :_\psi A) \to B) \bullet_\varphi t \longrightarrow (x :_\psi A) \to (B \bullet_\varphi (t \bullet_\psi x)) \qquad (3)$$
$$\text{otherwise}$$

Again we can use the Generation Lemma to obtain the shape of a derivation tree of the reducible expression, and obtain a valid typing for the reduct (also shown in Fig. 1. In this case the typing of the source involves the chain of rules PARAM-PROD on the left-hand side of APP. The typing of the reduct ends with PROD. □

We have taken the view in our presentation that the reduction rules are untyped, and therefore subject reduction must recover typings using the Generation Lemma. An alternative would be to have a typed reduction relation (this relation usually goes by the name of "judgmental equality"). In this case the reduction of proof trees shown in Fig. 1 would be part of the definition of reduction. The two approaches have been proved equivalent for arbitrary PTSs by Siles (2010).

Theorem 4 (Normalization)**.** *CCCC is strongly normalizing: every sequence of reductions eventually terminates.*

Proof. The new reduction rules, involving PARAM, are much easier to handle than β-reduction (rule 4). Indeed, the argument to the application is not duplicated by these reduction rules.

Hence, one can adapt the proofs of termination of CC to CCCC. At a high-level, the argument goes as follows. Consider reduction rules (1) to (3), that we collectively call PARAM-reductions below. Their effect is to make the left-hand-side of the $\bullet_\psi$ operator smaller. This can also be seen by examining the corresponding typing derivations: the reductions strictly decrease the size of the tree above the PARAM rule involved.

Consequently, between each β-reduction step, there can be only a finite number of PARAM-reductions. The question is now: do these PARAM-reductions create β-redexes? The reductions (1) and (2) do not, but (3) does. However, the new redex is harmless: it lies in an i-oblivious context, and therefore there is no risk of an (i-aware) PARAM-reduction to be created by that redex.

The situation is then that, by importing the proof techniques developed for CC, one can bound the chains of β-reductions, and use the above argument to finitely-bound the PARAM-reductions occurring between β-reduction steps. □

The above proof is an alternative to that done in earlier work (Bernardy and Moulin 2012), which works by construction a model by translation into CC. Even though the earlier proof can be adapted to the present system, we find the above proof more modular, and thus easier to grasp.

3.8 Type-Checking With Colors

The rules PARAM and the definition of and oblivious judgment Def. 1 are not syntax-directed, therefore it may be non-obvious how they should be used in type-checking. In this subsection we sketch a possible type-checking algorithm and briefly argue for its soundness. A full description of the algorithm, together with a completeness proof, is left for future work.

We assume that the user supplies a term t, a type T, a modality φ and a context Γ. The task is to reconstruct a derivation of $\Gamma \vdash t :_\varphi T$.

Most of the implementation of the new rules is realized at the point of checking a variable ($t = x$). Let us assume that looking up x in the context Γ yields $x :_\psi A$.

For each color $i \in \Gamma$ we have to take into account its status in φ and ψ. For simplicity we assume that φ and ψ contain at most one color; a full implementation will do the same task for each color independently.

1. $\dot\star \in \varphi$. If i is not mentioned in ψ, then the derivation is the following.

$$\dfrac{\dfrac{\dfrac{x : A \in \Gamma}{\Gamma \vdash x : A}\,\textsc{Var}}{\lfloor \Gamma \rfloor_i \vdash x : \lfloor A \rfloor_i}\,\text{Lem. 1}}{\Gamma \vdash x :_{\dot\star} \lfloor A \rfloor_i}\,\text{Def. 1}$$

Hence we simply check that $T = \lfloor A \rfloor_i$. For $i \in \psi$, then x is not accessible and type-checking reports failure. For $\dot{\imath} \in \psi$ one needs only to check $T = A$.

2. If the judgment is i-aware ($\varphi = \varnothing$), we have three cases. If $i \in \psi$, then x is not accessible and type-checking reports failure. Otherwise, then modalities match and we only need to check $T = A$. Notably, if $\dot{\imath} \in \psi$, x is more oblivious than required, but this is accepted by the START rule, which allows arbitrary extra anti-taints in the binding of the looked-up variable.
3. If $i \in \varphi$, we have two cases. If $i \in \psi$, then one simply checks $T = A$. Otherwise, one must take into account PARAM, that is, check $T = A \bullet_{\dot{\imath}} x$ instead of $T = A$.

Lastly, when checking a type used as a predicate, that is a term t of the form $(\Delta \to B) \bullet_\psi u$ (for any telescope Δ) or $s \bullet_\psi u$, we reduce the term t before checking it. The number of reduction steps is at most $|\Delta| + 1$ in the first case or 1 in the second case: the performance hit is minimal.

As usual in type-checking algorithms, all the equality-tests mentioned above have to be performed up-to the reduction relation, in order to take into account the CONV rule. This is done using standard means, for example normalizing terms before comparison, which is possible thanks to Th. 4.

4. Extensions

4.1 Inductive Definitions

The definitions presented in the previous section can be extended to work on inductive types in a straightforward manner, merely recursively applying the definitions on the types of each component of the inductive type, as we have done in the examples in Sec. 2. The extension of a similar system to inductive definitions is described in full detail by Bernardy et al. (2012). We conjecture that the addition of inductive definitions does not compromise any meta-theoretical property.

4.2 Colored Pairs

In this section we formally introduce colored pairs, and formalize a example presented in Sec. 2: inhabitants of the type $(a : \star) \to a \to a$ must be the identity function.

The formation and introduction rules (SUM, PAIR) are similar to the usual rules for dependent pairs, with the difference that the modalities track that the first component is oblivious to the color while the second component is tainted.

$$\textsc{Sum}\ \frac{\Gamma, x :_{\dot{\imath}} A \vdash B :_i s}{\Gamma \vdash (x : A) \times_i B : s} \qquad \textsc{Pair}\ \frac{\Gamma \vdash a :_{\dot{\imath}} A \qquad \Gamma \vdash b :_i B[a/x]}{\Gamma \vdash a,_i b : (x : A) \times_i B}$$

We do not provide special elimination rules for colored pairs. Instead, erasure and PARAM play this role. The erasure of a pair yields its first component if colors match, otherwise it acts structurally.

$$\begin{aligned} \lfloor (x : A) \times_i B \rfloor_i &= A \\ \lfloor (x : A) \times_j B \rfloor_i &= (x : \lfloor A \rfloor_i) \times_j \lfloor B \rfloor_i \\ \lfloor a,_i b \rfloor_i &= a \\ \lfloor a,_j b \rfloor_i &= \lfloor a \rfloor_i,_j \lfloor b \rfloor_i \end{aligned}$$

We remark that the property that erasure preserves typing is conserved. In particular:

$$\text{if } \Gamma \vdash p : (x : A) \times_i B \text{ then } \lfloor \Gamma \rfloor_i \vdash \lfloor p \rfloor_i : A$$

Interpreting a pair as a predicate yields the second component if colors match, otherwise it acts structurally:

$$\begin{aligned} ((x : A) \times_i B) \bullet_{\psi, \dot{\imath}} t &= B[t/x] \\ ((x : A) \times_j B) \bullet_\psi t &= (x : A \bullet_\psi \lfloor t \rfloor_j) \times_j (B \bullet_\psi t) \end{aligned}$$

We can now explain fully formally how one can derive that any function of type $(a : \star) \to a \to a$ is indeed an identity function. Let

$$\begin{aligned} \Gamma &\triangleq f : (a : \star) \to a \to a, \quad b : \star, \quad y : b \\ x \equiv_a y &\triangleq (P : a \to \star) \to P\, x \to P\, y \\ t &\triangleq f\, ((x : b) \times_i (x \equiv_b y))\, (y,_i \mathsf{refl}^i) \end{aligned}$$

Where $\equiv$ refers to the definition of the previous section. We first check that $(x : b) \times_i (x \equiv_b y)$ is well-colored (and well-sorted):

$$\textsc{Sum}\ \frac{\textsc{Erasure Def.}\ \dfrac{\textsc{Def. 1}\ \dfrac{\Gamma \vdash b : \star}{\lfloor \Gamma, i \rfloor_i \vdash b : \star}}{\Gamma, i \vdash b :_{\dot{\imath}} \star} \qquad \dfrac{\Gamma, i, x :_{\dot{\imath}} b \vdash x :_{\dot{\imath}} b}{\Gamma, i, x :_{\dot{\imath}} b \vdash (x \equiv_b y) :_i \star_i}}{\Gamma, i \vdash (x : b) \times_i (x \equiv_b y) : \star}$$

And proceed with the main result:

$$\textsc{App}\ \frac{\vdots \quad \dfrac{\Gamma \vdash y : b \qquad \Gamma, i \vdash \mathsf{refl}^i :_i (y \equiv_b y)}{\Gamma, i \vdash (y,_i \mathsf{refl}^i) : (x : b) \times_i (x \equiv_b y)}\ \textsc{Pair}}{\textsc{Def.}\ \dfrac{\Gamma, i \vdash f\, ((x : b) \times_i (x \equiv_b y))\, (y,_i \mathsf{refl}^i) : ((x : b) \times_i (x \equiv_b y))}{\textsc{Param}\ \dfrac{\Gamma, i \vdash t : (x : b) \times_i (x \equiv_b y)}{\textsc{Erasure Def.}\ \dfrac{\Gamma, i \vdash t :_i (\lfloor t \rfloor_i \equiv_b y)}{\Gamma, i \vdash t :_i f\, b\, y \equiv_b y}}}}$$

An essential component of the trick is that f is i-oblivious. Hence it cannot distinguish in any way between i-tainted and non i-tainted terms, thus we can pass it a colored pair as its type argument.

The trick generalizes: one is able to derive useful properties about a polymorphic term q of type A by construction of an adequate term p of type $(x : A) \times_i B$ such that $\lfloor p \rfloor_i = q$. The construction of q involves specializing a type parameter to a colored pair type involving the property of interest.

In fact, by pairing a type A with a predicate $B[x]$, colored pair type allows to override the default predicate interpretation of the type A with $B[x]$, for a given specific color.

4.3 Abstraction Over Colors

So far we have assumed that colors were available in the context. In a complete TTC, a mechanism to abstract over colors should be provided. At this stage we have thought of colors only as a first-order concept, that is, we propose only first-order quantification over colors.

As such, color abstraction is a relatively modest, straightforward addition. We need new syntax for abstraction, quantification, and application, as well as a constant color (written 0 in the following) used for erasure. A notable feature is that one cannot abstract over a color which is present in the modality; otherwise the color would escape the scope where it is introduced. Using k to range over color names or 0, the typing rules are as follows:

$$\textsc{Col. Abs}\ \frac{\Gamma, i \vdash A :_\theta B \qquad i \notin \theta}{\Gamma \vdash \lambda i.A :_\theta \forall i.B} \qquad \textsc{Col. App}\ \frac{\Gamma \vdash A :_\theta \forall i.B}{\Gamma \vdash A\, k :_\theta B[k/i]}$$

and the reduction rules:

$$(\forall i.T)\ \bullet_\psi\ t \longrightarrow \forall i.T\ \bullet_\psi\ (t\ i)$$
$$(\lambda i.t)\ 0 \longrightarrow \lfloor t \rfloor_i$$
$$(\lambda i.t)\ j \longrightarrow t[j/i]$$

Erasure must be extended as well to substitute occurrences of colors as arguments:

$$\lfloor t\ i \rfloor_i = \lfloor t \rfloor_i\ 0$$
$$\lfloor t\ i \rfloor_j = \lfloor t \rfloor_j\ i$$
$$\lfloor t\ 0 \rfloor_i = \lfloor t \rfloor_i\ 0$$
$$\lfloor \lambda i.\ t \rfloor_j = \lambda i.\lfloor t \rfloor_j$$

With this extension, the subject reduction property depends on Lem. 1.

5. Discussion and Related Work

COQ***-style erasure*** Thanks to Paulin-Mohring (1989), COQ features *program extraction*, which is an erasure of proofs to obtain programs. Such programs are external entities, that is, they cannot be referred to as an erasure from the COQ script they originate. This shortcoming is remedied in TTC. For the purpose of extraction, COQ separates types from propositions, using different sorts for each, and allows types to depend on the existence of proofs (but not their structure). In contrast, in the system we present here, untainted terms cannot even depend on the existence of a tainted term. It is likely that the two notions of erasure can be combined in a single system, but we leave the study of that combination to future work.

AGDA***-style erasure*** A number of systems with modalities for erasure have been proposed (Pfenning 2001; Mishra-Linger and Sheard 2008; Abel and Scherer 2012), with the interpretation of *irrelevance*: types marked with a special modality (usually written $x \div A$) are understood as proofs, whose inhabitants are irrelevant for the execution of the programs.

The system presented here bears some similarity to such systems, but also presents important differences:

- Our binding form $x :_i A$ corresponds to the irrelevant binding $x \div A$.
- We have, in addition to irrelevant bindings, the complementary notion (written $x :_{\not\bullet} A$). This allows us to mix erased terms with non-erased ones, and choose arbitrarily which version we mean. In contrast, systems with erasure usually fix a specific view on which parts of a term is accessible.
- We support an arbitrary number of colors instead of a single one, which is essential for compositionality and to support n-ary parametricity.
- We focus on erasure instead of irrelevance. Previous authors usually allow the use of the *ex falso quod libet* principle on irrelevant assumptions, while we forbid any use of a tainted variable in a non-tainted context.

Types for language-based security Our notion of taint is reminiscent of that used in language-based security. More precisely, our tainted variables would correspond to variables at high security levels: tainted variables may be used only in tainted contexts. The present work can be seen as a generalization of (Abadi et al. 1999) to dependent types. A difference is that we use modalities instead of a different type former for security levels. As a consequence, we do not need a monad to relate security levels. To our knowledge, the combination of dependent types and security-levels in a type-system has not been realized before.

Ornaments Relating variants of dependently-typed programs have been a concern for a long time. The idea of ornamenting inductive structures have been proposed to remedy this problem (Dagand and McBride 2012). Here, we instead focus on erasure (let the user specify an ornamented type and recover the relation with its erasure) instead of specifying ornamentation of already existing types. This is not much of a difference in practice, because ornaments typically can only be applied to a single type. An advantage of colors over ornaments is that colors integrate natively with existing type-theories, while ornamentation relies on encodings; additionally we get free equalities when working with erasures, and colored pairs reveal parametricity properties. The chief advantage of ornaments is that any algebra can be used to ornament datatypes, while we are limited to structural relations.

Ko and Gibbons (2013) have shown how to compose ornaments. Composition is lacking from the system presented here, but we plan to support it in future extensions of TTC. Indeed, if two terms A and B share an erasure (for example $C = \lfloor A \rfloor_i = \lfloor B \rfloor_j$), it means that A and B are to versions of C ornamented differently. Under the same assumption, it is possible to automatically construct a term D such that $\lfloor D \rfloor_j = A$, $\lfloor D \rfloor_i = B$, and $\lfloor D \rfloor_{ij} = C$. That is, D contains both the ornamentations coming from A and B.

Parametricity Bernardy and Moulin (2012) have described a calculus which internalizes parametricity, and have shown that higher dimensions are necessary to nest parametricity. The present work has the same model as the previous one (colors are dimensions under another name). Besides re-framing dimensions as colors, which we find allows an easier grasp of intuitions, the present system features a number of technical simplifications:

1. In the system of Bernardy and Moulin (2012), one has to be explicit about the number of dimensions that a term has. In contrast, here, the dimensionality of a term is implicit. Indeed, contexts can be extended with an arbitrary number of dimensions. This means that a term can always be used in a context which has more (distinct) colors, whereas previously an explicit conversion had to take place. In other words, terms can be seen as infinitely-dimensional; but if they do not mention a dimension explicitly they behave uniformly over it. In particular, usual λ-terms are uniform, and parametricity is a consequence of this uniformity.
2. In this paper, we name dimensions, whereas they are numbered in (Bernardy and Moulin 2012). The situation is analogous to the issue of the representation of variables in lambda-calculi. One can either use explicit names or De Brujin indices, and using names is usually more convenient.

It is worth underlining that the notion of erasure we employ here is not the same as that of (Bernardy and Lasson 2011). Here, we erase the colored component; whereas Bernardy and Lasson (2011) erase the oblivious components.

Parametricity has more "standard consequences" such as the deduction of induction principles. Unfortunately, many of these consequence also require extensionality (Wadler (2007) gives a complete development). Since extensionality is not well integrated to type-theory (let alone TTC) at the moment, we cannot derive such constructions, at least not without postulating extensionality.

Higher-dimensional Equality Recent work on the interpretation of the equality-type in intensional type theory suggests that it should be modeled using higher-dimensional structures (Licata and Harper 2012). In the present work we support the definition of higher-dimensional structures via dependencies on colors. In future work we wish to investigate whether the presence of colors can help encoding the higher-dimensional structure of equality.

A potential difficulty is that the structure of equality is simplectic, while we have here a cubic structure (colors are orthogonal). A simplex is however easily embedded in a cube, so there are grounds to believe that the two aspects can eventually be integrated.

6. Conlusion and Future Work

We have described an extension of type-theory with colored terms, a notion of color erasure, and a way to interpret colored types as predicates. We have shown how that extension provides a new kind of genericity, and how the coloring discipline enforces invariants when writing programs. We also shown how to reveal these invariants (by typing the judgment in another modality) and internalize some parametricity results.

We detailed extensively a core version of that colored type-theory, namely CCCC. In particular, we proved fundamental properties for that system, such as Church-Rosser's, subject-reduction, and strong normalization. We also implemented some features of CCCC in a prototype we aim to merge into the main stream AGDA proof assistant.

Future work chiefly involves unifying colors as presented here with previous similar notions. On the implementation side, we aim to complete our prototype with all the features presented in this paper. Besides, we would like to investigate the feasibility of inference of color annotations before merging our implementation of TTC into the main branch of AGDA. We will then be able to assess the practical power of TTC. A first step in this direction is the retrofitting of the AGDA standard library to use colors.

Acknowledgments

Many ideas underlying this paper have germinated and matured in discussions with Thierry Coquand, Simon Huber and Patrik Jansson. We thank Peter Dybjer, Cezar Ionescu, Nicolas Pouillard and Philip Wadler as well as anonymous reviewers for useful feedback.

References

M. Abadi, A. Banerjee, N. Heintze, and J. Riecke. A core calculus of dependency. In *POPL'99*, pages 147–160. ACM, 1999.

A. Abel and G. Scherer. On irrelevance and algorithmic equality in predicative type theory. *Logical Methods in Comp. Sci.*, 8(1):1–36, 2012. TYPES'10 special issue.

H. P. Barendregt. Lambda calculi with types. *Handbook of logic in computer science*, 2:117–309, 1992.

J.-P. Bernardy and M. Lasson. Realizability and parametricity in pure type systems. In M. Hofmann, editor, *FoSSaCS*, volume 6604 of *LNCS*, pages 108–122. Springer, 2011.

J.-P. Bernardy and G. Moulin. A computational interpretation of parametricity. In *Proc. of the Symposium on Logic in Comp. Sci.* IEEE, 2012.

J.-P. Bernardy, P. Jansson, and R. Paterson. Parametricity and dependent types. In *Proc. of ICFP 2010*, pages 345–356. ACM, 2010.

J.-P. Bernardy, P. Jansson, and R. Paterson. Proofs for free — parametricity for dependent types. *J. Funct. Program.*, 22(02):107–152, 2012.

T. Coquand and G. Huet. The calculus of constructions. Technical report, INRIA, 1986.

P.-E. Dagand and C. McBride. Transporting functions across ornaments. In *Proc. of ICFP 2012*, ICFP '12. ACM, 2012.

D. Licata and R. Harper. Canonicity for 2-dimensional type theory. In *Proc. of POPL 2012*. ACM, 2012.

P. Martin-Löf. *Intuitionistic type theory*. Bibliopolis, 1984.

N. Mishra-Linger and T. Sheard. Erasure and polymorphism in pure type systems. In *FoSSaCS 2008*, pages 350–364. Springer-Verlag, 2008.

U. Norell. *Towards a practical programming language based on dependent type theory*. PhD thesis, Chalmers Tekniska Högskola, 2007.

C. Paulin-Mohring. Extracting Fω's programs from proofs in the calculus of constructions. In *POPL'89*, pages 89–104. ACM, 1989.

F. Pfenning. Intensionality, extensionality, and proof irrelevance in modal type theory. In *Proc. 16th Annual IEEE Symposium on Logic in Comp. Sci.*, pages 221–230. IEEE, 2001.

V. Siles. *Investigation on the typing of equality in type systems*. Phd thesis, École Polytechnique, 2010.

The Coq development team. The Coq proof assistant, 2012.

P. Wadler. Call-by-value is dual to call-by-name. In *Proc. of ICFP 2003*, ICFP '03, pages 189–201. ACM, 2003.

P. Wadler. The Girard–Reynolds isomorphism (second edition). *Theor. Comp. Sci.*, 375(1–3):201–226, 2007.

P. Wadler. Propositions as sessions. In *Proc. of ICFP 2012*, ICFP '12, pages 273–286. ACM, 2012.

Appendix: Definition of CCCC

Definition 3 (Syntax).

Variable	$\ni x, y, z$			
Color	$\ni i, j$			
Sort	$\ni s$	$::=$	$\star_\theta \mid \Box_\theta$	
Taint	$\ni \theta, \iota$	$::=$	$\varnothing$	*empty*
		$\mid$	θ, i	*tainted*
Modality	$\ni \psi, \varphi$	$::=$	(θ, ι)	
Term	$\ni A, \ldots, Z$	$::=$	x	*variable*
	a, b, c, t, u	$\mid$	s	*sort*
		$\mid$	$(x :_\psi A) \to B$	*product*
		$\mid$	$\lambda x :_\psi A.b$	*abstraction*
		$\mid$	$F \bullet_\psi a$	*application*
Context	$\ni \Gamma, \Delta$	$::=$	$-$	*empty*
		$\mid$	$\Gamma, x :_\psi A$	*binding*
		$\mid$	Γ, i	*color*

Definition 4 (Typing rules $\boxed{\Gamma \vdash A :_\theta B}$).
$\Gamma \vdash A :_\theta B$ *is well-formed only if* $i \in \Gamma$ *for each* $i \in \theta$.

$$\textsc{Conv}\ \frac{\Gamma \vdash a :_\theta A \qquad A =_\beta A'}{\Gamma \vdash a :_\theta A'} \qquad \textsc{Axiom}\ \frac{\vdash \Gamma}{\Gamma \vdash \star_\theta :_\theta \Box_\theta}$$

$$\textsc{Var}\ \frac{\vdash \Gamma \qquad x :_\theta A \in \Gamma}{\Gamma \vdash x :_\theta A} \qquad \textsc{Prod}\ \frac{\Gamma, x :_\psi A \vdash B :_\theta s}{\Gamma \vdash (x :_\psi A) \to B :_\theta s}$$

$$\textsc{Abs}\ \frac{\Gamma, x :_\psi A \vdash b :_\theta B}{\Gamma \vdash (\lambda x :_\psi A.b) :_\theta (x :_\psi A) \to B}$$

$$\textsc{App}\ \frac{\Gamma \vdash F :_\theta (x :_\psi A) \to B \qquad \Gamma \vdash u :_\psi A}{\Gamma \vdash F \bullet_\psi u :_\theta B[u/x]}$$

$$\textsc{Param}\ \frac{\Gamma \vdash A :_\theta B \qquad i \notin \theta}{\Gamma \vdash A :_{\theta,i} B \bullet_{\theta,\dot{\imath}} \lfloor A \rfloor_i}$$

(To limit clutter we omit the well-sorted conditions of types A and B in the rule Abs*.) We also have*

$$\textit{If } \psi = (\theta, \iota) \textit{ then } \Gamma \vdash A :_\psi B \triangleq \lfloor \Gamma \rfloor_\iota \vdash A :_\theta B$$

Definition 5 (Accessible variable $\boxed{x :_\theta A \in \Gamma}$).

$$\textsc{Start}\ \frac{}{x :_\theta A \in \Gamma, x :_{(\theta,\iota)} A} \qquad \textsc{Col. Wk}\ \frac{x :_\theta A \in \Gamma \qquad i \notin \theta}{x :_\theta A \in \Gamma, i}$$

$$\textsc{Wk}\ \frac{x :_\theta A \in \Gamma}{x :_\theta A \in \Gamma, y :_\psi B}$$

Definition 6 (Well-formed contexts $\boxed{\vdash \Gamma}$).

$$\textsc{Empty}\ \frac{}{\vdash -} \qquad \textsc{Color}\ \frac{\vdash \Gamma}{\vdash \Gamma, i} \qquad \textsc{Bind}\ \frac{\vdash \Gamma \qquad \Gamma \vdash A :_\psi s}{\vdash \Gamma, x :_\psi A}$$

Definition 7 (erasure $\boxed{\lfloor T \rfloor_i}$). *The definition of erasure depends on the actual modality used. We write all the cases on the same line; the condition is written above each column.*

	$i \notin \psi$	$i \in \psi$	$\psi = \varphi, \dot{\imath}$
$\lfloor x \rfloor_i = x$			
$\lfloor s \rfloor_i = s$			
$\lfloor (x :_\psi A) \to B \rfloor_i = (x :_\psi \lfloor A \rfloor_i) \to \lfloor B \rfloor_i$	$\lfloor B \rfloor_i$	$(x :_\varphi A) \to \lfloor B \rfloor_i$	
$\lfloor \lambda x :_\psi A.b \rfloor_i = \lambda x :_\psi \lfloor A \rfloor_i . \lfloor b \rfloor_i$	$\lfloor b \rfloor_i$	$\lambda x :_\varphi A . \lfloor b \rfloor_i$	
$\lfloor F \bullet_\psi a \rfloor_i = (\lfloor F \rfloor_i) \bullet_\psi \lfloor a \rfloor_i$	$\lfloor F \rfloor_i$	$(\lfloor F \rfloor_i) \bullet_\varphi a$	
$\lfloor \Gamma, x :_\psi A \rfloor_i = \lfloor \Gamma \rfloor_i, x :_\psi \lfloor A \rfloor_i$	$\lfloor \Gamma \rfloor_i$	$\lfloor \Gamma \rfloor_i, x :_\varphi A$	
$\lfloor \Gamma, j \rfloor_i = \lfloor \Gamma \rfloor_i, j$			
$\lfloor \Gamma, i \rfloor_i = \Gamma$			

Erasure is extended to taints as follows:

Definition 8 (erasure $\boxed{\lfloor T \rfloor_\iota}$).

$$\lfloor T \rfloor_\varnothing = T$$
$$\lfloor T \rfloor_{\iota,i} = \lfloor \lfloor T \rfloor_i \rfloor_\iota$$

Definition 9 (Reduction $\boxed{t \longrightarrow u}$).

$$s_\theta \bullet_\varphi t \longrightarrow (z :_\varphi t) \to s_{\theta \cup \iota} \qquad (1)$$
where $\varphi = (\theta, \iota)$

$$((x :_\psi A) \to B) \bullet_\varphi t \longrightarrow (x :_\psi A) \to (B \bullet_\varphi t) \qquad (2)$$
if $\exists i$ *such that* $i \in \psi$ *and* $\dot{\imath} \in \varphi$

$$((x :_\psi A) \to B) \bullet_\varphi t \longrightarrow (x :_\psi A) \to (B \bullet_\varphi (t \bullet_\psi x)) \qquad (3)$$
otherwise

$$(\lambda x :_\psi A.b) \bullet_\psi t \longrightarrow b[t/x] \qquad (4)$$

and congruences.

Definition 10 (Substitution). *The substitution of variables in i-oblivious contexts erases i from the substitutees.*

$$(F \bullet_\psi a)[u/x] = F[u/x] \bullet_\psi a[u\{\psi\}/x]$$
$$((y :_\psi A) \to B)[u/x] = (y :_\psi A[u\{\psi\}/x]) \to B[u/x]$$
$$(\lambda y :_\psi A.b)[u/x] = \lambda y :_\psi A[u\{\psi\}/x].b[u/x]$$

Where

$$u\{(\theta, \iota)\} = \lfloor u \rfloor_\iota$$