Parametricity for Haskell with Imprecise Error Semantics

Florian Stenger[*] Janis Voigtländer

Institut für Theoretische Informatik
Technische Universität Dresden
01062 Dresden, Germany

{stenger,voigt}@tcs.inf.tu-dresden.de

Abstract

Types play an important role both in reasoning about Haskell and for its implementation. For example, the Glasgow Haskell Compiler performs certain fusion transformations that are intended to improve program efficiency and whose semantic justification is derived from polymorphic function types. At the same time, GHC adopts a scheme of error raising, propagation, and handling which is nondeterministic in the sense that there is some freedom as to which of a number of potential failure events hidden somewhere in a program is actually triggered. Implemented for good pragmatic reasons, this scheme complicates the meaning of programs and thus necessitates extra care when reasoning about them. In particular, since every erroneous value now represents a whole set of potential (but not arbitrary) failure causes, and since the associated propagation rules are askew to standard notions of program flow and value dependence, some standard laws suddenly fail to hold. This includes laws derived from polymorphic types, popularized as free theorems and at the base of the mentioned kind of fusion. We study this interaction between type-based reasoning and imprecise errors by revising and extending the foundational notion of relational parametricity, as well as further material required to make it applicable. More generally, we believe that our development and proofs help direct the way for incorporating further and other extensions and semantic features that deviate from the "naive" setting in which reasoning about Haskell programs often takes place.

[*]This author was supported by the DFG under grant VO 1512/1-1.

1 Introduction

Functional languages come with a rich set of conceptual tools for reasoning about programs. For example, structural induction and equational reasoning tell us that the standard Haskell functions

$$\begin{array}{llll} \mathit{takeWhile} :: (\alpha \to \mathsf{Bool}) \to [\alpha] \to [\alpha] \\ \mathit{takeWhile}\ p\ [\,] & = [\,] \\ \mathit{takeWhile}\ p\ (x : y) & \mid\ p\ x & = x : \mathit{takeWhile}\ p\ y \\ & \mid\ \mathit{otherwise} & = [\,] \end{array}$$

and

$$\begin{array}{ll} \mathit{map} :: (\alpha \to \beta) \to [\alpha] \to [\beta] \\ \mathit{map}\ h\ [\,] & = [\,] \\ \mathit{map}\ h\ (x : y) & = h\ x : \mathit{map}\ h\ y \end{array}$$

satisfy the following law for appropriately typed p, h, and l:

$$\mathit{takeWhile}\ p\ (\mathit{map}\ h\ l) = \mathit{map}\ h\ (\mathit{takeWhile}\ (p \circ h)\ l)\,, \tag{1}$$

where $\circ$ is function composition.

But programming language reality can be a tough game, leading to unexpected failures of such near-obvious laws. For example, Peyton Jones et al. (1999) propose a design for error handling based on a certain degree of impreciseness. The major implementations GHC and Hugs (as well as one distribution of the language Clean) have integrated this design years ago. However, the attendant semantics betrays law (1) to be wrong. An instantiation showing this is $p = \mathit{null}$, $h = \mathit{tail}$, and $l = [[i] \mid i \leftarrow [1..(\mathit{div}\ 1\ 0)]]$ (or any other immediately failing expression of type list-of-lists), where

$$\begin{array}{llll} \mathit{null} :: [\alpha] \to \mathsf{Bool} & & \mathit{tail} :: [\alpha] \to [\alpha] \\ \mathit{null}\ [\,] & = \mathsf{True} & \mathit{tail}\ [\,] & = \mathit{error}\ \text{“tail: empty list”} \\ \mathit{null}\ (x : y) & = \mathsf{False} & \mathit{tail}\ (x : y) & = y \end{array}$$

are standard Haskell functions as well. The problem with (1) now is that its left-hand side yields exactly the “divide by zero”-error coming from l, whereas its right-hand side may also yield the “tail: empty list”-error. This is so due to the semantics of pattern-matching in the design of Peyton Jones et al. (1999) (and also Moran et al. (1999)). In short, it prescribes that when pattern-matching on an erroneous value as scrutinee, not only are any errors associated with it propagated, but also are the branches of the pattern-match investigated in “error-finding mode” to detect any errors that may arise there independently of the scrutinee. This is done in order to give the language implementation more freedom in arranging computations, thus allowing more transformations on the code prior to execution. But here it means that when $\mathit{takeWhile}\ (\mathit{null} \circ \mathit{tail})$ encounters an erroneous value, also $(\mathit{null} \circ \mathit{tail})\ x$ is evaluated, with x bound to a special value $\mathit{Bad}\ \emptyset$ that exists only to trigger the error-finding mode. And indeed, the application of tail on that x raises the “tail:

empty list"-error, which is propagated by *null* and then unioned with the "divide by zero"-error from l. In contrast, *takeWhile null* on an erroneous value does not add any further errors, because the definition of *null* raises none. And, on both sides of (1), *map h* only ever propagates, but never introduces errors.

Thus, if we do not want to take the risk of introducing previously nonexistent errors, we cannot use (1) as a transformation from left to right, even though this might have been beneficial (by bringing p and h together for further analysis or for subsequent transformations potentially improving efficiency). The supposed semantic equivalence simply does not hold. Note that this does not necessarily mean that for a given language implementation we will always, or ever, see different errors on the left- and right-hand sides of (1). After all, for the example instantiation above the semantics prescribes that the right-hand side may yield either of the two errors in question, so for a given interpreter or compiler it might very well happen that always the same as on the left-hand side appears. But that would be pure coincidence on which we cannot rely. If all the guarantee we have is that the language implementation builds on the semantics of Peyton Jones et al. (1999), then we have to accept that the arbitration between the two potential errors may in principle vary with set of flags, time of day, phase of the moon, and so on. Impreciseness in the semantics has its price, and if we are not ready to abandon the overall design (which would be tantamount to taking away considerable freedom from language implementers), then we better learn how to cope with it when reasoning about programs.

The above discussion regarding a concrete instantiation of p, h, and l gives negative information only, namely that (1) may break down in some cases. It does not provide any positive information about conditions on p, h, and l under which (1) actually *is* a semantic equivalence. Moreover, it is relative to the particular definition of *takeWhile* given at the very beginning, whereas laws like (1) are often derived more generally as *free theorems* (Reynolds 1983; Wadler 1989) from types alone, without considering concrete definitions. In this paper we undertake to develop the theory of free theorems for Haskell with imprecise error semantics. This continues earlier work by Johann and Voigtländer (2004) for Haskell with all potential error causes (including nontermination) conflated into a single erroneous value $\perp$. That earlier work gives that in this setting (1) is a semantic equivalence provided $p \neq \perp$ and h is strict and total in the sense that $h\ \perp = \perp$ and for every $x \neq \perp$, $h\ x \neq \perp$. The task before us involves finding the right generalizations of such conditions for a setting in which not all errors are equal. Questions like the following ones arise:

- From which erroneous values should p be different?
- For strictness, is it enough that h preserves the least element $\perp$, which in the design of Peyton Jones et al. (1999) denotes the union of all error causes, including nontermination?
- Or do we need that also every other erroneous value (denoting a collection of only some potential error causes, maybe just a singleton set) is mapped to an erroneous one? To the same one? Or to $\perp$?

- For totality, is it enough that "non-$\bot$ goes to non-$\bot$"?
- Does this allow "non-$\bot$ goes to non-$\bot$ but erroneous"?
- Or do we need "nonerroneous goes to nonerroneous"?

It is not to be expected that the answers are trivial to come by. The two settings are simply too different. This is also evidenced by the failure of an inequational variant of (1) which is given by the results of Johann and Voigtländer (2004) and in which strictness of h suffices to guarantee that the right-hand side is at least as defined as the left-hand side, without further conditions on p or h. Most Haskell programmers, even when aware of the imprecise error semantics, would probably agree that *tail* is a strict function. The immediate pattern-match makes it so. And yet, we saw above an instantiation with $h = \mathit{tail}$ in which the right-hand side is less defined (due to more potential errors) than the left-hand side. Finally, it is worth pointing out that the failures of (1) and its inequational variant occur for a very innocently-looking definition of *takeWhile* here. Note that it does not, by itself, introduce any errors or nontermination, nor does it use selective strictness via Haskell's *seq*-primitive. Thus, the features that made life hard[1] in the setting of Johann and Voigtländer (2004) are actually absent, and still the laws break down. In fact, were it not for the imprecise error semantics, (1) would hold for the given definition of *takeWhile* as a semantic equivalence for arbitrary p, h, and l, even ones involving $\bot$ and *seq* in arbitrary ways and without strictness or totality conditions. By contraposition, this indicates that distinguishing different failure causes and treating them the imprecision-through-sets-of-errors way poses genuinely new challenges.

Fortunately, we are not left groping in the dark. Our investigation can be very much goal-directed by studying proof cases of the *(relational) parametricity theorem*, which is the foundation for all free theorems, and trying to adapt the proof to the imprecise error setting. This leads us to discover, among other formal details and ingredients, the appropriate generalized conditions sought above (first as restrictions on relations, then specialized to the function level). In fact, we think that beside the results we provide for the imprecise error setting, this paper can also serve as a guide on how to go about extending relational parametricity to new language features and semantic designs in general. We establish both equational and inequational parametricity theorems, including one for the refinement order of Moran et al. (1999). And while we do not deal with error recovery through exception handling in the IO monad, we will make some initial steps into the realm of exceptions as first class citizens by integrating a primitive (Haskell's *mapException*) that allows manipulating already raised errors (respectively, their descriptive arguments) from inside the language.

[1]Indeed, Johann and Voigtländer (2004) had to add rather ad-hoc occurrences of *seq* to a definition of the *filter*-function (of same type as *takeWhile*) in order to "provoke" failures of the corresponding standard free theorem. Here, instead, even the most natural specification of *takeWhile* leads to problems.

Also note that even though we do not allow the full power of exception handling in the functions *for which* we derive free theorems, our results are nevertheless immediately relevant as well in the larger context of programs that *do* error recovery in the IO monad. Just imagine alternatively the left- or right-hand side of the offending instantiation of (1) in the place of the "$\cdots$" in the following code snippet:

$$\begin{array}{l}\mathsf{Control.Exception}.\mathit{catch}\ (\mathit{evaluate}\ \cdots)\ (\lambda s \to \mathit{if}\ \ s \neq \mathsf{ErrorCall}\ \text{“tail: empty list”}\\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \mathit{then}\ \mathit{return}\ [[42]]\\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \mathit{else}\ \ \mathit{return}\ [\,])\end{array}$$

Then depending on whether the left- or right-hand side of (1) is put there, we might observe different nonerroneous program outcomes. This is even more severe than "just" a confusion between different erroneous values.

With the results from this paper both kinds of problems are settled. For example, we will derive (cf. Example 4.9) that (1) is a true semantic equivalence provided p and h are nonerroneous, h acts as identity on erroneous values, and h never maps a nonerroneous value to an erroneous one. Slightly weaker conditions suffice for an inequational variant (cf. Example 5.10). And similar fixes exist for all the other free theorems we love and treasure.

2 Standard Parametricity

We start out from a standard denotational semantics for a polymorphic lambda-calculus that corresponds to Haskell without distinguishing error causes, i.e., with only one erroneous value $\bot$.

The syntax of types and terms is given in Figure 1, where α ranges over type variables, x over term variables, and n over the integers. We include integers and lists as representatives for numeric types and algebraic datatypes, and addition as an exemplary numeric operation. Note that the calculus is explicitly typed and that type abstraction and application are explicit in the syntax as well. General recursion is captured via a fixpoint combinator, while selective strictness (à la *seq*) is provided via a strict-let construct.

$$\begin{array}{rl}\tau ::= & \alpha \mid \mathsf{Int} \mid [\tau] \mid \tau \to \tau \mid \forall \alpha.\tau\\ t\ ::= & x \mid n \mid t + t \mid [\,]_\tau \mid t : t \mid \mathbf{case}\ t\ \mathbf{of}\ \{[\,] \to t\,;\ x : x \to t\} \mid\\ & \lambda x : \tau.t \mid t\ t \mid \Lambda\alpha.t \mid t\ \tau \mid \mathbf{fix} \mid \mathbf{let!}\ x = t\ \mathbf{in}\ t\end{array}$$

Figure 1: Syntax of Types τ and Terms t.

Figure 2 gives the typing rules for our calculus. Standard conventions apply here. In particular, typing environments Γ take the form $\alpha_1, \ldots, \alpha_k, x_1 : \tau_1, \ldots, x_l : \tau_l$ with distinct α_i and x_j, where all free variables occurring in a τ_j have to be among the listed type variables.

$$\Gamma, x : \tau \vdash x : \tau \qquad \Gamma \vdash n : \mathsf{Int} \qquad \Gamma \vdash [\,]_\tau : [\tau] \qquad \Gamma \vdash \mathbf{fix} : \forall \alpha.(\alpha \to \alpha) \to \alpha$$

$$\frac{\Gamma \vdash t_1 : \mathsf{Int} \qquad \Gamma \vdash t_2 : \mathsf{Int}}{\Gamma \vdash (t_1 + t_2) : \mathsf{Int}} \qquad \frac{\Gamma \vdash t_1 : \tau \qquad \Gamma \vdash t_2 : [\tau]}{\Gamma \vdash (t_1 : t_2) : [\tau]} \qquad \frac{\alpha, \Gamma \vdash t : \tau}{\Gamma \vdash (\Lambda \alpha.t) : \forall \alpha.\tau}$$

$$\frac{\Gamma \vdash t : [\tau_1] \qquad \Gamma \vdash t_1 : \tau_2 \qquad \Gamma, x_1 : \tau_1, x_2 : [\tau_1] \vdash t_2 : \tau_2}{\Gamma \vdash (\mathbf{case}\ t\ \mathbf{of}\ \{[\,] \to t_1\,;\ x_1 : x_2 \to t_2\}) : \tau_2} \qquad \frac{\Gamma \vdash t : \forall \alpha.\tau_1}{\Gamma \vdash (t\ \tau_2) : \tau_1[\tau_2/\alpha]}$$

$$\frac{\Gamma, x : \tau_1 \vdash t : \tau_2}{\Gamma \vdash (\lambda x : \tau_1.t) : \tau_1 \to \tau_2} \qquad \frac{\Gamma \vdash t_1 : \tau_1 \to \tau_2 \qquad \Gamma \vdash t_2 : \tau_1}{\Gamma \vdash (t_1\ t_2) : \tau_2}$$

$$\frac{\Gamma \vdash t_1 : \tau_1 \qquad \Gamma, x : \tau_1 \vdash t_2 : \tau_2}{\Gamma \vdash (\mathbf{let!}\ x = t_1\ \mathbf{in}\ t_2) : \tau_2}$$

Figure 2: Typing Rules.

For example, the function *map* can be defined as the following term and then satisfies $\vdash map : \tau$, where $\tau = \forall \alpha.\forall \beta.(\alpha \to \beta) \to [\alpha] \to [\beta]$:

$$\begin{aligned}&\mathbf{fix}\ \tau\ (\lambda m : \tau.\Lambda \alpha.\Lambda \beta.\lambda h : \alpha \to \beta.\lambda l : [\alpha].\\ &\qquad \mathbf{case}\ l\ \mathbf{of}\ \{[\,] \to [\,]_\beta\,;\ x : y \to (h\ x) : (m\ \alpha\ \beta\ h\ y)\})\,.\end{aligned}$$

The denotational semantics interprets types as *pointed complete partial orders* (for short, *pcpos*; least element always denoted $\bot$). The definition in Figure 3, assuming θ to be a mapping from type variables to pcpos, is entirely standard. The operation $lift_\bot$ takes a complete partial order, adds a new element $\bot$ to the carrier set, defines this new $\bot$ to be below every other element, and leaves the ordering otherwise unchanged. To avoid confusion, the original elements are tagged, i.e.,

$$lift_\bot\, S = \{\bot\} \cup \{\lfloor s \rfloor \mid s \in S\}\,.$$

The complete partial order lifted in the definition of $[\![\mathsf{Int}]\!]_\theta$ is the flat one without ordering between integers. For list types, prior to lifting, $[\,]$ is only related to itself, while the ordering between "$- : -$"-values is component-wise. Also note the use of the greatest fixpoint to provide for infinite lists.[2] The function space lifted in the definition of $[\![\tau_1 \to \tau_2]\!]_\theta$ is the one of monotonic and continuous maps between $[\![\tau_1]\!]_\theta$ and $[\![\tau_2]\!]_\theta$, ordered point-wise. Finally, polymorphic types are interpreted as sets of functions from pcpos to values restricted as in the last line of Figure 3, and again ordered point-wise (i.e., $g_1 \sqsubseteq g_2$ iff for every pcpo D, $g_1\ D \sqsubseteq g_2\ D$).

The semantics of terms in Figure 4 is also standard. It uses λ for denoting anonymous functions, and the following operator:

$$h\ \$\ a = \begin{cases} f\ a & \text{if } h = \lfloor f \rfloor \\ \bot & \text{if } h = \bot\,. \end{cases}$$

[2]There is no distinction between *gfp* and *lfp* if we right away demand a pcpo, not just a set, satisfying $S = lift_\bot\,(\{[\,]\} \cup \{a : b \mid a \in [\![\tau]\!]_\theta,\ b \in S\}$.

$$
\begin{aligned}
[\![\alpha]\!]_\theta &= \theta(\alpha)\\
[\![\mathsf{Int}]\!]_\theta &= \mathit{lift}_\bot\,\{\ldots,\,-2,\,-1,\,0,\,1,\,2,\,\ldots\}\\
[\![[\tau]]\!]_\theta &= \mathit{gfp}\,(\lambda S.\mathit{lift}_\bot\,(\{[\,]\}\cup\{a:b\mid a\in[\![\tau]\!]_\theta,\,b\in S\}))\\
[\![\tau_1\to\tau_2]\!]_\theta &= \mathit{lift}_\bot\,\{f:[\![\tau_1]\!]_\theta\to[\![\tau_2]\!]_\theta\}\\
[\![\forall\alpha.\tau]\!]_\theta &= \{g\mid\forall D\text{ pcpo. }(g\ D)\in[\![\tau]\!]_{\theta[\alpha\mapsto D]}\}
\end{aligned}
$$

Figure 3: Standard Semantics of Types.

$$
\begin{aligned}
[\![x]\!]_{\theta,\sigma} &= \sigma(x)\\
[\![n]\!]_{\theta,\sigma} &= \lfloor n\rfloor\\
[\![t_1+t_2]\!]_{\theta,\sigma} &= \begin{cases}\lfloor n_1+n_2\rfloor & \text{if } [\![t_1]\!]_{\theta,\sigma}=\lfloor n_1\rfloor,\ [\![t_2]\!]_{\theta,\sigma}=\lfloor n_2\rfloor\\ \bot & \text{otherwise}\end{cases}\\
[\![[\,]_\tau]\!]_{\theta,\sigma} &= \lfloor[\,]\rfloor\\
[\![t_1:t_2]\!]_{\theta,\sigma} &= \lfloor[\![t_1]\!]_{\theta,\sigma}:[\![t_2]\!]_{\theta,\sigma}\rfloor\\
[\![\mathbf{case}\ t\ \mathbf{of}\ \{[\,]\to t_1\,;\ x_1:x_2\to t_2\}]\!]_{\theta,\sigma} &= \begin{cases}[\![t_1]\!]_{\theta,\sigma} & \text{if } [\![t]\!]_{\theta,\sigma}=\lfloor[\,]\rfloor\\ [\![t_2]\!]_{\theta,\sigma[x_1\mapsto a,\,x_2\mapsto b]} & \text{if } [\![t]\!]_{\theta,\sigma}=\lfloor a:b\rfloor\\ \bot & \text{if } [\![t]\!]_{\theta,\sigma}=\bot\end{cases}\\
[\![\lambda x:\tau.t]\!]_{\theta,\sigma} &= \lfloor\lambda a.[\![t]\!]_{\theta,\sigma[x\mapsto a]}\rfloor\\
[\![t_1\ t_2]\!]_{\theta,\sigma} &= [\![t_1]\!]_{\theta,\sigma}\ \$\ [\![t_2]\!]_{\theta,\sigma}\\
[\![\Lambda\alpha.t]\!]_{\theta,\sigma} &= \lambda D.[\![t]\!]_{\theta[\alpha\mapsto D],\sigma}\\
[\![t\ \tau]\!]_{\theta,\sigma} &= [\![t]\!]_{\theta,\sigma}\ [\![\tau]\!]_\theta\\
[\![\mathbf{fix}]\!]_{\theta,\sigma} &= \lambda D.\lfloor\lambda h.\textstyle\bigsqcup((h\,\$)^i\ \bot)\rfloor\\
[\![\mathbf{let!}\ x=t_1\ \mathbf{in}\ t_2]\!]_{\theta,\sigma} &= \begin{cases}[\![t_2]\!]_{\theta,\sigma[x\mapsto a]} & \text{if } [\![t_1]\!]_{\theta,\sigma}=a\neq\bot\\ \bot & \text{if } [\![t_1]\!]_{\theta,\sigma}=\bot\end{cases}
\end{aligned}
$$

Figure 4: Standard Semantics of Terms.

The expression $\bigsqcup((h\,\$)^i\ \bot)$ in the definition for **fix** means the supremum of the chain $\bot\sqsubseteq h\ \$\ \bot\sqsubseteq h\ \$\,(h\ \$\ \bot)\cdots$. Altogether, we have that if $\Gamma\vdash t:\tau$ and $\sigma(x)\in[\![\tau']\!]_\theta$ for every $x:\tau'$ occurring in Γ, then $[\![t]\!]_{\theta,\sigma}\in[\![\tau]\!]_\theta$.

The key to parametricity results is the definition of a family of relations by induction on a calculus' type structure. The appropriate such *logical relation* for our current setting is defined in Figure 5, assuming ρ to be a mapping from type variables to binary relations between pcpos. We use id_D to denote the identity relation on the pcpo D. The operation *list* takes a relation $\mathcal{R}$ and maps it to

$$\mathit{list}\,\mathcal{R}=\mathit{gfp}\,(\lambda\mathcal{S}.\{(\bot,\bot),\,(\lfloor[\,]\rfloor,\lfloor[\,]\rfloor)\}\cup\{(\lfloor a:b\rfloor,\lfloor c:d\rfloor)\mid(a,c)\in\mathcal{R},\,(b,d)\in\mathcal{S}\})\,,$$

where again the greatest fixpoint is taken. For two pcpos D_1 and D_2, $\mathit{Rel}(D_1,D_2)$ collects all relations between them that are *strict*, *continuous*, and *bottom-reflecting*. Strictness and continuity are just the standard notions, i.e., membership of the pair

$$\begin{array}{ll}
\Delta_{\alpha,\rho} & = \rho(\alpha) \\
\Delta_{\mathsf{Int},\rho} & = id_{\mathit{lift}_\bot\{\ldots,-2,-1,0,1,2,\ldots\}} \\
\Delta_{[\tau],\rho} & = \mathit{list}\, \Delta_{\tau,\rho} \\
\Delta_{\tau_1\to\tau_2,\rho} & = \{(f,g) \mid f=\bot \text{ iff } g=\bot,\, \forall (a,b)\in\Delta_{\tau_1,\rho}.\ (f \$ a, g \$ b)\in\Delta_{\tau_2,\rho}\} \\
\Delta_{\forall\alpha.\tau,\rho} & = \{(u,v) \mid \forall D_1,D_2 \text{ pcpos}, \mathcal{R}\in \mathit{Rel}(D_1,D_2).\ (u\ D_1, v\ D_2)\in\Delta_{\tau,\rho[\alpha\mapsto\mathcal{R}]}\}
\end{array}$$

Figure 5: Standard Logical Relation.

$(\bot,\bot)$ and closure under suprema. A relation $\mathcal{R}$ is bottom-reflecting if $(a,b)\in\mathcal{R}$ implies that $a=\bot$ iff $b=\bot$. The corresponding explicit condition on f and g in the definition of $\Delta_{\tau_1\to\tau_2,\rho}$ serves the purpose of ensuring that bottom-reflectingness is preserved throughout the logical relation. Overall, reasoning like Johann and Voigtländer (2004) gives the following important lemma (by induction on τ), where *Rel* is the union of all $\mathit{Rel}(D_1,D_2)$.

Lemma 2.1. *If ρ maps only to relations in Rel, then $\Delta_{\tau,\rho}\in \mathit{Rel}$ as well.*

The lemma is crucial for then proving the following parametricity theorem.

Theorem 2.2. *If $\Gamma\vdash t:\tau$, then for every θ_1, θ_2, ρ, σ_1, and σ_2 such that*

- *for every α occurring in Γ, $\rho(\alpha)\in \mathit{Rel}(\theta_1(\alpha),\theta_2(\alpha))$, and*
- *for every $x:\tau'$ occurring in Γ, $(\sigma_1(x),\sigma_2(x))\in\Delta_{\tau',\rho}$,*

we have $([\![t]\!]_{\theta_1,\sigma_1},[\![t]\!]_{\theta_2,\sigma_2})\in\Delta_{\tau,\rho}$.

One peculiarity of the semantic setup considered so far is that the type semantics of all except the polymorphic types are lifted. That is, the complete partial order interpretations of all other types are not just pointed, i.e., contain a least element $\bot$, but actually lifted, i.e., all non-$\bot$ elements are explicitly tagged with $\lfloor - \rfloor$. That we do not need to lift the interpretation of polymorphic types has to do with the fact that type abstraction and application are implicit in surface Haskell and carry no computational content. In other words, there is no difference in Haskell between $\Lambda\alpha.\bot$ and $\bot$, while there *is* a difference between $\lambda x:\tau.\bot$ and $\bot$ (with regard to selective strictness). That is why $[\![\tau_1\to\tau_2]\!]_\theta$ is lifted in Figure 3, why $[\![\lambda x:\tau.t]\!]_{\theta,\sigma}$ and $[\![t_1\ t_2]\!]_{\theta,\sigma}$ involve $\lfloor - \rfloor$ and \$ in Figure 4, and why $\Delta_{\tau_1\to\tau_2,\rho}$ requires the "$f=\bot$ iff $g=\bot$"-condition in Figure 5, while $[\![\forall\alpha.\tau]\!]_\theta$ is not lifted, $[\![\Lambda\alpha.t]\!]_{\theta,\sigma}$ and $[\![t\ \tau]\!]_{\theta,\sigma}$ are defined more directly, and $\Delta_{\forall\alpha.\tau,\rho}$ requires no extra condition, and yet Lemma 2.1 holds (and so does, ultimately, Theorem 2.2).

For preparing the development in the next section, though, it would be more beneficial if all the cases of the type semantics were homogeneous in their lifting behavior. This is indeed possible without disrupting the overall approach and results. Figure 6 collects the necessary adaptations to Figures 3–5, explained in some more detail below. One important thing to note is that the adapted semantics is truly

$$\begin{aligned}
[\![\forall\alpha.\tau]\!]_\theta &= \mathit{lift}_\perp \{g \mid \forall D \text{ pcpo. } (g\ D) \in [\![\tau]\!]_{\theta[\alpha\mapsto D]} \setminus \{\perp\}\} \\
[\![\Lambda\alpha.t]\!]_{\theta,\sigma} &= \begin{cases} \lfloor \lambda D.[\![t]\!]_{\theta[\alpha\mapsto D],\sigma} \rfloor & \text{if } [\![t]\!]_{\theta[\alpha\mapsto\{\perp\}],\sigma} \neq \perp \\ \perp & \text{if } [\![t]\!]_{\theta[\alpha\mapsto\{\perp\}],\sigma} = \perp \end{cases} \\
[\![t\ \tau]\!]_{\theta,\sigma} &= [\![t]\!]_{\theta,\sigma} \ \$\$\ [\![\tau]\!]_\theta \\
[\![\mathbf{fix}]\!]_{\theta,\sigma} &= \lfloor \lambda D. \lfloor \lambda h. \bigsqcup ((h\,\$)^i\ \perp) \rfloor \rfloor \\
\Delta_{\forall\alpha.\tau,\rho} &= \{(u,v) \mid \forall D_1, D_2 \text{ pcpos}, \mathcal{R} \in \mathit{Rel}(D_1,D_2). \\
&\qquad\qquad (u\ \$\$\ D_1, v\ \$\$\ D_2) \in \Delta_{\tau,\rho[\alpha\mapsto\mathcal{R}]}\}
\end{aligned}$$

Figure 6: Adaptations for Explicit Lifting at Polymorphic Types.

equivalent to the standard one from a user's perspective, i.e., with respect to which terms of the calculus are equated or found to be in an approximation relationship.

For $[\![\forall\alpha.\tau]\!]_\theta$, we explicitly add a new $\perp$ via $\mathit{lift}_\perp$, while at the same time excluding, via a refined condition, the least element already present in $\{g \mid \forall D \text{ pcpo. } (g\ D) \in [\![\tau]\!]_{\theta[\alpha\mapsto D]}\}$. Actually, the additional condition that for every pcpo D, $(g\ D) \neq \perp$, excludes more than just the least g. It excludes every g that maps *any* D to $\perp$, not just the single g that maps *every* D to $\perp$. But this is fine, given that it is actually impossible to define a polymorphic value in Haskell (or as a term in our calculus) of which the instantiation at some type is $\perp$ and at some other type is non-$\perp$. This is justified by Lemma 7.17 of Voigtländer and Johann (2007) and also the type-erasing operational semantics of Johann and Voigtländer (2008).

For $[\![\Lambda\alpha.t]\!]_{\theta,\sigma}$, we need to analyze the semantics of t to sort out the exceptional case that every D is mapped to $\perp$, in which case the semantics of $\Lambda\alpha.t$ should itself be the $\perp$ explicitly added via $\mathit{lift}_\perp$. By the observation made in the previous paragraph, it is not actually necessary to check the behavior for every D. Instead, the test can be performed with a single, arbitrary pcpo. We choose the simplest one, namely just $\{\perp\}$. If we find that we are not in the exceptional case, the denotation is just as before, but appropriately lifted via $\lfloor - \rfloor$. A simple lifting also deals with $[\![\mathbf{fix}]\!]_{\theta,\sigma}$.

For adapting $[\![t\ \tau]\!]_{\theta,\sigma}$ and $\Delta_{\forall\alpha.\tau,\rho}$, we simply use an operator in the spirit of \$:

$$h\ \$\$\ D = \begin{cases} g\ D & \text{if } h = \lfloor g \rfloor \\ \perp & \text{if } h = \perp\,. \end{cases}$$

Overall, the adaptations leave Lemma 2.1 and Theorem 2.2, as well as their proofs, intact. Only for reference, the proof of Theorem 2.2 for the revised setting, and spelling out more formally the "narrative" of Johann and Voigtländer (2004), is given in Appendix A.

3 Imprecise Error Semantics

We now want to treat different failure causes as semantically different, rather than conflating them into a single erroneous value $\perp$. To this end, we add a new term-former to the syntax from Figure 1:

$$t \ ::= \ \cdots \mid \mathbf{error}\,,$$

and a new typing rule to the derivation system from Figure 2:

$$\Gamma \vdash \mathbf{error} : \forall\alpha.\mathsf{Int} \to \alpha\,.$$

Note that, deviating from Haskell, we use integers rather than strings as descriptive arguments to **error**. Of course, this is not an essential difference, and only done here for the sake of simplicity.

Our treatment of errors shall be that of Peyton Jones et al. (1999) and Moran et al. (1999). In particular, we will use, and embellish, the denotational semantics given by Peyton Jones et al. (1999), our main extension being that they do not explicitly describe how to deal with polymorphic types. Their main innovation, and the reason for calling the semantics "imprecise", is the use of *sets* of possible failure causes. Formally, let

$$\mathcal{E} = \{\mathsf{ErrorCall}\ n \mid n \in \{\ldots, -2, -1, 0, 1, 2, \ldots\}\} \tag{2}$$

and

$$\mathcal{E}^{nt} = \{\mathsf{NonTermination}\} \cup \mathcal{E}\,,$$

where NonTermination and ErrorCall are (for now) only descriptive tags for use in the denotational semantics, but without direct syntactical counterparts in the underlying calculus. The set of all erroneous values is then

$$V_{err} = \{Bad\ e \mid e \in \mathcal{P}(\mathcal{E}) \cup \{\mathcal{E}^{nt}\}\}$$

and its elements are ordered by

$$Bad\ e \sqsubseteq Bad\ e' \ \text{ iff } \ e \supseteq e'\,. \tag{3}$$

Peyton Jones et al. (1999) then replace the standard operation $lift_\perp$ by

$$lift_{err}\ S = V_{err} \cup \{Ok\ s \mid s \in S\}\,.$$

The approximation order on such error-lifted complete partial orders (henceforth, for short, *elcpos*) is given by (3) on erroneous values, by taking over the order from S for nonerroneous values, and by mandating that $\perp = Bad\ \mathcal{E}^{nt}$ is below all, even nonerroneous, values, while otherwise erroneous and nonerroneous values are pairwise incomparable.

With these definitions in place, the first four lines of Figure 7 should hold no surprises, as they are in complete analogy to Figure 3. Of course, we now assume that θ maps to elcpos only. The last line of Figure 7 is in analogy to the first one of Figure 6. By subtracting V_{err} from the possible ranges of g, we mandate that a nonerroneous polymorphic value does not have any erroneous instantiation,

$$
\begin{array}{ll}
[\![\alpha]\!]^{err}_\theta & = \theta(\alpha) \\
[\![\mathsf{Int}]\!]^{err}_\theta & = \mathit{lift}_{err}\,\{\ldots,\,-2,\,-1,\,0,\,1,\,2,\,\ldots\} \\
[\![[\tau]]\!]^{err}_\theta & = \mathit{gfp}\,(\lambda S.\mathit{lift}_{err}\,(\{[\,]\} \cup \{a : b \mid a \in [\![\tau]\!]^{err}_\theta,\, b \in S\})) \\
[\![\tau_1 \to \tau_2]\!]^{err}_\theta & = \mathit{lift}_{err}\,\{f : [\![\tau_1]\!]^{err}_\theta \to [\![\tau_2]\!]^{err}_\theta\} \\
[\![\forall\alpha.\tau]\!]^{err}_\theta & = \mathit{lift}_{err}\,\{g \mid \forall D \text{ elcpo. } (g\ D) \in [\![\tau]\!]^{err}_{\theta[\alpha\mapsto D]} \setminus V_{err}\}
\end{array}
$$

Figure 7: Error Semantics of Types.

and thus in particular again exclude (as in Haskell) polymorphic values of which the instantiation at some type is erroneous and at some other type is nonerroneous. More specifically, we also expect that an erroneous polymorphic value exhibits exactly the same potential failing behavior in each of its instantiations. Of course, ensuring all this also depends on the term semantics, to be considered next.

Some of the definitions in Figure 8 are directly taken over from Figure 4, modulo lifting via Ok rather than via $\lfloor - \rfloor$, and thus need not be further discussed here. The definitions of $[\![t_1\ t_2]\!]^{err}_{\theta,\sigma}$ and $[\![\mathbf{fix}]\!]^{err}_{\theta,\sigma}$ are as in Figures 4 and 6, respectively, but use the following variant of the operator \$:

$$
h \ \$\ a = \begin{cases} f\ a & \text{if } h = \mathit{Ok}\ f \\ \mathit{Bad}\ (e \cup E(a)) & \text{if } h = \mathit{Bad}\ e\,, \end{cases}
$$

where

$$
E(a) = \begin{cases} \emptyset & \text{if } a = \mathit{Ok}\ v \\ e & \text{if } a = \mathit{Bad}\ e\,. \end{cases}
$$

The crucial point here, taken over from Peyton Jones et al. (1999), is that application of an erroneous function value incurs all potential failures of the argument as well. Also essentially taken over are the definitions of $[\![t_1 + t_2]\!]^{err}_{\theta,\sigma}$ and $[\![\mathbf{case}\ t\ \mathbf{of}\ \{[\,] \to t_1\,;\ x_1 : x_2 \to t_2\}]\!]^{err}_{\theta,\sigma}$, except that we do not check for overflow in the case of addition. To bring about erroneous values other than $\bot$ in the first place, we have the obvious definition of $[\![\mathbf{error}]\!]^{err}_{\theta,\sigma}$. The definitions of $[\![\Lambda\alpha.t]\!]^{err}_{\theta,\sigma}$ and $[\![t\ \tau]\!]^{err}_{\theta,\sigma}$ follow the corresponding ones in Figure 6, but with the following variant of the operator \$\$:

$$
h \ \$\$\ D = \begin{cases} g\ D & \text{if } h = \mathit{Ok}\ g \\ \mathit{Bad}\ e & \text{if } h = \mathit{Bad}\ e\,. \end{cases}
$$

Finally, the definition of $[\![\mathbf{let!}\ x = t_1\ \mathbf{in}\ t_2]\!]^{err}_{\theta,\sigma}$ follows the one in Figure 4, but similarly to the definition of $[\![\mathbf{case}\ t\ \mathbf{of}\ \{[\,] \to t_1\,;\ x_1 : x_2 \to t_2\}]\!]^{err}_{\theta,\sigma}$, and in line with the operational semantics of Moran et al. (1999), t_2 is evaluated in "error-finding mode" to contribute further potential failure causes in case t_1 is already erroneous.

4 Parametricity for Imprecise Error Semantics

To establish an analogue of Theorem 2.2 for the imprecise error semantics, we first need to determine just the right set of restrictions to impose on relational inter-

$$
\begin{aligned}
&[\![x]\!]^{err}_{\theta,\sigma} && = \sigma(x)\\
&[\![n]\!]^{err}_{\theta,\sigma} && = \mathit{Ok}\ n\\
&[\![t_1 + t_2]\!]^{err}_{\theta,\sigma} && =\\
&\quad \begin{cases} \mathit{Ok}\ (n_1 + n_2) & \text{if } [\![t_1]\!]^{err}_{\theta,\sigma} = \mathit{Ok}\ n_1,\ [\![t_2]\!]^{err}_{\theta,\sigma} = \mathit{Ok}\ n_2\\ \mathit{Bad}\ (E([\![t_1]\!]^{err}_{\theta,\sigma}) \cup E([\![t_2]\!]^{err}_{\theta,\sigma})) & \text{otherwise}\end{cases}\\
&[\![[\,]_\tau]\!]^{err}_{\theta,\sigma} && = \mathit{Ok}\ [\,]\\
&[\![t_1 : t_2]\!]^{err}_{\theta,\sigma} && = \mathit{Ok}\ ([\![t_1]\!]^{err}_{\theta,\sigma} : [\![t_2]\!]^{err}_{\theta,\sigma})\\
&[\![\mathbf{case}\ t\ \mathbf{of}\ \{[\,] \to t_1\ ;\ x_1 : x_2 \to t_2\}]\!]^{err}_{\theta,\sigma} =\\
&\quad \begin{cases} [\![t_1]\!]^{err}_{\theta,\sigma} & \text{if } [\![t]\!]^{err}_{\theta,\sigma} = \mathit{Ok}\ [\,]\\ [\![t_2]\!]^{err}_{\theta,\sigma[x_1 \mapsto a,\, x_2 \mapsto b]} & \text{if } [\![t]\!]^{err}_{\theta,\sigma} = \mathit{Ok}\ (a : b)\\ \mathit{Bad}\ (e \cup E([\![t_1]\!]^{err}_{\theta,\sigma}) \cup E([\![t_2]\!]^{err}_{\theta,\sigma[x_1 \mapsto \mathit{Bad}\,\emptyset,\, x_2 \mapsto \mathit{Bad}\,\emptyset]})) & \text{if } [\![t]\!]^{err}_{\theta,\sigma} = \mathit{Bad}\ e\end{cases}\\
&[\![\lambda x : \tau . t]\!]^{err}_{\theta,\sigma} && = \mathit{Ok}\ (\lambda a . [\![t]\!]^{err}_{\theta,\sigma[x \mapsto a]})\\
&[\![t_1\ t_2]\!]^{err}_{\theta,\sigma} && = [\![t_1]\!]^{err}_{\theta,\sigma}\ \$\ [\![t_2]\!]^{err}_{\theta,\sigma}\\
&[\![\Lambda\alpha . t]\!]^{err}_{\theta,\sigma} && = \begin{cases} \mathit{Ok}\ (\lambda D . [\![t]\!]^{err}_{\theta[\alpha \mapsto D],\sigma}) & \text{if } [\![t]\!]^{err}_{\theta[\alpha \mapsto V_{err}],\sigma} = \mathit{Ok}\ v\\ \mathit{Bad}\ e & \text{if } [\![t]\!]^{err}_{\theta[\alpha \mapsto V_{err}],\sigma} = \mathit{Bad}\ e\end{cases}\\
&[\![t\ \tau]\!]^{err}_{\theta,\sigma} && = [\![t]\!]^{err}_{\theta,\sigma}\ \$\$\ [\![\tau]\!]^{err}_{\theta}\\
&[\![\mathbf{fix}]\!]^{err}_{\theta,\sigma} && = \mathit{Ok}\ (\lambda D . \mathit{Ok}\ (\lambda h . \textstyle\bigsqcup((h\,\$)^i\ \bot)))\\
&[\![\mathbf{let!}\ x = t_1\ \mathbf{in}\ t_2]\!]^{err}_{\theta,\sigma} && = \begin{cases} [\![t_2]\!]^{err}_{\theta,\sigma[x \mapsto \mathit{Ok}\,v]} & \text{if } [\![t_1]\!]^{err}_{\theta,\sigma} = \mathit{Ok}\ v\\ \mathit{Bad}\ (e \cup E([\![t_2]\!]^{err}_{\theta,\sigma[x \mapsto \mathit{Bad}\,\emptyset]})) & \text{if } [\![t_1]\!]^{err}_{\theta,\sigma} = \mathit{Bad}\ e\end{cases}\\
&[\![\mathbf{error}]\!]^{err}_{\theta,\sigma} && = \mathit{Ok}\ (\lambda D . \mathit{Ok}\ (\lambda a . \begin{cases} \mathit{Bad}\ \{\mathsf{ErrorCall}\ n\} & \text{if } a = \mathit{Ok}\ n\\ \mathit{Bad}\ e & \text{if } a = \mathit{Bad}\ e\end{cases}))
\end{aligned}
$$

Figure 8: Error Semantics of Terms.

pretations of types. So far, we required strict, continuous, and bottom-reflecting relations. It seems reasonable that continuity will still be required, as we still have to provide for general recursion via the fixpoint combinator. But for strictness and bottom-reflectingness, the situation is less clear, as we now have more than a single erroneous value $\bot$ to consider.

For example, strictness currently only states that the pair $(\bot, \bot)$, i.e. the pair $(Bad\ \mathcal{E}^{nt}, Bad\ \mathcal{E}^{nt})$, should be contained in every relation. But what about other erroneous values? Should any pair of them be related? Or only identical ones? Or is inclusion of $(\bot, \bot)$ actually enough?

The best way to answer such questions is to go through the proof of Theorem 2.2 and see where changes in the calculus and its semantics might require a change in the proof. In our case, it of course makes the most sense to first study the impact of the new primitive **error**. Recall that its typing rule is as follows:

$$\Gamma \vdash \textbf{error} : \forall \alpha.\mathsf{Int} \to \alpha\,.$$

So we will have to prove that for every θ_1, θ_2, ρ, σ_1, and σ_2 such that

- for every α occurring in Γ, $\rho(\alpha)$ is an appropriately restricted relation between $\theta_1(\alpha)$ and $\theta_2(\alpha)$, and
- for every $x : \tau'$ occurring in Γ, $(\sigma_1(x), \sigma_2(x)) \in \Delta_{\tau',\rho}$,

we have $([\![\textbf{error}]\!]^{err}_{\theta_1,\sigma_1}, [\![\textbf{error}]\!]^{err}_{\theta_2,\sigma_2}) \in \Delta_{\forall\alpha.\mathsf{Int}\to\alpha,\rho}$.

Apparently, by Figure 6, this will require to establish that for every (appropriate) D_1, D_2, and $\mathcal{R}$,

$$([\![\textbf{error}]\!]^{err}_{\theta_1,\sigma_1}\ \$\$\ D_1, [\![\textbf{error}]\!]^{err}_{\theta_2,\sigma_2}\ \$\$\ D_2) \in \Delta_{\mathsf{Int}\to\alpha,\rho[\alpha\mapsto\mathcal{R}]}\,.$$

Further unfolding the current definition of Δ, now via Figure 5, tells us that we will have to show that

$$[\![\textbf{error}]\!]^{err}_{\theta_1,\sigma_1}\ \$\$\ D_1 = \bot \ \text{ iff }\ [\![\textbf{error}]\!]^{err}_{\theta_2,\sigma_2}\ \$\$\ D_2 = \bot \tag{4}$$

(or a similar statement involving also non-$\bot$ erroneous values?), and that for every $(a, b) \in \Delta_{\mathsf{Int},\rho[\alpha\mapsto\mathcal{R}]}$,

$$([\![\textbf{error}]\!]^{err}_{\theta_1,\sigma_1}\ \$\$\ D_1\ \$\ a, [\![\textbf{error}]\!]^{err}_{\theta_2,\sigma_2}\ \$\$\ D_2\ \$\ b) \in \mathcal{R}\,.$$

Taking into account that the integer type should still be interpreted by an identity relation, and using the semantics definitions given in Section 3, the latter is the same as requiring that for every $a \in \mathit{lift}_{err}\,\{\ldots, -2, -1, 0, 1, 2, \ldots\}$, the value

$$\begin{cases} Bad\ \{\mathsf{ErrorCall}\ n\} & \text{if } a = Ok\ n \\ Bad\ e & \text{if } a = Bad\ e \end{cases} \tag{5}$$

is related to itself by $\mathcal{R}$, which is equivalent to requiring that every erroneous value is related to itself by $\mathcal{R}$. Therefore, we propose to generalize the notion of strictness as follows.

Definition 4.1. A relation $\mathcal{R}$ is *error-strict* if $id_{V_{err}} \subseteq \mathcal{R}$.

Similar questions as for strictness arise for bottom-reflectingness in the presence of different failure causes. Is it enough to maintain that two related values are either both $\perp$ or none of them is? Or should we generalize to requiring that either both are erroneous or none of them is? Or should we be more demanding by even expecting that only equal failure causes (or sets thereof) are related?

The relevant proof case to check here is the one for the strict-let construct, because selective strictness was what necessitated bottom-reflectingness in the first place (Johann and Voigtländer 2004). Recall that the typing rule is as follows:

$$\frac{\Gamma \vdash t_1 : \tau_1 \qquad \Gamma, x : \tau_1 \vdash t_2 : \tau_2}{\Gamma \vdash (\mathbf{let!}\ x = t_1\ \mathbf{in}\ t_2) : \tau_2}\,.$$

Inside the proof of an analogue of Theorem 2.2 by induction over typing derivations we will have to establish, for the induction conclusion in this case, that for θ_1, θ_2, ρ, σ_1, and σ_2 as above,

$$([\![\mathbf{let!}\ x = t_1\ \mathbf{in}\ t_2]\!]^{err}_{\theta_1,\sigma_1}, [\![\mathbf{let!}\ x = t_1\ \mathbf{in}\ t_2]\!]^{err}_{\theta_2,\sigma_2}) \in \Delta_{\tau_2,\rho}\,.$$

The semantics from Section 3 tells us that the two values in the relation of which we are interested here are equal to

$$\begin{cases} [\![t_2]\!]^{err}_{\theta_1,\sigma_1[x \mapsto Ok\ v_1]} & \text{if } [\![t_1]\!]^{err}_{\theta_1,\sigma_1} = Ok\ v_1 \\ Bad\ (e_1 \cup E([\![t_2]\!]^{err}_{\theta_1,\sigma_1[x \mapsto Bad\ \emptyset]})) & \text{if } [\![t_1]\!]^{err}_{\theta_1,\sigma_1} = Bad\ e_1 \end{cases} \tag{6}$$

and

$$\begin{cases} [\![t_2]\!]^{err}_{\theta_2,\sigma_2[x \mapsto Ok\ v_2]} & \text{if } [\![t_1]\!]^{err}_{\theta_2,\sigma_2} = Ok\ v_2 \\ Bad\ (e_2 \cup E([\![t_2]\!]^{err}_{\theta_2,\sigma_2[x \mapsto Bad\ \emptyset]})) & \text{if } [\![t_1]\!]^{err}_{\theta_2,\sigma_2} = Bad\ e_2\,, \end{cases} \tag{7}$$

respectively. The role of bottom-reflectingness in the $\perp$-only setting is to ensure, via the induction hypothesis corresponding to the precondition $\Gamma \vdash t_1 : \tau_1$, namely

$$([\![t_1]\!]^{err}_{\theta_1,\sigma_1}, [\![t_1]\!]^{err}_{\theta_2,\sigma_2}) \in \Delta_{\tau_1,\rho}\,, \tag{8}$$

that the same branch is chosen in (the analogues of) the two case distinctions above. Here the same can be achieved by introducing an auxiliary function extracting the tag of a value as follows:

$$T(a) = \begin{cases} Ok & \text{if } a = Ok\ v \\ Bad & \text{if } a = Bad\ e\,, \end{cases}$$

and generalizing bottom-reflectingness in such a way that related values are always required to have the same image under T.

But does this suffice? For the case that $[\![t_1]\!]^{err}_{\theta_1,\sigma_1} = \mathit{Ok}\ v_1$ and $[\![t_1]\!]^{err}_{\theta_2,\sigma_2} = \mathit{Ok}\ v_2$, yes, because we then get the desired

$$([\![t_2]\!]^{err}_{\theta_1,\sigma_1[x\mapsto \mathit{Ok}\, v_1]}, [\![t_2]\!]^{err}_{\theta_2,\sigma_2[x\mapsto \mathit{Ok}\, v_2]}) \in \Delta_{\tau_2,\rho}$$

from $(\mathit{Ok}\ v_1, \mathit{Ok}\ v_2) \in \Delta_{\tau_1,\rho}$ (cf. (8)) and the induction hypothesis corresponding to the precondition $\Gamma, x : \tau_1 \vdash t_2 : \tau_2$, namely that for every $(b, c) \in \Delta_{\tau_1,\rho}$,

$$([\![t_2]\!]^{err}_{\theta_1,\sigma_1[x\mapsto b]}, [\![t_2]\!]^{err}_{\theta_2,\sigma_2[x\mapsto c]}) \in \Delta_{\tau_2,\rho}\ . \tag{9}$$

However, in the case that $[\![t_1]\!]^{err}_{\theta_1,\sigma_1} = \mathit{Bad}\ e_1$ and $[\![t_1]\!]^{err}_{\theta_2,\sigma_2} = \mathit{Bad}\ e_2$, we need to show that

$$(\mathit{Bad}\ (e_1 \cup E([\![t_2]\!]^{err}_{\theta_1,\sigma_1[x\mapsto \mathit{Bad}\,\emptyset]})), \mathit{Bad}\ (e_2 \cup E([\![t_2]\!]^{err}_{\theta_2,\sigma_2[x\mapsto \mathit{Bad}\,\emptyset]}))) \in \Delta_{\tau_2,\rho}\ , \tag{10}$$

and do not yet have the means for doing so. Note that a supposed error-strictness of $\Delta_{\tau_2,\rho}$ would only allow us to conclude the desired membership if the sets $e_1 \cup E([\![t_2]\!]^{err}_{\theta_1,\sigma_1[x\mapsto \mathit{Bad}\,\emptyset]})$ and $e_2 \cup E([\![t_2]\!]^{err}_{\theta_2,\sigma_2[x\mapsto \mathit{Bad}\,\emptyset]})$ were equal. Revising the notion of error-strictness to guarantee that indeed any two erroneous values are related, independently of the sets of possible failures they represent, would risk completely blurring any distinction between different failure causes, and thus is no option. Instead, the proposed generalized notion of bottom-reflectingness is strengthened in a very natural way. Rather than just requiring that two related values always have the same image under T, we expect the same to be true under E.

Definition 4.2. A relation $\mathcal{R}$ is *error-reflecting* if $(a, b) \in \mathcal{R}$ implies that $T(a) = T(b)$ and $E(a) = E(b)$.[3]

Then, (8) and the assumption that $\Delta_{\tau_1,\rho}$ is error-reflecting imply that in the case $[\![t_1]\!]^{err}_{\theta_1,\sigma_1} = \mathit{Bad}\ e_1$ and $[\![t_1]\!]^{err}_{\theta_2,\sigma_2} = \mathit{Bad}\ e_2$ we even have $e_1 = e_2$. Moreover, (9) and $(\mathit{Bad}\ \emptyset, \mathit{Bad}\ \emptyset) \in \Delta_{\tau_1,\rho}$ (cf. supposed error-strictness of $\Delta_{\tau_1,\rho}$) give

$$([\![t_2]\!]^{err}_{\theta_1,\sigma_1[x\mapsto \mathit{Bad}\,\emptyset]}, [\![t_2]\!]^{err}_{\theta_2,\sigma_2[x\mapsto \mathit{Bad}\,\emptyset]}) \in \Delta_{\tau_2,\rho}\ ,$$

and thus by supposed error-reflectingness of $\Delta_{\tau_2,\rho}$,

$$E([\![t_2]\!]^{err}_{\theta_1,\sigma_1[x\mapsto \mathit{Bad}\,\emptyset]}) = E([\![t_2]\!]^{err}_{\theta_2,\sigma_2[x\mapsto \mathit{Bad}\,\emptyset]})$$

as well, which finally establishes (10), without having to revise the notion of error-strictness.

The above considerations lead us to focus on relations that are error-strict, continuous, and error-reflecting. Clearly, ensuring that these restrictions are preserved will require changes to the definition of Δ. For example, the operation *list* used

[3]Note that $E(a) = E(b)$ does not imply $T(a) = T(b)$, due to $\mathit{Bad}\ \emptyset$.

in Figure 5 will not suffice anymore (besides using the wrong kind of tagging: $\lfloor - \rfloor$ vs. *Ok*). But it is easy enough to replace it as follows:

$$\begin{aligned} \mathit{list}^{\mathit{err}}\,\mathcal{R} = \mathit{gfp}\,(\lambda\mathcal{S}.\mathit{id}_{V_{\mathit{err}}} &\cup \{(\mathit{Ok}\ [\,],\mathit{Ok}\ [\,])\} \\ &\cup \{(\mathit{Ok}\ (a:b),\mathit{Ok}\ (c:d)) \mid (a,c)\in\mathcal{R},\ (b,d)\in\mathcal{S}\})\,. \end{aligned}$$

For the case of function types, we clearly need an appropriate replacement for the "$f = \bot$ iff $g = \bot$"-condition occurring in the definition of $\Delta_{\tau_1\to\tau_2,\rho}$. It might seem that in order to guarantee error-reflectingness (instead of bottom-reflectingness, as earlier) we now have to require "$T(f) = T(g)$ and $E(f) = E(g)$". But actually it turns out that requiring just "$T(f) = T(g)$" is enough, as then the other conjunct can be established from relatedness of $f\ \$\ a$ and $g\ \$\ b$ for related a and b (see below). For the case of polymorphic types, we clearly have to restrict the quantified-over relations to error-strict, continuous, and error-reflecting ones. To this end, for given elcpos D_1 and D_2, let $\mathit{Rel}^{\mathit{err}}(D_1,D_2)$ collect all relations between them that are error-strict, continuous, and error-reflecting. (Also, let $\mathit{Rel}^{\mathit{err}}$ be the union of all $\mathit{Rel}^{\mathit{err}}(D_1,D_2)$.) Overall, we obtain the new logical relation defined in Figure 9. Comparing it to the first four definitions in Figure 5 and the last definition in Figure 6, note that it uses the versions of \$ and \$\$ from Section 3 rather than those from Section 2. We get the following analogue of Lemma 2.1.

Lemma 4.3. *If ρ maps only to relations in $\mathit{Rel}^{\mathit{err}}$, then $\Delta^{\mathit{err}}_{\tau,\rho} \in \mathit{Rel}^{\mathit{err}}$ as well.*

The proof is mostly routine, but we briefly sketch a few interesting parts related to the treatment of erroneous values:

- Error-strictness of $\Delta^{\mathit{err}}_{\tau_1\to\tau_2,\rho}$ follows from error-reflectingness of $\Delta^{\mathit{err}}_{\tau_1,\rho}$ and error-strictness of $\Delta^{\mathit{err}}_{\tau_2,\rho}$, because for every $e \in \mathcal{P}(\mathcal{E}) \cup \{\mathcal{E}^{nt}\}$ and a,b with $E(a) = E(b)$,
$$((\mathit{Bad}\ e)\ \$\ a, (\mathit{Bad}\ e)\ \$\ b) \in \mathit{id}_{V_{\mathit{err}}}\,.$$

- Error-reflectingness of $\Delta^{\mathit{err}}_{\tau_1\to\tau_2,\rho}$ follows from error-strictness of $\Delta^{\mathit{err}}_{\tau_1,\rho}$ and error-reflectingness of $\Delta^{\mathit{err}}_{\tau_2,\rho}$, because for every $e,e' \in \mathcal{P}(\mathcal{E}) \cup \{\mathcal{E}^{nt}\}$,
$$E((\mathit{Bad}\ e)\ \$\,(\mathit{Bad}\ \emptyset)) = E((\mathit{Bad}\ e')\ \$\,(\mathit{Bad}\ \emptyset))$$
implies $e = e'$.

- Error-strictness of $\Delta^{\mathit{err}}_{\forall\alpha.\tau,\rho}$ follows from error-strictness of $\Delta^{\mathit{err}}_{\tau,\rho[\alpha\mapsto\mathcal{R}]}$ for every error-strict, continuous, and error-reflecting relation $\mathcal{R}$, because for every $e \in \mathcal{P}(\mathcal{E}) \cup \{\mathcal{E}^{nt}\}$ and elcpos D_1 and D_2,
$$((\mathit{Bad}\ e)\ \$\$\ D_1, (\mathit{Bad}\ e)\ \$\$\ D_2) \in \mathit{id}_{V_{\mathit{err}}}\,.$$

- Error-reflectingness of $\Delta^{\mathit{err}}_{\forall\alpha.\tau,\rho}$ follows from error-reflectingness of $\Delta^{\mathit{err}}_{\tau,\rho[\alpha\mapsto\mathcal{R}]}$ for every error-strict, continuous, and error-reflecting relation $\mathcal{R}$, because for every (erroneous or nonerroneous) value h in $[\![\forall\alpha.\tau]\!]^{\mathit{err}}_{\theta}$ for some θ, and every elcpo D, $T(h\ \$\$\ D) = T(h)$ and $E(h\ \$\$\ D) = E(h)$.

$$\begin{array}{ll}
\Delta^{err}_{\alpha,\rho} & = \rho(\alpha) \\
\Delta^{err}_{\mathsf{Int},\rho} & = id_{\mathit{lift}_{err}\{\ldots,-2,-1,0,1,2,\ldots\}} \\
\Delta^{err}_{[\tau],\rho} & = \mathit{list}^{err}\ \Delta^{err}_{\tau,\rho} \\
\Delta^{err}_{\tau_1\to\tau_2,\rho} & = \{(f,g) \mid T(f) = T(g),\ \forall (a,b) \in \Delta^{err}_{\tau_1,\rho}.\ (f\ \$\ a, g\ \$\ b) \in \Delta^{err}_{\tau_2,\rho}\} \\
\Delta^{err}_{\forall\alpha.\tau,\rho} & = \{(u,v) \mid \forall D_1, D_2 \text{ elcpos}, \mathcal{R} \in \mathit{Rel}^{err}(D_1,D_2). \\
& \qquad\qquad (u\ \$\$\ D_1, v\ \$\$\ D_2) \in \Delta^{err}_{\tau,\rho[\alpha\mapsto\mathcal{R}]}\}
\end{array}$$

Figure 9: Logical Relation for Imprecise Error Semantics.

Being assured of Lemma 4.3 is nice, but not our ultimate goal in this section. Rather, we want the following analogue of Theorem 2.2.

Theorem 4.4. *If $\Gamma \vdash t : \tau$, then for every θ_1, θ_2, ρ, σ_1, and σ_2 such that*

- *for every α occurring in Γ, $\rho(\alpha) \in \mathit{Rel}^{err}(\theta_1(\alpha), \theta_2(\alpha))$, and*
- *for every $x : \tau'$ occurring in Γ, $(\sigma_1(x), \sigma_2(x)) \in \Delta^{err}_{\tau',\rho}$,*

we have $([\![t]\!]^{err}_{\theta_1,\sigma_1}, [\![t]\!]^{err}_{\theta_2,\sigma_2}) \in \Delta^{err}_{\tau,\rho}$.

Of course, Lemma 4.3 plays an important role in the proof, in particular in the inductive case for type application. The proof case for **error** was already discussed at the beginning of this section. The only change necessary to what was said there is that instead of (4) we actually need to establish that $T([\![\mathbf{error}]\!]^{err}_{\theta_1,\sigma_1}\ \$\$\ D_1) = T([\![\mathbf{error}]\!]^{err}_{\theta_2,\sigma_2}\ \$\$\ D_2)$. But this is straightforward from the term semantics, which forces both tags to be *Ok*.

Another proof case already given earlier in this section is the one for the strict-let construct. Clearly, it uses Lemma 4.3 as well, to deduce $T([\![t_1]\!]^{err}_{\theta_1,\sigma_1}) = T([\![t_1]\!]^{err}_{\theta_2,\sigma_2})$ from $([\![t_1]\!]^{err}_{\theta_1,\sigma_1}, [\![t_1]\!]^{err}_{\theta_2,\sigma_2}) \in \Delta^{err}_{\tau_1,\rho}$ and to deduce

$$(\mathit{Bad}\ (e_1 \cup E([\![t_2]\!]^{err}_{\theta_1,\sigma_1[x\mapsto \mathit{Bad}\,\emptyset]})), \mathit{Bad}\ (e_2 \cup E([\![t_2]\!]^{err}_{\theta_2,\sigma_2[x\mapsto \mathit{Bad}\,\emptyset]}))) \in \Delta^{err}_{\tau_2,\rho}$$

from $(\mathit{Bad}\ e_1, \mathit{Bad}\ e_2) \in \Delta^{err}_{\tau_1,\rho}$ and the statement that for every $(b,c) \in \Delta^{err}_{\tau_1,\rho}$, $([\![t_2]\!]^{err}_{\theta_1,\sigma_1[x\mapsto b]}, [\![t_2]\!]^{err}_{\theta_2,\sigma_2[x\mapsto c]}) \in \Delta^{err}_{\tau_2,\rho}$.

Two further proof cases also require Lemma 4.3, namely the one for **case** and the one for type abstraction. For the former one, we refer to Section 5 (where the corresponding, slightly more elaborate "inequational" proof case is discussed). For the latter one, consider

$$\frac{\alpha, \Gamma \vdash t : \tau}{\Gamma \vdash (\Lambda\alpha.t) : \forall\alpha.\tau}.$$

To establish

$$([\![\Lambda\alpha.t]\!]^{err}_{\theta_1,\sigma_1}, [\![\Lambda\alpha.t]\!]^{err}_{\theta_2,\sigma_2}) \in \Delta^{err}_{\forall\alpha.\tau,\rho}, \tag{11}$$

we first analyze the values $[\![t]\!]^{err}_{\theta_1[\alpha\mapsto V_{err}],\sigma_1}$ and $[\![t]\!]^{err}_{\theta_2[\alpha\mapsto V_{err}],\sigma_2}$. By the induction hypothesis corresponding to the precondition $\alpha,\Gamma \vdash t : \tau$ they are related by $\Delta^{err}_{\tau,\rho[\alpha\mapsto id_{V_{err}}]}$, which implies

$$T([\![t]\!]^{err}_{\theta_1[\alpha\mapsto V_{err}],\sigma_1}) = T([\![t]\!]^{err}_{\theta_2[\alpha\mapsto V_{err}],\sigma_2})$$

and

$$E([\![t]\!]^{err}_{\theta_1[\alpha\mapsto V_{err}],\sigma_1}) = E([\![t]\!]^{err}_{\theta_2[\alpha\mapsto V_{err}],\sigma_2})$$

by Lemma 4.3. So either

$$[\![t]\!]^{err}_{\theta_1[\alpha\mapsto V_{err}],\sigma_1} = [\![t]\!]^{err}_{\theta_2[\alpha\mapsto V_{err}],\sigma_2} = \mathit{Bad}\ e$$

for some $e \in \mathcal{P}(\mathcal{E}) \cup \{\mathcal{E}^{nt}\}$, or

$$T([\![t]\!]^{err}_{\theta_1[\alpha\mapsto V_{err}],\sigma_1}) = T([\![t]\!]^{err}_{\theta_2[\alpha\mapsto V_{err}],\sigma_2}) = \mathit{Ok}\,.$$

In the first case, we have $[\![\Lambda\alpha.t]\!]^{err}_{\theta_1,\sigma_1} = [\![\Lambda\alpha.t]\!]^{err}_{\theta_2,\sigma_2} = \mathit{Bad}\ e$ by the term semantics, and thus $([\![\Lambda\alpha.t]\!]^{err}_{\theta_1,\sigma_1}, [\![\Lambda\alpha.t]\!]^{err}_{\theta_2,\sigma_2}) \in \Delta^{err}_{\forall\alpha.\tau,\rho}$ by Lemma 4.3. In the second case, (11) follows from the definitions of $[\![\Lambda\alpha.t]\!]^{err}_{\theta_1,\sigma_1}$, $[\![\Lambda\alpha.t]\!]^{err}_{\theta_2,\sigma_2}$, and $\Delta^{err}_{\forall\alpha.\tau,\rho}$ and the fact that for every pair of elcpos D_1, D_2 and $\mathcal{R} \in \mathit{Rel}^{err}(D_1, D_2)$,

$$([\![t]\!]^{err}_{\theta_1[\alpha\mapsto D_1],\sigma_1}, [\![t]\!]^{err}_{\theta_2[\alpha\mapsto D_2],\sigma_2}) \in \Delta^{err}_{\tau,\rho[\alpha\mapsto\mathcal{R}]}\,,$$

which is another instance of the induction hypothesis corresponding to the precondition $\alpha,\Gamma \vdash t : \tau$. All other proof cases proceed like the corresponding ones for Theorem 2.2, up to very minor and obvious changes related to the different kinds of tagging.

Having established Theorem 4.4, we can use it to derive free theorems that hold with respect to the imprecise error semantics. When doing so, we typically want to specialize relations (arising from the quantification in the definition of $\Delta^{err}_{\forall\alpha.\tau,\rho}$) to functions. To this end, the following definition is useful. The notation $\emptyset$ is used for empty mappings from type or term variables to elcpos and values, respectively.

Definition 4.5. Let h be a term with $\vdash h : \tau_1 \to \tau_2$. The *graph* of h, denoted by $\mathcal{G}(h)$, is the relation

$$\{(a,b) \mid [\![h]\!]^{err}_{\emptyset,\emptyset}\ \$\ a = b\} \subseteq [\![\tau_1]\!]^{err}_{\emptyset} \times [\![\tau_2]\!]^{err}_{\emptyset}\,.$$

Note that it is actually a function, as h and a determine b.

Of course, we should restrict attention to such h for which $\mathcal{G}(h)$ fulfills all necessary requirements on relations, i.e., error-strictness, continuity, and error-reflectingness. Error-strictness is easily translated from a restriction on $\mathcal{G}(h)$ to one on h. Continuity is a general property of functions and function application in the underlying semantics. Half of error-reflectingness, in the case of functions, is already given by error-strictness. The other half requires to ensure that nonerroneous arguments are mapped to nonerroneous results. Altogether, we get the following definition and lemma.

Definition 4.6. A term h with $\vdash h : \tau_1 \to \tau_2$ and $[\![h]\!]^{err}_{\emptyset,\emptyset} = Ok\ f$ is

- *error-strict* if $f\ a = a$ for every $a \in V_{err}$, and
- *error-total* if $T(f\ a) = Ok$ for every $a \in [\![\tau_1]\!]^{err}_{\emptyset} \setminus V_{err}$.

An h with $T([\![h]\!]^{err}_{\emptyset,\emptyset}) = Bad$ is neither error-strict nor error-total.

For example, *null* is error-strict and error-total, while *tail* is neither of both. Also, Haskell's standard projection function *fst* is error-strict but not error-total, while (*const* 42) is error-total but not error-strict.

Lemma 4.7. *Let h be a term with $\vdash h : \tau_1 \to \tau_2$. Then $\mathcal{G}(h) \in Rel^{err}$ iff h is error-strict and error-total.*

We will only use the if-direction of this lemma, so we only sketch the proof of that direction, and only the parts related to the treatment of erroneous values:

- Error-strictness of $\mathcal{G}(h)$ follows from error-strictness of h by the definition of \$ from Section 3.
- Error-reflectingness of $\mathcal{G}(h)$ follows from error-strictness and error-totality of h by the definition of \$ and because for every $a \in [\![\tau_1]\!]^{err}_{\emptyset} \setminus V_{err}$, $T(a) = Ok$ and $E(a) = \emptyset$, and for every $b \in [\![\tau_2]\!]^{err}_{\emptyset}$, $T(b) = Ok$ implies $E(b) = \emptyset$.

One further auxiliary lemma we need has to do with a connection between $\mathcal{G}$, the function *map*, and $list^{err}$.

Lemma 4.8. *Let h be a term with $\vdash h : \tau_1 \to \tau_2$. Then $\mathcal{G}(map\ \tau_1\ \tau_2\ h) = list^{err}\ \mathcal{G}(h)$.*

The proof (by coinduction, using the definition of $list^{err}$ via a greatest fixpoint) holds no surprises and is thus omitted here.

We now have everything at hand to derive free theorems. For illustration, we take up the introductory example.

Example 4.9. Let t be a term with $\vdash t : \forall\alpha.(\alpha \to \mathsf{Bool}) \to [\alpha] \to [\alpha]$. Of course, this first requires to extend the calculus and proofs by integrating a Boolean type and associated term-formers with appropriate typing rules, semantics, and so on. Since the details are entirely straightforward, we omit them here. By Theorem 4.4 we have

$$([\![t]\!]^{err}_{\emptyset,\emptyset}, [\![t]\!]^{err}_{\emptyset,\emptyset}) \in \Delta^{err}_{\forall\alpha.(\alpha\to\mathsf{Bool})\to[\alpha]\to[\alpha],\emptyset}\ ,$$

where $\emptyset$ is now also used to denote the empty mapping from type variables to relations. Using the definition of Δ^{err}, we obtain that for every choice of elcpos

D_1, D_2, relation $\mathcal{R} \in \mathit{Rel}^{err}(D_1, D_2)$, values p_1, p_2 with $(p_1, p_2) \in \Delta^{err}_{\alpha \to \mathsf{Bool}, [\alpha \mapsto \mathcal{R}]}$, and l_1, l_2 with $(l_1, l_2) \in \mathit{list}^{err}\ \mathcal{R}$,

$$(\llbracket t \rrbracket^{err}_{\emptyset,\emptyset}\ \$\$\ D_1\ \$\ p_1\ \$\ l_1, \llbracket t \rrbracket^{err}_{\emptyset,\emptyset}\ \$\$\ D_2\ \$\ p_2\ \$\ l_2) \in \mathit{list}^{err}\ \mathcal{R}\,.$$

Let h be a term with $\vdash h : \tau_1 \to \tau_2$ that is error-strict and error-total. By Lemma 4.7 we have $\mathcal{G}(h) \in \mathit{Rel}^{err}(\llbracket \tau_1 \rrbracket^{err}_{\emptyset}, \llbracket \tau_2 \rrbracket^{err}_{\emptyset})$, so we can use it to instantiate $\mathcal{R}$ above. By Lemma 4.8 we then have $\mathit{list}^{err}\ \mathcal{R} = \mathcal{G}(\mathit{map}\ \tau_1\ \tau_2\ h)$, and thus for every choice of values p_1, p_2 with $(p_1, p_2) \in \Delta^{err}_{\alpha \to \mathsf{Bool}, [\alpha \mapsto \mathcal{G}(h)]}$ and $l_1 \in \llbracket [\tau_1] \rrbracket^{err}_{\emptyset}$,

$$\begin{gathered}\llbracket \mathit{map}\ \tau_1\ \tau_2\ h \rrbracket^{err}_{\emptyset,\emptyset}\ \$\,(\llbracket t \rrbracket^{err}_{\emptyset,\emptyset}\ \$\$\,\llbracket \tau_1 \rrbracket^{err}_{\emptyset}\ \$\ p_1\ \$\ l_1) \\ = \\ \llbracket t \rrbracket^{err}_{\emptyset,\emptyset}\ \$\$\,\llbracket \tau_2 \rrbracket^{err}_{\emptyset}\ \$\ p_2\ \$\,(\llbracket \mathit{map}\ \tau_1\ \tau_2\ h \rrbracket^{err}_{\emptyset,\emptyset}\ \$\ l_1)\,.\end{gathered}$$

The condition on p_1 and p_2 unfolds to $T(p_1) = T(p_2)$ and

$$\forall (a, b) \in \mathcal{G}(h).\ (p_1\ \$\ a, p_2\ \$\ b) \in \mathit{id}_{\mathit{lift}_{err}\{\mathsf{False},\mathsf{True}\}}\,,$$

i.e., to $T(p_1) = T(p_2)$ and for every $a \in \llbracket \tau_1 \rrbracket^{err}_{\emptyset}$,

$$p_1\ \$\ a = p_2\ \$\,(\llbracket h \rrbracket^{err}_{\emptyset,\emptyset}\ \$\ a)\,.$$

The latter is easy to satisfy by choosing $p_1 = \llbracket \lambda x : \tau_1.p\ (h\ x) \rrbracket^{err}_{\emptyset,\emptyset}$ and $p_2 = \llbracket p \rrbracket^{err}_{\emptyset,\emptyset}$ for some p with $\vdash p : \tau_2 \to \mathsf{Bool}$, but we need to take note of the requirement that $T(\llbracket \lambda x : \tau_1.p\ (h\ x) \rrbracket^{err}_{\emptyset,\emptyset}) = T(\llbracket p \rrbracket^{err}_{\emptyset,\emptyset})$, i.e., $T(\llbracket p \rrbracket^{err}_{\emptyset,\emptyset}) = \mathit{Ok}$.

Altogether, we now have for every $l_1 \in \llbracket [\tau_1] \rrbracket^{err}_{\emptyset}$,

$$\begin{gathered}\llbracket \mathit{map}\ \tau_1\ \tau_2\ h \rrbracket^{err}_{\emptyset,\emptyset}\ \$\,(\llbracket t \rrbracket^{err}_{\emptyset,\emptyset}\ \$\$\,\llbracket \tau_1 \rrbracket^{err}_{\emptyset}\ \$\,\llbracket \lambda x : \tau_1.p\ (h\ x) \rrbracket^{err}_{\emptyset,\emptyset}\ \$\ l_1) \\ = \\ \llbracket t \rrbracket^{err}_{\emptyset,\emptyset}\ \$\$\,\llbracket \tau_2 \rrbracket^{err}_{\emptyset}\ \$\,\llbracket p \rrbracket^{err}_{\emptyset,\emptyset}\ \$\,(\llbracket \mathit{map}\ \tau_1\ \tau_2\ h \rrbracket^{err}_{\emptyset,\emptyset}\ \$\ l_1)\,,\end{gathered}$$

and thus for every term l with $\vdash l : [\tau_1]$,

$$\llbracket \mathit{map}\ \tau_1\ \tau_2\ h\ (t\ \tau_1\ (\lambda x : \tau_1.p\ (h\ x))\ l) \rrbracket^{err}_{\emptyset,\emptyset} = \llbracket t\ \tau_2\ p\ (\mathit{map}\ \tau_1\ \tau_2\ h\ l) \rrbracket^{err}_{\emptyset,\emptyset}$$

under the conditions that h is error-strict and error-total and that $T(\llbracket p \rrbracket^{err}_{\emptyset,\emptyset}) = \mathit{Ok}$. This is the promised equivalence repair for (1).

5 Going Inequational

As mentioned in the introduction, Johann and Voigtländer (2004) consider an inequational version of parametricity to get (weaker) free theorems under weaker preconditions. Their key step is that of moving to an asymmetric logical relation by requiring all relations to be closed under left-composition with the approximation order.

Definition 5.1. Let $\mathcal{R}$ be a relation between elcpos. Then $\sqsubseteq\,;\,\mathcal{R}$ denotes the relation $\{(a,b) \mid \exists c.\ a \sqsubseteq c,\ (c,b) \in \mathcal{R}\}$ between the same elcpos, and $\mathcal{R}$ is *left-closed* if $\sqsubseteq\,;\,\mathcal{R} = \mathcal{R}$.

For base types like Int, one simply chooses the corresponding approximation order as relational interpretation. For algebraic datatypes like lists, explicit left-composition of the "structural lifting" (as obtained via *list* or list^{err}) with $\sqsubseteq$ suffices. To determine what to do about function and polymorphic types, and in particular what further restrictions on relations are necessary, the best strategy is again to step through the proof cases of a desired inequational variant of Theorem 4.4.

For the case of **error**, we arrive via reasoning very much like at the beginning of Section 4 at a point where we have to show that for every $a, b \in \mathit{lift}_{err}\,\{\ldots, -2, -1, 0, 1, 2, \ldots\}$ with $a \sqsubseteq b$, the values (5) and

$$\begin{cases} \mathit{Bad}\ \{\mathsf{ErrorCall}\ n\} & \text{if } b = \mathit{Ok}\ n \\ \mathit{Bad}\ e & \text{if } b = \mathit{Bad}\ e \end{cases} \tag{12}$$

are related. The only difference to earlier, namely that a and b might be different, comes from the change in interpreting Int by the approximation order rather than the identity relation. It is easy to see that requiring error-strictness and left-closedness of the relation in question already suffices to guarantee that the two values in question are indeed contained in it.[4]

Regarding error-reflectingness, we hope for a weakening of the restriction thanks to our move to an asymmetric setting, just as Johann and Voigtländer (2004) broke the symmetry to avoid full bottom-reflectingness. To determine just the right weakening, we reconsider the proof case for the strict-let construct. Again similarly to Section 4, we have to investigate the relation between values (6) and (7). If the first branch is chosen in both case distinctions, then the proof can proceed as before. If the second branch is chosen in both case distinctions, then we would also proceed as in Section 4, except that thanks to left-closedness we are now satisfied even with

$$e_1 \cup E(\llbracket t_2 \rrbracket^{err}_{\theta_1,\sigma_1[x\mapsto \mathit{Bad}\,\emptyset]}) \supseteq e_2 \cup E(\llbracket t_2 \rrbracket^{err}_{\theta_2,\sigma_2[x\mapsto \mathit{Bad}\,\emptyset]})$$

rather than expecting their equality. As a consequence, we could weaken the "$E(a) = E(b)$"-condition in Definition 4.2 to "$E(a) \supseteq E(b)$". It remains to discuss how to deal with the potential for choosing different branches in the two case distinctions. Before, we completely prevented this via the "$T(a) = T(b)$"-condition in Definition 4.2. Now, breaking the symmetry, we want to allow one of the two over-cross situations, but not the other. In particular, due to the nature of left-closedness, we should prevent the situation where the first branch is chosen in the first case distinction, but the second branch in the second one. Given the known

[4] The only possible cases for $a, b \in \mathit{lift}_{err}\,\{\ldots, -2, -1, 0, 1, 2, \ldots\}$ with $a \sqsubseteq b$ are as follows: (i) $a = b = \mathit{Ok}\ n$, (ii) $a = \bot$ and $b = \mathit{Ok}\ n$, and (iii) $a = \mathit{Bad}\ e$ and $b = \mathit{Bad}\ e'$ with $e \supseteq e'$. In each of these cases, (5) and (12) give erroneous values in an approximation relationship.

relatedness of $[\![t_1]\!]^{err}_{\theta_1,\sigma_1}$ and $[\![t_1]\!]^{err}_{\theta_2,\sigma_2}$ by an induction hypothesis akin to (8) in Section 4, this can be achieved by at least requiring "$T(a) \preccurlyeq T(b)$", where $\preccurlyeq$ is the total order corresponding to the strict total order $\mathit{Bad} \prec \mathit{Ok}$. Finally, for the other over-cross situation $[\![t_1]\!]^{err}_{\theta_1,\sigma_1} = \mathit{Bad}\ e_1$ and $[\![t_1]\!]^{err}_{\theta_2,\sigma_2} = \mathit{Ok}\ v_2$, we have to establish the relatedness of $\mathit{Bad}\ (e_1 \cup E([\![t_2]\!]^{err}_{\theta_1,\sigma_1[x\mapsto \mathit{Bad}\,\emptyset]}))$ and $[\![t_2]\!]^{err}_{\theta_2,\sigma_2[x\mapsto \mathit{Ok}\,v_2]}$. If $e_1 = \mathcal{E}^{nt}$, then it follows from

- the assumed left-closedness of every relation,
- the fact that then $\mathit{Bad}\ (e_1 \cup E([\![t_2]\!]^{err}_{\theta_1,\sigma_1[x\mapsto \mathit{Bad}\,\emptyset]}))$ equals $\bot$ and thus approximates $[\![t_2]\!]^{err}_{\theta_1,\sigma_1[x\mapsto \mathit{Bad}\,e_1]}$, and
- the relatedness of $[\![t_2]\!]^{err}_{\theta_1,\sigma_1[x\mapsto \mathit{Bad}\,e_1]}$ and $[\![t_2]\!]^{err}_{\theta_2,\sigma_2[x\mapsto \mathit{Ok}\,v_2]}$ (by induction hypotheses akin to (8) and (9)).

If instead $e_1 \in \mathcal{P}(\mathcal{E})$, then there is no general way to establish the desired relatedness, so we have to rule out this particular case as well. Overall, we arrive at the following definition.

Definition 5.2. A relation $\mathcal{R}$ is *error-approximating* if $(a, b) \in \mathcal{R}$ implies that $T(a) \preccurlyeq T(b)$, $E(a) \supseteq E(b)$, and $a \in V_{err} \setminus \{\bot\} \Rightarrow T(b) = \mathit{Bad}$.

Note that an equivalent definition would be to require for every $(a, b) \in \mathcal{R}$ that $T(a) \preccurlyeq T(b)$ and $T(a) = \mathit{Bad} \Rightarrow a \sqsubseteq b$, provided we allow ourselves the slight abuse of notation to use $\sqsubseteq$ between an erroneous value from one elcpo and an arbitrary value from another elcpo.

From now on, we use $\mathit{Rel}^{\sqsubseteq}$ to denote error-strict, continuous, left-closed, and error-approximating relations. The task still left towards establishing inequational parametricity for the imprecise error semantics is to adapt (possibly weaken) the "$T(f) = T(g)$"-condition from the definition of $\Delta^{err}_{\tau_1\to\tau_2,\rho}$ in Figure 9, so as to preserve this new set of restrictions. Clearly, due to the left-closedness requirement, we cannot longer insist on equality of the tags of f and g in all cases, because if any pair of functions (f, g) is in the relation, then $(\bot, g)$ must be so as well, regardless of $T(g)$, and we do not necessarily have $T(\bot) = T(g)$. On the other hand, requiring just $T(f) \preccurlyeq T(g)$ would not be enough, as we could then not guarantee that the resulting relational interpretation of a function type always satisfies the last condition from Definition 5.2. It turns out that the right decision is to still require $T(f) = T(g)$, but only when $f \neq \bot$.

Overall, our inequational logical relation is defined as in Figure 10. The proof of the following lemma is sufficiently similar to the one of Lemma 4.3 that we do not elaborate further on it.

Lemma 5.3. *If ρ maps only to relations in $\mathit{Rel}^{\sqsubseteq}$, then $\Delta^{\sqsubseteq}_{\tau,\rho} \in \mathit{Rel}^{\sqsubseteq}$ as well.*

$$
\begin{array}{ll}
\Delta^{\sqsubseteq}_{\alpha,\rho} & = \rho(\alpha) \\
\Delta^{\sqsubseteq}_{\mathsf{Int},\rho} & = \sqsubseteq_{\mathit{lift}_{err}\{...,-2,-1,0,1,2,...\}} \\
\Delta^{\sqsubseteq}_{[\tau],\rho} & = \sqsubseteq ; (\mathit{list}^{err}\, \Delta^{\sqsubseteq}_{\tau,\rho}) \\
\Delta^{\sqsubseteq}_{\tau_1\to\tau_2,\rho} & = \{(f,g) \mid f \neq \bot \Rightarrow T(f) = T(g),\, \forall (a,b) \in \Delta^{\sqsubseteq}_{\tau_1,\rho}.\ (f \ \$\ a, g\ \$\ b) \in \Delta^{\sqsubseteq}_{\tau_2,\rho}\} \\
\Delta^{\sqsubseteq}_{\forall\alpha.\tau,\rho} & = \{(u,v) \mid \forall D_1, D_2 \text{ elcpos}, \mathcal{R} \in \mathit{Rel}^{\sqsubseteq}(D_1,D_2). \\
& \qquad\qquad (u\ \$\$\ D_1, v\ \$\$\ D_2) \in \Delta^{\sqsubseteq}_{\tau,\rho[\alpha\mapsto\mathcal{R}]}\}
\end{array}
$$

Figure 10: Inequational Logical Relation.

Finally, the following parametricity theorem holds.

Theorem 5.4. *If $\Gamma \vdash t : \tau$, then for every θ_1, θ_2, ρ, σ_1, and σ_2 such that*

- *for every α occurring in Γ, $\rho(\alpha) \in \mathit{Rel}^{\sqsubseteq}(\theta_1(\alpha), \theta_2(\alpha))$, and*
- *for every $x : \tau'$ occurring in Γ, $(\sigma_1(x), \sigma_2(x)) \in \Delta^{\sqsubseteq}_{\tau',\rho}$,*

we have $([\![t]\!]^{err}_{\theta_1,\sigma_1}, [\![t]\!]^{err}_{\theta_2,\sigma_2}) \in \Delta^{\sqsubseteq}_{\tau,\rho}$.

Most of the proof cases not already discussed (earlier in this section, or in the previous section for the slightly simpler equational setting) are quite straightforward. We choose to elaborate on the one for **case**, mainly because the corresponding term semantics belongs to the more intricate aspects of the imprecise error semantics. Recall the following typing rule:

$$
\frac{\Gamma \vdash t : [\tau_1] \qquad \Gamma \vdash t_1 : \tau_2 \qquad \Gamma, x_1 : \tau_1, x_2 : [\tau_1] \vdash t_2 : \tau_2}{\Gamma \vdash (\mathbf{case}\ t\ \mathbf{of}\ \{[\,] \to t_1\, ;\ x_1 : x_2 \to t_2\}) : \tau_2} .
$$

To establish

$$
\begin{array}{l}
([\![\mathbf{case}\ t\ \mathbf{of}\ \{[\,] \to t_1\, ;\ x_1 : x_2 \to t_2\}]\!]^{err}_{\theta_1,\sigma_1}, \\
\ [\![\mathbf{case}\ t\ \mathbf{of}\ \{[\,] \to t_1\, ;\ x_1 : x_2 \to t_2\}]\!]^{err}_{\theta_2,\sigma_2}) \in \Delta^{\sqsubseteq}_{\tau_2,\rho} ,
\end{array}
$$

we analyze the values $[\![t]\!]^{err}_{\theta_1,\sigma_1}$ and $[\![t]\!]^{err}_{\theta_2,\sigma_2}$. By induction hypothesis they are related by $\Delta^{\sqsubseteq}_{[\tau_1],\rho} = \sqsubseteq ; (\mathit{list}^{err}\, \Delta^{\sqsubseteq}_{\tau_1,\rho})$, so we only have to consider the following five cases:

- $[\![t]\!]^{err}_{\theta_1,\sigma_1} = \mathit{Ok}\ [\,]$ and $[\![t]\!]^{err}_{\theta_2,\sigma_2} = \mathit{Ok}\ [\,]$, in which case the desired statement is equivalent to the induction hypothesis

$$
([\![t_1]\!]^{err}_{\theta_1,\sigma_1}, [\![t_1]\!]^{err}_{\theta_2,\sigma_2}) \in \Delta^{\sqsubseteq}_{\tau_2,\rho} , \tag{13}
$$

- $[\![t]\!]^{err}_{\theta_1,\sigma_1} = \bot$ and $[\![t]\!]^{err}_{\theta_2,\sigma_2} = \mathit{Ok}\ [\,]$, in which case we have to show $(\bot, [\![t_1]\!]^{err}_{\theta_2,\sigma_2}) \in \Delta^{\sqsubseteq}_{\tau_2,\rho}$, which follows from (13) by left-closedness of $\Delta^{\sqsubseteq}_{\tau_2,\rho}$ (cf. Lemma 5.3),

- $[\![t]\!]^{err}_{\theta_1,\sigma_1} = Ok\ (a : b)$ and $[\![t]\!]^{err}_{\theta_2,\sigma_2} = Ok\ (c : d)$ with $(a,c) \in \sqsubseteq ; \Delta^{\sqsubseteq}_{\tau_1,\rho}$ and $(b,d) \in \sqsubseteq ; (list^{err}\ \Delta^{\sqsubseteq}_{\tau_1,\rho}) = \Delta^{\sqsubseteq}_{[\tau_1],\rho}$, in which case the desired statement follows by left-closedness of $\Delta^{\sqsubseteq}_{\tau_1,\rho}$ (cf. Lemma 5.3 again) from the induction hypothesis that for every $(a,c) \in \Delta^{\sqsubseteq}_{\tau_1,\rho}$ and $(b,d) \in \Delta^{\sqsubseteq}_{[\tau_1],\rho}$,
$$([\![t_2]\!]^{err}_{\theta_1,\sigma_1[x_1\mapsto a,\, x_2\mapsto b]}, [\![t_2]\!]^{err}_{\theta_2,\sigma_2[x_1\mapsto c,\, x_2\mapsto d]}) \in \Delta^{\sqsubseteq}_{\tau_2,\rho}\,, \qquad (14)$$
- $[\![t]\!]^{err}_{\theta_1,\sigma_1} = \bot$, $[\![t]\!]^{err}_{\theta_2,\sigma_2} = Ok\ (c : d)$, and there exist a and b with $(a,c) \in \Delta^{\sqsubseteq}_{\tau_1,\rho}$ and $(b,d) \in list^{err}\ \Delta^{\sqsubseteq}_{\tau_1,\rho} \subseteq \Delta^{\sqsubseteq}_{[\tau_1],\rho}$, in which case we have to show
$$(\bot, [\![t_2]\!]^{err}_{\theta_2,\sigma_2[x_1\mapsto c,\, x_2\mapsto d]}) \in \Delta^{\sqsubseteq}_{\tau_2,\rho}\,,$$
which follows from (14) by left-closedness of $\Delta^{\sqsubseteq}_{\tau_2,\rho}$, and
- $[\![t]\!]^{err}_{\theta_1,\sigma_1} = Bad\ e_1$ and $[\![t]\!]^{err}_{\theta_2,\sigma_2} = Bad\ e_2$ with $e_1 \supseteq e_2$, in which case the desired statement follows by error-strictness and left-closedness of $\Delta^{\sqsubseteq}_{\tau_2,\rho}$ from
$$\begin{gathered} Bad\ (e_1 \cup E([\![t_1]\!]^{err}_{\theta_1,\sigma_1}) \cup E([\![t_2]\!]^{err}_{\theta_1,\sigma_1[x_1\mapsto Bad\,\emptyset,\, x_2\mapsto Bad\,\emptyset]})) \\ \sqsubseteq \\ Bad\ (e_2 \cup E([\![t_1]\!]^{err}_{\theta_2,\sigma_2}) \cup E([\![t_2]\!]^{err}_{\theta_2,\sigma_2[x_1\mapsto Bad\,\emptyset,\, x_2\mapsto Bad\,\emptyset]}))\,, \end{gathered}$$
which follows from $e_1 \supseteq e_2$, (13), (14), error-strictness of $\Delta^{\sqsubseteq}_{\tau_1,\rho}$ and $\Delta^{\sqsubseteq}_{[\tau_1],\rho}$, and error-approximation of $\Delta^{\sqsubseteq}_{\tau_2,\rho}$.

To use Theorem 5.4 for deriving inequational free theorems, we again need a way to bring functions into play where originally general relations are quantified. Due to the asymmetry we have introduced, the graph notion from Definition 4.5 is replaced by two variants now.

Definition 5.5. Let h be a term with $\vdash h : \tau_1 \to \tau_2$. The *right-graph* of h, denoted by $\mathcal{G}_{right}(h)$, is the relation
$$\{(a,b) \mid [\![h]\!]^{err}_{\emptyset,\emptyset}\ \$\ a \sqsubseteq b\} \subseteq [\![\tau_1]\!]^{err}_{\emptyset} \times [\![\tau_2]\!]^{err}_{\emptyset}\,,$$
while the *left-graph* of h, denoted by $\mathcal{G}_{left}(h)$, is the relation
$$\{(a,b) \mid a \sqsubseteq [\![h]\!]^{err}_{\emptyset,\emptyset}\ \$\ b\} \subseteq [\![\tau_2]\!]^{err}_{\emptyset} \times [\![\tau_1]\!]^{err}_{\emptyset}\,.$$

Note that both $\mathcal{G}_{right}(h)$ and $\mathcal{G}_{left}(h)$ are left-closed by definition (in the former case due to monotonicity of $(\lambda a.[\![h]\!]^{err}_{\emptyset,\emptyset}\ \$\ a)$). Of course, we also need them to fulfill the other relevant restrictions. For right-graphs, we obtain the following lemma.

Lemma 5.6. *Let h be a term with $\vdash h : \tau_1 \to \tau_2$. Then $\mathcal{G}_{right}(h) \in Rel^{\sqsubseteq}$ iff h is error-strict and error-total.*

Again we will only use the if-direction, the proof of which is similar as in the case of Lemma 4.7.

For left-graphs, it turns out that a slight weakening of error-totality suffices.

Definition 5.7. A term h with $\vdash h : \tau_1 \to \tau_2$ is *error-pretotal* if $[\![h]\!]^{err}_{\emptyset,\emptyset} = Ok\ f$ and for every $a \in [\![\tau_1]\!]^{err}_{\emptyset} \setminus V_{err}$, $f\ a = \bot$ or $T(f\ a) = Ok$.

Lemma 5.8. *Let h be a term with $\vdash h : \tau_1 \to \tau_2$. Then $\mathcal{G}_{left}(h) \in Rel^{\sqsubseteq}$ iff h is error-strict and error-pretotal.*

We only show here how error-approximation of $\mathcal{G}_{left}(h)$ follows from error-strictness and error-pretotality of h. Let $[\![h]\!]^{err}_{\emptyset,\emptyset} = Ok\ f$ and $(a, b) \in \mathcal{G}_{left}(h)$, i.e., $a \sqsubseteq f\ b$. Then we have:

- $T(a) \preccurlyeq T(b)$, because $T(b) = Bad$ implies $f\ b = b$ by error-strictness, thus $a \sqsubseteq b$, and hence $T(a) = Bad$ as well,
- $E(a) \supseteq E(b)$, because either $T(b) = Ok$, in which case $E(b) = \emptyset$, or $T(b) = Bad$, in which case as above $a \sqsubseteq b$, and
- if $a \in V_{err} \setminus \{\bot\}$, then $T(b) = Bad$, because otherwise, by error-pretotality, $f\ b = \bot$ or $T(f\ b) = Ok$, both of which would contradict $a \in V_{err} \setminus \{\bot\}$ by $a \sqsubseteq f\ b$.

We also get two statements in the spirit of Lemma 4.8, with routine proofs.

Lemma 5.9. *Let h be a term with $\vdash h : \tau_1 \to \tau_2$. Then $\mathcal{G}_{left}(map\ \tau_1\ \tau_2\ h) = \sqsubseteq ; (list^{err}\ \mathcal{G}_{left}(h))$. Moreover, if h is error-strict, then $\mathcal{G}_{right}(map\ \tau_1\ \tau_2\ h) = \sqsubseteq ; (list^{err}\ \mathcal{G}_{right}(h))$.*

To see the inequational setup in action, we reconsider the introductory example, respectively, Example 4.9.

Example 5.10. By Theorem 5.4 we have

$$([\![t]\!]^{err}_{\emptyset,\emptyset}, [\![t]\!]^{err}_{\emptyset,\emptyset}) \in \Delta^{\sqsubseteq}_{\forall\alpha.(\alpha\to\mathsf{Bool})\to[\alpha]\to[\alpha],\emptyset}\,.$$

Using the definition of $\Delta^{\sqsubseteq}$, we obtain that for every choice of elcpos D_1, D_2, relation $\mathcal{R} \in Rel^{\sqsubseteq}(D_1, D_2)$, values p_1, p_2 with $(p_1, p_2) \in \Delta^{\sqsubseteq}_{\alpha\to\mathsf{Bool},[\alpha\mapsto\mathcal{R}]}$, and l_1, l_2 with $(l_1, l_2) \in \sqsubseteq ; (list^{err}\ \mathcal{R})$,

$$([\![t]\!]^{err}_{\emptyset,\emptyset}\ \$\$\ D_1\ \$\ p_1\ \$\ l_1, [\![t]\!]^{err}_{\emptyset,\emptyset}\ \$\$\ D_2\ \$\ p_2\ \$\ l_2) \in \sqsubseteq ; (list^{err}\ \mathcal{R})\,.$$

Let h be a term with $\vdash h : \tau_1 \to \tau_2$. If h is error-strict and error-total, then by Lemma 5.6, $\mathcal{G}_{right}(h) \in Rel^{\sqsubseteq}([\![\tau_1]\!]^{err}_{\emptyset}, [\![\tau_2]\!]^{err}_{\emptyset})$, so we can use $\mathcal{G}_{right}(h)$ to instantiate

$\mathcal{R}$ above. Continuing with Lemma 5.9 instead of Lemma 4.8, we can then proceed essentially as in Example 4.9 and eventually conclude

$$[\![map\ \tau_1\ \tau_2\ h\ (t\ \tau_1\ (\lambda x:\tau_1.p\ (h\ x))\ l)]\!]^{err}_{\emptyset,\emptyset} \sqsubseteq [\![t\ \tau_2\ p\ (map\ \tau_1\ \tau_2\ h\ l)]\!]^{err}_{\emptyset,\emptyset}$$

under exactly the same conditions, in particular also $T([\![p]\!]^{err}_{\emptyset,\emptyset}) = Ok$, under which we even showed equivalence there.

More interestingly, though, we can also use Lemma 5.8 to obtain an appropriate instantiation for $\mathcal{R}$ above. So let h be error-strict and error-pretotal. Then we have $\mathcal{G}_{left}(h) \in Rel^{\sqsubseteq}([\![\tau_2]\!]^{err}_{\emptyset}, [\![\tau_1]\!]^{err}_{\emptyset})$, and by Lemma 5.9,

$$\sqsubseteq ; (list^{err}\ \mathcal{G}_{left}(h)) = \mathcal{G}_{left}(map\ \tau_1\ \tau_2\ h)\,,$$

and thus for every choice of values $(p_1, p_2) \in \Delta^{\sqsubseteq}_{\alpha\to\mathsf{Bool},[\alpha\mapsto\mathcal{G}_{left}(h)]}$ and $l_2 \in [\![[\tau_1]]\!]^{err}_{\emptyset}$,

$$\begin{gathered}[\![t]\!]^{err}_{\emptyset,\emptyset}\ \$\$\ [\![\tau_2]\!]^{err}_{\emptyset}\ \$\ p_1\ \$\ ([\![map\ \tau_1\ \tau_2\ h]\!]^{err}_{\emptyset,\emptyset}\ \$\ l_2)\\ \sqsubseteq \\ [\![map\ \tau_1\ \tau_2\ h]\!]^{err}_{\emptyset,\emptyset}\ \$\ ([\![t]\!]^{err}_{\emptyset,\emptyset}\ \$\$\ [\![\tau_1]\!]^{err}_{\emptyset}\ \$\ p_2\ \$\ l_2)\,.\end{gathered}$$

The condition on p_1 and p_2 unfolds to $p_1 \neq \bot \Rightarrow T(p_1) = T(p_2)$ and

$$\forall (a, b) \in \mathcal{G}_{left}(h).\ (p_1\ \$\ a, p_2\ \$\ b) \in \sqsubseteq_{lift_{err}\{\mathsf{False},\mathsf{True}\}}\,,$$

the latter being equivalent to, for every $a \in [\![\tau_2]\!]^{err}_{\emptyset}$ and $b \in [\![\tau_1]\!]^{err}_{\emptyset}$,

$$a \sqsubseteq [\![h]\!]^{err}_{\emptyset,\emptyset}\ \$\ b \Rightarrow p_1\ \$\ a \sqsubseteq p_2\ \$\ b\,.$$

This is easy to satisfy by choosing $p_1 = [\![p]\!]^{err}_{\emptyset,\emptyset}$ and $p_2 = [\![\lambda x:\tau_1.p\ (h\ x)]\!]^{err}_{\emptyset,\emptyset}$ for some p with $\vdash p : \tau_2 \to \mathsf{Bool}$, but we need to take note of the requirement that $[\![p]\!]^{err}_{\emptyset,\emptyset} \neq \bot \Rightarrow T([\![p]\!]^{err}_{\emptyset,\emptyset}) = T([\![\lambda x:\tau_1.p\ (h\ x)]\!]^{err}_{\emptyset,\emptyset})$, i.e., $[\![p]\!]^{err}_{\emptyset,\emptyset} \neq \bot \Rightarrow T([\![p]\!]^{err}_{\emptyset,\emptyset}) = Ok$.

Altogether, we get for every term l with $\vdash l : [\tau_1]$,

$$[\![t\ \tau_2\ p\ (map\ \tau_1\ \tau_2\ h\ l)]\!]^{err}_{\emptyset,\emptyset} \sqsubseteq [\![map\ \tau_1\ \tau_2\ h\ (t\ \tau_1\ (\lambda x:\tau_1.p\ (h\ x))\ l)]\!]^{err}_{\emptyset,\emptyset}$$

under the conditions that h is error-strict and error-pretotal, and $[\![p]\!]^{err}_{\emptyset,\emptyset} = \bot$ or $T([\![p]\!]^{err}_{\emptyset,\emptyset}) = Ok$.

A simple instantiation differentiating this from the purely equational result in Example 4.9 is as follows. Let

$$\begin{aligned} id &= \Lambda\alpha.\lambda x:\alpha.x\\ t &= \Lambda\alpha.\lambda f:\alpha\to\mathsf{Bool}.\mathbf{let!}\ x = f\ \mathbf{in}\ id\ [\alpha]\\ \tau_1 &= \tau_2 = \mathsf{Int}\\ p &= \mathbf{fix}\ (\mathsf{Int}\to\mathsf{Bool})\ (id\ (\mathsf{Int}\to\mathsf{Bool}))\\ h &= id\ \mathsf{Int}\\ l &= [\,]_{\mathsf{Int}}\,. \end{aligned}$$

Then we have $[\![t\ \tau_2\ p\ (map\ \tau_1\ \tau_2\ h\ l)]\!]^{err}_{\emptyset,\emptyset} = \bot$ but on the other hand

$$[\![map\ \tau_1\ \tau_2\ h\ (t\ \tau_1\ (\lambda x:\tau_1.p\ (h\ x))\ l)]\!]^{err}_{\emptyset,\emptyset} = Ok\ [\,]\,.$$

$$
\begin{aligned}
\Delta^{\underset{\sim}{\sqsubset}}_{\alpha,\rho} &= \rho(\alpha)\\
\Delta^{\underset{\sim}{\sqsubset}}_{\mathsf{Int},\rho} &= \underset{\sim}{\sqsubset}_{\mathit{lift}_{err}\{\ldots,-2,-1,0,1,2,\ldots\}}\\
\Delta^{\underset{\sim}{\sqsubset}}_{[\tau],\rho} &= \underset{\sim}{\sqsubset}\ ;(\mathit{list}^{err}\ \Delta^{\underset{\sim}{\sqsubset}}_{\tau,\rho})\\
\Delta^{\underset{\sim}{\sqsubset}}_{\tau_1\to\tau_2,\rho} &= \{(f,g) \mid T(f)=T(g),\, \forall (a,b)\in \Delta^{\underset{\sim}{\sqsubset}}_{\tau_1,\rho}.\ (f\ \$\ a, g\ \$\ b)\in \Delta^{\underset{\sim}{\sqsubset}}_{\tau_2,\rho}\}\\
\Delta^{\underset{\sim}{\sqsubset}}_{\forall\alpha.\tau,\rho} &= \{(u,v) \mid \forall D_1, D_2 \text{ elcpos}, \mathcal{R}\in \mathit{Rel}^{\underset{\sim}{\sqsubset}}(D_1,D_2).\\
&\qquad\qquad (u\ \$\$\ D_1, v\ \$\$\ D_2)\in \Delta^{\underset{\sim}{\sqsubset}}_{\tau,\rho[\alpha\mapsto\mathcal{R}]}\}
\end{aligned}
$$

Figure 11: Logical Relation for Refinement.

6 Treating Refinement

Instead of semantic equivalence or approximation, Moran et al. (1999) consider a *refinement* order. The motivation is to be more liberal than equivalence, by allowing to relate two erroneous values the second of which represents a smaller error set than the first of which does, but at the same time less liberal than approximation, by not allowing to relate $\perp$ to a nonerroneous value. Of course, this basic idea extends to complex values, e.g., the singleton list values $\mathit{Ok}\ (\perp : (\mathit{Ok}\ []))$ and $\mathit{Ok}\ ((\mathit{Ok}\ 42) : (\mathit{Ok}\ []))$ are not related. The refinement order $\underset{\sim}{\sqsubset}$ is simply defined like $\sqsubseteq$, except that when lifting via lift_{err}, we do not put $\perp$ in relation with any $\mathit{Ok}\ s$.

When aiming for a refinement version of relational parametricity, essentially all we need to do is to replace $\sqsubseteq$ by $\underset{\sim}{\sqsubset}$ throughout the development in the previous section (but *not* in the term semantics for **fix** in Figure 8). In particular, Definition 5.1 is modified in the obvious way to yield a variant of left-closedness for $\underset{\sim}{\sqsubset}$. To adapt Definition 5.2, we consider the remark given below it. Replacing $\sqsubseteq$ by $\underset{\sim}{\sqsubset}$ there, we are led to require that for every $(a,b)\in\mathcal{R}$, $T(a)\preccurlyeq T(b)$ and $T(a)=\mathit{Bad} \Rightarrow a \underset{\sim}{\sqsubset} b$. By the definition of $\underset{\sim}{\sqsubset}$, this can be simplified as follows.

Definition 6.1. A relation $\mathcal{R}$ is *error-refining* if $(a,b)\in\mathcal{R}$ implies that $T(a)=T(b)$ and $E(a)\supseteq E(b)$.

We use $\mathit{Rel}^{\underset{\sim}{\sqsubset}}$ to denote error-strict, continuous, $\underset{\sim}{\sqsubset}$-left-closed, and error-refining relations. Figure 11 should hold few surprises now. The only subtlety to note is that the definition for the function type case corresponds to the one from Figure 9 rather than to the one from Figure 10 as for all others. The reason is precisely the move from error-approximation to error-refinement.

Such mixing of cases from the equational setting of Section 4 and the inequational one of Section 5 extends also to the proof of the parametricity theorem. With the above definitions, a preservation lemma in the style of Lemma 5.3 (or Lemmas 2.1, 4.3) is straightforwardly established. Then we get the following theorem.

Theorem 6.2. *If $\Gamma \vdash t : \tau$, then for every θ_1, θ_2, ρ, σ_1, and σ_2 such that*

- *for every α occurring in Γ, $\rho(\alpha) \in Rel^{\sqsubseteq}(\theta_1(\alpha), \theta_2(\alpha))$, and*
- *for every $x : \tau'$ occurring in Γ, $(\sigma_1(x), \sigma_2(x)) \in \Delta^{\sqsubseteq}_{\tau',\rho}$,*

we have $([\![t]\!]^{err}_{\theta_1,\sigma_1}, [\![t]\!]^{err}_{\theta_2,\sigma_2}) \in \Delta^{\sqsubseteq}_{\tau,\rho}$.

We do not elaborate on the proof, except for noting that all proof cases are like those for Theorem 4.4 or 5.4 or blends thereof.

Of course, it is also possible to replay the definition of left- and right-graphs of functions, study appropriate restrictions for them, and so on. But since it turns out that for the particular case of our introductory example this does not give results beyond those obtained in Example 4.9 (i.e., we get two-way refinement under the same conditions under which we already showed equivalence there), we stop here for the moment.

7 Dealing with Exceptions

So far, we have dealt with errors as events that lead a program to fail, without any possibility to manipulate them from inside the language itself, or to even recover from them. While full exception handling, which in Haskell happens in the IO monad, is out of the scope of the present paper, we undertake some steps in the proper direction by discussing how to deal with the Haskell primitive *mapException*, also discussed by Peyton Jones et al. (1999).

First of all, we need an algebraic datatype for representing exceptions as (non-erroneous) values. Simplifying the corresponding Haskell type a bit, we add

$$\begin{aligned} \tau &::= \cdots \mid \mathsf{Exception} \\ t &::= \cdots \mid \mathsf{NonTermination} \mid \mathsf{ErrorCall} \end{aligned}$$

to the syntax from Figure 1, as well as the new typing rules

$$\Gamma \vdash \mathsf{NonTermination} : \mathsf{Exception} \qquad \Gamma \vdash \mathsf{ErrorCall} : \mathsf{Int} \to \mathsf{Exception}$$

to the derivation system from Figure 2. Instead of **error**, we introduce the primitive **throw** with typing rule

$$\Gamma \vdash \mathbf{throw} : \forall \alpha. \mathsf{Exception} \to \alpha\,.$$

Since the type semantics of Exception naturally is the lifting of the set (approximation-ordered in the obvious way) containing NonTermination and ErrorCall a for every a from the type semantics of Int, we also get a slightly different collection of erroneous values. Namely, instead of (2), $\mathcal{E}$ now contains all ErrorCall a with a either

$$
\begin{aligned}
&[\![\mathsf{NonTermination}]\!]^{err}_{\theta,\sigma} = Ok\ \mathsf{NonTermination}\\
&[\![\mathsf{ErrorCall}]\!]^{err}_{\theta,\sigma} = Ok\ (\lambda a. Ok\ (\mathsf{ErrorCall}\ a))
\end{aligned}
$$

$$
[\![\mathbf{throw}]\!]^{err}_{\theta,\sigma} = Ok\ (\lambda D. Ok\ (\lambda a. \begin{cases} \bot & \text{if } a = Ok\ \mathsf{NonTermination}\\ Bad\ \{\mathsf{ErrorCall}\ b\} & \text{if } a = Ok\ (\mathsf{ErrorCall}\ b)\\ Bad\ e & \text{if } a = Bad\ e \end{cases}))
$$

$$
[\![\mathbf{mapException}]\!]^{err}_{\theta,\sigma} =
$$

$$
Ok\ (\lambda D. Ok\ (\lambda h. Ok\ (\lambda a. \begin{cases} Ok\ v & \text{if } a = Ok\ v\\ \bot & \text{if } a = \bot\\ Bad\ \bigcup\limits_{v \in e} E([\![\mathbf{throw}]\!]^{err}_{\theta,\sigma} & \text{if } a = Bad\ e\\ \qquad \$\$\ D\ \$\ (h\ \$\ (Ok\ v))) & \quad \text{with } e \in \mathcal{P}(\mathcal{E}) \end{cases})))
$$

Figure 12: Additions for Error Semantics of Terms with Advanced Primitives.

being itself an erroneous value or being $Ok\ n$ for $n \in \{\ldots, -2, -1, 0, 1, 2, \ldots\}$. As a consequence of having erroneous values $Bad\ e$ with e containing objects like $\mathsf{ErrorCall}\ (Bad\ e')$, we need a slightly more sophisticated definition of the approximation (pre-)order. Namely, when comparing erroneous values a and b, instead of checking $E(a) \supseteq E(b)$ as in (3), we have to check whether

$$
\begin{gathered}
\mathsf{NonTermination} \in E(b) \Rightarrow \mathsf{NonTermination} \in E(a)\\
\wedge\ \forall\ (\mathsf{ErrorCall}\ c) \in E(b).\ \exists\ (\mathsf{ErrorCall}\ d) \in E(a).\ d \sqsubseteq c\,.
\end{gathered} \tag{15}
$$

Similarly, when checking such sets for equality, we now actually check (15) and its reverse.

Finally, the new primitive **mapException** with typing rule

$$
\Gamma \vdash \mathbf{mapException} : \forall \alpha. (\mathsf{Exception} \rightarrow \mathsf{Exception}) \rightarrow \alpha \rightarrow \alpha
$$

takes an exceptions-to-exceptions function and a further argument of arbitrary type and uses the former to transform every exception potentially appearing as a result of the latter, while preserving potential nontermination and leaving nonerroneous values completely unchanged. Note that **mapException** really needs to be added as a primitive, because the described behavior cannot otherwise be realized in our calculus (or in Haskell).

Of course, the term semantics from Figure 8, minus the definition for **error**, needs to be adapted now. The definitions to be added, carefully crafted to agree with the actual behavior of the primitives in Haskell, are shown in Figure 12. The most interesting case is in the last line, describing how **mapException** shall deal with a non-$\bot$ erroneous value in its second argument. For the sake of brevity, our definition there refers to the definition for **throw**. Clearly, this indirection could be avoided by simple inlining.

Having introduced the syntax and semantics of the extended calculus, we can now turn to relational parametricity. The changes necessary to the developments in Sections 4–6 are as follows:

- Each of Figures 9–11 gets an additional definition for $\Delta^{\cdot\cdot}_{\mathsf{Exception},\rho}$, where the right-hand side, in turn, is an identity, approximation, and refinement relation, respectively.
- In Definitions 5.2 and 6.1, "$E(a) \supseteq E(b)$" is replaced by (15).

And that is it. In particular, Theorems 4.4, 5.4, and 6.2 (as well as the consecutive developments leading to the results in Examples 4.9 and 5.10) remain valid. Of the new proof cases to investigate, the ones for the data constructors $\mathsf{NonTermination}$ and $\mathsf{ErrorCall}$ are straightforward, while the ones for **throw** are very similar to the earlier ones for **error**. We sketch only the case for **mapException** in the proof of Theorem 4.4 (the corresponding ones for Theorems 5.4 and 6.2 being only slightly more involved). To establish that for every pair of elcpos D_1, D_2 and relation $\mathcal{R} \in \mathit{Rel}^{\mathit{err}}(D_1, D_2)$,

$$m_1 = [\![\mathbf{mapException}]\!]^{\mathit{err}}_{\theta_1,\sigma_1} \ \$\$ \ D_1$$

and

$$m_2 = [\![\mathbf{mapException}]\!]^{\mathit{err}}_{\theta_2,\sigma_2} \ \$\$ \ D_2$$

are related by

$$\Delta^{\mathit{err}}_{(\mathsf{Exception}\to\mathsf{Exception})\to\alpha\to\alpha,\rho[\alpha\mapsto\mathcal{R}]} \, ,$$

first note that $T(m_1) = T(m_2)$. So it remains to show that for every $(h_1, h_2) \in \Delta^{\mathit{err}}_{\mathsf{Exception}\to\mathsf{Exception},\rho[\alpha\mapsto\mathcal{R}]}$, $(m_1 \ \$ \ h_1, m_2 \ \$ \ h_2) \in \Delta^{\mathit{err}}_{\alpha\to\alpha,\rho[\alpha\mapsto\mathcal{R}]}$, i.e.,

$$T(m_1 \ \$ \ h_1) = T(m_2 \ \$ \ h_2)$$

and for every $(a_1, a_2) \in \mathcal{R}$,

$$(m_1 \ \$ \ h_1 \ \$ \ a_1, m_2 \ \$ \ h_2 \ \$ \ a_2) \in \mathcal{R} \, .$$

By the definition of the term semantics for **mapException** and by error-reflectingness of $\mathcal{R}$, this reduces to showing that:

- for every $a_1 = \mathit{Ok} \ v_1$ and $a_2 = \mathit{Ok} \ v_2$ with $(a_1, a_2) \in \mathcal{R}$, $(\mathit{Ok} \ v_1, \mathit{Ok} \ v_2) \in \mathcal{R}$,
- $(\bot, \bot) \in \mathcal{R}$, and
- for $a_1 = \mathit{Bad} \ e_1$ and $a_2 = \mathit{Bad} \ e_2$ with $e_1 = e_2 \in \mathcal{P}(\mathcal{E})$,

$$\begin{aligned}(\mathit{Bad} \ &\bigcup_{v \in e_1} E([\![\mathbf{throw}]\!]^{\mathit{err}}_{\theta_1,\sigma_1} \ \$\$ \ D_1 \ \$ \, (h_1 \ \$ \, (\mathit{Ok} \ v))), \\ \mathit{Bad} \ &\bigcup_{v \in e_2} E([\![\mathbf{throw}]\!]^{\mathit{err}}_{\theta_2,\sigma_2} \ \$\$ \ D_2 \ \$ \, (h_2 \ \$ \, (\mathit{Ok} \ v)))) \in \mathcal{R} \, .\end{aligned}$$

The first requirement is trivial, while the second one holds by error-strictness of $\mathcal{R}$. For the third one, recall that $(h_1, h_2) \in \Delta^{\mathit{err}}_{\mathsf{Exception}\to\mathsf{Exception},\rho[\alpha\mapsto\mathcal{R}]}$ and thus for every $v \in e_1$ $(= e_2)$, $h_1 \ \$ \, (\mathit{Ok} \ v) = h_2 \ \$ \, (\mathit{Ok} \ v)$. Given this, the desired statement follows from the definition of the term semantics for **throw** (in particular its disregard of the concrete elcpo D_1 or D_2 supplied as argument) and error-strictness of $\mathcal{R}$.

Extending the results in this paper to Haskell's *catch* and *try* (which replace the *getException*-primitive of Peyton Jones et al. (1999)) is the natural next step in our research.

8 Related Work

The work most closely related to ours here is that of Johann and Voigtländer (2008), who also study relational parametricity for a setting in which different failure causes are semantically distinguished. However, they do not work with the imprecise error semantics embodied in Haskell. Rather, their error treatment is completely deterministic, but results are given modulo a presumed, and then fixed, order on erroneous values. It might be tempting to try and encode the "contents" of erroneous values in the imprecise error semantics, namely sets of error causes, into the unstructured erroneous values of the setup in Johann and Voigtländer (2008), and to choose the order on these unstructured values to agree with the reversed subset order, as used in (3), on the encoded sets. But this approach cannot faithfully model how errors are propagated and combined in the imprecise error semantics, e.g., by taking unions in the term semantics for **case**. In fact, it is unclear whether or how the formal development of Johann and Voigtländer (2008) can be adapted for integration of an "error-finding mode".

At the outset of this paper, we gave an example of how the imprecise error semantics impacts seemingly standard laws. Since this example was in no way specific to relational parametricity as a sole way of establishing the equivalence in question for the given definition of *takeWhile*, it is natural to ask about the impact of imprecise error semantics on other reasoning techniques, in particular structural induction. Recently, Filinski and Støvring (2007) established *rigid induction* as a generalized principle dealing with a range of computational effects, starting with partiality à la $\perp$. It should be interesting to investigate how imprecise error semantics fits in there.

References

A. Filinski and K. Støvring. Inductive reasoning about effectful data types. In *International Conference on Functional Programming, Proceedings*, volume 42(9) of *SIGPLAN Notices*, pages 97–110. ACM Press, 2007. DOI: 10.1145/1291220.1291168.

P. Johann and J. Voigtländer. Free theorems in the presence of seq. In *Principles of Programming Languages, Proceedings*, volume 39(1) of *SIGPLAN Notices*, pages 99–110. ACM Press, 2004. DOI: 10.1145/982962.964010.

P. Johann and J. Voigtländer. A family of syntactic logical relations for the semantics of Haskell-like languages. *Information and Computation*, 2008. DOI: 10.1016/j.ic.2007.11.009.

A. Moran, S.B. Lassen, and S.L. Peyton Jones. Imprecise exceptions, Co-inductively. In *Higher Order Operational Techniques in Semantics, Proceedings*, volume 26 of *ENTCS*, pages 122–141. Elsevier, 1999. DOI: 10.1016/S1571-0661(05)80288-9.

S.L. Peyton Jones, A. Reid, C.A.R. Hoare, S. Marlow, and F. Henderson. A semantics for imprecise exceptions. In *Programming Language Design and Implementation, Proceedings*, volume 34(5) of *SIGPLAN Notices*, pages 25–36. ACM Press, 1999. DOI: 10.1145/301631.301637.

J.C. Reynolds. Types, abstraction and parametric polymorphism. In *Information Processing, Proceedings*, pages 513–523. Elsevier, 1983.

J. Voigtländer and P. Johann. Selective strictness and parametricity in structural operational semantics, inequationally. *Theoretical Computer Science*, 388(1–3): 290–318, 2007. DOI: 10.1016/j.tcs.2007.09.014.

P. Wadler. Theorems for free! In *Functional Programming Languages and Computer Architecture, Proceedings*, pages 347–359. ACM Press, 1989. DOI: 10.1145/99370.99404.

A Proof of Theorem 2.2

The proof is by induction over typing derivations with respect to the system from Figure 2.

The cases $\Gamma, x : \tau \vdash x : \tau$, $\Gamma \vdash n : \mathsf{Int}$, and $\Gamma \vdash [\,]_\tau : [\tau]$ are immediate.

In the case
$$\frac{\Gamma \vdash t_1 : \mathsf{Int} \qquad \Gamma \vdash t_2 : \mathsf{Int}}{\Gamma \vdash (t_1 + t_2) : \mathsf{Int}}\,,$$
we have to show
$$([\![t_1 + t_2]\!]_{\theta_1,\sigma_1}, [\![t_1 + t_2]\!]_{\theta_2,\sigma_2}) \in \Delta_{\mathsf{Int},\rho}\,,$$
which is equivalent to showing that the values
$$\begin{cases} \lfloor n_1 + n_2 \rfloor & \text{if } [\![t_1]\!]_{\theta_1,\sigma_1} = \lfloor n_1 \rfloor,\ [\![t_2]\!]_{\theta_1,\sigma_1} = \lfloor n_2 \rfloor \\ \bot & \text{otherwise} \end{cases}$$
and
$$\begin{cases} \lfloor n'_1 + n'_2 \rfloor & \text{if } [\![t_1]\!]_{\theta_2,\sigma_2} = \lfloor n'_1 \rfloor,\ [\![t_2]\!]_{\theta_2,\sigma_2} = \lfloor n'_2 \rfloor \\ \bot & \text{otherwise} \end{cases}$$
are equal. But this follows from $[\![t_1]\!]_{\theta_1,\sigma_1} = [\![t_1]\!]_{\theta_2,\sigma_2}$ and $[\![t_2]\!]_{\theta_1,\sigma_1} = [\![t_2]\!]_{\theta_2,\sigma_2}$, which in turn follow from the induction hypotheses $([\![t_1]\!]_{\theta_1,\sigma_1}, [\![t_1]\!]_{\theta_2,\sigma_2}) \in \Delta_{\mathsf{Int},\rho}$ and $([\![t_2]\!]_{\theta_1,\sigma_1}, [\![t_2]\!]_{\theta_2,\sigma_2}) \in \Delta_{\mathsf{Int},\rho}$.

Similarly, in the case
$$\frac{\Gamma \vdash t_1 : \tau \qquad \Gamma \vdash t_2 : [\tau]}{\Gamma \vdash (t_1 : t_2) : [\tau]}\,,$$

we have

$$\begin{array}{rl} & ([\![t_1 : t_2]\!]_{\theta_1,\sigma_1}, [\![t_1 : t_2]\!]_{\theta_2,\sigma_2}) \in \Delta_{[\tau],\rho} \\ \Leftrightarrow & (\lfloor [\![t_1]\!]_{\theta_1,\sigma_1} : [\![t_2]\!]_{\theta_1,\sigma_1} \rfloor, \lfloor [\![t_1]\!]_{\theta_2,\sigma_2} : [\![t_2]\!]_{\theta_2,\sigma_2} \rfloor) \in \mathit{list}\,\Delta_{\tau,\rho} \\ \Leftrightarrow & ([\![t_1]\!]_{\theta_1,\sigma_1}, [\![t_1]\!]_{\theta_2,\sigma_2}) \in \Delta_{\tau,\rho},\ ([\![t_2]\!]_{\theta_1,\sigma_1}, [\![t_2]\!]_{\theta_2,\sigma_2}) \in \Delta_{[\tau],\rho}\,, \end{array}$$

so the induction hypotheses suffice.

In the case

$$\frac{\Gamma \vdash t : [\tau_1] \qquad \Gamma \vdash t_1 : \tau_2 \qquad \Gamma, x_1 : \tau_1, x_2 : [\tau_1] \vdash t_2 : \tau_2}{\Gamma \vdash (\mathbf{case}\ t\ \mathbf{of}\ \{[\,] \to t_1\,;\ x_1 : x_2 \to t_2\}) : \tau_2}\,,$$

we have to show that the values

$$\begin{cases} [\![t_1]\!]_{\theta_1,\sigma_1} & \text{if } [\![t]\!]_{\theta_1,\sigma_1} = \lfloor [\,] \rfloor \\ [\![t_2]\!]_{\theta_1,\sigma_1[x_1 \mapsto a,\, x_2 \mapsto b]} & \text{if } [\![t]\!]_{\theta_1,\sigma_1} = \lfloor a : b \rfloor \\ \bot & \text{if } [\![t]\!]_{\theta_1,\sigma_1} = \bot \end{cases}$$

and

$$\begin{cases} [\![t_1]\!]_{\theta_2,\sigma_2} & \text{if } [\![t]\!]_{\theta_2,\sigma_2} = \lfloor [\,] \rfloor \\ [\![t_2]\!]_{\theta_2,\sigma_2[x_1 \mapsto c,\, x_2 \mapsto d]} & \text{if } [\![t]\!]_{\theta_2,\sigma_2} = \lfloor c : d \rfloor \\ \bot & \text{if } [\![t]\!]_{\theta_2,\sigma_2} = \bot \end{cases}$$

are related by $\Delta_{\tau_2,\rho}$. Since $([\![t]\!]_{\theta_1,\sigma_1}, [\![t]\!]_{\theta_2,\sigma_2}) \in \Delta_{[\tau_1],\rho} = \mathit{list}\,\Delta_{\tau_1,\rho}$ by induction hypothesis, we only have to consider the following three cases:

- $[\![t]\!]_{\theta_1,\sigma_1} = \lfloor [\,] \rfloor$ and $[\![t]\!]_{\theta_2,\sigma_2} = \lfloor [\,] \rfloor$, in which case the induction hypothesis $([\![t_1]\!]_{\theta_1,\sigma_1}, [\![t_1]\!]_{\theta_2,\sigma_2}) \in \Delta_{\tau_2,\rho}$ suffices,

- $[\![t]\!]_{\theta_1,\sigma_1} = \lfloor a : b \rfloor$ and $[\![t]\!]_{\theta_2,\sigma_2} = \lfloor c : d \rfloor$ with $(a,c) \in \Delta_{\tau_1,\rho}$ and $(b,d) \in \mathit{list}\,\Delta_{\tau_1,\rho} = \Delta_{[\tau_1],\rho}$, in which case the induction hypothesis that for every such a, b, c, and d,

  $$([\![t_2]\!]_{\theta_1,\sigma_1[x_1 \mapsto a,\, x_2 \mapsto b]}, [\![t_2]\!]_{\theta_2,\sigma_2[x_1 \mapsto c,\, x_2 \mapsto d]}) \in \Delta_{\tau_2,\rho}\,,$$

  suffices, and

- $[\![t]\!]_{\theta_1,\sigma_1} = \bot$ and $[\![t]\!]_{\theta_2,\sigma_2} = \bot$, in which case we have to show $(\bot, \bot) \in \Delta_{\tau_2,\rho}$, which follows from strictness of $\Delta_{\tau_2,\rho}$ (cf. Lemma 2.1).

In the case

$$\frac{\Gamma, x : \tau_1 \vdash t : \tau_2}{\Gamma \vdash (\lambda x : \tau_1.t) : \tau_1 \to \tau_2}\,,$$

we have

$$\begin{array}{rl} & ([\![\lambda x : \tau_1.t]\!]_{\theta_1,\sigma_1}, [\![\lambda x : \tau_1.t]\!]_{\theta_2,\sigma_2}) \in \Delta_{\tau_1 \to \tau_2,\rho} \\ \Leftrightarrow & (\lfloor \lambda a.[\![t]\!]_{\theta_1,\sigma_1[x \mapsto a]} \rfloor, \lfloor \lambda b.[\![t]\!]_{\theta_2,\sigma_2[x \mapsto b]} \rfloor) \in \Delta_{\tau_1 \to \tau_2,\rho} \\ \Leftrightarrow & \forall (a,b) \in \Delta_{\tau_1,\rho}.\ ([\![t]\!]_{\theta_1,\sigma_1[x \mapsto a]}, [\![t]\!]_{\theta_2,\sigma_2[x \mapsto b]}) \in \Delta_{\tau_2,\rho}\,, \end{array}$$

so the induction hypothesis suffices.

In the case
$$\frac{\Gamma \vdash t_1 : \tau_1 \to \tau_2 \qquad \Gamma \vdash t_2 : \tau_1}{\Gamma \vdash (t_1\ t_2) : \tau_2}\,,$$
we have
$$\begin{array}{rl} & ([\![t_1\ t_2]\!]_{\theta_1,\sigma_1}, [\![t_1\ t_2]\!]_{\theta_2,\sigma_2}) \in \Delta_{\tau_2,\rho} \\ \Leftrightarrow & ([\![t_1]\!]_{\theta_1,\sigma_1}\ \$\ [\![t_2]\!]_{\theta_1,\sigma_1}, [\![t_1]\!]_{\theta_2,\sigma_2}\ \$\ [\![t_2]\!]_{\theta_2,\sigma_2}) \in \Delta_{\tau_2,\rho} \\ \Leftarrow & ([\![t_2]\!]_{\theta_1,\sigma_1}, [\![t_2]\!]_{\theta_2,\sigma_2}) \in \Delta_{\tau_1,\rho}, \\ & \forall (a,b) \in \Delta_{\tau_1,\rho}.\ ([\![t_1]\!]_{\theta_1,\sigma_1}\ \$\ a, [\![t_1]\!]_{\theta_2,\sigma_2}\ \$\ b) \in \Delta_{\tau_2,\rho} \\ \Leftarrow & ([\![t_2]\!]_{\theta_1,\sigma_1}, [\![t_2]\!]_{\theta_2,\sigma_2}) \in \Delta_{\tau_1,\rho}, \\ & ([\![t_1]\!]_{\theta_1,\sigma_1}, [\![t_1]\!]_{\theta_2,\sigma_2}) \in \Delta_{\tau_1 \to \tau_2,\rho}\,, \end{array}$$
so the induction hypotheses suffice.

In the case
$$\frac{\alpha, \Gamma \vdash t : \tau}{\Gamma \vdash (\Lambda \alpha.t) : \forall \alpha.\tau}\,,$$
we have to show that the values
$$\begin{cases} \lfloor \lambda D_1.[\![t]\!]_{\theta_1[\alpha \mapsto D_1],\sigma_1} \rfloor & \text{if } [\![t]\!]_{\theta_1[\alpha \mapsto \{\bot\}],\sigma_1} \neq \bot \\ \bot & \text{if } [\![t]\!]_{\theta_1[\alpha \mapsto \{\bot\}],\sigma_1} = \bot \end{cases}$$
and
$$\begin{cases} \lfloor \lambda D_2.[\![t]\!]_{\theta_2[\alpha \mapsto D_2],\sigma_2} \rfloor & \text{if } [\![t]\!]_{\theta_2[\alpha \mapsto \{\bot\}],\sigma_2} \neq \bot \\ \bot & \text{if } [\![t]\!]_{\theta_2[\alpha \mapsto \{\bot\}],\sigma_2} = \bot \end{cases}$$
are related by $\Delta_{\forall\alpha.\tau,\rho}$. By induction hypothesis, $[\![t]\!]_{\theta_1[\alpha \mapsto \{\bot\}],\sigma_1}$ and $[\![t]\!]_{\theta_2[\alpha \mapsto \{\bot\}],\sigma_2}$ are related by $\Delta_{\tau,\rho[\alpha \mapsto \{(\bot,\bot)\}]}$, so by bottom-reflectingness of the latter (cf. Lemma 2.1) we know that either both are non-$\bot$ or both are $\bot$. In the latter case, our proof obligation reduces to $(\bot,\bot) \in \Delta_{\forall\alpha.\tau,\rho}$, which holds by strictness of $\Delta_{\forall\alpha.\tau,\rho}$ (cf. Lemma 2.1 again). In the former case, it reduces to
$$\begin{array}{rl} & (\lfloor \lambda D_1.[\![t]\!]_{\theta_1[\alpha \mapsto D_1],\sigma_1} \rfloor, \lfloor \lambda D_2.[\![t]\!]_{\theta_2[\alpha \mapsto D_2],\sigma_2} \rfloor) \in \Delta_{\forall\alpha.\tau,\rho} \\ \Leftrightarrow & \forall D_1, D_2 \text{ pcpos}, \mathcal{R} \in \mathit{Rel}(D_1,D_2).\ ([\![t]\!]_{\theta_1[\alpha \mapsto D_1],\sigma_1}, [\![t]\!]_{\theta_2[\alpha \mapsto D_2],\sigma_2}) \in \Delta_{\tau,\rho[\alpha \mapsto \mathcal{R}]}\,, \end{array}$$
so (another invocation of) the induction hypothesis suffices.

In the case
$$\frac{\Gamma \vdash t : \forall \alpha.\tau_1}{\Gamma \vdash (t\ \tau_2) : \tau_1[\tau_2/\alpha]}\,,$$
we have
$$\begin{array}{rl} & ([\![t\ \tau_2]\!]_{\theta_1,\sigma_1}, [\![t\ \tau_2]\!]_{\theta_2,\sigma_2}) \in \Delta_{\tau_1[\tau_2/\alpha],\rho} \\ \Leftrightarrow & ([\![t]\!]_{\theta_1,\sigma_1}\ \$\$\ [\![\tau_2]\!]_{\theta_1}, [\![t]\!]_{\theta_2,\sigma_2}\ \$\$\ [\![\tau_2]\!]_{\theta_2}) \in \Delta_{\tau_1,\rho[\alpha \mapsto \Delta_{\tau_2,\rho}]} \\ \Leftarrow & \forall D_1, D_2 \text{ pcpos}, \mathcal{R} \in \mathit{Rel}(D_1,D_2).\ ([\![t]\!]_{\theta_1,\sigma_1}\ \$\$\ D_1, [\![t]\!]_{\theta_2,\sigma_2}\ \$\$\ D_2) \in \Delta_{\tau_1,\rho[\alpha \mapsto \mathcal{R}]} \\ \Leftrightarrow & ([\![t]\!]_{\theta_1,\sigma_1}, [\![t]\!]_{\theta_2,\sigma_2}) \in \Delta_{\forall\alpha.\tau_1,\rho}\,, \end{array}$$

so the induction hypothesis suffices. Note that the equivalence

$$\Delta_{\tau_1[\tau_2/\alpha],\rho} = \Delta_{\tau_1,\rho[\alpha\mapsto\Delta_{\tau_2,\rho}]}\,,$$

used in the first step above, holds by an easy induction on τ_1. Also note that the consecutive step uses $\Delta_{\tau_2,\rho} \in \mathit{Rel}$, as justified by Lemma 2.1.

In the case

$$\Gamma \vdash \mathbf{fix} : \forall\alpha.(\alpha \to \alpha) \to \alpha\,,$$

we have

$$\begin{array}{ll} & ([\![\mathbf{fix}]\!]_{\theta_1,\sigma_1}, [\![\mathbf{fix}]\!]_{\theta_2,\sigma_2}) \in \Delta_{\forall\alpha.(\alpha\to\alpha)\to\alpha,\rho} \\ \Leftrightarrow & (\lfloor\lambda D_1.\lfloor\lambda h_1.\bigsqcup((h_1\,\$)^i\,\bot)\rfloor\rfloor, \lfloor\lambda D_2.\lfloor\lambda h_2.\bigsqcup((h_2\,\$)^i\,\bot)\rfloor\rfloor) \in \Delta_{\forall\alpha.(\alpha\to\alpha)\to\alpha,\rho} \\ \Leftrightarrow & \forall D_1, D_2 \text{ pcpos}, \mathcal{R} \in \mathit{Rel}(D_1, D_2). \\ & (\lfloor\lambda h_1.\bigsqcup((h_1\,\$)^i\,\bot)\rfloor, \lfloor\lambda h_2.\bigsqcup((h_2\,\$)^i\,\bot)\rfloor) \in \Delta_{(\alpha\to\alpha)\to\alpha,\rho[\alpha\mapsto\mathcal{R}]} \\ \Leftrightarrow & \forall D_1, D_2 \text{ pcpos}, \mathcal{R} \in \mathit{Rel}(D_1, D_2). \\ & \forall (h_1, h_2) \in \Delta_{\alpha\to\alpha,\rho[\alpha\mapsto\mathcal{R}]}.\ (\bigsqcup((h_1\,\$)^i\,\bot), \bigsqcup((h_2\,\$)^i\,\bot)) \in \mathcal{R}\,. \end{array}$$

The precondition $(h_1, h_2) \in \Delta_{\alpha\to\alpha,\rho[\alpha\mapsto\mathcal{R}]}$ implies that for every $(a, b) \in \mathcal{R}$, also $(h_1\ \$\ a, h_2\ \$\ b) \in \mathcal{R}$. Together with strictness and continuity of $\mathcal{R}$, this gives the desired statement.

Finally, in the case

$$\frac{\Gamma \vdash t_1 : \tau_1 \qquad \Gamma, x : \tau_1 \vdash t_2 : \tau_2}{\Gamma \vdash (\mathbf{let!}\ x = t_1\ \mathbf{in}\ t_2) : \tau_2}\,,$$

we have to show that the values

$$\begin{cases} [\![t_2]\!]_{\theta_1,\sigma_1[x\mapsto a]} & \text{if } [\![t_1]\!]_{\theta_1,\sigma_1} = a \neq \bot \\ \bot & \text{if } [\![t_1]\!]_{\theta_1,\sigma_1} = \bot \end{cases}$$

and

$$\begin{cases} [\![t_2]\!]_{\theta_2,\sigma_2[x\mapsto b]} & \text{if } [\![t_1]\!]_{\theta_2,\sigma_2} = b \neq \bot \\ \bot & \text{if } [\![t_1]\!]_{\theta_2,\sigma_2} = \bot \end{cases}$$

are related by $\Delta_{\tau_2,\rho}$. By the induction hypothesis $([\![t_1]\!]_{\theta_1,\sigma_1}, [\![t_1]\!]_{\theta_2,\sigma_2}) \in \Delta_{\tau_1,\rho}$ and bottom-reflectingness of $\Delta_{\tau_1,\rho}$ (cf. Lemma 2.1) we only have to consider the following two cases:

1. $[\![t_1]\!]_{\theta_1,\sigma_1} = a \neq \bot$ and $[\![t_1]\!]_{\theta_2,\sigma_2} = b \neq \bot$, in which case the induction hypothesis that for every $(a, b) \in \Delta_{\tau_1,\rho}$,

   $$([\![t_2]\!]_{\theta_1,\sigma_1[x\mapsto a]}, [\![t_2]\!]_{\theta_2,\sigma_2[x\mapsto b]}) \in \Delta_{\tau_2,\rho}\,,$$

   suffices, and

2. $[\![t_1]\!]_{\theta_1,\sigma_1} = \bot$ and $[\![t_1]\!]_{\theta_2,\sigma_2} = \bot$, in which case we have to show $(\bot, \bot) \in \Delta_{\tau_2,\rho}$, which follows from strictness of $\Delta_{\tau_2,\rho}$ (cf. Lemma 2.1 again).

This completes the proof.