Type-Dependent Equality (TDE)

Florian Rabe

Summer 2021

Motivation

Soft vs. Hard Typing

Hard Typing

- typing is function from terms to types
- types exist independently of terms
- terms exist as inhabitants of types
- type-checking (usually) decidable
- examples: HOL, dependent type theory (Coq, Agda, HoTT, . . .)

Soft Typing

- typing is relation between terms and types
- types are predicates on terms
- terms exist independently of types
- type-checking (usually) undecidable
- examples: Nuprl, Mizar

TDE only an option in soft-typed systems

Side note: Soft-Hard Intermediates

PVS

- hard-typed HOL
- plus predicate subtypes at any type
- plus anonymous record types with structural (horizontal) subtyping

OOP

- hard-typed language for base types, enums
- nominal subtyping between user-defined types (= classes)
- is-instance-of operator and type casts

Duck typing

- formal type system deemphasized
- types represent expectations (presence of methods) about objects

Equality in Soft-Typed Systems

Without TDE

- binary relation $x = y$ on untyped terms
- independent of type

Mizar, formal set theory

With TDE

- ternary relation $x =_A y$
- allows $x =_A y$ but $x \neq_B y$

Nuprl, occasionally in informal mathematics

- Quotients:

$$1 \neq_{\mathbb{Z}} 3 \quad \text{but} \quad 1 =_{\mathbb{Z}\,\mathtt{mod}\,2} 3$$

- Structures:

$$(\mathbb{N}, +) \neq_{Monoid} (\mathbb{N}, \cdot) \quad \text{but} \quad (\mathbb{N}, +) =_{Set} (\mathbb{N}, \cdot)$$

- Generated sets:

$$X \cdot Y \neq_{Group\langle X,Y \rangle} Y \cdot X \quad \text{but} \quad X \cdot Y =_{\mathbb{R}[X,Y]} Y \cdot X$$

Subtyping in Soft-Typed Systems

Subtyping $A <: B$ iff

Without TDE

- for all x, if $x : A$, then $x : B$
- identity map is injection $A \to B$

With TDE

- for all x, if $x : A$, then $x : B$, and
- for all x, y, if $x =_A y$, then $x =_B y$
- identity map is function $A \to B$ preserves membership and equality
- not necessarily injective: B may equate more terms than A

Type Theoretical Features: Predicate Types

Overview

- Motivation: subtype $A|p$ of A by unary predicate $p : A \to \texttt{bool}$ — $\{x \in A \mid p(x)\}$ in math
- Example: positive integers

$$\mathbb{Z}|(\lambda x.x > 0)$$

- Subtyping: $A|p <: A$
- Representation: inhabitants of A reused as inhabitants of $A|p$ — injection is no-op
  - elegant on paper
  - efficient in implementations

Typing Rules

Type formation:

$$\frac{\vdash A : \mathtt{type} \qquad \vdash p : A \to \mathtt{bool}}{\vdash A|p : \mathtt{type}}$$

Introduction:

$$\frac{\vdash t : A \qquad \vdash p\,t}{\vdash t : A|p}$$

Elimination:

$$\frac{\vdash t : A|p}{\vdash t : A} \qquad \frac{\vdash t : A|p}{\vdash p\,t}$$

Type-dependent equality:

$$\frac{\vdash s : A|p \qquad \vdash t : A|p \qquad \vdash s =_A t}{\vdash s =_{A|p} t}$$

Note:

- introduction and elimination are no-ops — no new syntax
- introduction rule is type-dependent definedness

Type Theoretical Features: Quotient Types

Overview

- Motivation: quotient A/r of A by binary predicate $r : A \to A \to \texttt{bool}$
- Example: integers modulo 3

$$\mathbb{Z}/(\lambda xy.x \equiv y \bmod 3)$$

- Subtyping: no obvious subtyping between A/r and A
- Representation: equivalence classes as elements of A/r
  awkard on paper and in implementations
- Idea with TDE
  - reuse inhabitants of A as inhabitants of A/r
  - projection $A \to A/r$ as no-op
  - TDE: use different equalities $=_A$ and $=_{A/r}$

Typing Rules

Type formation:

$$\frac{\vdash A : \mathtt{type} \qquad \vdash r : A \to A \to \mathtt{bool}}{\vdash A/r : \mathtt{type}}$$

Introduction (type-dependent definedness):

$$\frac{\vdash t : A}{\vdash t : A/r}$$

Elimination:

$$\frac{\vdash s : A/r \qquad x : A \vdash t(x) : B \qquad x : A, y : A, r\,x\,y \vdash t(x) =_B t(y)}{\vdash t(s) : B}$$

Type-dependent equality:

$$\frac{\vdash s : A \qquad \vdash t : A \qquad \vdash r\,s\,t}{\vdash s =_{A/r} t}$$

Note:

- r need not be equivalence — closure taken by equality rules
- elimination form $t(s)$ applies t to any representative s

Predicate-Quotient Type Duality

Motivation

Predicate types $A|p$

- canonical injections into A
- every function f into A uniquely factors through minimal $A|p$

image of f

Quotient types A/r

- canonical projections out of A
- every function f out of A uniquely factors through maximal A/r

kernel of f

dual in the sense of category theory

Duality not fully exploited in typical mathematics

- injections are no-ops
- projections via equivalence classes

with TDE: projections are no-ops

Predicate/Quotient Subtype Hierarchy with TDE

$A\|(\lambda x.\texttt{false}) = \varnothing$		initial object, empty type
$<:$		
$A\|p$		predicate types
$<:$	for $\forall x. p\, x \Rightarrow q\, x$	using increasingly
$A\|q$		true predicates
$<:$		
$A\|(\lambda x.\texttt{true})$		
$= A =$		base type
$A/(\lambda xy.\texttt{false})$		
$<:$		
A/r		quotient types
$<:$	for $\forall xy. r\, x\, y \Rightarrow s\, x\, y$	using increasingly
A/s		true relations
$<:$		
$A/(\lambda xy.\texttt{true}) = \top$		terminal object, unit type

Type Theoretical Features: Record Types

Strict vs. Lax Records

Strict record types R

- records of type R have exactly the fields of R — $\{\}$ unit type
- forgetful functor $S \to R$ copies/removes fields — R quotient of S
- equality at R: all fields equal — normal equality

Lax record types R (also called *extensible*)

- records of type R have at least the fields of R — $\{\}$ type of all records
- forgetful functor $S \to R$ is no-op — $S <: R$
- equality at R: fields required by R equal,
  extraneous fields may differ $\to$ type-dependent equality

Lax record example: $R = \{x : \mathbb{N}\}$ and $S = \{x : \mathbb{N}, y : \mathbb{N}\}$

$$[x = 0, y = 1] =_R [x = 0, y = 2]$$

$$[x = 0, y = 1] \neq_S [x = 0, y = 2]$$

Are mathematical structures strict or lax?

Mathematical practice abstracts from distinction

Case pro strictness

- Typical definitions define structures as certain tuples

  A group is a tuple $(G, \circ, e, ^{-1})$ such that ...

- Yields different categories with explicit forgetful functors

Case pro laxness

- Subtyping routinely used
  - Every group is a monoid
  - Every topological group is a group
  - Every vector space with distinguished base is a vector space

  forgetful functor is no-op, same letter used

- Elements of tuples routinely seen as flexible

  Groups also given as tuples $(G, \circ)$

A Language with Type-Dependent Equality

Syntax

Minimal syntax for function+predicate+quotient types

$$
\begin{array}{lcl}
A, B & ::= & \texttt{bool} \mid A \to B \mid A|p \mid A/r \\
s, t, p, r & ::= & x \mid \text{(usual logical operators)} \\
& \mid & \lambda x : A.t \mid t\,t \mid s =_A t \mid t \in A
\end{array}
$$

intro/elim for predicate/quotient types are no-ops
easy to add dependent types, lax record types etc.

Rules:

- for function types as usual
- for predicate and quotient types as above
- for equality: see below

Semantics

Based on partial equivalence relations (PER) on universe U

well-known trick

PER on U = equivalence relation on subset of U

Interprets

- type A as PER $[\![A]\!] \subseteq U \times U$

  U restricted to domain of $[\![A]\!]$ and quotiented by $[\![A]\!]$

- term $t : A$ as elements $[\![t]\!] \in U$

  equivalence class of t relative to $[\![A]\!]$

- typing $t : A$

  $[\![t]\!]$ in domain of $[\![A]\!]$

- equality $s =_A t$ as

  $$([\![s]\!], [\![t]\!]) \in [\![A]\!]$$

Rules for Equality

Introduction rule (reflexivity)

$$\frac{\vdash t : A}{\vdash t =_A t}$$

Elimination rule (substitution):

$$\frac{\vdash s =_A s' \qquad x : A \vdash F(x) : \texttt{bool} \qquad \vdash F(s)}{\vdash F(s')}$$

Main problem with TDE: soundness of substitution very brittle

Need to check interaction between

- each type and — e.g., quotient types, lax record types
- each generic operation — e.g., substitution, $\in$-operator

Soundness of Substitution: Failures

Boolean operator $t \in A$

- $t \in (A|p)$ can simulate ill-typed application $p(t)$
- for $x : (\mathbb{N}/\mathtt{mod}_2)$
  - $\mathrm{IsPrime}(x)$ ill-typed
  - $x \in (\mathbb{N}|\mathrm{IsPrime})$ well-typed

$$2 =_{\mathbb{N}/\mathtt{mod}_2} 4$$

$$2 \in (\mathbb{N}|\mathrm{IsPrime}) \quad \text{and} \quad 4 \notin (\mathbb{N}|\mathrm{IsPrime})$$

Lax record types with access to extraneous fields

- record types $AbelianGroup <: Group$
- function $\lambda x : Group.\ \mathtt{if}(x\ \mathtt{hasField}\ commutative) \ldots$
- may treat $Group$-equal inputs differently

Handling in Proof Assistants

Mizar

Type System

- Soft typing
- No TDE
- Equality
  - syntax: $x = y$
  - semantics $x =_A y$ where A is smallest type containing x and y

Records

- Only named record types declared individually at toplevel
- Inheritance between records yields
  - nominal subtyping (special case of lax records)
  - explicit forgetful functors
- Equality between records
  - equality of all shared fields
  - forgetful functors applied to compare fewer fields

Nuprl

Type System

- Soft typing
- With TDE
- Quotients as before

Records

- No primitive records
- lax records defined via other type operators

Handling of substitution

$$\frac{\vdash s =_A s' \quad x : A \vdash F(x) : \texttt{bool} \quad \vdash F(s)}{\vdash F(s')}$$

$$\frac{\vdash A : \texttt{type} \quad t \text{ closed}}{t \in A : \texttt{bool}} \qquad \text{thus not: } \frac{A <: B}{x : B \vdash x \in A : \texttt{bool}}$$

Conclusion

Big-Picture Message

Hard typing arguably dominant paradigm

- type theoretical programming languages — Haskell, ML
- formalized mathematics — most ITPs

But soft typing inherent feature of mathematics

hard typing doomed as formalism for math?

Soft typing worth revisiting

- main drawback: theorem proving needed for type checking
- but today
  - type systems much better understood
  - ATPs much stronger

Quote:

- Me: What would you change if starting from scratch?
- Main developer of a hard-typed ITP: I'd do everything soft-typed like in Mizar.

Type-Dependent Equality (TDE)

Optional feature in soft-typed systems

Advantages

- elegant representation of quotients — projection is no-op
- better capture of duality of predicate/quotient typing
- good handling of equality for lax records
- maybe closer to informal mathematics

Disadvantages

- not well-understood
- soundness of substitution subtly difficult
- not combinable with every other language feature
  e.g., inspecting lax record fields

"Type-Dependent Equality" is an impractical name

- causes misunderstandings, impossible to google
- tell me if you have a better suggestion