UNIVERSITY OF TARTU
1632

Poor man's dependent types for Isabelle/HOL

Dominique Unruh

University of Tartu
RWTH Aachen

The setting

- Isabelle/HOL theorem prover:
  - Nice UI, large math library
  - HOL: Logic without dependent types
    - Makes things often easier
    - Sometimes dead ends
- We try to get around one of these dead ends

Existential types – some use cases

- ∃ countably-infinite dimensional vector space
- ∀ prime p, ∃ field of size p
- For any subspace of a Hilbert space, exists an isomorphic Hilbert space
- For any lens $A \to B$, exists T and bijection $f : A \times T \to B$ describing it
- Any *-homo f can be written as $f(a) = U(a \otimes 1)U^*$

How to express this in Isabelle/HOL?

- ∃ countably-infinite dimensional vector space

```
typedef myvs = ‹{f::nat⇒real. finite (- f -` {0})}›
  by (auto intro!: exI[of _ ‹λ_. 0›])

instance myvs :: real_vector
  sorry

lemma ‹countable_dimension TYPE(myvs)›
  sorry
```

- ∀ prime p, ∃ field of size p

```
typedef myfield = ‹{x::nat. x < p}›
Illegal variables in representing set: "p"
```

How to get around this?

- Can express the group example,
  and all the other examples

- Without introducing dependent or existential types into the logic

- With dirty trickery, yes! ☺

Aside: typedef in Isabelle/HOL

- Type definitions distinct from value definitions

```
typedef myvs = ‹{f::nat⇒real. finite (- f -` {0})}›
```

$=: S$

- Introduces new type `myvs`.
- Axiomatizes a bijection Rep between the type `myvs` and the set S.
- Adds a bit of convenient infrastructure
- Can be parametric in other types, not values!

Our solution

- New (derived) predicate:

```
let 't = S in P
```

- Definition (roughly):
  If ‘t is a type isomorphic to S, then P holds.
- Expresses: “there is a type ‘t so that P holds”

Rules

Uses Types_to_Set extension

$$\frac{\texttt{P}\ \textit{(given typedef)}}{\texttt{let 'a::type = w in P}}\ \textsc{Intro}$$

$$\frac{\texttt{let 'a::type = w in P} \qquad \texttt{'a}\ \textit{does not occur in}\ \texttt{P}}{\texttt{P}}\ \textsc{Elim}$$

$$\frac{\texttt{let 'a::type = w in P} \qquad \texttt{P ==> Q}\ \textit{(given typedef)}}{\texttt{let 'a::type = w in Q}}\ \textsc{ModusPonens}$$

- Here *(given typedef)* means:
  Holds in a context where the typedef-axioms are assumed for ‘a

How it's defined

```
(let 't::type = (S::'s set) in P rep_t)
 ⟷
∀'t S ≠ {} ∧ (∀rep::'t⇒'s. bij_betw rep UNIV S ⟶ P rep)
```

- Notice: free type variables always implicitly all-quantified on top.
- Consequence: not meaningful in a negated position (∀'t "floats up")
- Still useful!

Use example

```
lemma tensor_ccsubspace_INF_left: ‹(INF x∈X. S x) ⊗s T = (INF x∈X. S x ⊗s T)› if ‹X ≠ {}›
proof (cases ‹T=0›) [3 lines]
next
  case False
  from ccsubspace_as_whole_type[OF False]
  have ‹let 't::type = some_onb_of T in
        (INF x∈X. S x) ⊗s T = (INF x∈X. S x ⊗s T)›
  proof with_type_mp
    with_type_case
    from with_type_mp.premise
    obtain U :: ‹'t ell2 ⇒CL 'c ell2› where [simp]: ‹isometry U› and imU: ‹U *s ⊤ = T› [1 lines]
    have ‹(id_cblinfun ⊗o U) *s ((⨅x∈X. S x) ⊗s ⊤) = (id_cblinfun ⊗o U) *s (⨅x∈X. S x ⊗s ⊤)›
    then show ‹(⨅x∈X. S x) ⊗s T = (⨅x∈X. S x ⊗s T)› [1 lines]
  qed
  from this[cancel_with_type]
  show ?thesis
    by -
qed
```