Quotient Types — A Modular Approach

Part II

Aleksey Nogin

April 16, 2001

Overview of the Previous Seminar

- Introduction and motivation:
  - I propose splitting complex rules and concepts into simpler ones
    - More flexible rules
    - Easier to understand
    - Allow reusing and recombining
  - I concentrate on the quotient type
    - Quotient rules are the most complex ones in NuPRL
    - We had trouble using quotient types efficiently
- Basic building blocks (modularizing the quotient type)
  - The Squash type
  - The extensional Squash type
  - The explicit nondeterminicity type
  - The intensional quotient type

  Criteria:
  - · "single–step" rules
  - · balanced introduction and elimination rules
  - · reversible rules

Plan for Today

- A quick review of the types introduced in the previous seminar
- Formalization of the intensional quotient type
- Formalization of collection types

The ''Squash'' Type

Informally, $[A] = \{\texttt{Unit} \mid A\}$

$$\frac{\Gamma \vdash A}{\Gamma \vdash [A]} \quad (\textit{Squash-intro})$$

$$\frac{\Gamma;\, x : A;\, \Delta \vdash t \in C}{\Gamma;\, x : [A];\, \Delta \vdash t \in C} \quad (\textit{Squash-elim})$$

$$\frac{\Gamma \vdash [A]}{\Gamma \vdash \bullet \in [A]} \quad (\textit{Squash-mem-intro})$$

$$\frac{\Gamma;\, x : [A];\, \Delta[\bullet] \vdash C[\bullet]}{\Gamma;\, x : [A];\, \Delta[x] \vdash C[x]} \quad (\textit{Squash-mem-elim})$$

Extensional Squash Operator A Semantics

For a closed term A,

$$[\![A]\!] \in \mathbb{U}_i \quad \textit{iff} \quad A \in \mathbb{U}_i$$

$$t_1 = t_2 \in [\![A]\!] \quad \textit{iff} \quad t_1 = t_2 \in [A]$$

For closed A and B,

$$[\![A]\!] = [\![B]\!] \in \mathbb{U}_i \quad \textit{iff} \quad A \in \mathbb{U}_i \,\&\, B \in \mathbb{U}_i \,\&\, A \Leftrightarrow B$$

Notes

- A nondeterministic pair is in some type when the elements of that pair are equal in that type.
- When a *deterministic* function is applied to a nondeterministic pair, the result is a nondeterministic pair of function applications.

Extensional Squash Operator A Rules

$$\frac{\Gamma \vdash [\![A]\!] \Leftrightarrow [\![B]\!]}{\Gamma \vdash [\![A]\!] = [\![B]\!]} \qquad (\textit{Esquash-equality})$$

$$\frac{\Gamma \vdash [A]}{\Gamma \vdash [\![A]\!]} \qquad (\textit{Esquash-intro})$$

$$\frac{\Gamma \vdash [\![A]\!] \qquad \Gamma \vdash A \ \mathsf{Type}}{\Gamma \vdash [A]} \qquad (\textit{Esquash-elim})$$

$$\frac{\Gamma \vdash [\![A]\!]}{\Gamma \vdash \bullet \in [\![A]\!]} \qquad (\textit{Esquash-mem-intro})$$

$$\frac{\Gamma; x : [\![A]\!]; \Delta[\bullet] \vdash C[\bullet]}{\Gamma; x : [\![A]\!]; \Delta[x] \vdash C[x]} \qquad (\textit{Esquash-mem-elim})$$

Explicit Nondeterminism A Semantics

Informally, ND $= \mathbb{B}/\!\!/\mathsf{True}$ Γ the type of "nondeterministic bits".

$\mathsf{ND}_x\{t_1; t_2\} = \texttt{if } x \texttt{ then } t_1 \texttt{ else } t_2 \texttt{ fi}$ Γ a "nondeterministic pair" (when $x \in \mathsf{ND}$).

Example. $x : \mathsf{ND} \vdash \mathsf{ND}_x\{2; 4\} \in \mathbb{Z}_2$ is valid, but $x : \mathsf{ND} \vdash \mathsf{ND}_x\{2; 4\} \in \mathbb{Z}$ is not.

Explicit Nondeterminism A Rules

$$\frac{\Gamma;\ u : A[\mathtt{tt}] = A[\mathtt{ff}];\ y : A[\mathtt{tt}];\ x : \mathsf{ND};\ \Delta[x;y] \vdash C[x;y]}{\Gamma;\ x : \mathsf{ND};\ y : A[x];\ \Delta[x;y] \vdash C[x;y]} \quad (\textit{ND-elim})$$

$$\frac{\Gamma \vdash C[\mathtt{tt}] = C[\mathtt{ff}] \qquad \Gamma \vdash C[\mathtt{tt}]}{\Gamma;\ x : \mathsf{ND} \vdash C[x]} \quad (\textit{ND-elim}')$$

Derived rule:

$$\frac{\Gamma \vdash t[\mathtt{tt}] = t[\mathtt{ff}] \in A}{\Gamma;\ x : \mathsf{ND} \vdash t[x] \in A} \quad (\textit{ND-memb})$$

Notes

- Notice that (*ND-elim*) does not completely eliminate ND, in order to eliminate it we need to iterate (*ND-elim*) and then apply (*ND-elim'*).
- The ruleset for ND is intensionally very incomplete. In this context, ND is only meant to be an auxiliary type since our mechanism for expressing rules can not express the result of completely eliminating ND.
- (*ND-memb*) is here to show what (*ND-elim'*) means in a simple particular case where it is especially interesting.
- However, this type seems to also be interesting on its own. If we decide to use it on its own, we'll need more rules about it, but this is outside of the scope of this talk.
- Again, all the rules are reversible.

- Now we have all we need to be able to describe the quotient type.
- As we discussed a week ago, NuPRL 4 has intensional set type, but an extensional quotient type. With esquash, it's easy to convert intensional set and quotient types into extensional ones by esquashing the predicate, so I will focus on the intensional quotient type.

Intensional Quotient Type Base Rules

$$\frac{E \text{ is an ER over } A \qquad \Gamma \vdash A = A' \qquad \Gamma;\ x : A;\ y : A \vdash E[x;y] = E'[x;y]}{\Gamma \vdash (A \underset{i}{/\!/} E) = (A' \underset{i}{/\!/} E')} \quad (\textit{Iquot-equal})$$

$$\frac{\Gamma \vdash x \in A \qquad \Gamma \vdash (A \underset{i}{/\!/} E)\ \mathsf{Type}}{\Gamma \vdash x \in (A \underset{i}{/\!/} E)} \quad (\textit{Iquot-mem-intro})$$

Intensional Quotient Type Equality Rules

$$\frac{\Gamma \vdash [\![E[x;y]]\!] \qquad \Gamma \vdash x \in (A \underset{i}{/\!/} E) \qquad \Gamma \vdash y \in (A \underset{i}{/\!/} E)}{\Gamma \vdash x = y \in (A \underset{i}{/\!/} E)} \quad (\textit{Iquot-equal-intro})$$

$$\frac{\Gamma;\ u : x = y \in (A \underset{i}{/\!/} E);\ v : [\![E[x;y]]\!];\ \Delta[u] \vdash C[u]}{\Gamma;\ u : x = y \in (A \underset{i}{/\!/} E);\ \Delta[u] \vdash C[u]} \quad (\textit{Iquot-equal-elim})$$

Intensional Quotient Type
the Elimination Rule

$$\frac{\Gamma;\ u_1 : A;\ u_2 : A;\ v : E[u_1; u_2];\ x : \mathrm{ND};\ \ \Delta[\mathrm{ND}_x\{u_1; u_2\}] \vdash \mathbf{[}C[\mathrm{ND}_x\{u_1; u_2\}]\mathbf{]}}{\Gamma;\ u : A \underset{i}{/\!\!/} E; \Delta[u] \vdash \mathbf{[}C[u]\mathbf{]}} \quad (\textit{Iquot-elim})$$

Intensional Quotient Type
All the Rules

$$\frac{E \text{ is an ER over } A \qquad \Gamma \vdash A = A' \qquad \Gamma; x : A; y : A \vdash E[x; y] = E'[x; y]}{\Gamma \vdash (A \underset{i}{/\!\!/} E) = (A' \underset{i}{/\!\!/} E')} \quad (\textit{Iquot-equal})$$

$$\frac{\Gamma \vdash x \in A \qquad \Gamma \vdash (A \underset{i}{/\!\!/} E) \text{ Type}}{\Gamma \vdash x \in (A \underset{i}{/\!\!/} E)} \quad (\textit{Iquot-mem-intro})$$

$$\frac{\Gamma \vdash [\![E[x; y]]\!] \qquad \Gamma \vdash x \in (A \underset{i}{/\!\!/} E) \qquad \Gamma \vdash y \in (A \underset{i}{/\!\!/} E)}{\Gamma \vdash x = y \in (A \underset{i}{/\!\!/} E)} \quad (\textit{Iquot-equal-intro})$$

$$\frac{\Gamma; u : x = y \in (A \underset{i}{/\!\!/} E); v : [\![E[x; y]]\!]; \Delta[u] \vdash C[u]}{\Gamma; u : x = y \in (A \underset{i}{/\!\!/} E); \Delta[u] \vdash C[u]} \quad (\textit{Iquot-equal-elim})$$

$$\frac{\Gamma;\ u_1 : A;\ u_2 : A;\ v : E[u_1; u_2];\ x : \mathrm{ND};\ \ \Delta[\mathrm{ND}_x\{u_1; u_2\}] \vdash \mathbf{[}C[\mathrm{ND}_x\{u_1; u_2\}]\mathbf{]}}{\Gamma;\ u : A \underset{i}{/\!\!/} E; \Delta[u] \vdash \mathbf{[}C[u]\mathbf{]}} \quad (\textit{Iquot-elim})$$

Intensional Quotient Type
Derived Rules

$$\frac{\Gamma;\ u_1 : A;\ u_2 : A;\ v : E[u_1; u_2];\ x : \mathrm{ND};\ \ \Delta[\mathrm{ND}_x\{u_1; u_2\}] \vdash t[u_1] = t[u_2] \in C}{\Gamma;\ u : A \underset{i}{/\!\!/} E; \Delta[u] \vdash t[u] \in C} \quad (\textit{Iquot-elim2})$$

$$\frac{\Gamma; u : x = y \in (A \underset{i}{/\!\!/} E); v : \mathbf{[}E[x; y]\mathbf{]}; \Delta[u] \vdash C[u] \qquad \Gamma; u : x = y \in (A \underset{i}{/\!\!/} E); \Delta[u] \vdash E[x; y] \text{ Type}}{\Gamma; u : x = y \in (A \underset{i}{/\!\!/} E); \Delta[u] \vdash C[u]} \quad (\textit{Iquot-equal-elim2})$$

Notes

- The only irreversible rule here is (*Iquot-mem-intro*).
- While (*Iquot-mem-intro*) is the irreversible part of the intro rule, the (*Iquot-equal-intro*) is the reversible part of it.
- It's interesting to note that even for intensional quotient type we need the extensional squash operator. Indeed, in (*Iquot-equal-intro*), x and y could potentially range over elements of an equivalence class of A, but E is not necessary functional over these classes, it's only functional over A. Of course, we could restrict this rule to the case when E is indeed functional Γ this is what (*Iquot-equal-elim2*) does and what the corresponding NuPRL rule does. However, (*Iquot-equal-elim2*) is weaker then (*Iquot-equal-intro*) and sometimes it is insufficient.
- (*Iquot-elim2*) shows what (*Iquot-elim*) means when the conclusion is a membership. This rule is very similar to what NuPRL has (although NuPRL has two versions of it and (*Iquot-elim*) is a generalization of *both* of them.

Collections

Collections of elements of type T: $I : \mathbb{U}_i \times (I \to T)$ or just $T \to \mathbb{P}_i$.

The two types are isomorphic:

$$\langle I, c\rangle \;\rightsquigarrow\; \lambda t : T.\, \exists i : I.\, c(i) = t \in T)$$

$$P \;\rightsquigarrow\; \langle \{t : T \mid P(t)\}, \lambda t.\, t\rangle$$

We will use $T \to \mathbb{P}$; we need to quotient it over $\Leftrightarrow$.

Collections — the Problem

$$\mathsf{Col}_i\{T\} := c_1, c_2 : (T \to \mathbb{P}_i) /\!\!/_i (\forall t : T.\, c_1(t) \Leftrightarrow c_2(t))$$

We want to define `mem` and `union`:

1. $\mathtt{mem} : \; \mathsf{Col}_i\{T\} \to T \to \mathbb{P}_i$
2. $\mathtt{union} : \; I : \mathbb{U}_i \to (I \to \mathsf{Col}_i\{T\}) \to \mathsf{Col}_i\{T\}$
3. $\forall c : T \to \mathbb{U}_i.\, \forall t : T.\, c(t) \Rightarrow \mathtt{mem}(c; t)$
4. $\forall c : T \to \mathbb{U}_i.\, \forall t : T.\, \neg c(t) \Rightarrow \neg\mathtt{mem}(c; t)$
5. $\forall c_1, c_2 : \mathsf{Col}_i\{T\}.$
   $$\Big(\Big(\forall t : T.\, \big(\mathtt{mem}(c_1; t) \Leftrightarrow \mathtt{mem}(c_2; t)\big)\Big) \Leftrightarrow c_1 = c_2 \in \mathsf{Col}_i\{T\}\Big)$$
6. $\forall I : \mathbb{U}_i.\, \forall C : I \to \mathsf{Col}_i\{T\}.\, \forall t : T.\, \big(\exists i : I.\, \mathtt{mem}(C(i); t) \Rightarrow \mathtt{mem}(\bigcup_{i:I} C(i); t)\big)$
7. $\forall I : \mathbb{U}_i.\, \forall C : I \to \mathsf{Col}_i\{T\}.\, \forall t : T.$
   $$\Big(\neg\Big(\exists i : I.\, \mathtt{mem}\big(C(i); t\big)\Big) \Rightarrow \neg\Big(\mathtt{mem}\big(\bigcup_{i:I} C(i); t\big)\Big)\Big)$$

Notes

- Note in 4 that we can not require $\mathtt{mem} \Rightarrow c$ since that would mean that we'll have to be able to reconstruct proofs of c which is not realistic. Same in 7.
- In 5, the $\Leftarrow$ direction is easy Γ it follows from 1.
- Problems: the solutions we considered would either give us `mem` in $\mathbb{P}_i /\!\!/ \Leftrightarrow$, or `mem` and/or `union` would raise the universe level, or something else would go wrong.
- And now the solution.

Collections — a Solution

$$\mathtt{mem}(c; t) := [\![c(t)]\!]$$

$$\bigcup_{i:I} C(i) := \lambda t. \exists i : I. \mathtt{mem}(C(i); t)$$