

# (Quotient) Containers are BNFs

Jasmin Christian Blanchette      Andrei Popescu  
Dmitriy Traytel

September 21, 2018

## 1 Containers are BNFs

**typedecl**  $S$  — A type/set of shapes.

**typedecl**  $U$  — A type/set of positions. Not present in the container formulation as they work directly with the dependent type  $P\ s$  for a fixed shape  $s$ . It can be thought of as the dependent sum type  $U = (\text{Sum } s : S. P\ s)$

**consts**  $P :: S \Rightarrow U\ \text{set}$  — The actual assignments of positions to shapes.

The following type  $'a\ F$  is the extension of a container.

We emulate the dependent sum type used in the containers paper using a subtype of the independent product type.  $\text{Func } A\ B$  denotes the set of functions from  $A$  to  $B$ . Since HOL functions are total any  $f \in \text{Func } A\ B$  is restricted to return some fixed unspecified value outside of the domain  $A$ :  $\forall x. x \notin A \longrightarrow f\ x = \text{undefined}$ .  $UNIV$  is the set of all elements of type  $'a$ . In HOL it is necessarily non-empty.

**typedef** (**overloaded**)  $'a\ F = \{(s :: S, f). f \in \text{Func } (P\ s) (UNIV :: 'a\ \text{set})\}$   
**by** (*auto simp: Func-def*)

**setup-lifting** *type-definition-F*

Forces a function to be *undefined* outside the given domain (later this will be always  $P\ s$  for some fixed shape  $s$ )

**abbreviation** *restr where*  
 $\text{restr } A\ f\ x \equiv (\text{if } x \in A \text{ then } f\ x \text{ else undefined})$

**lift-definition**  $\text{map-}F :: ('a \Rightarrow 'b) \Rightarrow 'a\ F \Rightarrow 'b\ F$  — Functorial action on the container extension.

**is**  $\lambda g\ (s, f). (s, \text{restr } (P\ s) (g \circ f))$   
**by** (*auto simp: Func-def*)

**lift-definition**  $\text{set-}F :: 'a\ F \Rightarrow 'a\ \text{set}$  — The elements contained in the container extension.

**is**  $\lambda(s, f). f\ 'P\ s$ .

The container extension is a BNF.

```

bnf 'a F
  map: map-F
  sets: set-F
  bd: natLeq +c card-of (UNIV :: U set)
  subgoal by (rule ext, transfer) (auto simp: Func-def)
  subgoal by (rule ext, transfer) (auto simp: Func-def)
  subgoal by (transfer) (auto simp: Func-def)
  subgoal by (rule ext, transfer) (auto simp: Func-def)
  subgoal by (simp add: card-order-csum natLeq-card-order)
  subgoal by (simp add: cinfinite-csum natLeq-cinfinite)
  subgoal apply (transfer, clarsimp)
    apply (rule ordLeq-transitive[OF card-of-image])
    apply (rule ordLeq-transitive[OF - ordLeq-csum2])
    apply simp-all
  done
subgoal for R S
  apply (rule predicate2I, transfer fixing: R S, clarsimp simp: Func-def)
  subgoal for s f g
    by (rule exI[of - restr (P s) (λu. (fst (f u), snd (g u)))]
      (auto simp: relcompp-apply image-subset-iff split-beta fun-eq-iff split: if-splits))
  done
done

```

The relator  $rel-F$  is defined internally in terms of  $map-F$  and  $set-F$ :  $rel-F R a b = (\exists z. z \in \{x. set-F x \subseteq \{(x, y). R x y\} \} \wedge map-F fst z = a \wedge map-F snd z = b)$ .

Moreover, the above **bnf** command proves a wealth of useful BNF properties, including the parametricity of most involved entities:

$$\begin{aligned}
& ((Rb \implies Sd) \implies rel-F Rb \implies rel-F Sd) \text{ map-F map-F} \\
& (rel-F R \implies rel-set R) set-F set-F \\
& ((Sa \implies Sc \implies (=)) \implies rel-F Sa \implies rel-F Sc \implies (=)) rel-F rel-F
\end{aligned}$$

## 2 Quotient Containers are BNFs

Quotient Containers additionally allow to identify different elements in the container extension that only differ by certain "allowed" permutations of positions.  $G$  (for a shape  $s$ ) is some set of allowed permutations (bijections) closed under composition and inverses and containing identity.

The restriction of all functions to  $P s$  is necessary due to the lack of dependent types.

**axiomatization**  $G$  **where**

$G\text{-bij}: f \in G s \implies \text{bij-betw } f (P s) (P s)$  **and**

***G-id***:  $id \in G\ s$  **and**  
***G-comp***:  $f \in G\ s \implies g \in G\ s \implies \text{restr } (P\ s) (g\ o\ f) \in G\ s$  **and**  
***G-inv***:  $f \in G\ s \implies \text{restr } (P\ s) (\text{the-inv-into } (P\ s)\ f) \in G\ s$

The equivalence relation  $eq$  on the container extension that allows to permute positions according to the functions in  $G$ .

**lift-definition**  $eq :: 'a\ F \Rightarrow 'a\ F \Rightarrow bool$  **is**  
 $\lambda(s1, f1). \lambda(s2, f2). s1 = s2 \wedge (\exists g \in G\ s1. f1 = \text{restr } (P\ s1) (f2\ o\ g))$  .

**lemma**  $eq\text{-refl}[simp]$ :  $eq\ x\ x$   
**by** *transfer* (*auto simp: fun-eq-iff Func-def G-id intro!: bexI[of - id]*)

**lemma**  $eq\text{-sym}$ :  $eq\ x\ y \implies eq\ y\ x$   
**apply** (*transfer; clarsimp*)  
**subgoal for**  $s\ f1\ f2$   
**apply** (*frule G-bij*)  
**apply** (*auto simp: fun-eq-iff Func-def G-inv*  
 $f\text{-the-inv-into-f-bij-betw}\ bij\text{-betw-def}\ the\text{-inv-into-into}$   
 $intro!: bexI[of - \text{restr } (P\ s) (\text{the-inv-into } (P\ s)\ f2)]$ )  
**done**  
**done**

**lemma**  $eq\text{-trans}$ :  $eq\ x\ y \implies eq\ y\ z \implies eq\ x\ z$   
**apply** (*transfer; clarsimp*)  
**subgoal for**  $s\ f1\ f\ g$   
**apply** (*frule G-bij*)  
**apply** (*auto simp: fun-eq-iff Func-def G-comp*  
 $f\text{-the-inv-into-f-bij-betw}\ bij\text{-betw-def}\ the\text{-inv-into-into}$   
 $intro!: bexI[of - \text{restr } (P\ s) (g\ o\ f)]$ )  
**done**  
**done**

The extension of a quotient container is the container extension  $'a\ F$ , quotiented by  $eq$

**quotient-type (overloaded)**  $'a\ Q = 'a\ F / eq$   
**by** (*intro equivpI reflpI sympI transpI eq-refl | elim eq-sym eq-trans | assumption*)<sup>+</sup>

**lift-definition**  $\text{map-}Q :: ('a \Rightarrow 'b) \Rightarrow 'a\ Q \Rightarrow 'b\ Q$  — Functorial action on the quotient container extension simply lifted from the container extension.

**is**  $\text{map-}F$   
**subgoal for**  $g\ f1\ f2$   
**supply**  $F.\text{map-transfer}[transfer\text{-rule}\ del]$   
**apply** (*transfer fixing: g, clarsimp*)  
**subgoal for**  $s\ f1\ f2$   
**apply** (*frule G-bij*)  
**apply** (*auto simp: fun-eq-iff Func-def bij-betw-def intro!: bexI[of - f2]*)  
**done**  
**done**

done

**lift-definition** *set-Q* :: 'a Q  $\Rightarrow$  'a set — The set of elements in the quotient container extension are the ones in the underlying container extension (equivalent container extension elements have equal sets of elements).

is *set-F*

subgoal for *f1 f2*

supply *F.set-transfer*[*transfer-rule del*]

apply (*transfer*, *clarsimp*)

subgoal for *s g f*

apply (*frule G-bij*)

apply (auto simp: *Func-def image-iff bij-betw-def intro: bexI*[*of - f -*])

apply (*metis image-iff*)

done

done

done

The quotient container extension is a BNF.

**bnf** 'a Q

map: *map-Q*

sets: *set-Q*

bd: *natLeq +c card-of (UNIV :: U set)*

subgoal by (rule *ext*, *transfer*) (auto simp: *F.map-id*)

subgoal by (rule *ext*, *transfer*) (auto simp: *F.map-comp*)

subgoal by (*transfer*) (auto cong: *F.map-cong*)

subgoal by (rule *ext*, *transfer*) (auto simp: *F.set-map*)

subgoal by (simp add: *card-order-csum natLeq-card-order*)

subgoal by (simp add: *cinfinite-csum natLeq-cinfinite*)

subgoal by (*transfer*, *clarsimp*, rule *F.set-bd*)

subgoal for *R S*

apply (rule *predicate2I*, *transfer fixing: R S*, *clarsimp simp: Func-def*)

supply *F.map-transfer*[*transfer-rule del*] *F.set-transfer*[*transfer-rule del*]

apply (*transfer fixing: R S*; *clarsimp*)

subgoal for *f1 s f2 f3 l r g1 g2 g3 g4*

apply (rule *exI*[*of - restr (P s) (λu. (fst (l u), snd (r (the-inv-into (P s) g3 (g2 u))))*])

apply (auto simp: *relcompp-apply image-subset-iff split-beta fun-eq-iff Func-def split: if-splits*)

apply (*smt G-bij bij-betw-def f-the-inv-into-f image-eqI the-inv-into-onto*)

apply (rule *bexI*[*of - restr (P s) (g4 o restr (P s) (restr (P s) (the-inv-into (P s) g3) o g2))*], *clarsimp*)

apply (*metis G-bij bij-betw-def image-eqI the-inv-into-onto*)

apply (*intro G-comp G-inv; assumption*)

done

done

done

As before relator *rel-Q* is defined internally in terms of *map-Q* and *set-Q*:  
 $rel-Q\ R\ a\ b = (\exists z. z \in \{x. set-Q\ x \subseteq \{(x, y). R\ x\ y\}\} \wedge map-Q\ fst\ z = a$

$\wedge \text{map-}Q \text{ snd } z = b).$

Moreover, the above **bnf** command proves a wealth of useful BNF properties, including the parametricity of most involved entities:

$$\begin{aligned}
& ((Rb ==> Sd) ==> \text{rel-}Q Rb ==> \text{rel-}Q Sd) \text{ map-}Q \text{ map-}Q \\
& \quad (\text{rel-}Q R ==> \text{rel-set } R) \text{ set-}Q \text{ set-}Q \\
& ((Sa ==> Sc ==> (=)) ==> \text{rel-}Q Sa ==> \text{rel-}Q Sc ==> (=)) \text{ rel-}Q \text{ rel-}Q
\end{aligned}$$