(Quotient) Containers are BNFs

Jasmin Christian Blanchette Lorenzo Gheri
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 = (Sum s : S. P s)

consts $P :: S \Rightarrow U\ set$ — 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. *Func A B* denotes the set of functions from A to B. Since HOL functions are total any $f \in Func\ A\ B$ is restricted to return some fixed unspecified value outside of the domain A: $\forall x.\ x \notin A \longrightarrow f\ x = undefined$. *UNIV* is the set of all elements of type $'a$. In HOL it is necessarily non-empty.

typedef (**overloaded**) $'a\ F = \{(s :: S, f).\ f \in Func\ (P\ s)\ (UNIV :: 'a\ set)\}$
  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**
  $restr\ A\ f\ x \equiv (if\ x \in A\ then\ f\ x\ else\ undefined)$

lift-definition *map-F* :: $('a \Rightarrow 'b) \Rightarrow 'a\ F \Rightarrow 'b\ F$ — Functorial action on the container extension.
  is $\lambda g\ (s, f).\ (s, restr\ (P\ s)\ (g\ o\ f))$
  by (*auto simp*: *Func-def*)

lift-definition *set-F* :: $'a\ F \Rightarrow 'a\ set$ — The elemants 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*) (λ*u*. (*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* = (∃ *z*. *z* ∈ {*x*. *set-F x* ⊆ {(*x*, *y*). *R x y*}} ∧ *map-F fst z* = *a* ∧ *map-F snd z* = *b*).

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

$$((Rb \mathrel{===>} Sd) \mathrel{===>} \mathit{rel\text{-}F}\ Rb \mathrel{===>} \mathit{rel\text{-}F}\ Sd)\ \mathit{map\text{-}F}\ \mathit{map\text{-}F}$$
$$(\mathit{rel\text{-}F}\ R \mathrel{===>} \mathit{rel\text{-}set}\ R)\ \mathit{set\text{-}F}\ \mathit{set\text{-}F}$$
$$((Sa \mathrel{===>} Sc \mathrel{===>} (=)) \mathrel{===>} \mathit{rel\text{-}F}\ Sa \mathrel{===>} \mathit{rel\text{-}F}\ Sc \mathrel{===>} (=))\ \mathit{rel\text{-}F}\ \mathit{rel\text{-}F}$$

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-bij: *f* ∈ *G s* ⟹ *bij-betw f* (*P s*) (*P s*) **and**

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

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**
$\lambda(s1,\ f1).\ \lambda(s2,\ f2).\ s1 = s2 \wedge (\exists g \in G\ s1.\ f1 = restr\ (P\ s1)\ (f2\ o\ g))$ **.**

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

lemma *eq-sym*: *eq x y* $\Longrightarrow$ *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-the-inv-into-f-bij-betw bij-betw-def the-inv-into-into
intro!: *bexI*[*of* - *restr* (*P s*) (*the-inv-into* (*P s*) *f2*)])
done
done

lemma *eq-trans*: *eq x y* $\Longrightarrow$ *eq y z* $\Longrightarrow$ *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-the-inv-into-f-bij-betw bij-betw-def the-inv-into-into
intro!: *bexI*[*of* - *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*)+

lift-definition *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 *map-F*
subgoal for *g f1 f2*
supply *F.map-transfer*[*transfer-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* ⇒ *'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*) (λ*u*. (*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* = (∃ *z*. *z* ∈ {*x*. *set-Q x* ⊆ {(*x*, *y*). *R x y*}} ∧ *map-Q fst z* = *a*

$\wedge$ *map-Q snd z* = *b*).
Moreover, the above **bnf** command proves a wealth of useful BNF properties, including the parametricity of most involved entities:

$$((Rb \Longrightarrow Sd) \Longrightarrow \textit{rel-Q}\ Rb \Longrightarrow \textit{rel-Q}\ Sd)\ \textit{map-Q}\ \textit{map-Q}$$
$$(\textit{rel-Q}\ R \Longrightarrow \textit{rel-set}\ R)\ \textit{set-Q}\ \textit{set-Q}$$
$$((Sa \Longrightarrow Sc \Longrightarrow (=)) \Longrightarrow \textit{rel-Q}\ Sa \Longrightarrow \textit{rel-Q}\ Sc \Longrightarrow (=))\ \textit{rel-Q}\ \textit{rel-Q}$$