NOTES ON A PUSHOUT OF SETS

1. HoTT concepts

Definition 1. A **set** is a type $\mathtt{S}$ such that for any elements $x, y \colon \mathtt{S}$, and $p, q \colon x = y$, we have $p = q$.

Definition 2. Let $f \colon \mathtt{A} \to \mathtt{B}$. For any $x, y \colon \mathtt{A}$, we get a function

$$\mathtt{ap}_f \colon x =_{\mathtt{A}} y \to fx =_{\mathtt{B}} fy$$

On Identity types.

This can be interpreted in three ways:

(1) type morphisms preserve equality,
(2) functions of spaces are continuous,
(3) groupoid morphisms given functions on hom-sets.

Because $\mathtt{ap}$ preserve paths, all functions in HoTT are continuous. There are more results showing that f is functorial in that it preserves refl's and path concatenation.

Note: we can take the categorical notation and write $f(p)$ for a path $p \colon x = y$ instead of $\mathtt{ap}_f(p)$, but for now we stick with the latter.

Definition 3. Types can be defined by constructors. For example the circle type $\mathtt{S}^1$ is given by a 0-cell s and a 2-cell $p \colon s = s$.

Higher induction says that to define a map out of such a type, it suffices to define the map on the constructors. Hence a map

$$f \colon \mathtt{S}^1 \to \mathtt{A}$$

is given by $f(s)$ and $\mathtt{ap}_f(p)$.

Definition 4. Given a span

$$\begin{array}{ccc} \mathtt{A} & \xrightarrow{f} & \mathtt{B} \\ {\scriptstyle g}\downarrow & & \\ \mathtt{C} & & \end{array}$$

its pushout $\mathtt{B} +_{\mathtt{A}} \mathtt{C}$ is defined by

- a function $\mathtt{inl} \colon \mathtt{B} \to \mathtt{B} +_{\mathtt{A}} \mathtt{C}$
- a function $\mathtt{inr} \colon \mathtt{C} \to \mathtt{B} +_{\mathtt{A}} \mathtt{C}$
- for each $a \in \mathtt{A}$ and path $\mathtt{glue}(a) \colon fa = ga$

Hence, all functions $F \colon \mathtt{B} +_{\mathtt{A}} \mathtt{C} \to \mathtt{D}$ are given by higher induction:

- define $F(\mathtt{inl}(b))$ for all $b \colon \mathtt{B}$
- define $F(\mathtt{inr}(c))$ for all $c \colon \mathtt{C}$
- define $\mathtt{ap}_F(\mathtt{glue}(a)) \colon F(\mathtt{inl}(fa)) = F(\mathtt{inr}(ga))$ for all $a \colon \mathtt{A}$

2. The setup

The idea is that we have types $\mathtt{A}$, $\mathtt{B}$, and $\mathtt{C}$, all of which are sets. The question: is the pushout given by the square

$$\begin{tikzcd} \mathtt{A} \arrow[r, hook, "f"] \arrow[d, "g"'] & \mathtt{B} \arrow[d, "\mathtt{inl}"] \\ \mathtt{C} \arrow[r, "\mathtt{inr}"] & \mathtt{B} +_{\mathtt{A}} \mathtt{C} \end{tikzcd}$$

also a set when f is a monomorphism?

Thus to determine whether $\mathtt{B} +_{\mathtt{A}} \mathtt{C}$ is a set, we need to access its identity types. We do this with an *encode-decode* style proof.

Roughly, a proof of this sort begins by guessing what the identity types are. That is, for each x and y in $\mathtt{B} +_{\mathtt{A}} \mathtt{C}$, we define a type

$$\mathtt{code} \colon \mathtt{B} +_{\mathtt{A}} \mathtt{C} \to \mathtt{B} +_{\mathtt{A}} \mathtt{C} \to \mathtt{Type}$$

so that $\mathtt{code}(x, y)$ serves as our guess as to what $x =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} y$ actually is. Then we define functions

$$\mathtt{encode}_{x,y} \colon (x = y) \to \mathtt{code}(x, y) \text{ and } \mathtt{decode}_{x,y} \colon \mathtt{code}(x, y) \to (x = y)$$

for each x and y in $\mathtt{B} +_{\mathtt{A}} \mathtt{C}$. Hopefully, these are mutually inverse.

3. DEFINING $\texttt{code}$

Let's try to define

$$\texttt{code}\colon \texttt{B} +_{\texttt{A}} \texttt{C} \to \texttt{B} +_{\texttt{A}} \texttt{C} \to \texttt{Type}.$$

Note that $\texttt{code}$ is a map from a pushout, so we define it using induction of higher types, as in Definition 4. Hence we need three types schemes:

$$\begin{aligned} \texttt{code}(\texttt{inl}(b))&\colon \texttt{B} +_{\texttt{A}} \texttt{C} \to \texttt{Type} \\ \texttt{code}(\texttt{inr}(c))&\colon \texttt{B} +_{\texttt{A}} \texttt{C} \to \texttt{Type} \\ \texttt{code}(\texttt{ap}_{\texttt{glue}}(a))&\colon \texttt{B} +_{\texttt{A}} \texttt{C} \to \texttt{Type} \end{aligned}$$

These schemes run through $a\colon A$, $b\colon B$, and $c\colon C$. They are also functions on the same coproduct! To define $\texttt{code}(\texttt{inl}(b))$, we use higher induction which gives the type schemes:

$$\texttt{code}(\texttt{inl}(b), \texttt{inl}(b')),\;\; \texttt{code}(\texttt{inl}(b), \texttt{inr}(c')),\;\; \texttt{code}(\texttt{inl}(b), \texttt{ap}_{\texttt{glue}}(a')).$$

Similarly, we define $\texttt{code}(\texttt{inr}(c))$ by

$$\texttt{code}(\texttt{inr}(c), \texttt{inl}(b')),\;\; \texttt{code}(\texttt{inr}(c), \texttt{inr}(c')),\;\; \texttt{code}(\texttt{inr}(c), \texttt{ap}_{\texttt{glue}}(a')).$$

and $\texttt{code}(\texttt{glue}(a))$ by

$$\texttt{code}(\texttt{ap}_{\texttt{glue}}(a), \texttt{inl}(b')),\;\; \texttt{code}(\texttt{ap}_{\texttt{glue}}(a), \texttt{inr}(c')),\;\; \texttt{code}(\texttt{ap}_{\texttt{glue}}(a), \texttt{ap}_{\texttt{glue}}(a')).$$

The $\texttt{code}$'s that have no $\texttt{ap}_{\texttt{glue}}$'s in the arguments correspond to our guesses for the identity types. The $\texttt{code}$'s that have one $\texttt{ap}_{\texttt{glue}}$ in the arguments give a pre- or post-composition of paths. The $\texttt{code}$'s that have two $\texttt{ap}_{\texttt{glue}}$'s in the arguments ensure that this pre- and post-composition action is coherent. This fits together in a nice little diagram:

$$\begin{tikzcd}
\texttt{code}(\texttt{inl}(b), \texttt{inl}(b')) \arrow[r, "{\texttt{code}(\texttt{inl}(b), \texttt{ap}_{\texttt{glue}}(a')}"] \arrow[d, "{\texttt{code}(\texttt{ap}_{\texttt{glue}}(a), \texttt{inl}(b')}"'] \arrow[dr, phantom, "{\simeq \texttt{code}(\texttt{ap}_{\texttt{glue}}(a), \texttt{ap}_{\texttt{glue}}(a'))}"] & \texttt{code}(\texttt{inl}(b), \texttt{inr}(c')) \arrow[d, "{\texttt{code}(\texttt{ap}_{\texttt{glue}}(a), \texttt{inr}(c')}"] \\
\texttt{code}(\texttt{inr}(c), \texttt{inl}(b')) \arrow[r, "{\texttt{code}(\texttt{inr}(c), \texttt{ap}_{\texttt{glue}}(a')}"'] & \texttt{code}(\texttt{inr}(c), \texttt{inr}(c'))
\end{tikzcd}$$

code(b,b'). $\mathtt{code}(\mathtt{inl}(b), \mathtt{inl}(b'))$ is the most complicated. In order to incorporate $\mathtt{refl}_b$ when b is not in the image of f, we define this type to be the pushout of the span

$$\begin{tikzcd} \sum_{a,a':A}(b =_B f(a)) \times (b' =_B f(a')) \times (b =_B b') \arrow[r, "\alpha"] \arrow[d, "\beta"'] & (b =_B b') \\ \sum_{a,a':A}(b =_B f(a)) \times (b' =_B f(a')) \times (g(a) =_C g(a')) & \end{tikzcd}$$

Here, α is a projection. Also, β is a projection of the first two factors and places $\mathtt{ap}_g(p)$ in the third factor. This uses the injectivity of f to get a p: $a = a'$ if the upper left is populated.

This is a proposition. Indeed, the span feet are propositions and the only way for both to be populated is if the apex is also populated. But this would identify the left and right included elements with a glue.

But this can be simplified via some case analysis. If (p, q, r) is in the upper left, then since we know that we get a path $qrp^{-1} : fa =_{\mathtt{B}} fa'$ which gives a $\ell : a =_{\mathtt{A}} a'$ by injectivity of f. Thus we can β-reduce fa' to fa which gives us $(b =_{\mathtt{B}} fa) \times (b' =_{\mathtt{B}} fa) \times (b =_{\mathtt{B}} b')$. Any witness to that has form (p, q', r) where $q^{-1}p : b =_{\mathtt{B}} b'$ so since $\mathtt{B}$ is a set, $q^{-1}p = r$ so this $(p, q', r) = (p, q', q^{-1}p)$. Since the last factor depends on the first two, we can ignore it so we reduce the upper left further to $(b =_{\mathtt{B}} fa) \times (b' =_{\mathtt{B}} fa)$. But maybe a nicer thing to work with is $(b =_{\mathtt{B}} fa) \times (b' =_{\mathtt{B}} fa') \times (a =_{\mathtt{A}} a')$. Then α assembles those maps into one of form $b =_{\mathtt{B}} b'$ and β applies g to the last factor. That is, we can work instead with the pushout

$$\begin{tikzcd} \sum_{a,a':A}(b =_B fa) \times (b' =_{\mathtt{B}} fa') \times (a =_{\mathtt{A}} a') \arrow[r, "\alpha"] \arrow[d, "\beta"'] & (b =_{\mathtt{B}} b') \\ \sum_{a,a':A}(b =_{\mathtt{B}} fa) \times (b' =_{\mathtt{B}} fa') \times (ga =_C ga') & \end{tikzcd}$$

We can simplify this further due to the injectivity of f. The apex of the span can be boiled down to

$$(b =_{\mathtt{B}} fa) \times (b' =_{\mathtt{B}} fa') \times (a =_{\mathtt{A}} a')$$

is equivalent to

$$(b =_{\mathtt{B}} fa) \times (b' =_{\mathtt{B}} fa)$$

because if all three factors are populated, then we can β-reduce a' to a and $a =_{\mathtt{A}} a$ contains only $\mathtt{refl}$ because $\mathtt{A}$ is a set. Thus, our pushout becomes

$$\begin{tikzcd} \sum_{a:\mathtt{A}}(b =_{\mathtt{B}} fa) \times (b' =_{\mathtt{B}} fa) \arrow[r, "\alpha"] \arrow[d, "\beta"'] & (b =_{\mathtt{B}} b') \\ \sum_{a,a':A}(b =_{\mathtt{B}} fa) \times (b' =_{\mathtt{B}} fa') \times (ga =_C ga') & \end{tikzcd}$$

Later, when defining $\mathtt{decode}$, we'll need to know what constructors this pushout has.

- $\sum_{a,a':A}(b =_B fa) \times (b' =_{\mathtt{B}} fa') \times (a =_{\mathtt{A}} a')$. Any witnesses (p, q, ℓ) assembles into a path $b =_{\mathtt{B}} b'$ so α is injective. Since $\mathtt{B}$ is a set, this is a proposition.

- $\sum_{a,a':A}(b =_B fa) \times (b' =_{\mathtt{B}} fa') \times (a =_{\mathtt{A}} a')$. This can only contain one element. Indeed, suppose that $(p, q, r) : (b =_B fa) \times (b' =_{\mathtt{B}} fa') \times (a =_{\mathtt{A}} a')$ and $(p', q', r') : (b =_B fa'')\times(b' =_{\mathtt{B}} fa''')\times(a'' =_{\mathtt{A}} a''')$ witness the lower left. Then p and p' assemble to witness $fa =_{\mathtt{B}} fa''$. Similarly, q and q' assemble to witness $fa' =_{\mathtt{B}} fa'''$. By injectivity of f, we get than $a =_{\mathtt{A}} a''$ and $a' =_{\mathtt{A}} a'''$. So $(b =_B fa'') \times (b' =_{\mathtt{B}} fa''') \times (a'' =_{\mathtt{A}} a''')$ beta-reduces to $(b =_B fa) \times (b' =_{\mathtt{B}} fa'') \times (a' =_{\mathtt{A}} a')$. This reduction identifies (p, q, r) and (p', q', r') because each factor is a set. So the lower left is a proposition.

From this, it follows that both α and $\beta$are injections and so the pushout is a set by some older result (ask mike). Anyway, depending on a few things we have different cases.

Remark 1.
- If $p : b =_{\mathtt{B}} b'$ neither b, b' in the image of f, then the only constructor of the pushout is $\mathtt{inl} p$.
- If $p : b =_{\mathtt{B}} b'$ and b is in the image of f, then so is b'. Then the upper left is populated, which implies the lower left is and so the constructor are the elements included from the upper right and lower left which are then glued together.
- If $b =_{\mathtt{B}} b'$ is empty is either b or b' are not in the image of f then both other corners are empty too, so no constructors.
- If $b =_{\mathtt{B}} b'$ is empty, $p : b =_{\mathtt{B}} fa$ and $q : b' =_{\mathtt{B}} fa'$, then $a =_{\mathtt{A}} a'$ must be empty, else we can prove $b =_{\mathtt{B}} b'$. Hence upper right and upper left are empty. So either $ga =_{\mathtt{C}} ga'$ is empty or not. If not, we get a constructor included from the lower left, else the pushout is empty.

code (b,c). $\mathtt{code}\,(\mathtt{inl}(b), \mathtt{inr}(c')) :\equiv \sum_{a:A}(b =_B f(a)) \times (c' =_C g(a))$ This is a proposition. Indeed, if there does not exist an $a : A$ such that $b =_B f(a)$ and $c' =_C g(a)$ are both populated, then $\mathtt{code}\,(\mathtt{inl}(b), \mathtt{inr}(c'))$ is empty. If there exists a single $a : A$ such that $b =_B f(a)$ and $c' =_C g(a)$ are both populated, then because they are each equivalent to $\mathtt{1}$, $\mathtt{code}\,(\mathtt{inl}(b), \mathtt{inr}(c'))$ is also equivalent to $\mathtt{1}$. If there is $a, a' : A$ such that $b =_B f(a)$ and $c' =_C g(a)$, and also $b =_B f(a')$ and $c' =_C g(a')$, then the injectivity of f and $f(a) =_B b =_B f(a')$ implies that $a =_A a'$ which also gives us that $\mathtt{code}\,(\mathtt{inl}(b), \mathtt{inr}(c'))$ is equivalent to $\mathtt{1}$.

code (c,b). $\texttt{code}\,(\texttt{inr}(c), \texttt{inl}(b')) :\equiv \sum_{a:A}(c =_C g(a)) \times (b' =_B f(a))$ This is a proposition by the same sort of argument from above.

code (c,c'). $\mathtt{code}\,(\mathtt{inr}(c), \mathtt{inr}(c')) :\equiv \sum_{a,a':A}(c =_C g(a)) \times (c' =_C g(a')) \times (f(a) =_B f(a'))$ The injectivity of f gives us that $f(a) =_B f(a')$ imples that $a =_A a'$ which in turn implies that $g(a) =_C g(a')$, hence $c =_C c'$. Therefore, $\mathtt{code}\,(\mathtt{inr}(c), \mathtt{inr}(c')) = (c =_C c')$. Hence $\mathtt{code}\,(\mathtt{inr}(c), \mathtt{inr}(c'))$ is a proposition.

code (b, glue a). These are all equivalences, hence by univalence we define them as identity types. To show this, we show each is populated.

$\texttt{code}\left(\texttt{inl}(b), \texttt{ap}_{\texttt{glue}}(a')\right) : (\texttt{code}(\texttt{inl}(b), \texttt{inl}(f(a'))) = \texttt{code}(\texttt{inl}(b), \texttt{inr}(g(a'))))$ Because both sides of the identity type are propositions, to show that this equivalence holds it suffices to show that either $\texttt{code}(\texttt{inl}(b), \texttt{inl}(f(a')))$ and $\texttt{code}(\texttt{inl}(b), \texttt{inr}(g(a')))$ are both empty or both populated. This follows from post-composition with $\texttt{glue}(\texttt{a}')$ or its inverse.

code (glue a, b). $\mathtt{code}\left(\mathtt{ap}_{\mathtt{glue}}(a), \mathtt{inl}(b')\right) : \left(\mathtt{code}(\mathtt{inl}(f(a)), \mathtt{inl}(b')) = \mathtt{code}(\mathtt{inr}(g(a)), \mathtt{inl}(b'))\right)$ This follows from a similar argument to that above, with post-composition replaced with pre-composition.

code (c, glue a). $\mathtt{code}\left(\mathtt{inr}(c), \mathtt{ap}_{\mathtt{glue}}(a')\right) :\equiv (\mathtt{code}(\mathtt{inr}(c), \mathtt{inl}(f(a'))) = \mathtt{code}(\mathtt{inr}(c), \mathtt{inr}(g(a'))))$
This follows from a similar argument.

code (glue a , c). $\texttt{code}\left(\texttt{ap}_{\texttt{glue}}(a), \texttt{inr}(c')\right) :\equiv (\texttt{code}(\texttt{inl}(f(a)), \texttt{inr}(c')) = \texttt{code}(\texttt{inr}(g(a)), \texttt{inr}(c')))$
This follows from a similar argument.

code (glue a, glue a'). $\texttt{code}(\texttt{ap}_{\texttt{glue}}(a), \texttt{ap}_{\texttt{glue}}(a'))$ is uniquely determined because everything involved is a proposition. Because we have that the 1-cells in the square are equalities, there is only a single way to commute. This single way is how we define our 2-cell.

4. DEFINING encode

Now that `code` is defined, we define maps between it and the identity types inside of $B +_A C$. The first map we consider is encode, which is of type

$$\texttt{encode} : \prod_{x:\mathtt{B}+_{\mathtt{A}}\mathtt{C}} \prod_{y:\mathtt{B}+_{\mathtt{A}}\mathtt{C}} (x =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} y) \to \texttt{code}(x, y).$$

What are the non-empty identity types in $B +_A C$ that `encode` must map from?

- for $b, b' : B$ and $p : b =_B b'$, a path

$$\texttt{ap}_{\texttt{inl}}(p) : \texttt{inl}(b) =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} \texttt{inl}(b')$$

- for $c, c' : C$ and $q : c =_C c'$, a path

$$\texttt{ap}_{\texttt{inr}}(q) : \texttt{inr}(c) =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} \texttt{inl}(c')$$

- for $a : A$, a path

$$\texttt{glue}(a) : \texttt{inl}(fa) =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} \texttt{inr}(ga)$$

Thus it suffices to define the value of `encode` for $\texttt{ap}_{\texttt{inl}}(p)$, $\texttt{ap}_{\texttt{inr}}(q)$, and $\texttt{glue}(a)$. Also, since we are mapping out of identity types, we can use path induction.

- Define

$$\texttt{encode} \colon (\texttt{inl}(b) =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} \texttt{inl}(b)) \to \texttt{code}(\texttt{inl}(b), \texttt{inl}(b))$$

  by $\texttt{refl}_{\texttt{inl}b} \mapsto \texttt{refl}_{\texttt{inl}'b}$.
- Define

$$\texttt{encode} \colon (\texttt{inr}(c) =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} \texttt{inr}(c)) \to \texttt{code}(\texttt{inr}(c), \texttt{inr}(c))$$

  by $\texttt{refl}_c \mapsto \texttt{ap}_{\texttt{inr}'}(\texttt{refl}_c)$. Note that this `inr` is coming from the span used to define $\texttt{code}(\texttt{inr}(c), \texttt{inr}(c))$.
- Define

$$\texttt{encode} \colon (\texttt{inl}(fa) =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} \texttt{inr}(ga)) \to \texttt{code}(\texttt{inl}(fa), \texttt{inr}(ga))$$

  by $\texttt{glue}(a) \mapsto (\texttt{refl}_{fa}, \texttt{refl}_{ga})$

5. DEFINING decode

Define

$$\texttt{decode} : \prod_{x:\mathtt{B}+_{\mathtt{A}}\mathtt{C}} \prod_{y:\mathtt{B}+_{\mathtt{A}}\mathtt{C}} \texttt{code}(x,y) \to (x =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} y).$$

We can't use path induction because we are not mapping out of an identity type. The general strategy will be to give values for decode when feeding it the four different types of inputs— b/b', c/c', b/c, and c/b—as well as higher paths coming from glue. Of the former four, b/b' is the most difficult because of the complicated definition for $\texttt{code}(\texttt{inl}b, \texttt{inl}b')$. When dealing with glue, we must ensure naturality, meaning that there will be commuting diagrams to check.

- The type
$$\texttt{decode}(\texttt{inr}c, \texttt{inr}c')\colon \texttt{code}(\texttt{inr}c, \texttt{inr}c') \to (c =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}})c')$$
is given by $p \mapsto \texttt{ap}_{\texttt{inl}}p$
- The type
$$\texttt{decode}(\texttt{inl}b, \texttt{inr}c)\colon \texttt{code}(\texttt{inl}b, \texttt{inr}c) \to (b =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}})c)$$
is trivial unless $b = fa$ and $c = ga$ hold. Define
$$\texttt{decode}(\texttt{inl}b, \texttt{inr}c)\colon \texttt{code}(\texttt{inl}fa, \texttt{inr}ga) \to (fa =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}})ga)$$
given by $(p,q) \mapsto (\texttt{ap}_{\texttt{inr}}q^{-1})(\texttt{glue}a)(\texttt{ap}_{\texttt{inl}}p)$.
- The type
$$\texttt{decode}(\texttt{inr}c, \texttt{inl}b)\colon \texttt{code}(\texttt{inr}c, \texttt{inl}b) \to (c =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}})b)$$
is trivial unless $b = fa$ and $c = ga$ hold. Define
$$\texttt{decode}(\texttt{inr}c, \texttt{inl}b)\colon \texttt{code}(\texttt{inr}ga, \texttt{inl}fa) \to (ga =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}})fa)$$
given by $(q,p) \mapsto (\texttt{ap}_{\texttt{inl}}p^{-1})(\texttt{glue}a)(\texttt{ap}_{\texttt{inr}}q)$.
- The type
$$\texttt{decode}(\texttt{inl}b, \texttt{inl}b')\colon \texttt{code}(\texttt{inl}b, \texttt{inl}b') \to (b =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}})b')$$
is more involved because $\texttt{code}(\texttt{inl}b, \texttt{inl}b')$ is a pushout. To define a map out of a pushout, define it on the constructors. Hence, to define $\texttt{decode}(\texttt{inl}b, \texttt{inl}b')$ we need to produce values for
  - $\texttt{decode}(\texttt{inl}b, \texttt{inl}b')(\texttt{inl}p)$ for $p : b =_{\mathtt{B}} b'$
  - for each $a, a' : \mathtt{A}$, define $\texttt{decode}(\texttt{inl}b, \texttt{inl}b')(\texttt{inr}(p,q,r))$ for $p : (b +_{\mathtt{B}} fa)$, $q : (b' =_{\mathtt{B}} fa')$, and $r : (ga =_{\mathtt{C}} ga')$, and
  - $\texttt{ap}_{\texttt{decode}(\texttt{inl}b,\texttt{inl}b')}(\texttt{glue}a)$ for $a : \mathtt{A}$.

Now let's make the definitions depending on the four cases laid out in Remark 1

- Case 1. For $p : b =_{\mathtt{B}} b'$, then define $\texttt{decode}(\texttt{inl}b, \texttt{inl}b')(\texttt{inl}p) = \texttt{ap}_{\texttt{inl}}\texttt{inl}p$
- Case 2. For $\alpha(p,q,r) = p(\texttt{ap}_f r)q^{-1} : b =_{\mathtt{B}} b'$ and $\beta(p,q,r) = (p,q,\texttt{ap}_g r) : \sum_{a,a'}(b = b') \times (b' = fa') \times (ga = ga')$, then define $\texttt{decode}(\texttt{inl}b, \texttt{inl}b')(\texttt{inl}p(\texttt{ap}_f r)q^{-1})$ to be $\texttt{inl}p(\texttt{ap}_f r)q^{-1}$. Define $\texttt{decode}(\texttt{inl}b, \texttt{inl}b')(\texttt{inr}(p,q,\texttt{ap}_g r))$ to be $(\texttt{inr}q'^{-1})(\texttt{glue}a')^{-1}(\texttt{ap}_g r)(\texttt{glue}a)$ This is well defined because there is a $\texttt{glue}(p,q,r)$ connecting $\texttt{inl}(p(\texttt{ap}_f r)q^{-1})$ to $\texttt{inr}(p,q,\texttt{ap}_g r)$ in the $\texttt{code}(\texttt{inl}b, \texttt{inl}b')$ definition which we apply decode to, identifying their images.
- Case 3. Nothing to do, no constructors.

- Case 4. There is only one constructor, $\mathtt{inr}(p, q, \mathtt{ap}_g r)$. Define $\mathtt{decode}(\mathtt{inl}b, \mathtt{inl}b')(\mathtt{inr}(p, q, \mathtt{ap}_g r))$ to be the path $(\mathtt{inr}q'^{-1})(\mathtt{glue}a')^{-1}(\mathtt{ap}_g r)(\mathtt{glue}a)(\mathtt{inr}p)$.

To show decode is well defined, we must have that

$$\mathtt{decode}(\mathtt{code}(\mathtt{inl}fa, \mathtt{inl}b)) = \mathtt{decode}(\mathtt{code}(\mathtt{inr}ga, \mathtt{inl}b)).$$

Recall,

$$\mathtt{code}(\mathtt{inr}ga, \mathtt{inl}b) :\equiv \Sigma_{x:\mathtt{A}}(ga =_{\mathtt{C}} gx) \times (b =_{\mathtt{B}} fx)$$

and

$$\mathtt{code}(\mathtt{inl}fa, \mathtt{inl}b)$$

is the pushout of

$$\begin{array}{ccc} \sum_{x,x':\mathtt{A}}(fa =_{\mathtt{B}} fx) \times (b =_{\mathtt{B}} fx') \times (fa =_{\mathtt{B}} b) & \xrightarrow{\alpha} & (fa =_{\mathtt{B}} b) \\ \downarrow{\scriptstyle\beta} & & \\ \sum_{x,x':\mathtt{A}}(fa =_{\mathtt{B}} fx) \times (b =_{\mathtt{B}} fx') \times (gx =_{\mathtt{C}} gx') & & \end{array}$$

Clearly, $\mathtt{code}(\mathtt{inr}ga, \mathtt{inl}b)$ it empty unless $b =_{\mathtt{B}} fa'$. The same can be said for $\mathtt{code}(\mathtt{inl}fa, \mathtt{inl}b)$ because the constructors of the pushout require either a populated $fa =_{\mathtt{B}} b$ or a populated $b =_{\mathtt{B}} fx'$ for any $x' : \mathtt{A}$. So, we might as well assume that $b =_{\mathtt{B}} fa$. Why not $b =_{\mathtt{B}} fa'$ for some other $a' : \mathtt{A}$? I should think about this later.

Now, we have that

$$\mathtt{code}(\mathtt{inr}ga, \mathtt{inl}fa) :\equiv \Sigma_{x:\mathtt{A}}(ga =_{\mathtt{C}} gx) \times (fa =_{\mathtt{B}} fx)$$

and

$$\mathtt{code}(\mathtt{inl}fa, \mathtt{inl}fa)$$

is the pushout of

$$\begin{array}{ccc} \sum_{x,x':\mathtt{A}}(fa =_{\mathtt{B}} fx) \times (fa =_{\mathtt{B}} fx') \times (fa =_{\mathtt{B}} fa) & \xrightarrow{\alpha} & (fa =_{\mathtt{B}} fa) \\ \downarrow{\scriptstyle\beta} & & \\ \sum_{x,x':\mathtt{A}}(fa =_{\mathtt{B}} fx) \times (fa =_{\mathtt{B}} fx') \times (gx =_{\mathtt{C}} gx') & & \end{array}$$

which should just be $\{\mathtt{refl}_{fa}\}$ since $\mathtt{B}$ is a set. It follows that $\mathtt{decode}(\mathtt{code}(\mathtt{inl}fa, \mathtt{inl}fa))$ maps $\mathtt{refl}_{fa}$ to $(\mathtt{refl}_{fa})$ as required. We must now show that $\mathtt{decode}(\mathtt{code}(\mathtt{inr}ga, \mathtt{inl}fa))$ is also $\mathtt{refl}_{fa}$. But this follows from

$$\mathtt{code}(\mathtt{inr}ga, \mathtt{inl}fa) = (ga =_{\mathtt{C}} ga) \times (fa_{\mathtt{B}} fa) = \{(\mathtt{refl}_{ga}, \mathtt{refl}_{fa})$$

which is mapped via decode to $(\mathtt{ap}_{\mathtt{inl}}\mathtt{refl}_{fa})(\mathtt{glue}a)(\mathtt{ap}_{\mathtt{inr}}\mathtt{refl}_{ga}$, which is equal to $\mathtt{glue}a$. But since $\mathtt{glue}a$ is contractible, it is homotopic to $\mathtt{refl}_{fa}$.

6. ANOTHER SHOT

This section contains the latest strategy for proving that $\mathtt{B} +_{\mathtt{A}} \mathtt{C}$ is a pushout.

Thus far, we've defined all of the `code` spaces, and the map `encode`. It remains to define `decode`. To do this, we are fixing basepoints $\mathtt{inl}(b)$ and $\mathtt{inr}(c)$ in the pushout and will define

$$\mathtt{decode}(\mathtt{inl}b, -) \colon \prod_{x \colon \mathtt{B}+_{\mathtt{A}}\mathtt{C}} \mathtt{code}(\mathtt{inl}b, x) \to \mathtt{inl}b =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} x$$

and also

$$\mathtt{decode}(\mathtt{inr}c, -) \colon \prod_{x \colon \mathtt{B}+_{\mathtt{A}}\mathtt{C}} \mathtt{code}(\mathtt{inr}c, x) \to \mathtt{inr}c =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} x.$$

Why does this work? We've already shown that $\mathtt{code}(\mathtt{inl}b, x)$ and $\mathtt{code}(\mathtt{inr}c, x)$ are propositions for any x. It will follow that if $\mathtt{decode}(\mathtt{inl}b, -)$ and $\mathtt{decode}(\mathtt{inr}c, -)$ are equivalences, then we'll have that both $(\mathtt{inl}b =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} x)$ and $(\mathtt{inr}c =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} x)$ are also propositions. But once we have that the latter are propositions, we know that all identity spaces are propositions because any type $y =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} x$ is merely equal to $\mathtt{inl}b =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} x$ or $\mathtt{inr}c =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} x$.

6.1. Define $\mathtt{decode}(\mathtt{inl}b, -)$.

To define the map

$$\mathtt{decode}(\mathtt{inl}b, -) \colon \prod_{x \colon \mathtt{B}+_{\mathtt{A}}\mathtt{C}} \mathtt{code}(\mathtt{inl}b, x) \to \mathtt{inl}b =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} x \tag{1}$$

we use induction on $\mathtt{B} +_{\mathtt{A}} \mathtt{C}$ which, in practice, means that we will have the following two function types:

- $\mathtt{decode}(\mathtt{inl}b, -)(\mathtt{inl}b') \colon \mathtt{code}(\mathtt{inl}b, \mathtt{inl}b') \to (\mathtt{inl}b =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} \mathtt{inl}b')$
- $\mathtt{decode}(\mathtt{inl}b, -)(\mathtt{inl}c) \colon \mathtt{code}(\mathtt{inl}b, \mathtt{inr}c) \to (\mathtt{inl}b =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} \mathtt{inr}c$

and a witness

$$\begin{aligned}\mathtt{ap}_{\mathtt{decode}(\mathtt{inl}b,-)}(\mathtt{glue}a) \colon (&\mathtt{code}(\mathtt{inl}b, \mathtt{inl}fa) \to (\mathtt{inl}b =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} \mathtt{inl}fa) \\ &= (\mathtt{code}(\mathtt{inl}b, \mathtt{inl}ga) \to (\mathtt{inl}b =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} \mathtt{inl}ga)\end{aligned}$$

6.1.1. *Define* $\mathtt{decode}(\mathtt{inl}b, -)(\mathtt{inl}b')$.

When x is $\mathtt{inl}b' \colon \mathtt{B} +_{\mathtt{A}} \mathtt{C}$, then 3 becomes

$$\mathtt{decode}(\mathtt{inl}b, -) \colon \mathtt{code}(\mathtt{inl}b, \mathtt{inl}b') \to \mathtt{inl}b =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} \mathtt{inl}b'$$

This is defined by induction on $\mathtt{code}(\mathtt{inl}b, \mathtt{inl}b')$ which, recall, is the pushout of the span

$$\begin{tikzcd} \sum_{a,a':A}(b =_{\mathtt{B}} fa) \times (b' =_{\mathtt{B}} fa') \times (b =_{\mathtt{B}} b') \arrow[r, "\alpha"] \arrow[d, "\beta"'] & (b =_{\mathtt{B}} b') \\ \sum_{a,a':A}(b =_{\mathtt{B}} fa) \times (b' =_{\mathtt{B}} fa') \times (ga =_{\mathtt{C}} ga') & \end{tikzcd}$$

Since there are two pushouts running around, denote the inclusion maps associated to $\mathtt{B} +_{\mathtt{A}} \mathtt{C}$ by `inl` & `inr` and denote the inclusion maps assiciated to $\mathtt{code}(\mathtt{inl}b, \mathtt{inl}b')$ by $\mathtt{inl}'$ & $\mathtt{inr}'$.

We need to define values

- $\mathtt{decode}(\mathtt{inl}b, -)(\mathtt{inl}'p) \colon (b =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} b')$ for $p \colon b =_{\mathtt{B}} b'$
- $\mathtt{decode}(\mathtt{inl}b, -)(\mathtt{inr}'(q, r, s)) \colon (b = \mathtt{B} +_{\mathtt{A}} \mathtt{C}b')$ for $(q, r, s) \colon (b =_B fa) \times (b' =_{\mathtt{B}} fa') \times (ga =_{\mathtt{C}} ga')$

- $$\mathtt{ap}_{\mathtt{decode}(\mathtt{inl}b,-)}(\mathtt{glue}(t,u,v))\colon (\mathtt{decode}(\mathtt{inl}b,-)(\mathtt{inl}'(\alpha(t,u,v))) =_{\mathtt{inl}b=_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}}\mathtt{inl}b'} \mathtt{decode}(\mathtt{inl}b,--)(\mathtt{inr}'(\beta(t,u,v)))$$
  for $(t,u,v)\colon (b =_{\mathtt{B}} fa) \times (b' =_{\mathtt{B}} fa') \times (b =_{\mathtt{B}} b')$.

These are respecively

- let $\mathtt{decode}(\mathtt{inl}b,--)(\mathtt{inl}'p)$ be $\mathtt{ap}_{\mathtt{inl}}(p)$
- let $\mathtt{decode}(\mathtt{inl}b,--)(q,r,s)$ be the path $ap_{\mathtt{inl}}q; \mathtt{glue}a; \mathtt{ap}_{\mathtt{inr}}s; \mathtt{glue}^{-1}a'; r^{-1}$ in $\mathtt{inl}b =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} \mathtt{inl}b'$
- we show that $\mathtt{ap}_{\mathtt{decode}(\mathtt{inl}b,--)}(t,u,v)$ is trivial.

Lemma 1. *When inhabited, the type*

$$\mathtt{T} :\equiv \sum_{a,a'\,:\,\mathtt{A}} (b =_{\mathtt{B}} fa) \times (b' =_{\mathtt{B}} fa') \times (b =_{\mathtt{B}} b')$$

is equivalent to $\mathtt{1}$.

Proof. Let $(t,u,v)\colon \mathtt{T}$. Then $t^{-1}; v; u\colon fa =_{\mathtt{B}} fa'$. By the injectivity of f, there exists $x\colon a =_{\mathtt{A}} a'$. We can reduce $\mathtt{T}$ to

$$\sum_{a'',a'''\,:\,\mathtt{A}} (fa =_{\mathtt{B}} fa'') \times (fa' =_{\mathtt{B}} fa''') \times (fa =_{\mathtt{B}} fa').$$

The injectivity of f allows further reduction to

$$(a =_{\mathtt{A}} a'') \times (a' =_{\mathtt{A}} a''') \times (a =_{\mathtt{A}} a').$$

This is a proposition since $\mathtt{A}$ is a set, but since we are starting with $\mathtt{T}$ inhabited, we further reduce the above to $\mathtt{1}$. □

Lemma 2. *When* $\mathtt{T}$ *from above is inhabited, then* $(b =_{\mathtt{B}} b') \sim \mathtt{1}$.

Proof. Combine the two facts: $\mathtt{B}$ is a set and $\mathtt{T}$ being inhabitied implies that $b =_{\mathtt{B}} b'$ is inhabitied. □

Lemma 3. *When the type* $\mathtt{T}$ *from above is inhabited, the type*

$$\mathtt{S} :\equiv \sum_{a,a'\,:\,\mathtt{A}} (b =_{\mathtt{B}} fa) \times (b' =_{\mathtt{B}} fa') \times (ga =_{\mathtt{C}} ga')$$

is equivalent to $\mathtt{1}$.

Proof. That $\mathtt{T}$ being inhabited implies that $\mathtt{S}$ is. But $\mathtt{S}$ is equivalent to

$$(fa =_{\mathtt{B}} fa) \times (fa' =_{\mathtt{B}} fa') \times (ga =_{\mathtt{C}} ga').$$

Because $\mathtt{B}$ is a set, the first two factors above are equivalent to $\mathtt{1}$, whence $\mathtt{S}$ is equivalent to $ga =_{\mathtt{C}} ga'$. That is inhabitied by assumption and $\mathtt{C}$ is a set, and so it—and therefore $\mathtt{S}$ — is equivalent to $\mathtt{1}$. □

Theorem 1. *When* $\mathtt{T}$ *is inhabited,* $\mathtt{code}(\mathtt{inl}b, \mathtt{inl}b')$ *is equivalent to* $\mathtt{1}$.

Proof. The preceeding three lemmas imply the span over which we define $\mathtt{code}(\mathtt{inl}b, \mathtt{inl}b')$ is equivalent to the span

$$\mathtt{1} \leftarrow \mathtt{1} \rightarrow \mathtt{1}$$

whose pushout is $\mathtt{1}$. □

We conclude that $\mathtt{code}(\mathtt{inl}b, \mathtt{inl}b')$ is a 0-type and so the relevant part of $\mathtt{decode}$ is automatically continuous/functorial.

6.1.2. *Define* $\mathtt{decode}(\mathtt{inl}b, -)(\mathtt{inl}c)$. When x is $\mathtt{inr}c \colon \mathtt{B} +_{\mathtt{A}} \mathtt{C}$, then (3) becomes

$$\mathtt{decode}(\mathtt{inl}b, -) \colon \mathtt{code}(\mathtt{inl}b, \mathtt{inr}c) \to (\mathtt{inl}b =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} \mathtt{inr}c)$$

We have that

$$\mathtt{code}(\mathtt{inl}b, \mathtt{inr}c) = \sum_{a \colon \mathtt{A}} (b =_{\mathtt{B}} fa) \times (c =_{\mathtt{C}} ga)$$

Define

$$\mathtt{decode}(\mathtt{inl}b, -)(p, q) :\equiv \mathtt{inl}p; \mathtt{glue}a; \mathtt{inr}q$$

6.1.3. *Define* $\mathtt{ap}_{\mathtt{decode}(\mathtt{inl}b,-)}(\mathtt{glue}a)$. Right now, we define

$$\begin{aligned} \mathtt{ap}_{\mathtt{decode}(\mathtt{inl}b,-)}(\mathtt{glue}a) \colon & (\mathtt{code}(\mathtt{inl}b, \mathtt{inl}fa) \to (\mathtt{inl}b =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} \mathtt{inl}fa) \\ & = (\mathtt{code}(\mathtt{inl}b, \mathtt{inl}ga) \to (\mathtt{inl}b =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} \mathtt{inl}ga) \end{aligned}$$

To define this is to construct a commuting square

$$\begin{tikzcd} \mathtt{code}(\mathtt{inl}b, \mathtt{inl}fa) \arrow[r, "\mathtt{decode}(\mathtt{inl}b,-)(\mathtt{inl}b')"] \arrow[d, "\theta"'] & (\mathtt{inl}b =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} \mathtt{inl}fa) \arrow[d, "\theta'"'] \\ \mathtt{code}(\mathtt{inl}b, \mathtt{inl}ga) \arrow[r, "\mathtt{decode}(\mathtt{inl}b,-)(\mathtt{inl}c)"] & (\mathtt{inl}b =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} \mathtt{inl}ga) \end{tikzcd}$$

We have already defined the two $\mathtt{decode}$ maps. The map θ' is post-composition by $\mathtt{glue}a$. The map θ requires induction to define because we are mapping out of a pushout. Therefore, we need the following three values to define θ:

- $\theta(\mathtt{inl}'p)$ for $p \colon b =_B fa$
- $\theta(\mathtt{inr}'(q, r, s))$ for $(q, r, s) \colon \sum_{a', a'' \colon \mathtt{A}} (b =_{\mathtt{B}} fa') \times (fa =_B fa'') \times (ga' =_C ga'')$
- $\mathtt{ap}_\theta(\mathtt{glue}(t, u, v)) \colon \theta(\mathtt{inl}'(\alpha(t, u, v))) = \theta(\mathtt{inr}'(\beta(t, u, v)))$.

A quick remark on notation: since f is monic and $\mathtt{B}$ is a set, we can pull back any paths of form $r \colon fa =_{\mathtt{B}} fa'$ to a path $\hat{r} \colon a = \mathtt{A}a'$.

Define $\theta(\mathtt{inl}'p)$ for $p \colon b =_B fa$ to be $(p, \mathtt{refl}_{ga})$. Define $\theta(q, r, s)$ to be $(q, \mathtt{ap}_g(\hat{r}); s^{-1})$.

Now we know that $\mathtt{ap}_\theta(\mathtt{glue}(t, u, v))$ is a path from $\theta(\mathtt{inl}'(v)) = (v, \mathtt{refl}_{ga})$ to $\theta(\mathtt{inr}'(\beta(t, u, v))) = \theta(\mathtt{inr}'(t, u, \mathtt{ap}_g(\hat{v}))) = (t, \mathtt{ap}_g(\hat{u}); \mathtt{ap}_g(\hat{v})^{-1}) = (t, \mathtt{ap}_g(\hat{u}; \hat{v}^{-1}))$. With this in mind, we will define $\mathtt{ap}_\theta(\mathtt{glue}(t, u, v))$ to be post-composition by

$$(\mathtt{ap}_f(\hat{u}; \hat{v}^{-1}), \mathtt{ap}_g(\hat{u}; \hat{v}^{-1})).$$

Now we need to check that the diagram commutes. Since $\mathtt{code}(\mathtt{inl}b, \mathtt{inl}fa)$ is a set, it suffices to check that

$$\theta'(\mathtt{decode}(\mathtt{inl}b,-)(\mathtt{inl}fa)(x)) =_{\mathtt{inl}b =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} \mathtt{inr}ga} \mathtt{decode}(\mathtt{inl}b,-)(\mathtt{inr}ga)(\theta(x))$$

where $x\colon \mathtt{code}(\mathtt{inl}b, \mathtt{inl}fa)$ is

- $\mathtt{inl}'p$ for $p\colon b =_{\mathtt{B}} fa$, and
- $\mathtt{inr}'(q,r,s)$ for $(q,r,s)\colon \sum_{a',a''\colon \mathtt{A}} (b =_{\mathtt{B}} fa') \times (fa =_{\mathtt{B}} fa'') \times (ga' =_{\mathtt{C}} ga'')$

and we also need to check that

$$\mathtt{ap}_{\theta';\mathtt{decode}(\mathtt{inl}b,-)(\mathtt{inl}fa))}(\mathtt{glue}(t,u,v)) =_{\mathtt{inl}b =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} \mathtt{inr}ga} \mathtt{ap}_{\mathtt{decode}(\mathtt{inl}b,-)(\mathtt{inr}ga);\theta}(\mathtt{glue}(t,u,v))$$

Set $x :\equiv \mathtt{inl}'p$. Then

$$\begin{aligned}\theta'(\mathtt{decode}(\mathtt{inl}b,-)(\mathtt{inl}fa)(\mathtt{inl}'p)) &= \theta'((\mathtt{ap}_{\mathtt{inl}'}p)) \\ &= \mathtt{ap}_{\mathtt{inl}'}p; \mathtt{glue}a\end{aligned}$$

Also,

$$\begin{aligned}\mathtt{decode}(\mathtt{inl}b,-)(\mathtt{inr}ga)(\theta(\mathtt{inl}'p)) &= \mathtt{decode}(\mathtt{inl}b,-)(\mathtt{inr}ga)((p, \mathtt{refl}_{ga})) \\ &= \mathtt{ap}_{\mathtt{inl}'}p; \mathtt{glue}a; \mathtt{ap}_{\mathtt{inr}'}\mathtt{refl}_{ga} \\ &= \mathtt{ap}_{\mathtt{inl}'}p; \mathtt{glue}a; \mathtt{refl}_{\mathtt{inr}'ga} \\ &= \mathtt{ap}_{\mathtt{inl}'}p; \mathtt{glue}a\end{aligned}$$

Therefore, the square commutes on $\mathtt{inl}'p$

Set $x :\equiv \mathtt{inr}'(q,r,s)$. Then

$$\begin{aligned}\theta'(\mathtt{decode}(\mathtt{inl}b,-)(\mathtt{inl}fa)(\mathtt{inr}'(q,r,s))) &= \theta'(\mathtt{ap}_{\mathtt{inl}}q; \mathtt{glue}a'; \mathtt{ap}_{\mathtt{inr}}s; \mathtt{glue}a''; \mathtt{ap}_{\mathtt{inl}}r^{-1}) \\ &= \mathtt{ap}_{\mathtt{inl}}q; \mathtt{glue}a'; \mathtt{ap}_{\mathtt{inr}}s; \mathtt{glue}^{-1}a''; \mathtt{ap}_{\mathtt{inl}}r^{-1}; \mathtt{glue}a \\ &= \mathtt{ap}_{\mathtt{inl}}q; \mathtt{glue}a'; \mathtt{ap}_{\mathtt{inr}}s; \mathtt{glue}^{-1}a''; \mathtt{ap}_{\mathtt{inl}}\mathtt{ap}_f\hat{r}^{-1}; \mathtt{glue}a\end{aligned}$$

Also,

$$\begin{aligned}\mathtt{decode}(\mathtt{inl}b,-)(\mathtt{inr}ga)(\theta(\mathtt{inr}'(q,r,s))) &= \mathtt{decode}(\mathtt{inl}b,-)(\mathtt{inr}ga)(q, \mathtt{ap}_g(\hat{r}; s^{-1})) \\ &= \mathtt{ap}_{\mathtt{inl}}q; \mathtt{glue}a'; (\mathtt{ap}_{\mathtt{inr}}(\mathtt{ap}_g\hat{r}; s^{-1})^{-1}) \\ &= \mathtt{ap}_{\mathtt{inl}}q; \mathtt{glue}a'; \mathtt{ap}_{\mathtt{inr}}(\mathtt{ap}_g(\hat{r}); (\mathtt{ap}_g(s^{-1}))^{-1}) \\ &= \mathtt{ap}_{\mathtt{inl}}q; \mathtt{glue}a'; \mathtt{ap}_{\mathtt{inr}}(s^{-1})^{-1}; \mathtt{ap}_g(\hat{r})^{-1} \\ &= \mathtt{ap}_{\mathtt{inl}}q; \mathtt{glue}a'; \mathtt{ap}_{\mathtt{inr}}(s^{-1})^{-1}; \mathtt{ap}_g(\hat{r})^{-1} \\ &= \mathtt{ap}_{\mathtt{inl}}q; \mathtt{glue}a'; \mathtt{ap}_{\mathtt{inr}}s; \mathtt{ap}_g(\hat{r})^{-1}\end{aligned}$$

We just need to check that

$$\mathtt{ap}_g(\hat{r})^{-1} = \mathtt{glue}^{-1}a''; \mathtt{ap}_{\mathtt{inl}}\mathtt{ap}_f\hat{r}^{-1}; \mathtt{glue}a$$

But this follows from the continuity of $\mathtt{glue}$, that is, it preserves paths. Therefore, the square commutes on $\mathtt{inr}'(q,r,s)$.

It remains to check that

$$\mathtt{ap}_{\theta';\mathtt{decode}(\mathtt{inl}b,-)(\mathtt{inl}fa))}(\mathtt{glue}(t,u,v)) =_{\mathtt{inl}b =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} \mathtt{inr}ga} \mathtt{ap}_{\mathtt{decode}(\mathtt{inl}b,-)(\mathtt{inr}ga);\theta}(\mathtt{glue}(t,u,v))$$

for $(t, u, v)\colon \sum_{a', a''\colon \mathtt{A}} (b =_{\mathtt{B}} fa') \times (fa =_{\mathtt{B}} fa'') \times (b =_{\mathtt{B}} fa)$. Here, we make some reductions.

- Because f is monic, we get that $fa =_{\mathtt{B}} fa''$ reduces to $fa =_{\mathtt{B}} fa$. Since $\mathtt{B}$ is a set, we can take $u = \mathtt{refl}_{fa}$.
- The type $b =_{\mathtt{B}} fa'$ reduces to $fa =_{\mathtt{B}} fa$ because $\mathtt{B}$ is monic, so $t = \mathtt{refl}_{fa}$.
- The type $b =_{\mathtt{B}} fa$ reduces to $fa =_{\mathtt{B}} fa$ because $\mathtt{B}$ is monic, so $v = \mathtt{refl}_{fa}$.

Hence without loss of generality, we can take $(t, u, v) = (\mathtt{refl}_{fa}, \mathtt{refl}_{fa}, \mathtt{refl}_{fa})$. Therefore, it suffices to check that

$$\mathtt{ap}_{\theta';\mathtt{decode}(\mathtt{inl}b,-)(\mathtt{inl}fa))}(\mathtt{glue}(\mathtt{refl}_{fa}, \mathtt{refl}_{fa}, \mathtt{refl}_{fa})) =_{\mathtt{inl}b =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} \mathtt{inr}ga} \mathtt{ap}_{\mathtt{decode}(\mathtt{inl}b,-)(\mathtt{inr}ga);\theta}(\mathtt{glue}(\mathtt{refl}_{fa}, \mathtt{ref}$$

But because the square commutes on points as shown above, the two paths on the left and right of the above equation are certainly parallel. Then functorality gives us that $\mathtt{refl}_{fa}$ is preserved. Hence the square commutes.

6.2. **Define decode(c,-).** To define the map

$$\mathtt{decode}(\mathtt{inr}c, -)\colon \prod_{x\colon \mathtt{B}+_{\mathtt{A}}\mathtt{C}} \mathtt{code}(\mathtt{inr}c, x) \to \mathtt{inr}c =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} x \tag{2}$$

we use induction on $\mathtt{B} +_{\mathtt{A}} \mathtt{C}$ which, in practice, means that we will have the following two function types:

- $\mathtt{decode}(\mathtt{inr}c, -)(\mathtt{inl}b)\colon \mathtt{code}(\mathtt{inr}c, \mathtt{inl}b) \to (\mathtt{inr}c =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} \mathtt{inl}b)$
- $\mathtt{decode}(\mathtt{inr}c, -)(\mathtt{inr}c')\colon \mathtt{code}(\mathtt{inr}c, \mathtt{inr}c') \to (\mathtt{inr}c =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} \mathtt{inr}c'$

and a witness

$$\begin{aligned}\mathtt{ap}_{\mathtt{decode}(\mathtt{inr}c,-)}(\mathtt{glue}a)\colon (\mathtt{code}(\mathtt{inr}c, \mathtt{inl}fa) &\to (\mathtt{inr}c =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} \mathtt{inl}fa) \\ = (\mathtt{code}(\mathtt{inr}c, \mathtt{inl}ga) &\to (\mathtt{inr}c =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} \mathtt{inl}ga)\end{aligned}$$

6.2.1. *Define* $\mathtt{decode}(\mathtt{inr}c, -)(\mathtt{inl}b)$*.* Here, we define a map

$$\mathtt{decode}(\mathtt{inr}c, -)(\mathtt{inl}b)\colon \mathtt{code}(\mathtt{inr}c, \mathtt{inl}b) \to (\mathtt{inr}c =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} \mathtt{inl}b).$$

Recall that $\mathtt{code}(\mathtt{inr}c, \mathtt{inl}b) :\equiv \sum_{a\colon \mathtt{A}} (c =_{\mathtt{C}} ga) \times (b =_{\mathtt{B}} fa)$. Thus for any (p, q) in $\mathtt{code}(\mathtt{inr}c, \mathtt{inl}b)$, we define a path $\mathtt{decode}(\mathtt{inr}c, -)(\mathtt{inl}b)(p, q)$ in $\mathtt{B} +_{\mathtt{A}} \mathtt{C}$ from $\mathtt{inr}c$ to $\mathtt{inl}b$. Take this path to be $\mathtt{inr}p; \mathtt{glue}^{-1}a; \mathtt{inl}q^{-1}$.

6.2.2. *Define* $\mathtt{decode}(\mathtt{inr}c, -)(\mathtt{inr}c')$*.* Here, we define a map

$$\mathtt{decode}(\mathtt{inr}c, -)(\mathtt{inr}c')\colon \mathtt{code}(\mathtt{inr}c, \mathtt{inr}c') \to (\mathtt{inr}c =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} \mathtt{inr}c').$$

Recall that

$$\mathtt{code}(\mathtt{inr}c, \mathtt{inr}c') :\equiv \sum_{a\colon \mathtt{A}} (c =_{\mathtt{C}} ga) \times (c' =_{\mathtt{C}} ga') \times (fa =_{\mathtt{B}} fa')$$

Thus for any (p, q, r) in $\mathtt{code}(\mathtt{inr}c, \mathtt{inr}c')$, we define a path $\mathtt{decode}(\mathtt{inr}c, -)(\mathtt{inr}c')(p, q, r)$ in $\mathtt{B} +_{\mathtt{A}} \mathtt{C}$ from $\mathtt{inr}c$ to $\mathtt{inr}c'$. Take this path to be $\mathtt{inr}p; \mathtt{glue}^{-1}a; \mathtt{inl}r\mathtt{glue}a'; \mathtt{inr}q^{-1}$.

6.2.3. *Define* $\mathtt{ap}_{\mathtt{decode}(\mathtt{inr}c,-)}(\mathtt{glue}a)$. Here, we define a path

$$\begin{aligned}\mathtt{ap}_{\mathtt{decode}(\mathtt{inr}c,-)}(\mathtt{glue}a)\colon (\mathtt{code}(\mathtt{inr}c,\mathtt{inl}fa) &\to (\mathtt{inr}c =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} \mathtt{inl}fa)\\ &= (\mathtt{code}(\mathtt{inr}c,\mathtt{inl}ga) \to (\mathtt{inr}c =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} \mathtt{inl}ga).\end{aligned}$$

Placing the definitions in for the two `code` expressions, this path is constructed as a commuting square

$$\begin{tikzcd}
\sum_{a'\colon \mathtt{A}} (c =_{\mathtt{C}} ga') \times (fa =_{\mathtt{B}} fa') \arrow[r, "\mathtt{decode}"] \arrow[d, "\theta"'] & (\mathtt{inr}c =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} \mathtt{inl}fa) \arrow[d, "\theta'"] \\
\sum_{a',a''\colon \mathtt{A}} (c =_{\mathtt{C}} ga') \times (ga =_{\mathtt{C}} ga') \times (fa' =_{\mathtt{B}} fa'') \arrow[r, "\mathtt{decode}"] & \mathtt{inr}c =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} \mathtt{inl}ga
\end{tikzcd}$$

The easier to define is θ' which is simply concatenation with $\mathtt{glue}a$. To define θ, we send $(p,q) \mapsto (p, \mathtt{refl}_{ga}, q)$. Now we need to see if this square commutes. We have that

$$\begin{aligned}\theta'(\mathtt{decode}(\mathtt{inr}c,-)(\mathtt{inl}fa)(p,q)) &= \theta'(\mathtt{inr}p; \mathtt{glue}^{-1}a'; \mathtt{inl}q^{-1})\\ &= \mathtt{inr}p; \mathtt{glue}^{-1}a'; \mathtt{inl}q^{-1}; \mathtt{glue}a.\end{aligned}$$

We also have that

$$\begin{aligned}\mathtt{decode}(\mathtt{inr}c,-)(\mathtt{inl}ga)(\theta(p,q)) &= \mathtt{decode}(\mathtt{inr}c,-)(\mathtt{inl}ga)(p,\mathtt{refl}_{ga},q)\\ &= \mathtt{inr}p; \mathtt{glue}^{-1}a'; \mathtt{inl}q^{-1}; \mathtt{glue}a; \mathtt{inrrefl}_{ga}\\ &= \mathtt{inr}p; \mathtt{glue}^{-1}a'; \mathtt{inl}q^{-1}; \mathtt{glue}a\end{aligned}$$

Hence the square commutes.

7. Composing encode and decode

Let's compose `encode` and `decode` maps to see if they are mutual inverses.

Are we required to define this using induction on the pushout or no? We've already defined encode and decode, so maybe it's unnecessary. For now, I'll include both.

7.1. encode;decode with induction.

First, we look at the composite

$$\mathtt{encode};\mathtt{decode}\colon \prod_{x,y\colon \mathtt{B}+_{\mathtt{A}}\mathtt{C}} x =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} yx =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} y$$

This map can be computed using single, as opposed to double, induction on $\mathtt{B} +_{\mathtt{A}} \mathtt{C}$. That is, we can instead compute values

- $\mathtt{encode};\mathtt{decode}(\mathtt{inl}b,-)\colon \prod_{x\colon \mathtt{B}+_{\mathtt{A}}\mathtt{C}} (\mathtt{inl}b =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} x) \to (\mathtt{inl}b =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} x)$
- $\mathtt{encode};\mathtt{decode}(\mathtt{inr}c,-)\colon \prod_{x\colon \mathtt{B}+_{\mathtt{A}}\mathtt{C}} \mathtt{inr}c =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} x \to (\mathtt{inr}c =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} x)$
- $\mathtt{ap}_{\mathtt{encode};\mathtt{decode}}(\mathtt{glue}a)\colon \mathtt{encode};\mathtt{decode}(\mathtt{inl}fa,-) =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} \mathtt{encode};\mathtt{decode}(\mathtt{inr}ga,-)$

The first two of these will be computed using induction on $\mathtt{B}+_{\mathtt{A}}\mathtt{C}$. To check $\mathtt{encode};\mathtt{decode}(\mathtt{inl}b,-)$, we need the following:

- $\texttt{encode};\texttt{decode}(\texttt{inl}b,-)(\texttt{inl}b')\colon (\texttt{inl}b =_{\texttt{B}+_\texttt{A}\texttt{C}} \texttt{inl}b') \to (\texttt{inl}b =_{\texttt{B}+_\texttt{A}\texttt{C}} \texttt{inl}b')$. By path induction, suffice to check $\texttt{refl}_{\texttt{inl}b}$:
$$\texttt{refl}_{\texttt{inl}b} \mapsto \texttt{ap}_{\texttt{inl}'}\texttt{refl}_{\texttt{inl}b} = \texttt{refl}_{\texttt{inl}'b} \mapsto \texttt{ap}_{\texttt{inl}}\texttt{refl}_{\texttt{inl}'b} = \texttt{refl}_{\texttt{inl}b}.$$
So this one checks out.
- $\texttt{encode};\texttt{decode}(\texttt{inl}b,-)(\texttt{inl}b')\colon (\texttt{inl}b =_{\texttt{B}+_\texttt{A}\texttt{C}} \texttt{inr}c) \to (\texttt{inl}b =_{\texttt{B}+_\texttt{A}\texttt{C}} \texttt{inr}c)$. This is only non-trivial when $b =_\texttt{B} fa$ and $c =_\texttt{C} ga$. Hence
$$\texttt{glue}a \mapsto (\texttt{refl}_{fa}, \texttt{refl}_{ga}) \mapsto \texttt{ap}_{\texttt{inr}}\texttt{refl}_{ga};\texttt{glue}a;\texttt{ap}_{\texttt{inl}}\texttt{refl}_{fa} = \texttt{glue}a.$$
This one checks out too.
- $\texttt{ap}_{\texttt{encode};\texttt{decode}(\texttt{inl}b,-)}(\texttt{glue}a)\texttt{encode};\texttt{decode}(\texttt{inl}b,-)(\texttt{inl}fa) = \texttt{encode};\texttt{decode}(\texttt{inl}b,-)(\texttt{inr}ga)$. Note that the left hand side of this path space lives in
$$\texttt{inl}b =_{\texttt{B}+_\texttt{A}\texttt{C}} \texttt{inl}fa$$
and the righthand side lives in
$$\texttt{inl}b =_{\texttt{B}+_\texttt{A}\texttt{C}} \texttt{inr}ga$$
There is an equivalence between these path spaces obtained by concatenating $\texttt{glue}a$ and its inverse. Applying univalence to this equivalence gives us $\texttt{ap}_{\texttt{encode};\texttt{decode}(\texttt{inl}b,)}\texttt{glue}a$.

That completes the definition of $\texttt{encode};\texttt{decode}(\texttt{inl}b,-)$. Now, move on to $\texttt{encode};\texttt{decode}(\texttt{inr}c,-)$, which as we are inducting over $\texttt{B} +_\texttt{A} \texttt{C}$, requires obtaining three values:

- $\texttt{encode};\texttt{decode}(\texttt{inr}c,-)(\texttt{inl}b)\colon (\texttt{inr}c =_{\texttt{B}+_\texttt{A}\texttt{C}} \texttt{inl}b) \to (\texttt{inr}c =_{\texttt{B}+_\texttt{A}\texttt{C}} \texttt{inl}b)$. This is symmetric to the above case, so checks out.
- $\texttt{encode};\texttt{decode}(\texttt{inr}c,-)(\texttt{inr}c')\colon (\texttt{inr}c =_{\texttt{B}+_\texttt{A}\texttt{C}} \texttt{inr}c') \to (\texttt{inr}c =_{\texttt{B}+_\texttt{A}\texttt{C}} \texttt{inr}c')$. By path induction, that
$$\texttt{refl}_{\texttt{inr}c} \mapsto \texttt{refl}_c \mapsto \texttt{ap}_{\texttt{inr}}\texttt{refl}_c = \texttt{refl}_{\texttt{inr}c}$$
suffices.
- $\texttt{ap}_{\texttt{encode};\texttt{decode}(\texttt{inr}c,-)}(\texttt{glue}a)\colon \texttt{encode};\texttt{decode}(\texttt{inl}b,-)(\texttt{inl}fa) = \texttt{encode};\texttt{decode}(\texttt{inr}c,-)(\texttt{inr}ga)$. Note that the left hand side of this path space is
$$\texttt{inr}c =_{\texttt{B}+_\texttt{A}\texttt{C}} \texttt{inl}fa$$
and the righthand side is
$$\texttt{inr}c =_{\texttt{B}+_\texttt{A}\texttt{C}} \texttt{inr}ga$$
There is an equivalence between these path spaces obtained by concatenating $\texttt{glue}a$ and its inverse. Applying univalence to this equivalence gives us $\texttt{ap}_{\texttt{encode};\texttt{decode}(\texttt{inr}c,)}\texttt{glue}a$.

The final thing to construct is a path $\texttt{ap}_{\texttt{encode};\texttt{decode}}(\texttt{glue}a)\colon \texttt{encode};\texttt{decode}(\texttt{inl}fa,-) =_{\texttt{B}+_\texttt{A}\texttt{C}} \texttt{encode};\texttt{decode}(\texttt{inr}ga,-)$. Univalence allows us to instead find and equivalence
$$\prod_{x\colon \texttt{B}+_\texttt{A}\texttt{C}} (\texttt{inl}fa =_{\texttt{B}+_\texttt{A}\texttt{C}} x) \simeq \prod_{x\colon \texttt{B}+_\texttt{A}\texttt{C}} (\texttt{inr}ga =_{\texttt{B}+_\texttt{A}\texttt{C}} x)$$
This equivalence is obtained by concatenation by $\texttt{glue}a$ and its inverse.

And so, we have prove that `decode` is a section for `encode`. The opposite direction remains.

7.2. **encode;decode without induction.** First, we look at the composite

$$\mathtt{encode};\mathtt{decode}\colon \prod_{x,y\colon \mathtt{B}+_\mathtt{A}\mathtt{C}} x =_{\mathtt{B}+_\mathtt{A}\mathtt{C}} yx =_{\mathtt{B}+_\mathtt{A}\mathtt{C}} y$$

To show that this map is the identity up to homotopy, we compute two values:

- $\mathtt{encode};\mathtt{decode}(\mathtt{inl}b,-)\colon \prod_{x\colon \mathtt{B}+_\mathtt{A}\mathtt{C}} (\mathtt{inl}b =_{\mathtt{B}+_\mathtt{A}\mathtt{C}} x) \to (\mathtt{inl}b =_{\mathtt{B}+_\mathtt{A}\mathtt{C}} x)$
- $\mathtt{encode};\mathtt{decode}(\mathtt{inr}c,-)\colon \prod_{x\colon \mathtt{B}+_\mathtt{A}\mathtt{C}} \mathtt{inr}c =_{\mathtt{B}+_\mathtt{A}\mathtt{C}} x \to (\mathtt{inr}c =_{\mathtt{B}+_\mathtt{A}\mathtt{C}} x)$

but computing these values actually requires the computation of the following four maps:

- $\mathtt{encode};\mathtt{decode}(\mathtt{inl}b,-)(\mathtt{inl}b')\colon \mathtt{code}(\mathtt{inl}b,\mathtt{inl}b') \to (\mathtt{inl}b =_{\mathtt{B}+_\mathtt{A}\mathtt{C}} \mathtt{inl}b')$. By path induction, it suffices to check $\mathtt{refl}_{\mathtt{inl}b}$:

  $$\mathtt{refl}_{\mathtt{inl}b} \mapsto \mathtt{ap}_{\mathtt{inl}'}\mathtt{refl}_{\mathtt{inl}b} = \mathtt{refl}_{\mathtt{inl}'b} \mapsto \mathtt{ap}_{\mathtt{inl}}\mathtt{refl}_{\mathtt{inl}'b} = \mathtt{refl}_{\mathtt{inl}b}.$$

  So this one checks out.
- $\mathtt{encode};\mathtt{decode}(\mathtt{inl}b,-)(\mathtt{inr}c)\colon \mathtt{code}(\mathtt{inl}b,\mathtt{inr}c) \to (\mathtt{inl}b =_{\mathtt{B}+_\mathtt{A}\mathtt{C}} \mathtt{inl}c)$. glue(a) *is* the only thing to check here, right? This is only non-trivial when $b =_\mathtt{B} fa$ and $c =_\mathtt{C} ga$. Hence

  $$\mathtt{glue}a \mapsto (\mathtt{refl}_{fa},\mathtt{refl}_{ga}) \mapsto \mathtt{ap}_{\mathtt{inr}}\mathtt{refl}_{ga};\mathtt{glue}a;\mathtt{ap}_{\mathtt{inl}}\mathtt{refl}_{fa} = \mathtt{glue}a.$$

  This one checks out too.
- $\mathtt{encode};\mathtt{decode}(\mathtt{inr}c,-)(\mathtt{inl}b)\colon \mathtt{code}(\mathtt{inr}c,\mathtt{inl}b) \to (\mathtt{inr}c =_{\mathtt{B}+_\mathtt{A}\mathtt{C}} \mathtt{inl}b)$. This is symmetric to the above case, so checks out.
- $\mathtt{encode};\mathtt{decode}(\mathtt{inr}c,-)(\mathtt{inr}c')\colon \mathtt{code}(\mathtt{inr}c,\mathtt{inr}c') \to (\mathtt{inr}c =_{\mathtt{B}+_\mathtt{A}\mathtt{C}} \mathtt{inr}c')$. By path induction,

  $$\mathtt{refl}_{\mathtt{inr}c} \mapsto \mathtt{refl}_c \mapsto \mathtt{ap}_{\mathtt{inr}}\mathtt{refl}_c = \mathtt{refl}_{\mathtt{inr}c}$$

And so, we have prove that `decode` is a section for `encode`. The opposite direction remains.

7.3. **decode;encode.** Now, we look at the composite

$$\mathtt{decode};\mathtt{encode}\colon \prod_{x,y\colon \mathtt{B}+_\mathtt{A}\mathtt{C}} \mathtt{code}(x,y) \to \mathtt{code}(x,y)$$

We can compute this composite by computing values

- $\mathtt{decode};\mathtt{encode}(\mathtt{inl}b,-)(\mathtt{inl}b')\colon \mathtt{code}(\mathtt{inl}b,\mathtt{inl}b') \to \mathtt{code}(\mathtt{inl}b,\mathtt{inl}b')$
- $\mathtt{decode};\mathtt{encode}(\mathtt{inl}b,-)(\mathtt{inr}c)\colon \mathtt{code}(\mathtt{inl}b,\mathtt{inr}c) \to \mathtt{code}(\mathtt{inl}b,\mathtt{inr}c)$
- $\mathtt{decode};\mathtt{encode}(\mathtt{inr}c,-)(\mathtt{inl}b)\colon \mathtt{code}(\mathtt{inr}c,\mathtt{inl}b) \to \mathtt{code}(\mathtt{inr}c,\mathtt{inl}b)$
- $\mathtt{decode};\mathtt{encode}(\mathtt{inr}c,-)(\mathtt{inr}c')\colon \mathtt{code}(\mathtt{inr}c,\mathtt{inr}c') \to \mathtt{code}(\mathtt{inr}c,\mathtt{inr}c')$

But these maps are all identity since $\mathtt{code}(x,y)$ is a proposition for $x,y\colon \mathtt{B}+_\mathtt{A}\mathtt{C}$.

8. The 'pushout is a set' theorem presented

This section is devoted to proving the following theorem.

Theorem 2. *Given mere sets* $\mathtt{A}$, $\mathtt{B}$, *and* $\mathtt{C}$ *configured in a span*

$$\mathtt{C} \xleftarrow{g} \mathtt{A} \xrightarrow{f} \mathtt{B}$$

where f is monic, then the pushout $\mathtt{B}+_\mathtt{A}\mathtt{C}$ *is a mere set.*

For the remainder of the section, we denote the canonical pushout maps as $\mathtt{inl}\colon \mathtt{B} \to \mathtt{B} +_{\mathtt{A}} \mathtt{C}$ and $\mathtt{inr}\colon \mathtt{C} \to \mathtt{B} +_{\mathtt{A}} \mathtt{C}$.

Consider the types $\mathtt{P} :\equiv \{(x, y)\colon (\mathtt{B} +_{\mathtt{A}} \mathtt{C}) \times (\mathtt{B} +_{\mathtt{A}} \mathtt{C}) | x =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} y \text{ is a mere proposition}\}$ and $\mathtt{Q} :\equiv (\mathtt{B} +_{\mathtt{A}} \mathtt{C}) \times (\mathtt{B} +_{\mathtt{A}} \mathtt{C})$. To prove the theorem, it is sufficient to show that the inclusion $\mathtt{P} \hookrightarrow \mathtt{Q}$ is an equivalence.

Lemma 4. *The inclusion* $\mathtt{P} \to \mathtt{Q}$ *is an embedding.*

Proof. We need to show that, for each $(x_0, y_0)\colon \mathtt{Q}$, the homotopy fibre

$$\mathtt{F} :\equiv \sum_{(x,y)\colon \mathtt{P}} (x, y) =_{\mathtt{Q}} (x_0, y_0)$$

is a mere proposition. Any $p\colon \mathtt{F}$, is a pair of paths $p_x\colon x =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} x_0$ and $p_y\colon y =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} y_0$. The existance of such a p implies that $(x_0, y_0)\colon \mathtt{P}$ by β-reducing (x, y) to (x_0, y_0). Therefore, $\mathtt{refl}_{(x_0,y_0)}\colon \mathtt{F}$. The cube of paths

$$\begin{tikzcd}
x \arrow[rr, dash] \arrow[dr, dash, "p_x"] \arrow[dd, dash, "p_x"'] & & y \arrow[dr, dash, "p_y"] \arrow[dd, dash] & \\
& x_0 \arrow[rr, dash, "p_y"] \arrow[dd, dash] & & y_0 \arrow[dd, dash] \\
x_0 \arrow[rr, dash] \arrow[dr, dash] & & y_0 \arrow[dr, dash] & \\
& x_0 \arrow[rr, dash] & & y_0
\end{tikzcd}$$

commutes trivially and is filled because $\mathtt{B} +_{\mathtt{A}} \mathtt{C}$ is at most a 1-type. Hence, we have that $p =_{\mathtt{F}} \mathtt{refl}_{(x_0,y_0)}$ and $\mathtt{F}$ is contractible.

Where do we use that 'being a proposition' is a proposition? □

To prove that $\mathtt{P} \hookrightarrow \mathtt{Q}$ is an equivalnece, it remains to show that $\mathtt{P}$ contains a point in every connected component of $\mathtt{Q}$. Note that every connected component of $\mathtt{B} +_{\mathtt{A}} \mathtt{C}$ contains a point of form $\mathtt{inl}b$ or $\mathtt{inr}c$. Therefore, if $\mathtt{inl}b =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} x$ and $\mathtt{inr}c =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} x$ are always mere propositions, then $\mathtt{P}$ contains a point in every component of $\mathtt{Q}$ as desired.

To actually do this, we employ the *encode-decode method*. The strategy is to introduce types $\prod_{x\colon \mathtt{B}+_{\mathtt{A}}\mathtt{C}} \mathtt{code}(\mathtt{inl}b, x)$ and $\prod_{x\colon \mathtt{B}+_{\mathtt{A}}\mathtt{C}} \mathtt{code}(\mathtt{inl}c, x)$. After showing these are mere propositions—an easier task than proving the identity types in $\mathtt{B} +_{\mathtt{A}} \mathtt{C}$ are mere propositions—we build an equivalence between $\prod_{x\colon \mathtt{B}+_{\mathtt{A}}\mathtt{C}} \mathtt{code}(\mathtt{inl}b, x)$ and $\prod_{x\colon \mathtt{B}+_{\mathtt{A}}\mathtt{C}} \mathtt{inl}b =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} x$, and likewise between $\prod_{x\colon \mathtt{B}+_{\mathtt{A}}\mathtt{C}} \mathtt{code}(\mathtt{inr}c, x)$ and $\prod_{x\colon \mathtt{B}+_{\mathtt{A}}\mathtt{C}} \mathtt{inr}c =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} x$.

8.1. Defining $\prod_{x\colon \mathtt{B}+_{\mathtt{A}}\mathtt{C}} \mathtt{code}(\mathtt{inl}b, x)$.

To define $\prod_{x\colon \mathtt{B}+_{\mathtt{A}}\mathtt{C}} \mathtt{code}(\mathtt{inl}b, x)$, we induct on x, which requires the values

- $\mathtt{code}(\mathtt{inl}b, \mathtt{inl}b')$,
- $\mathtt{code}(\mathtt{inl}b, \mathtt{inr}c)$, and
- $\mathtt{code}(\mathtt{inl}b, \mathtt{glue}a)$.

We also show that the first two are mere propositions.

Define $\texttt{code}(\texttt{inl}b, \texttt{inl}b')$ as the pushout of the span

$$\begin{tikzcd} \sum_{a,a' : \texttt{A}}(b =_{\texttt{B}} fa) \times (b' =_{\texttt{B}} fa') \times (b =_{\texttt{B}} b') \arrow[r, "\alpha"] \arrow[d, "\beta"'] & (b =_{\texttt{B}} b') \\ \sum_{a,a' : \texttt{A}}(b =_{\texttt{B}} fa) \times (b' =_{\texttt{B}} fa') \times (ga =_{\texttt{C}} ga') & \end{tikzcd} \qquad \begin{tikzcd} (p,q,r) \arrow[r, mapsto, "\alpha"] \arrow[d, mapsto, "\beta"'] & r \\ (p,q,\texttt{ap}_g(\widehat{p^{-1};r;q})) & \end{tikzcd}$$

where $\widehat{p^{-1};r;q}$ is the preimage of the path $p^{-1};r;q \colon fa =_{\texttt{B}} fa'$ Its existance and uniqueness follows from the injectivity of f.

To distinguish these pushout maps from those associated to $\texttt{B} +_{\texttt{A}} \texttt{C}$, we denote them by $\texttt{inl}'$ and $\texttt{inr}'$.

In the following lemma, we use the denotation $\texttt{X} \coloneqq \sum_{a,a':\texttt{A}}(b =_{\texttt{B}} fa) \times (b' =_{\texttt{B}} fa') \times (ga =_{\texttt{C}} ga')$, $\texttt{Y} \coloneqq \sum_{a,a':\texttt{A}}(b =_{\texttt{B}} fa) \times (b' =_{\texttt{B}} fa') \times (b =_{/} Bb')$, and $\texttt{Z} \coloneqq b =_{\texttt{B}} b'$.

Lemma 5. *The type* $\texttt{code}(\texttt{inl}b, \texttt{inl}b')$ *is a mere proposition.*

Proof. The feet of the span are mere propositions; $\texttt{Z}$ trivially so. Showing that $\texttt{X}$ is also a mere proposition requires some work. Pick $(r,s,t) \colon (b =_{\texttt{B}} fa) \times (b' =_{\texttt{B}} fa') \times (ga =_{\texttt{C}} ga')$ and $(r',s',t') \colon (b =_{\texttt{B}} fa'') \times (b' =_{\texttt{B}} fa''') \times (ga'' =_{\texttt{C}} ga'''')$. Then $r^{-1};r' \colon fa =_{\texttt{B}} fa''$ and $s^{-1};s' \colon fa' =_{\texttt{B}} fa'''$, along with the injectivity of f, ensure the existance of paths $\hat{r} \colon a =_{\texttt{A}} a''$ and $\hat{s} \colon a' =_{\texttt{A}} a'''$. Beta-reduction provides that $(b =_{\texttt{B}} fa) \times (b' =_{\texttt{B}} fa') \times (ga =_{\texttt{C}} ga')$ is equivalent to $(b =_{\texttt{B}} fa'') \times (b' =_{\texttt{B}} fa''') \times (ga'' =_{\texttt{C}} ga'''')$, both of which are contractable because $\texttt{B}$ and $\texttt{C}$ are mere sets. It follows that $(r,s,t) = (r',s',t')$.

If $\texttt{code}(\texttt{inl}b, \texttt{inl}b')$ is not empty, then one of the span's feet must also be non-empty. This leads to three cases:

- $\texttt{X}$ is empty and $\texttt{Z}$ is non-empty. This forces $\texttt{Y}$ to also be empty and so the lemma holds;
- $\texttt{X}$ is non-empty and (Z) is empty. This again forces $\texttt{Y}$ to be empty, so the lemma holds;
- $\texttt{X}$ and $\texttt{Z}$ are non-empty. It is quick to check that this forces $\texttt{Y}$ to be non-empty. We dedicate the remainder of this proof to showing $\texttt{code}(\texttt{inl}b, \texttt{inl}b')$ is a proposition in this case.

A point in the apex of the span provides a witness to $fa =_{\texttt{B}} fa'$ which in turn provides a witness to $a =_{\texttt{A}} a'$. Beta-reduction plus univalence equates

$$\texttt{Y} = \sum_{a \colon \texttt{A}}(b =_{\texttt{B}} fa) \times (b' =_{\texttt{B}} fa) \times (b =_{\texttt{B}} b')$$

and also

$$\texttt{X} = \sum_{a \colon \texttt{A}}(b =_{\texttt{B}} fa) \times (b' =_{\texttt{B}} fa) \times (ga =_{\texttt{C}} ga)$$

The third terms of each of these types are redudant because $\texttt{B}$ and $\texttt{C}$ are mere sets. Applying beta-reduction and univalence again equates

$$\texttt{Y} = \sum_{a \colon \texttt{A}}(b =_{\texttt{B}} fa) \times (b' =_{\texttt{B}} fa)$$

and

$$\mathtt{X} = \sum_{a\colon \mathtt{A}} (b =_{\mathtt{B}} fa) \times (b' =_{\mathtt{B}} fa),$$

hence $\mathtt{X} = \mathtt{Y}$. It follows that $\mathtt{code}(\mathtt{inl}b, \mathtt{inl}b')$ is equivalent to the pushout of the span

$$\begin{tikzcd} \mathtt{X} \arrow[r, "!"] \arrow[d, "\mathrm{id}"'] & \mathtt{1} \\ \mathtt{X} & \end{tikzcd}$$

which is a mere set. ☐

This completes our current work with $\mathtt{code}(\mathtt{inl}b, \mathtt{inl}b')$.

Next, define

$$\mathtt{code}(\mathtt{inl}b, \mathtt{inr}c) :\equiv \sum_{a:\mathtt{A}} (b =_{\mathtt{B}} fa) \times (c =_{\mathtt{C}} ga).$$

Finally, we need to construct a witness to

$$\mathtt{code}(\mathtt{inl}b, \mathtt{ap}_{\mathtt{glue}}a)\colon \mathtt{code}(\mathtt{inl}b, \mathtt{inl}fa) = \mathtt{code}(\mathtt{inl}b, \mathtt{inr}ga).$$

Because both sides of the equation are mere propositions, it suffices to show that they are simultaneously empty or populated.

Lemma 6. *There exist functions* $\mathtt{code}(\mathtt{inl}b, \mathtt{inl}fa) \to \mathtt{code}(\mathtt{inl}b, \mathtt{inr}ga)$ *and* $\mathtt{code}(\mathtt{inl}b, \mathtt{inl}fa) \to \mathtt{code}(\mathtt{inl}b, \mathtt{inr}ga)$. *Moreover, they form an equivalence.*

Proof. Suppose $\mathtt{code}(\mathtt{inl}b, \mathtt{inl}ga)$ is inhabited. This point is equal to $(p, \mathtt{refl}_{ga})$ and p is pushed forward to populate $\mathtt{code}(\mathtt{inl}b, \mathtt{inl}fa)$.

Suppose that $\mathtt{code}(\mathtt{inl}b, \mathtt{inl}fa)$ is inhabited. This implies that either $p\colon b =_{\mathtt{B}} fa$ or $(q, r, s)\colon (b =_{\mathtt{B}} fa') \times (fa =_{\mathtt{B}} fa'') \times (ga' =_{\mathtt{C}} ga'')$ In the first case, $(p, \mathtt{refl}_{ga})\colon \mathtt{code}(\mathtt{inl}b, \mathtt{inr}ga)$ In the second case, the injectivity of f allows us to beta-reduce a to a''. That is, we actually have that $(q, r, s)\colon (b =_{\mathtt{B}} fa') \times (fa =_{\mathtt{B}} fa) \times (ga' =_{\mathtt{C}} ga)$ Hence $(q, s^{-1})\colon \mathtt{code}(\mathtt{inl}b, \mathtt{inr}ga)$.

The functions form an equivalence Because both $\mathtt{code}(\mathtt{inl}b, \mathtt{inl}fa)$ and $\mathtt{code}(\mathtt{inl}b, \mathtt{inr}ga)$ are propositions. ☐

8.2. **Define $\prod_{x\colon \mathtt{B}+_{\mathtt{A}}\mathtt{C}} \mathtt{code}(\mathtt{inr}c, x)$.** To define $\prod\limits_{x\colon \mathtt{B}+_{\mathtt{A}}\mathtt{C}} \mathtt{code}(\mathtt{inr}c, x)$ requires us to compute three values:

- $\mathtt{code}(\mathtt{inr}c, \mathtt{inl}b)$,
- $\mathtt{code}(\mathtt{inr}c, \mathtt{inr}c')$, and
- $\mathtt{code}(\mathtt{inr}c, \mathtt{glue}a)$,

the first two of which we show are mere propositions.

Next, define

$$\mathtt{code}(\mathtt{inr}c, \mathtt{inl}b) :\equiv \sum_{a:\mathtt{A}} (c =_{\mathtt{C}} ga) \times (b =_{\mathtt{B}} fa).$$

Lemma 7. *The type* $\mathtt{code}(\mathtt{inr}c, \mathtt{inl}b)$ *is a proposition.*

Proof. See Lemma 7. ☐

Define

$$\mathtt{code}(\mathtt{inr}c, \mathtt{inr}c') :\equiv \sum_{a,a':A} (c =_{\mathtt{C}} ga) \times (c' =_{\mathtt{C}} ga') \times (fa =_{\mathtt{B}} fa').$$

Lemma 8. *The type* $\mathtt{code}(\mathtt{inr}c, \mathtt{inr}c')$ *is a mere proposition.*

Proof. Because f is in injection, $a =_{\mathtt{A}} a'$ is populated so $\mathtt{code}(\mathtt{inr}c, \mathtt{inr}c')$ beta-reduces to

$$\sum_{a,a' : \mathtt{A}} (c =_{\mathtt{C}} ga) \times (c' =_{\mathtt{C}} ga) \times (a =_{\mathtt{A}} a)$$

which further reduces to

$$\mathtt{X} :\equiv \sum_{a : \mathtt{A}} (c =_{\mathtt{C}} ga) \times (c' =_{\mathtt{C}} ga)$$

because $\mathtt{A}$ is a set. Let (p, q) and (p', q') be in $\mathtt{X}$. The cube

$$\begin{tikzcd}
ga \arrow[rr, dash] \arrow[dd, dash, "p'"'] \arrow[dr, dash, "p"] & & ga \arrow[dd, dash, "q'"] \arrow[dr, dash, "q"] & \\
& c \arrow[rr, dash] \arrow[dd, dash] & & c \arrow[dd, dash] \\
ga' \arrow[rr, dash] \arrow[dr, dash] & & ga' \arrow[dr, dash] & \\
& c \arrow[rr, dash] & & c
\end{tikzcd}$$

commutes and is filled because $\mathtt{C}$ is a set. Therefore $(p, q) = (p', q')$ □

We now construct a point

$$\mathtt{code}(\mathtt{inr}c, \mathtt{ap}_{\mathtt{glue}}a) \colon \mathtt{code}(\mathtt{inr}c, \mathtt{inl}fa) = \mathtt{code}(\mathtt{inr}c, \mathtt{inl}ga).$$

Because both sides of the equality are propositions, a unique definition for $\mathtt{code}(\mathtt{inr}c, \mathtt{ap}_{\mathtt{glue}}a)$ exists if we construct an equivalnce.

Lemma 9. *The functions*

$$\mathtt{code}(\mathtt{inr}c, \mathtt{inl}fa) \to \mathtt{code}(\mathtt{inr}c, \mathtt{inr}ga),\ (p, q) \mapsto p$$

and

$$\mathtt{code}(\mathtt{inr}c, \mathtt{inr}ga) \to \mathtt{code}(\mathtt{inr}c, \mathtt{inl}fa),\ r \mapsto (r, \mathtt{refl}_{\mathtt{inl}fa})$$

form an equivalence.

Proof. First, let us show these actually are functions.

Let $(p, q) \colon \mathtt{code}(\mathtt{inr}c, \mathtt{inl}fa)$. Since f is injective, we have that $a =_{\mathtt{A}} a'$. Beta-reducing ga' to ga, gives us that $p \colon \mathtt{code}(\mathtt{inr}c, \mathtt{inr}ga)$. This provides the first function.

Given $r \colon \mathtt{code}(\mathtt{inr}c, \mathtt{inr}ga)$, r is equal to a point in $c =_{\mathtt{C}} ga$. This provides a point $(r, \mathtt{refl}_{fa}) \colon \mathtt{code}(\mathtt{inr}c, \mathtt{inl}fa)$. This gives the second function.

The functions form an equivalence Because both $\mathtt{code}(\mathtt{inr}c, \mathtt{inl}fa)$ and $\mathtt{code}(\mathtt{inr}c, \mathtt{inr}ga)$ are propositions. □

The next stage in proving Theorem 2 is showing that $\mathtt{inl}b =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} x$ and $\mathtt{inr}c =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} x$ are mere propositions for any x. To do so, we construct equivalences between, respectively,

$\texttt{code}(\texttt{inl}b, x)$ and $\texttt{code}(\texttt{inr}c, x)$ which we know are mere propositions. Specifically, we define maps

$$
\begin{aligned}
\texttt{encode}(\texttt{inl}b)&\colon \prod_{x\colon \texttt{B}+_{\texttt{A}}\texttt{C}} (\texttt{inl}b =_{\texttt{B}+_{\texttt{A}}\texttt{C}} x) \to \texttt{code}(\texttt{inl}b, x)\\
\texttt{encode}(\texttt{inr}c)&\colon \prod_{x\colon \texttt{B}+_{\texttt{A}}\texttt{C}} (\texttt{inr}c =_{\texttt{B}+_{\texttt{A}}\texttt{C}} x) \to \texttt{code}(\texttt{inr}c, x)\\
\texttt{decode}(\texttt{inl}b)&\colon \prod_{x\colon \texttt{B}+_{\texttt{A}}\texttt{C}} \texttt{code}(\texttt{inl}b, x) \to (\texttt{inl}b =_{\texttt{B}+_{\texttt{A}}\texttt{C}} x)\\
\texttt{decode}(\texttt{inr}c)&\colon \prod_{x\colon \texttt{B}+_{\texttt{A}}\texttt{C}} \texttt{code}(\texttt{inr}c, x) \to (\texttt{inr}c =_{\texttt{B}+_{\texttt{A}}\texttt{C}} x)
\end{aligned}
$$

with corresponding `encode` and `decode` pairs forming mutual equivalences.

8.3. **Defining `encode`.** We define $\texttt{encode}(\texttt{inl}b)$ and $\texttt{encode}(\texttt{inr}c)$ by inducting on $x\colon \texttt{B}+_{\texttt{A}}\texttt{C}$. In doing so, we make use of path induction.

Define

$$\texttt{encode}(\texttt{inl}(b))\colon \prod x\colon \texttt{B} +_{\texttt{A}} \texttt{C}(\texttt{inl}(b) =_{\texttt{B}+_{\texttt{A}}\texttt{C}} x) \to \texttt{code}(\texttt{inl}(b), x)$$

by the assignment $\texttt{refl}_{\texttt{inl}b} \mapsto \texttt{refl}_{\texttt{inl}'b}$. Recall, `inl` corresponds to the pushout map into $\texttt{B} +_{\texttt{A}} \texttt{C}$ and $\texttt{inl}'$ to the pushout map into $\texttt{code}(\texttt{inl}b, \texttt{inl}b')$.

Define

$$\texttt{encode}(\texttt{inr}c)\colon (\texttt{inr}c =_{\texttt{B}+_{\texttt{A}}\texttt{C}} x) \to \texttt{code}(\texttt{inr}c, x)$$

by the assignemnt $\texttt{refl}_{\texttt{inr}c} \mapsto \texttt{refl}_c$. Note, we use that $\texttt{code}(\texttt{inr}c, \texttt{inr}c)$ is equivalent to $c =_{\texttt{C}} c$ as shown in Lemma 8.

Define

$$\texttt{ap}_{\texttt{encode}(\texttt{inl}b)}\colon ((b =_{\texttt{B}+_{\texttt{A}}\texttt{C}} fa) \to \texttt{code}(\texttt{inl}b, \texttt{inl}fa)) = ((b =_{\texttt{B}+_{\texttt{A}}\texttt{C}} ga) \to \texttt{code}(\texttt{inl}b, \texttt{inr}ga))$$

to be the diagram

$$
\begin{tikzcd}
(b =_{\texttt{B}+_{\texttt{A}}\texttt{C}} fa) \arrow[r, "\texttt{encode}(\texttt{inl}b)(\texttt{inl}fa)"] \arrow[d, "(-);\texttt{glue}a"'] & \texttt{code}(\texttt{inl}b, \texttt{inl}fa) \arrow[d, "\texttt{code}(\texttt{inl}b, \texttt{glue}a)"] \\
(b =_{\texttt{B}+_{\texttt{A}}\texttt{C}} ga) \arrow[r, "\texttt{encode}(\texttt{inl}b)(\texttt{inr}ga)"] & \texttt{code}(\texttt{inl}b, \texttt{inr}ga)
\end{tikzcd}
$$

which commutes because $\texttt{code}(\texttt{inl}b, ga)$ is a mere proposition.

Define

$$\texttt{ap}_{\texttt{encode}(\texttt{inr}c)}\colon ((c =_{\texttt{B}+_{\texttt{A}}\texttt{C}} fa) \to \texttt{code}(\texttt{inr}c, \texttt{inl}fa)) = ((c =_{\texttt{B}+_{\texttt{A}}\texttt{C}} ga) \to \texttt{code}(\texttt{inr}c, \texttt{inr}ga))$$

to be the diagram

$$\begin{tikzcd}
(c =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} fa) \arrow[r, "\mathtt{encode}(\mathtt{inr}c)(\mathtt{inl}fa)"] \arrow[d, "(-);\mathtt{glue}a"'] & \mathtt{code}(\mathtt{inr}c, \mathtt{inl}fa) \arrow[d, "\mathtt{code}(\mathtt{inr}c, \mathtt{glue}a)"] \\
(c =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} ga) \arrow[r, "\mathtt{encode}(\mathtt{inr}c)(\mathtt{inr}ga)"] & \mathtt{code}(\mathtt{inr}c, \mathtt{inr}ga)
\end{tikzcd}$$

which commutes because $\mathtt{code}(\mathtt{inr}c, ga)$ is a mere proposition.

8.4. **Defining decode.** To define the map

$$\mathtt{decode}(\mathtt{inl}b)\colon \prod_{x\colon \mathtt{B}+_{\mathtt{A}}\mathtt{C}} \mathtt{code}(\mathtt{inl}b, x) \to (\mathtt{inl}b =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} x) \tag{3}$$

we use induction on $x\colon \mathtt{B} +_{\mathtt{A}} \mathtt{C}$ which, in practice, means that we need two function types

- $\mathtt{decode}(\mathtt{inl}b)(\mathtt{inl}b')\colon \mathtt{code}(\mathtt{inl}b, \mathtt{inl}b') \to (\mathtt{inl}b =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} \mathtt{inl}b')$
- $\mathtt{decode}(\mathtt{inl}b)(\mathtt{inl}c)\colon \mathtt{code}(\mathtt{inl}b, \mathtt{inr}c) \to (\mathtt{inl}b =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} \mathtt{inr}c$

and a witness

$$\begin{aligned}\mathtt{ap}_{\mathtt{decode}(\mathtt{inl}b)}(\mathtt{glue}a)\colon &(\mathtt{code}(\mathtt{inl}b, \mathtt{inl}fa) \to (\mathtt{inl}b =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} \mathtt{inl}fa)\\ &= (\mathtt{code}(\mathtt{inl}b, \mathtt{inl}ga) \to (\mathtt{inl}b =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} \mathtt{inl}ga)\end{aligned}$$

Define

$$\mathtt{decode}(\mathtt{inl}b)\colon \mathtt{code}(\mathtt{inl}b, \mathtt{inl}b') \to (\mathtt{inl}b =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} \mathtt{inl}b')$$

by inducting on $\mathtt{code}(\mathtt{inl}b, \mathtt{inl}b')$ This requires three values.

- Let $\mathtt{decode}(\mathtt{inl}b)(\mathtt{inl}'p)\colon (b =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} b')$, where $p\colon b =_{\mathtt{B}} b'$, be $\mathtt{ap}_{\mathtt{inl}}(p)$;
- Let $\mathtt{decode}(\mathtt{inl}b)(\mathtt{inr}'(q, r, s))\colon (b = \mathtt{B} +_{\mathtt{A}} \mathtt{C}b')$, where $(q, r, s)\colon \sum_{a,a'\colon \mathtt{A}} (b =_{\mathtt{B}} fa) \times (b' =_{\mathtt{B}} fa') \times (ga =_{\mathtt{C}} ga')$, be $ap_{\mathtt{inl}}q; \mathtt{glue}a; \mathtt{ap}_{\mathtt{inr}}s; \mathtt{glue}^{-1}a'; r^{-1}$ and
- The path
$$\begin{aligned}\mathtt{ap}_{\mathtt{decode}(\mathtt{inl}b)}(\mathtt{glue}(t, u, v))\colon (\mathtt{decode}(\mathtt{inl}b)(\mathtt{inl}'(\alpha(t, u, v))) &=_{\mathtt{inl}b =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} \mathtt{inl}b'}\\ \mathtt{decode}(\mathtt{inl}b)(\mathtt{inr}'(\beta(t, u, v))),&\end{aligned}$$
where $(t, u, v)\colon (b =_{\mathtt{B}} fa) \times (b' =_{\mathtt{B}} fa') \times (b =_{\mathtt{B}} b')$, is trivial because $\mathtt{code}(\mathtt{inl}b, \mathtt{inl}b')$ is a mere set.

To define

$$\mathtt{decode}(\mathtt{inl}b)\colon \mathtt{code}(\mathtt{inl}b, \mathtt{inr}c) \to (\mathtt{inl}b =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} \mathtt{inr}c),$$

recall that

$$\mathtt{code}(\mathtt{inl}b, \mathtt{inr}c) = \sum_{a\colon \mathtt{A}} (b =_{\mathtt{B}} fa) \times (c =_{\mathtt{C}} ga)$$

Define

$$\mathtt{decode}(\mathtt{inl}b)(p, q) :\equiv \mathtt{inl}p; \mathtt{glue}a; \mathtt{inr}q$$

Right now, we define

$$\begin{aligned}\mathtt{ap}_{\mathtt{decode}(\mathtt{inl}b)}(\mathtt{glue}a)\colon &(\mathtt{code}(\mathtt{inl}b, \mathtt{inl}fa) \to (\mathtt{inl}b =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} \mathtt{inl}fa)\\ &= (\mathtt{code}(\mathtt{inl}b, \mathtt{inl}ga) \to (\mathtt{inl}b =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} \mathtt{inl}ga)\end{aligned}$$

To define this is to construct a commuting square

$$\begin{tikzcd}
\mathtt{code}(\mathtt{inl}b, \mathtt{inl}fa) \arrow[r, "\mathtt{decode}(\mathtt{inl}b)(\mathtt{inl}b')"] \arrow[d, "\theta"'] & (\mathtt{inl}b =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} \mathtt{inl}fa) \arrow[d, "\theta'"'] \\
\mathtt{code}(\mathtt{inl}b, \mathtt{inl}ga) \arrow[r, "\mathtt{decode}(\mathtt{inl}b)(\mathtt{inl}c)"] & (\mathtt{inl}b =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} \mathtt{inl}ga)
\end{tikzcd}$$

We have already defined the two $\mathtt{decode}$ maps. The map θ' is post-composition by $\mathtt{glue}a$. The map θ requires induction to define because we are mapping out of a pushout. Therefore, we need the following three values to define θ:

- $\theta(\mathtt{inl}'p)$ for $p\colon b =_{\mathtt{B}} fa$
- $\theta(\mathtt{inr}'(q,r,s))$ for $(q,r,s)\colon \sum_{a',a''\colon \mathtt{A}} (b =_{\mathtt{B}} fa') \times (fa =_B fa'') \times (ga' =_C ga'')$
- $\mathtt{ap}_\theta(\mathtt{glue}(t,u,v))\colon \theta(\mathtt{inl}'(\alpha(t,u,v))) = \theta(\mathtt{inr}'(\beta(t,u,v)))$.

A quick remark on notation: since f is monic and $\mathtt{B}$ is a set, we can pull back any path of form $r\colon fa =_{\mathtt{B}} fa'$ to a determined path $\hat{r}\colon a =_{\mathtt{A}} a'$.

Define $\theta(\mathtt{inl}'p)$ for $p\colon b =_{\mathtt{B}} fa$ to be $(p, \mathtt{refl}_{ga})$. Define $\theta(q,r,s)$ to be $(q, \mathtt{ap}_g(\hat{r}); s^{-1})$.

Now we know that $\mathtt{ap}_\theta(\mathtt{glue}(t,u,v))$ is a path from $\theta(\mathtt{inl}'(v)) = (v, \mathtt{refl}_{ga})$ to $\theta(\mathtt{inr}'(\beta(t,u,v)))$ But that can be reduced:

$$\begin{aligned}
\theta(\mathtt{inr}'(t,u,\mathtt{ap}_g(\hat{v}))) &= (t, \mathtt{ap}_g(\hat{u}); \mathtt{ap}_g(\hat{v})^{-1}) \\
&= (t, \mathtt{ap}_g(\hat{u}; \hat{v}^{-1})).
\end{aligned}$$

With this in mind, we define $\mathtt{ap}_\theta(\mathtt{glue}(t,u,v))$ to be post-composition by

$$(\mathtt{ap}_f(\hat{u}; \hat{v}^{-1}), \mathtt{ap}_g(\hat{u}; \hat{v}^{-1})).$$

Now we need to check that the diagram commutes. Since $\mathtt{code}(\mathtt{inl}b, \mathtt{inl}fa)$ is a set, it suffices to check that

$$\theta'(\mathtt{decode}(\mathtt{inl}b)(\mathtt{inl}fa)(x)) =_{\mathtt{inl}b =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} \mathtt{inr}ga} \mathtt{decode}(\mathtt{inl}b)(\mathtt{inr}ga)(\theta(x))$$

where $x\colon \mathtt{code}(\mathtt{inl}b, \mathtt{inl}fa)$ takes values

- $\mathtt{inl}'p$ for $p\colon b =_{\mathtt{B}} fa$, and
- $\mathtt{inr}'(q,r,s)$ for $(q,r,s)\colon \sum_{a',a''\colon \mathtt{A}} (b =_{\mathtt{B}} fa') \times (fa =_{\mathtt{B}} fa'') \times (ga' =_{\mathtt{C}} ga'')$.

We also need to check that

$$\mathtt{ap}_{\theta';\mathtt{decode}(\mathtt{inl}b)(\mathtt{inl}fa))}(\mathtt{glue}(t,u,v)) =_{\mathtt{inl}b =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} \mathtt{inr}ga} \mathtt{ap}_{\mathtt{decode}(\mathtt{inl}b)(\mathtt{inr}ga);\theta}(\mathtt{glue}(t,u,v))$$

Set $x :\equiv \mathtt{inl}'p$. Then

$$\begin{aligned}\theta'(\mathtt{decode}(\mathtt{inl}b)(\mathtt{inl}fa)(\mathtt{inl}'p)) &= \theta'((\mathtt{ap}_{\mathtt{inl}'}p)) \\ &= \mathtt{ap}_{\mathtt{inl}'}p; \mathtt{glue}a\end{aligned}$$

Also,

$$\begin{aligned}\mathtt{decode}(\mathtt{inl}b)(\mathtt{inr}ga)(\theta(\mathtt{inl}'p)) &= \mathtt{decode}(\mathtt{inl}b)(\mathtt{inr}ga)((p, \mathtt{refl}_{ga})) \\ &= \mathtt{ap}_{\mathtt{inl}'}p; \mathtt{glue}a; \mathtt{ap}_{\mathtt{inr}'}\mathtt{refl}_{ga} \\ &= \mathtt{ap}_{\mathtt{inl}'}p; \mathtt{glue}a; \mathtt{refl}_{\mathtt{inr}'ga} \\ &= \mathtt{ap}_{\mathtt{inl}'}p; \mathtt{glue}a\end{aligned}$$

Therefore, the square commutes on $\mathtt{inl}'p$

Set $x :\equiv \mathtt{inr}'(q, r, s)$. Then

$$\begin{aligned}\theta'(\mathtt{decode}(\mathtt{inl}b)(\mathtt{inl}fa)(\mathtt{inr}'(q, r, s))) &= \theta'(\mathtt{ap}_{\mathtt{inl}}q; \mathtt{glue}a'; \mathtt{ap}_{\mathtt{inr}}s; \mathtt{glue}a''; \mathtt{ap}_{\mathtt{inl}}r^{-1}) \\ &= \mathtt{ap}_{\mathtt{inl}}q; \mathtt{glue}a'; \mathtt{ap}_{\mathtt{inr}}s; \mathtt{glue}^{-1}a''; \mathtt{ap}_{\mathtt{inl}}r^{-1}; \mathtt{glue}a \\ &= \mathtt{ap}_{\mathtt{inl}}q; \mathtt{glue}a'; \mathtt{ap}_{\mathtt{inr}}s; \mathtt{glue}^{-1}a''; \mathtt{ap}_{\mathtt{inl}}\mathtt{ap}_f\hat{r}^{-1}; \mathtt{glue}a\end{aligned}$$

Also,

$$\begin{aligned}\mathtt{decode}(\mathtt{inl}b)(\mathtt{inr}ga)(\theta(\mathtt{inr}'(q, r, s))) &= \mathtt{decode}(\mathtt{inl}b)(\mathtt{inr}ga)(q, \mathtt{ap}_g(\hat{r}; s^{-1})) \\ &= \mathtt{ap}_{\mathtt{inl}}q; \mathtt{glue}a'; (\mathtt{ap}_{\mathtt{inr}}(\mathtt{ap}_g\hat{r}; s^{-1})^{-1}) \\ &= \mathtt{ap}_{\mathtt{inl}}q; \mathtt{glue}a'; \mathtt{ap}_{\mathtt{inr}}(\mathtt{ap}_g(\hat{r}); (\mathtt{ap}_g(s^{-1}))^{-1}) \\ &= \mathtt{ap}_{\mathtt{inl}}q; \mathtt{glue}a'; \mathtt{ap}_{\mathtt{inr}}(s^{-1})^{-1}; \mathtt{ap}_g(\hat{r})^{-1} \\ &= \mathtt{ap}_{\mathtt{inl}}q; \mathtt{glue}a'; \mathtt{ap}_{\mathtt{inr}}(s^{-1})^{-1}; \mathtt{ap}_g(\hat{r})^{-1} \\ &= \mathtt{ap}_{\mathtt{inl}}q; \mathtt{glue}a'; \mathtt{ap}_{\mathtt{inr}}s; \mathtt{ap}_g(\hat{r})^{-1}\end{aligned}$$

We just need to check that

$$\mathtt{ap}_g(\hat{r})^{-1} = \mathtt{glue}^{-1}a''; \mathtt{ap}_{\mathtt{inl}}\mathtt{ap}_f\hat{r}^{-1}; \mathtt{glue}a$$

But this follows from the continuity of $\mathtt{glue}$, that is, it preserves paths. Therefore, the square commutes on $\mathtt{inr}'(q, r, s)$.

It remains to check that

$$\mathtt{ap}_{\theta';\mathtt{decode}(\mathtt{inl}b)(\mathtt{inl}fa))}(\mathtt{glue}(t, u, v)) =_{\mathtt{inl}b =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} \mathtt{inr}ga} \mathtt{ap}_{\mathtt{decode}(\mathtt{inl}b)(\mathtt{inr}ga);\theta}(\mathtt{glue}(t, u, v))$$

for $(t, u, v) \colon \sum_{a', a'' \colon \mathtt{A}} (b =_{\mathtt{B}} fa') \times (fa =_{\mathtt{B}} fa'') \times (b =_{\mathtt{B}} fa)$. Here, we make some reductions.

- Because f is monic, we get that $fa =_{\mathtt{B}} fa''$ reduces to $fa =_{\mathtt{B}} fa$. Since $\mathtt{B}$ is a set, we can take $u = \mathtt{refl}_{fa}$.
- The type $b =_{\mathtt{B}} fa'$ reduces to $fa =_{\mathtt{B}} fa$ because $\mathtt{B}$ is monic, so $t = \mathtt{refl}_{fa}$..
- The type $b =_{\mathtt{B}} fa$ reduces to $fa =_{\mathtt{B}} fa$ because $\mathtt{B}$ is monic, so $v = \mathtt{refl}_{fa}$.

Without loss of generality, we can take $(t,u,v) = (\mathtt{refl}_{fa}, \mathtt{refl}_{fa}, \mathtt{refl}_{fa})$. Therefore, it suffices to check that

$$\begin{aligned}&\mathtt{ap}_{\theta';\mathtt{decode}(\mathtt{inl}b)(\mathtt{inl}fa))}(\mathtt{glue}(\mathtt{refl}_{fa}, \mathtt{refl}_{fa}, \mathtt{refl}_{fa}))\\ &=_{\mathtt{inl}b =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} \mathtt{inr}ga}\ \mathtt{ap}_{\mathtt{decode}(\mathtt{inl}b)(\mathtt{inr}ga);\theta}(\mathtt{glue}(\mathtt{refl}_{fa}, \mathtt{refl}_{fa}, \mathtt{refl}_{fa})).\end{aligned}$$

But because the square commutes on points as shown above, the two paths on the left and right of the above equation are certainly parallel. Then functorality gives us that $\mathtt{refl}_{fa}$ is preserved. Hence the square commutes.

To define the map

$$\mathtt{decode}(\mathtt{inr}c) \colon \prod_{x\colon \mathtt{B}+_{\mathtt{A}}\mathtt{C}} \mathtt{code}(\mathtt{inr}c, x) \to \mathtt{inr}(c =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} x)$$

we use induction on $\mathtt{B} +_{\mathtt{A}} \mathtt{C}$ which, in practice, means that we require the two function types

- $\mathtt{decode}(\mathtt{inr}c)(\mathtt{inl}b) \colon \mathtt{code}(\mathtt{inr}c, \mathtt{inl}b) \to (\mathtt{inr}c =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} \mathtt{inl}b)$
- $\mathtt{decode}(\mathtt{inr}c)(\mathtt{inr}c') \colon \mathtt{code}(\mathtt{inr}c, \mathtt{inr}c') \to (\mathtt{inr}c =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} \mathtt{inr}c')$

and a witness

$$\begin{aligned}\mathtt{ap}_{\mathtt{decode}(\mathtt{inr}c)}(\mathtt{glue}a) \colon &(\mathtt{code}(\mathtt{inr}c, \mathtt{inl}fa) \to (\mathtt{inr}c =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} \mathtt{inl}fa)\\ &= (\mathtt{code}(\mathtt{inr}c, \mathtt{inl}ga) \to (\mathtt{inr}c =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} \mathtt{inl}ga)\end{aligned}$$

First, let us define the map

$$\mathtt{decode}(\mathtt{inr}c)(\mathtt{inl}b) \colon \mathtt{code}(\mathtt{inr}c, \mathtt{inl}b) \to (\mathtt{inr}c =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} \mathtt{inl}b).$$

Recall that $\mathtt{code}(\mathtt{inr}c, \mathtt{inl}b) :\equiv \sum_{a\colon \mathtt{A}} (c =_{\mathtt{C}} ga) \times (b =_{\mathtt{B}} fa)$. Thus for any (p,q) in $\mathtt{code}(\mathtt{inr}c, \mathtt{inl}b)$, we define a path $\mathtt{decode}(\mathtt{inr}c)(\mathtt{inl}b)(p,q)$ in $\mathtt{B} +_{\mathtt{A}} \mathtt{C}$ from $\mathtt{inr}c$ to $\mathtt{inl}b$. Take this path to be $\mathtt{inr}p; \mathtt{glue}^{-1}a; \mathtt{inl}q^{-1}$.

Next, we define

$$\mathtt{decode}(\mathtt{inr}c)(\mathtt{inr}c') \colon \mathtt{code}(\mathtt{inr}c, \mathtt{inr}c') \to (\mathtt{inr}c =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} \mathtt{inr}c').$$

Recall that

$$\mathtt{code}(\mathtt{inr}c, \mathtt{inr}c') :\equiv \sum_{a\colon \mathtt{A}} (c =_{\mathtt{C}} ga) \times (c' =_{\mathtt{C}} ga') \times (fa =_{\mathtt{B}} fa').$$

Thus for any (p,q,r) in $\mathtt{code}(\mathtt{inr}c, \mathtt{inr}c')$, we define a path $\mathtt{decode}(\mathtt{inr}c)(\mathtt{inr}c')(p,q,r)$ in $\mathtt{B} +_{\mathtt{A}} \mathtt{C}$ from $\mathtt{inr}c$ to $\mathtt{inr}c'$. Take this path to be $\mathtt{inr}p; \mathtt{glue}^{-1}a; \mathtt{inl}r\mathtt{glue}a'; \mathtt{inr}q^{-1}$.

Finally, we define a path

$$\begin{aligned}\mathtt{ap}_{\mathtt{decode}(\mathtt{inr}c)}(\mathtt{glue}a) \colon &(\mathtt{code}(\mathtt{inr}c, \mathtt{inl}fa) \to (\mathtt{inr}c =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} \mathtt{inl}fa)\\ &= (\mathtt{code}(\mathtt{inr}c, \mathtt{inl}ga) \to (\mathtt{inr}c =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} \mathtt{inl}ga).\end{aligned}$$

Replacing the two $\mathtt{code}$ expressions with their definitions, this path is constructed as a commuting square

$$\begin{tikzcd}
\sum_{a'\colon \mathtt{A}} (c =_{\mathtt{C}} ga') \times (fa =_{\mathtt{B}} fa') \arrow[r, "\mathtt{decode}"] \arrow[d, "\theta"'] & (\mathtt{inr}c =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} \mathtt{inl}fa) \arrow[d, "\theta'"] \\
\sum_{a',a''\colon \mathtt{A}} (c =_{\mathtt{C}} ga') \times (ga =_{\mathtt{C}} ga') \times (fa' =_{\mathtt{B}} fa'') \arrow[r, "\mathtt{decode}"] & \mathtt{inr}c =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} \mathtt{inl}ga
\end{tikzcd}$$

The easier map to define is θ' which concatenates with $\mathtt{glue}a$. Then θ is given by $(p, q) \mapsto (p, \mathtt{refl}_{ga}, q)$. Now we check whether the square commutes. We have

$$\begin{aligned} \theta'(\mathtt{decode}(\mathtt{inr}c, -)(\mathtt{inl}fa)(p, q)) &= \theta'(\mathtt{inr}p; \mathtt{glue}^{-1}a'; \mathtt{inl}q^{-1}) \\ &= \mathtt{inr}p; \mathtt{glue}^{-1}a'; \mathtt{inl}q^{-1}; \mathtt{glue}a. \end{aligned}$$

We also have that

$$\begin{aligned} \mathtt{decode}(\mathtt{inr}c, -)(\mathtt{inl}ga)(\theta(p, q)) &= \mathtt{decode}(\mathtt{inr}c, -)(\mathtt{inl}ga)(p, \mathtt{refl}_{ga}, q) \\ &= \mathtt{inr}p; \mathtt{glue}^{-1}a'; \mathtt{inl}q^{-1}; \mathtt{glue}a; \mathtt{inr}\mathtt{refl}_{ga} \\ &= \mathtt{inr}p; \mathtt{glue}^{-1}a'; \mathtt{inl}q^{-1}; \mathtt{glue}a \end{aligned}$$

Hence the square commutes.

8.5. **Composing encode and decode.** Consider the composite

$$\mathtt{encode}; \mathtt{decode} \colon \prod_{x, y \colon \mathtt{B}+_{\mathtt{A}}\mathtt{C}} x =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} y x =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} y$$

To show that this map is the identity up to homotopy, we compute both

- $\mathtt{encode}; \mathtt{decode}(\mathtt{inl}b) \colon \prod_{x \colon \mathtt{B}+_{\mathtt{A}}\mathtt{C}} (\mathtt{inl}b =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} x) \to (\mathtt{inl}b =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} x)$, and
- $\mathtt{encode}; \mathtt{decode}(\mathtt{inr}c) \colon \prod_{x \colon \mathtt{B}+_{\mathtt{A}}\mathtt{C}} \mathtt{inr}c =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} x \to (\mathtt{inr}c =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} x)$.

But computing these values actually requires the computation of the following four maps:

- $\mathtt{encode}; \mathtt{decode}(\mathtt{inl}b)(\mathtt{inl}b') \colon \mathtt{code}(\mathtt{inl}b, \mathtt{inl}b') \to (\mathtt{inl}b =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} \mathtt{inl}b')$. By path induction, it suffices to check $\mathtt{refl}_{\mathtt{inl}b}$:

$$\begin{aligned} \mathtt{encode}; \mathtt{decode}(\mathtt{inl}b)(\mathtt{inl}b')(\mathtt{refl}_{\mathtt{inl}b}) &= \mathtt{encode}(\mathtt{ap}_{\mathtt{inl}'}\mathtt{refl}_{\mathtt{inl}b}) \\ &= \mathtt{encode}(\mathtt{refl}_{\mathtt{inl}'b}) \\ &= \mathtt{ap}_{\mathtt{inl}}\mathtt{refl}_{\mathtt{inl}'b} \\ &= \mathtt{refl}_{\mathtt{inl}b}. \end{aligned}$$

  So this one checks out.
- $\mathtt{encode}; \mathtt{decode}(\mathtt{inl}b)(\mathtt{inr}c) \colon \mathtt{code}(\mathtt{inl}b, \mathtt{inr}c) \to (\mathtt{inl}b =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} \mathtt{inl}c)$. glue(a) *is* the only thing to check here, right? This is only non-trivial when $b =_{\mathtt{B}} fa$ and $c =_{\mathtt{C}} ga$. Hence

$$\mathtt{encode}; \mathtt{decode}(\mathtt{inl}b)(\mathtt{inr}c)(\mathtt{glue}a) = \mathtt{decode}(\mathtt{inl}b)(\mathtt{inr}c)(\mathtt{refl}_{fa}, \mathtt{refl}_{ga}) = \mathtt{ap}_{\mathtt{inr}}\mathtt{refl}_{ga}; \mathtt{glue}a; \mathtt{ap}_{\mathtt{inl}}\mathtt{re}$$

  This one checks out too.
- $\mathtt{encode}; \mathtt{decode}(\mathtt{inr}c, -)(\mathtt{inl}b) \colon \mathtt{code}(\mathtt{inr}c, \mathtt{inl}b) \to (\mathtt{inr}c =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} \mathtt{inl}b)$. This is symmetric to the above case, so checks out.
- $\mathtt{encode}; \mathtt{decode}(\mathtt{inr}c, -)(\mathtt{inr}c') \colon \mathtt{code}(\mathtt{inr}c, \mathtt{inr}c') \to (\mathtt{inr}c =_{\mathtt{B}+_{\mathtt{A}}\mathtt{C}} \mathtt{inr}c')$ is, by path induction,

$$\mathtt{encode}; \mathtt{decode}(\mathtt{inr}c, -)(\mathtt{inr}c')(\mathtt{refl}_{\mathtt{inr}c}) = \mathtt{decode}(\mathtt{inr}c, -)(\mathtt{inr}c')(\mathtt{refl}_{c}) = \mathtt{ap}_{\mathtt{inr}}\mathtt{refl}_{c} = \mathtt{refl}_{\mathtt{inr}c}$$

And so, we have proved that $\texttt{decode}$ is a section for $\texttt{encode}$. The opposite direction remains.

Now, we look at the composite

$$\texttt{decode};\texttt{encode}\colon \prod_{x,y\colon \texttt{B}+_\texttt{A}\texttt{C}} \texttt{code}(x,y) \to \texttt{code}(x,y)$$

We can compute this composite by computing the values

- $\texttt{decode};\texttt{encode}(\texttt{inl}b)(\texttt{inl}b')\colon \texttt{code}(\texttt{inl}b,\texttt{inl}b') \to \texttt{code}(\texttt{inl}b,\texttt{inl}b')$,
- $\texttt{decode};\texttt{encode}(\texttt{inl}b)(\texttt{inr}c)\colon \texttt{code}(\texttt{inl}b,\texttt{inr}c) \to \texttt{code}(\texttt{inl}b,\texttt{inr}c)$,
- $\texttt{decode};\texttt{encode}(\texttt{inr}c)(\texttt{inl}b)\colon \texttt{code}(\texttt{inr}c,\texttt{inl}b) \to \texttt{code}(\texttt{inr}c,\texttt{inl}b)$, and
- $\texttt{decode};\texttt{encode}(\texttt{inr}c)(\texttt{inr}c')\colon \texttt{code}(\texttt{inr}c,\texttt{inr}c') \to \texttt{code}(\texttt{inr}c,\texttt{inr}c')$

But these maps are all identity since $\texttt{code}(x,y)$ is a proposition for $x,y\colon \texttt{B}+_\texttt{A}\texttt{C}$.

We now know that $\texttt{encode}$ and $\texttt{decode}$ are mutual inverses. Therefore, $\texttt{inl}b =_{\texttt{B}+_\texttt{A}\texttt{C}} x$ and $\texttt{inr}c =_{\texttt{B}+_\texttt{A}\texttt{C}} x$ are mere propositions for any $x\colon \texttt{B}+_\texttt{A}\texttt{C}$. As discussed above, the embedding $\texttt{P} \hookrightarrow \texttt{Q}$ is actually an equivlance. Theorem 2 follows.

9. Cohomology in $\mathbb{Z}/2\mathbb{Z}$

Strategy: Use that $\pi_{n+m}(K(G,n)\wedge K(H,m)) = G\otimes H$ to show, for $R \coloneqq \mathbb{Z}/2\mathbb{Z}$, that $||K(R,n)\wedge K(R,m)||_{n+m} = K(R\otimes R, n+m)$. Then we'll lift the ring multiplication $R\otimes R \to R$ to a cup product on cohomology with R-coefficients.

First, we show that $||K(R,n)\wedge K(R,m)||_{n+m} = K(R\otimes R, n+m)$. We use the fact that $K(R\otimes R, n+m)$ is the unique up to homotopy space having the property that

$$\pi_k(K(R\otimes R, n+m)) = \begin{cases} 0, & k\neq n+m \\ R\otimes R, & k = n+m \end{cases}$$

Thus by showing that $||K(R,n)\wedge K(R,m)||_{n+m}$ has the property, we have the desired equivalence. First, we have that

$$\pi_{n+m}(||K(R,n)\wedge K(R,m)||_{n+m}) = R\otimes R$$

Second, we ahve that

$$\pi_k(||K(R,n)\wedge K(R,m)||_{n+m}) = 0$$

for $k > n+m$ because $||K(R,n)\wedge K(R,m)||_{n+m}$ is an $(n+m)$-type. Third, we have that

$$\pi_k(||K(R,n)\wedge K(R,m)||_{n+m}) = 0$$

for $k \leq n+m-1$ because $||K(R,n)\wedge K(R,m)||_{n+m}$ is $(n+m-1)$-connected, which follows from $K(R,k)$ being $(k-1)$-connected as seen in Guillame's thesis Prop 4.3.1. From these three facts, we conclude that

$$\pi_k(||K(R,n)\wedge K(R,m)||_{n+m}) = \begin{cases} 0, & k\neq n+m \\ R\otimes R, & k = n+m \end{cases}$$

and, in particular, we get that

$$||K(R,n)\wedge K(R,m)||_{n+m} \simeq K(R\otimes R, n+m).$$

Evoking the univalence axiom, we get that

$$||K(R,n)\wedge K(R,m)||_{n+m} = K(R\otimes R, n+m).$$

Now, we construct the ring structure on cohomology. The strategy is to define structures and property on EM-spaces then lift those up to cohomology. We begin by defining our EM-spaces. Following Licata and Finster's construction of EM-spaces, we define as follows:

$$K(\mathbb{Z}/2\mathbb{Z}, 0) :\equiv \mathbb{Z}/2\mathbb{Z}$$
$$K(\mathbb{Z}/2\mathbb{Z}, n) :\equiv ||\Sigma_{n-1}\mathbb{R}\mathbf{P}^2||_n$$

To simplify the notation, we write K_n for $K(\mathbb{Z}/2\mathbb{Z}, n)$ when feeling laze. Addition and subtraction operations of type $K_n \times K_n \to K_n$ follow directly from Guillaume'sthesis Proposition 5.1.4. Let's do multiplication. From the multiplication $\mathbb{Z}/2\mathbb{Z} \otimes \mathbb{Z}/2\mathbb{Z} \to \mathbb{Z}/2\mathbb{Z}$, we get a multiplication $K(\mathbb{Z}/2\mathbb{Z} \otimes \mathbb{Z}/2\mathbb{Z}, n) \to K(\mathbb{Z}/2\mathbb{Z}, n)$ by applying the functor

$$K(-, n)\colon \mathbf{Ring} \to \mathbf{Ho}(\mathbf{Top}_*)$$

But

$$||K(R, n) \wedge K(R, m)||_{n+m} = K(R \otimes R, n + m).$$

for any ring R as we showed above, and so we can treat our multiplication as

$$\widehat{\mu}\colon ||K(\mathbb{Z}/2\mathbb{Z}, n) \wedge K(\mathbb{Z}/2\mathbb{Z}, m)||_{n+m} \to K(\mathbb{Z}/2\mathbb{Z}, n + m).$$

However, we *really* want the domain of multiplication to be $K(\mathbb{Z}/2\mathbb{Z}, n) \times K(\mathbb{Z}/2\mathbb{Z}, m)$. To get this, we precompose $\widehat{\mu}$ by several canonical arrows resulting in the composite

$$K(\mathbb{Z}/2\mathbb{Z}, n)\times K(\mathbb{Z}/2\mathbb{Z}, m) \xrightarrow{\texttt{proj}} K(\mathbb{Z}/2\mathbb{Z}, n)\wedge K(\mathbb{Z}/2\mathbb{Z}, m) \xrightarrow{|-|_{n+m}} ||K(\mathbb{Z}/2\mathbb{Z}, n)\wedge K(\mathbb{Z}/2\mathbb{Z}, m)||_{n+m} \xrightarrow{\widehat{\mu}} K(\mathbb{Z}$$

that we call *multiplication* μ. This induces the cup product

$$\smile\colon H^n(X) \times H^m(X) \to H^{n+m}(X)$$

on cohomology. Recall that we define the n-th cohomology with $\mathbb{Z}/2\mathbb{Z}$-coefficients to be $H^n(X) :\equiv ||X \to_* K_n||_0$. This is the standard definition in homotopy type theory, we do not use singular cochains, because taking the set up cochains is not a continuous process. Define $\smile (|\alpha|, |\beta|))$, for $\alpha\colon X \to_* K_n$ and $\beta]fromX \to K_m$, to be the truncation of the pairing $\langle \alpha, \beta \rangle$ followed by μ:

$$\smile (|\alpha|, |\beta|)) : ||X \to K_n \times K_m \to K_{n+m}||_0.$$

Thus $\smile (|\alpha|, |\beta|))$ type-checks. It remains to show the usual ring properties hold for $H^n(X)$. Distributivity follows directly from Guillaume's argument in Proposition 5.1.7 which uses only connectivity hence carries through to our context without issue. To show 'graded commutativity', it suffices to show standard commutativity because of the $\mathbb{Z}/2\mathbb{Z}$-coefficients.