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

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

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 A, B, and C, all of which are sets. The question: is the pushout given by the square

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

also a set when f is a monomorphism?

Thus to determine whether $B +_A C$ 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 $B +_A C$, we define a type

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

so that $\mathtt{code}(x, y)$ serves as our guess as to what $x =_{B+_AC} y$ 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 $B +_A C$. Hopefully, these are mutually inverse.

3. Defining code

Let's try to define

$$\mathtt{code}\colon B +_A C \to B +_A C \to \mathtt{Type}.$$

Note that 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:

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

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

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

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

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

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

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

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

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

Define the coherence 0-cells

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

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

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 $p\colon a = a'$ 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.

$\mathtt{code}\,(\mathtt{inl}(b), \mathtt{inr}(c')) :\equiv \sum_{a:A}(b =_B f(a)) \times (c' =_C g(a))$ 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 $\mathtt{1}$.

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

$\mathtt{code}\,(\mathtt{inr}(c), \mathtt{inr}(c')) :\equiv \sum_{a,a':A}(c =_C g(a)) \times (c' =_C g(a')) \times (f(a) =_B f(a'))$ 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.

Define the coherence 1-cells

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.

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

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

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

Define the coherence 2-cell

$\texttt{code}(\texttt{ap}_{\texttt{glue}}(a), \texttt{ap}_{\texttt{glue}}(a'))$ 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.