Notes on 1-HITs

Gabe Dijkstra

1 Introduction

2 Describing higher inductive types

If we define an ordinary inductive type, we start out by writing down a list of constructors, e.g.:

$$
\begin{aligned}
&\textbf{data}\ \mathsf{T}\ :\ \mathsf{Type}\ \textbf{where} \\
&\quad \mathsf{c0}\ :\ \mathsf{F0}\ \mathsf{T}\ \rightarrow\ \mathsf{T} \\
&\quad \mathsf{c1}\ :\ \mathsf{F1}\ \mathsf{T}\ \rightarrow\ \mathsf{T} \\
&\qquad \vdots \\
&\quad \mathsf{ck}\ :\ \mathsf{Fk}\ \mathsf{T}\ \rightarrow\ \mathsf{T}
\end{aligned}
$$

where all $\mathsf{Fi}\ :\ \mathsf{Type}\ \rightarrow\ \mathsf{Type}$ are strictly positive functors. Equivalently, we can define an inductive type with a single constructor:

$$
\begin{aligned}
&\textbf{data}\ \mathsf{T}\ :\ \mathsf{Type}\ \textbf{where} \\
&\quad \mathsf{c}\ :\ \mathsf{F}\ \mathsf{T}\ \rightarrow\ \mathsf{T}
\end{aligned}
$$

where $\mathsf{F}\ \mathsf{X}\ :\equiv\ \mathsf{F0}\ \mathsf{X} + \mathsf{F1}\ \mathsf{X} + \mathsf{hdots} + \mathsf{Fk}\ \mathsf{X}$, so a single strictly positive functor is all we need to describe an ordinary inductive type.

In the case of higher inductive types, the situation is more involved. Consider for example the circle data type:

$$
\begin{aligned}
&\textbf{data}\ \mathsf{S1}\ :\ \mathsf{Type}\ \textbf{where} \\
&\quad \mathsf{base}\ :\ \mathsf{S1} \\
&\quad \mathsf{loop}\ :\ \mathsf{base}\ =\ \mathsf{base}
\end{aligned}
$$

There are two things that are different from our previous situation. Firstly, the result type of loop is not $\mathsf{S1}$, but a path space of $\mathsf{S1}$: constructors are no longer algebras of functor, but a kind of *dialgebra*: the arguments as well as the result type of a constructor may vary. Secondly, the loop constructor refers to the previous constructor base.

The result type of a constructor can also depend on the value of its arguments, as we see in the definition of propositional truncation as a higher inductive type:

```
data // A // : Type where
  [_] : A → // A //
  trunc : (x y : // A //) → x = y
```

Constructors of a higher inductive type are *dependent dialgebras*. In general, a higher inductive type looks as follows:

```
data T : Type where
  c0    : (x : F0 T)             → G0 T c0 x
  c1    : (x : F1 T c0)          → G1 T c0 x
  c2    : (x : F2 T c0 c1)       → G2 T c0 c1 x
        ⋮
  ck+1  : (x : Fk T c0 hdots ck) → Gk T c0 hdots ck x
```

We will refer to the Fi functors as *argument* functors and the Gi functors as *target* functors. The types of the argument functors are:

```
F0   : Type            → Type
F1   : (F0, G0) -alg   → Type
F2   : (F1, G1) -alg   → Type
Fk+1 : (Fk, Gk) -alg   → Type
```

where (F0, G0) -alg) is the category whose objects are dependent dialgebras (X : Type) $\times$ (θ : (x : F0 X) → G0 X x). The category (F (i+1), G (i+1) -alghas as objects: (X : (Fi, Gi) -alg) $\times$ (θ : (x : F (i+1) X) → G (i+1) X x). The target functors also take the value of the arguments as an argument, so they have the following types:

```
G0   : ∫ Type           F0      → Type
G1   : ∫ (F0, G0) -alg  F1      → Type
G2   : ∫ (F1, G1) -alg  F2      → Type
Gk+1 : ∫ (Fk, Gk) -alg  F (k+1) → Type
```

We see that the general shape of a constructor is a dependent dialgebra:

```
c : (x : F X) → G (X, x)
```

where C : Cat, F : C → Type and G : $\int$ C F → Type, where C is Type or some category of dependent dialgebras.

When describing higher inductive types, we do not allow for any target functor G: it must either return the type we are defining or a (possibly iterated) path space of that type.

A 0-*constructor* or *point constructor* is a dialgebra:

```
c : (x : F X) → U X
```

where C : Cat, F : C → Type and U : C → Type its forgetful functor.

A 1-*constructor* is a dependent dialgebra of which the target functor returns a path space:

$$\mathsf{c} \;:\; (\mathsf{x} \;:\; \mathsf{F}\,\mathsf{X}) \;\to\; \mathsf{Eq0}\,\mathsf{X}\,\mathsf{x}$$

where $\mathsf{C} : \mathsf{Cat}$, $\mathsf{F}, \mathsf{U} : \mathsf{C} \to \mathsf{Type}$, and $\mathsf{Eq0} : \int \mathsf{C}\;\mathsf{F} \to \mathsf{Type}$ is the functor:

$$\mathsf{Eq0}\;(\mathsf{X}, \mathsf{x}) \;:\equiv\; (\mathsf{l0}\,\mathsf{X}\,\mathsf{x} \;=\; \mathsf{r0}\,\mathsf{X}\,\mathsf{x})$$

where $\mathsf{l0}, \mathsf{r0} : \mathsf{F} \to \mathsf{U}$ are natural transformations.

For higher path constructors, we have to specify a tower of natural transformations, specifying the end points at every level. To specify an ($\mathsf{n}{+}1$)-constructor, we need to give the natural transformations:

$$\begin{aligned}
&\mathsf{l0}, \mathsf{r0} \;:\; \mathsf{F} \;\to\; \mathsf{U}\\
&\mathsf{l1}, \mathsf{r1} \;:\; 1 \;\to\; \mathsf{Eq0}\\
&\mathsf{l2}, \mathsf{r2} \;:\; 1 \;\to\; \mathsf{Eq1}\\
&\qquad\vdots\\
&\mathsf{ln}, \mathsf{rn} \;:\; 1 \;\to\; \mathsf{Eqn\text{-}1}
\end{aligned}$$

where $\mathsf{Eqi}\;\mathsf{X}\;\mathsf{x} \;:\equiv\; (\mathsf{li}\;\mathsf{X}\;\mathsf{x} \;=\; \mathsf{ri}\;\mathsf{X}\;\mathsf{x})$. Eqn is then the target functor we are interested in.

2.1 Strict positivity

Apart from the restrictions on targets, we also want all functors involved to be strictly positive. In the case of ordinary inductive types, we also have this restriction. An example of an inductive type which constructor is not strictly positive is the following:

```
data Term : Type where
  lam : (Term → Term) → Term
```

Using the type Term, we can find inhabitants of the empty type. We therefore only consider inductive types defined by strictly positive functors, i.e. containers. Containers however only describe functors Type → Type. An example of that shows that we need a notion of strict positivity for any functor into Type is that of the "initial field". If we write down the axioms of an algebraic structure as constructors, the inductive type we get is then the initial object in the category of that algebraic structure, i.e. we can define a type T with the monoid axioms as constructors:

```
data T : Type where
  uip : (x y : T) (p q : x = y) → p = q
  _•_ : T → T → T
  assoc : (x y z : T) → (x • y) • z = x • (y • z)
  e : T
```

$$\mathsf{e\text{-}unit\text{-}l} : (\mathsf{x} : \mathsf{T}) \to \mathsf{e} \bullet \mathsf{x} = \mathsf{x}$$
$$\mathsf{e\text{-}unit\text{-}r} : (\mathsf{x} : \mathsf{T}) \to \mathsf{x} \bullet \mathsf{e} = \mathsf{x}$$

T is equivalent to the unit type, which is the initial object in the category of monoids. If we now were to write down the axioms of a field, we run into trouble: there is no initial object in the category of fields. The culprit is the inverse operation, which has the type:

$$\mathsf{inv} : (\mathsf{x} : \mathsf{T}) \to (\mathsf{x} = 0 \to \bot) \to \mathsf{T}$$

The constructor $\mathsf{0}$ occurs negatively in this constructor.

To generalise the notion of strict positivity, we can generalise the notion of containers to not only describe functors $\mathsf{Type} \to \mathsf{Type}$, but functors *into* Type from any $\mathsf{C} : \mathsf{Cat}$. A *generalised container* is given by:

$$\mathsf{S} : \mathsf{Type}$$
$$\mathsf{P} : \mathsf{S} \to /\ \mathsf{C}\ /$$

Its extension is then defined as:

$$[\![\ \mathsf{S} \triangleleft \mathsf{P}\ ]\!] : \mathsf{C} \to \mathsf{Type}$$
$$[\![\ \mathsf{S} \triangleleft \mathsf{P}\ ]\!]\ \mathsf{X} :\equiv (\mathsf{s} : \mathsf{S}) \times \mathsf{C}\,(\mathsf{P}\,\mathsf{s}, \mathsf{X})$$

with the action on morphisms defined as:

$$[\![\ \mathsf{F}\ ]\!] : \mathsf{C}\,(\mathsf{X}, \mathsf{Y}) \to [\![\ \mathsf{F}\ ]\!]\ \mathsf{X} \to [\![\ \mathsf{F}\ ]\!]\ \mathsf{Y}$$
$$[\![\ \mathsf{F}\ ]\!]\ \mathsf{f}\,(\mathsf{s}, \mathsf{t}) :\equiv (\mathsf{s}, \mathsf{f} \circ \mathsf{t})$$

2.2 Higher inductive types as a sequence of monads

3 Restricted 1-HITs

coherence and other reasons why things are difficult

4 Algebras

The type of F-*algebras*, or simply *algebras*, can be defined as follows:

$$\mathsf{Alg} :\equiv (\mathsf{X} : \mathsf{Type}) \times (\theta : \mathsf{F}\,\mathsf{X} \to \mathsf{X})$$

where the type morphisms is defined as follows:

$$\mathsf{Alg\text{-}hom} : \mathsf{Alg} \to \mathsf{Alg} \to \mathsf{Type}$$
$$\mathsf{Alg\text{-}hom}\,(\mathsf{X}, \theta)\,(\mathsf{Y}, \rho) :\equiv$$

$$
\begin{aligned}
&(\mathsf{f} : \mathsf{X} \rightarrow \mathsf{Y}) \\
\times\ &(\mathsf{f}\text{-}\beta_0 : (\mathsf{x} : \mathsf{F}\,\mathsf{X}) \rightarrow \mathsf{f}\,(\theta\,\mathsf{x}) = \rho\,(\mathsf{F}\,\mathsf{f}\,\mathsf{x}))
\end{aligned}
$$

The witness of commutativity has suggestively been named $\mathsf{f}\text{-}\beta_0$ as this gives us the β-rule for the recursion and induction principles.

Something about having $\mathsf{f}_0 : \mathsf{f} \circ \theta \equiv \rho \circ \mathsf{F}\,\mathsf{f}$ instead. We need function extensionality either way, but this way it makes the arguments later on a bit easier. Also, the dependent versions don't work as pointfree as these ones

4.1 Homotopy initial algebras

We call an algebra (X, θ) *homotopy initial* if it has the property that for every algebra (Y, ρ), $\mathsf{Alg\text{-}hom}\ (\mathsf{X}, \theta)\ (\mathsf{Y}, \rho)$ is contractible, i.e.:

$$
\begin{aligned}
&\mathsf{is\text{-}initial} : \mathsf{Alg} \rightarrow \mathsf{Type} \\
&\mathsf{is\text{-}initial}\ \theta :\equiv (\rho : \mathsf{Alg}) \rightarrow \mathsf{is\text{-}contr}\ (\mathsf{Alg\text{-}hom}\ \theta\ \rho) \\
&\qquad\qquad\quad \equiv (\rho : \mathsf{Alg}) \rightarrow (\mathsf{f} : \mathsf{Alg\text{-}hom}\ \theta\ \rho) \times ((\mathsf{g} : \mathsf{Alg\text{-}hom}\ \theta\ \rho) \rightarrow \mathsf{f} = \mathsf{g})
\end{aligned}
$$

4.1.1 Equality of algebra morphisms

As we see in the definition of homotopy initiality, we need to be able to talk about equality of algebra morphisms. Given algebras (X, θ) and (Y, ρ) and morphisms $(\mathsf{f}, \mathsf{f}\text{-}\beta_0)$ and $(\mathsf{g}, \mathsf{g}\text{-}\beta_0)$ between them, we know that, by equality on Σ-types, the following holds:

$$
\begin{aligned}
&(\mathsf{f}, \mathsf{f}\text{-}\beta_0) = (\mathsf{g}, \mathsf{g}\text{-}\beta_0) \\
\simeq\ &(\mathsf{p} : \mathsf{f} = \mathsf{g}) \\
\times\ &(\mathsf{p}\text{-}\beta_0 : \mathsf{transport}\ (\lambda\,\mathsf{h} \circ (\mathsf{x} : \mathsf{F}\,\mathsf{X}) \rightarrow \mathsf{h}\,(\theta\,\mathsf{x}) = \rho\,(\mathsf{F}\,\mathsf{h}\,\mathsf{x}))\ \mathsf{f}\text{-}\beta_0 = \mathsf{g}\text{-}\beta_0)
\end{aligned}
$$

We not only need to show that the functions f and g are equal, but also that their β-laws are in some sense compatible with eachother, respecting the equality $\mathsf{f} = \mathsf{g}$.

As it turns out, the above is equivalent to something which is more convenient in subsequent proofs:

$$
\begin{aligned}
&\mathsf{transport}\ (\lambda\,\mathsf{h} \rightarrow (\mathsf{x} : \mathsf{F}\,\mathsf{X}) \rightarrow \mathsf{h}\,(\theta\,\mathsf{x}) = \rho\,(\mathsf{F}\,\mathsf{h}\,\mathsf{x}))\ \mathsf{p}\ \mathsf{f}\text{-}\beta_0 = \mathsf{g}\text{-}\beta_0 \\
\simeq\ &\{\mathsf{transport\ over\ \Pi\text{-}types}\} \\
&(\lambda\,\mathsf{x} \rightarrow \mathsf{transport}\ (\lambda\,\mathsf{h} \rightarrow \mathsf{h}\,(\theta\,\mathsf{x}) = \rho\,(\mathsf{F}\,\mathsf{h}\,\mathsf{x}))\ \mathsf{p}\ (\mathsf{f}\text{-}\beta_0\,\mathsf{x})) = \mathsf{g}\text{-}\beta_0 \\
\simeq\ &\{\mathsf{function\ extensionality}\} \\
&((\mathsf{x} : \mathsf{A}) \rightarrow \mathsf{transport}\ (\lambda\,\mathsf{h} \rightarrow \mathsf{h}\,(\theta\,\mathsf{x}) = \rho\,(\mathsf{F}\,\mathsf{h}\,\mathsf{x}))\ \mathsf{p}\ (\mathsf{f}\text{-}\beta_0\,\mathsf{x}) = \mathsf{g}\text{-}\beta_0\,\mathsf{x}) \\
\simeq\ &\{\mathsf{transporting\ over\ equalities}\} \\
&!\ (\mathsf{ap}\ (\lambda\,\mathsf{h} \rightarrow \mathsf{h}\,(\theta\,\mathsf{x}))\ \mathsf{p}) \cdot \mathsf{f}\text{-}\beta_0\,\mathsf{x} \cdot \mathsf{ap}\ (\lambda\,\mathsf{h} \rightarrow \rho\,(\mathsf{F}\,\mathsf{h}\,\mathsf{x}))\ \mathsf{p} = \mathsf{g}\text{-}\beta_0\,\mathsf{x} \\
\simeq\ &\{\mathsf{path\ algebra}\} \\
&\mathsf{f}\text{-}\beta_0\,\mathsf{x} \cdot \mathsf{ap}\ (\lambda\,\mathsf{h} \rightarrow \rho\,(\mathsf{F}\,\mathsf{h}\,\mathsf{x}))\ \mathsf{p} = \mathsf{ap}\ (\lambda\,\mathsf{h} \rightarrow \mathsf{h}\,(\theta\,\mathsf{x}))\ \mathsf{p} \cdot \mathsf{g}\text{-}\beta_0\,\mathsf{x}
\end{aligned}
$$

The last equation tells us that showing that two algebra morphisms are equal is somewhat like giving an algebra morphism from the witness of the β-law for f to that of g, as we can see if we draw the corresponding diagram:

comm diag

4.2 Algebras for restricted 1-HIT descriptions

$$
\begin{aligned}
&\mathsf{Alg} \;:\equiv \\
&\quad\;\; (\mathsf{X} : \mathsf{Type}) \\
&\quad \times (\theta_0 : \mathsf{F}_0\,\mathsf{X} \to \mathsf{X}) \\
&\quad \times (\theta_1 : (\mathsf{x} : \mathsf{F}_1\,\mathsf{X}) \to \mathsf{l}\,(\theta_0 * \mathsf{x}) = \mathsf{r}\,(\theta_0 * \mathsf{x}))
\end{aligned}
$$

θ_0 is an F_0-algebra and θ_1 can be seen as a dependent dialgebra by defining the functor G_1:

$$
\begin{aligned}
&\mathsf{G}_1 : \textstyle\int \mathsf{F}_0\text{-}\mathsf{Algebra}\,\mathsf{F}_1 \to \mathsf{Type} \\
&\mathsf{G}_1\,((\mathsf{X},\theta),\mathsf{x}) :\equiv (\mathsf{l}\,(\theta_0 * \mathsf{x}) = \mathsf{r}\,(\theta_0 * \mathsf{x}))
\end{aligned}
$$

Its hom-types can be defined as follows:

$$
\begin{aligned}
&\mathsf{Alg\text{-}hom} : \mathsf{Alg} \to \mathsf{Alg} \to \mathsf{Type} \\
&\mathsf{Alg\text{-}hom}\,(\mathsf{X},\theta)\,(\mathsf{Y},\rho) :\equiv \\
&\quad\;\; (\mathsf{f} : \mathsf{X} \to \mathsf{Y}) \\
&\quad \times (\mathsf{f}\text{-}\beta_0 : (\mathsf{x} : \mathsf{F}_0\,\mathsf{X}) \to \mathsf{f}\,(\theta_0\,\mathsf{x}) = \rho_0\,(\mathsf{F}_0\,\mathsf{f}\,\mathsf{x})) \\
&\quad \times (\mathsf{f}\text{-}\beta_1 : (\mathsf{x} : \mathsf{F}_1\,\mathsf{X}) \to \mathsf{G}_1\,\mathsf{x}\,\mathsf{f},\mathsf{f}_0\,(\theta_1\,\mathsf{x}) = \rho_1\,(\mathsf{F}_1\,\mathsf{f}\,\mathsf{x}))
\end{aligned}
$$

5 Induction

Given a functor $\mathsf{F} : \mathsf{Type} \to \mathsf{Type}$ given as a container $\mathsf{F} :\equiv \mathsf{S} \lhd \mathsf{P}$, the W-type W is defined as having the following introduction rule / constructor:

$$
\mathsf{c} : \mathsf{F}\,\mathsf{W} \to \mathsf{W}
$$

as well as an elimination rule / induction principle:

$$
\begin{aligned}
&\mathsf{ind} : \\
&\qquad (\mathsf{B} : \mathsf{W} \to \mathsf{Type}) \\
&\qquad (\mathsf{m} : (\mathsf{x} : \mathsf{F}\,\mathsf{W}) \to \Box\,\mathsf{F}\,\mathsf{B}\,\mathsf{x} \to \mathsf{B}\,(\mathsf{c}\,\mathsf{x})) \\
&\qquad (\mathsf{x} : \mathsf{W}) \\
&\quad \to \mathsf{B}\,\mathsf{x}
\end{aligned}
$$

with computation rule:

$$
\begin{aligned}
&\mathsf{ind}\text{-}\beta_0 : \\
&\qquad (\mathsf{B} : \mathsf{W} \to \mathsf{Type}) \\
&\qquad (\mathsf{m} : (\mathsf{x} : \mathsf{F}\,\mathsf{W}) \to \Box\,\mathsf{F}\,\mathsf{B}\,\mathsf{x} \to \mathsf{B}\,(\mathsf{c}\,\mathsf{x})) \\
&\qquad (\mathsf{x} : \mathsf{F}\,\mathsf{W}) \\
&\quad \to \mathsf{ind}\,\mathsf{B}\,\mathsf{m}\,(\mathsf{c}\,\mathsf{x}) = \mathsf{m}\,\mathsf{x}\,(\Box\text{-}\mathsf{lift}\,\mathsf{F}\,(\mathsf{ind}\,\mathsf{B}\,\mathsf{m})\,\mathsf{x})
\end{aligned}
$$

5.1 Initiality implies induction

The induction principle tells us that for a family $\mathsf{B} : \mathsf{W} \to \mathsf{Type}$ and a motive $\mathsf{m} : (\mathsf{x} : \mathsf{F}\ \mathsf{W}) \to \Box\ \mathsf{F}\ \mathsf{B}\ \mathsf{x} \to \mathsf{B}\ (\mathsf{c}\ \mathsf{x})$, we get a dependent function $\mathsf{ind} : (\mathsf{x} : \mathsf{W}) \to \mathsf{B}\,\mathsf{x}$ along with a computation rule.

Note that m, along with c, can be seen as a morphism between the families (W, B) and $(\mathsf{F}\ \mathsf{W}, \Box\ \mathsf{F}\ \mathsf{B})$.

Another way to say this is that given a function $\mathsf{p} : \mathsf{B} \to \mathsf{W}$ for some $\mathsf{B} : \mathsf{Type}$ and that we want $\theta : \mathsf{F}\ \mathsf{B} \to \mathsf{B}$ such that θ, along with c, becomes a morphism $\mathsf{Fp} \to \mathsf{p}$ in the arrow category. This is equivalent to asking for an algebra (B, θ) along with an algebra morphism $(\mathsf{B}, \theta) \to (\mathsf{W}, \mathsf{c})$.

Give arguments why section induction is correct

Show that initiality implies section induction

5.2 Induction principle for restricted 1-HITs

To get the induction principle of restricted 1-HITs , we need to take that of the ordinary W-types and extend it with the new (path) constructor: we have to add a method to ind and the appropriate computation rule for that.

Consider the method for the 0-constructor:

$$\mathsf{m}_0 : (\mathsf{x} : \mathsf{F}_0\ \mathsf{X}) \to \Box\ \mathsf{F}_0\ \mathsf{B}\ \mathsf{x} \to \mathsf{B}\ (\mathsf{c}_0\ \mathsf{x})$$

We can read this as showing that if for all subterms of $\mathsf{c}_0\ \mathsf{x}$ B holds, B also holds for $\mathsf{c}_0\ \mathsf{x}$ itself. Since we are dealing with a 0-constructor, we don't have any issues with the target of the constructor: the target functor is the identity functor. For the motive of the 1-constructor, we want something like:

$$\mathsf{m}_1 : (\mathsf{x} : \mathsf{F}_1\ \mathsf{X}) \to \Box\ \mathsf{F}_1\ \mathsf{B}\ \mathsf{x} \to \Box\ \mathsf{G}\ (\mathsf{c}_1\ \mathsf{x})$$

We need to figure out what the lifting of B to the target functor G is.

6 Induction implies homotopy initiality

We want to show that, given $\mathsf{T} : \mathsf{Type}$ and $\mathsf{c} : \mathsf{F}\ \mathsf{T} \to \mathsf{T}$ that satisfies the induction principle, the algebra (T, c) is initial, i.e. for any algebra (X, θ), $\mathsf{Alg\text{-}hom}\ (\mathsf{T}, \mathsf{c})\ (\mathsf{X}, \theta) \simeq 1$. We will first show that we have an algebra morphism $\mathsf{f} : (\mathsf{T}, \mathsf{c}) \to (\mathsf{X}, \theta)$ and will then show that this algebra morphism is unique.

6.1 W-types

6.1.1 Existence

We can use the induction principle to produce the algebra morphism we want:

$$\mathsf{f} : \mathsf{T} \to \mathsf{X}$$
$$\mathsf{f} :\equiv \mathsf{ind}\ (\lambda\, \mathsf{x} \to \mathsf{X})\ (\lambda\ (\mathsf{s}, \_)\ \mathsf{t} \to \theta\ (\mathsf{s}, \mathsf{t}))$$

The computation rule is then given directly by the computation rule for the induction rule:

$$\mathsf{f}_0 : (\mathsf{x} : \mathsf{F}\ \mathsf{T}) \to \mathsf{f}\ (\theta\ \mathsf{x}) = \theta\ (\mathsf{F}\ \mathsf{f}\ \mathsf{x})$$
$$\mathsf{f}_0 :\equiv \mathsf{ind}\text{-}\beta$$

6.1.2 Uniqueness

Assuming that we have an algebra morphism $(\mathsf{g}, \mathsf{g}_0) : (\mathsf{T}, \mathsf{c}) \to (\mathsf{X}, \theta)$, we need to show that $(\mathsf{f}, \mathsf{f}_0) = (\mathsf{g}, \mathsf{g}_0)$. Showing that the $\mathsf{f} = \mathsf{g}$ can be done by induction, using the motive $\lambda\ \mathsf{x} \to \mathsf{f}\ \mathsf{x} = \mathsf{g}\ \mathsf{x}$. The induction step is then proven as follows:

$$\mathsf{step} : (\mathsf{x} : [\![\ \mathsf{F}\ ]\!]_0\ \mathsf{T}) \to \Box\ \mathsf{F}\ \mathsf{f{=}g\text{-}B}\ \mathsf{x} \to \mathsf{f{=}g\text{-}B}\ (\mathsf{c}\ \mathsf{x})$$
$$\mathsf{step}\ \mathsf{x}\ \mathsf{u} :\equiv$$
$$\mathsf{f}\ (\mathsf{c}\ \mathsf{x})$$
$$=\{\ \mathsf{f}_0\ \mathsf{x}\ \}$$
$$\theta\ ([\![\ \mathsf{F}\ ]\!]_1\ \mathsf{f}\ \mathsf{x})$$
$$=\{\ \mathsf{ap}\ \theta\ (\mathsf{ind\text{-}hyp}\ \mathsf{x}\ \mathsf{u})\ \}$$
$$\theta\ ([\![\ \mathsf{F}\ ]\!]_1\ \mathsf{g}\ \mathsf{x})$$
$$=\{\ !\ (\mathsf{g}_0\ \mathsf{x})\ \}$$
$$\mathsf{g}\ (\mathsf{c}\ \mathsf{x})\ \blacksquare$$

Explain ind-hyp

We can define the witness of $\mathsf{f} = \mathsf{g}$ as follows:

$$\mathsf{p} : (\mathsf{x} : \mathsf{T}) \to \mathsf{f}\ \mathsf{x} = \mathsf{g}\ \mathsf{x}$$
$$\mathsf{p} :\equiv \mathsf{ind}\ (\lambda\ \mathsf{x} \to \mathsf{f}\ \mathsf{x} = \mathsf{g}\ \mathsf{x})\ \mathsf{step}$$

which comes with the computation rule:

$$\mathsf{p}\text{-}\beta_0 : (\mathsf{x} : \mathsf{FT}) \to \mathsf{p}\ (\mathsf{c}\ \mathsf{x}) = \mathsf{f}\text{-}\beta_0\ \mathsf{x} \cdot \mathsf{ap}\ \theta\ (\mathsf{ind\text{-}hyp}\ \mathsf{x}\ (\Box\text{-}\mathsf{lift}\ \mathsf{F}\ \mathsf{p}\ \mathsf{x})) \cdot\ !\ (\mathsf{g}\text{-}\beta_0\ \mathsf{x})$$

say that $\lambda{=}$ is fun ext

We now need to show that our witness p satisfies the "computation rule" $(\mathsf{x} : \mathsf{FT}) \to \mathsf{f}\text{-}\beta_0\ \mathsf{x} \cdot \mathsf{ap}\ (\lambda\ \mathsf{h} \to \rho\ (\mathsf{F}\ \mathsf{h}\ \mathsf{x}))\ \mathsf{p} = \mathsf{ap}\ (\lambda\ \mathsf{h} \to \mathsf{h}\ (\theta\ \mathsf{x}))\ \mathsf{p} \cdot \mathsf{g}\text{-}\beta_0\ \mathsf{x}$. Let $\mathsf{x} : \mathsf{FT}$, then we can calculate:

$\mathsf{f}\text{-}\beta_0\ \mathsf{x} \cdot \mathsf{ap}\ (\lambda\ \mathsf{h} \to \theta\ (\mathsf{F}\ \mathsf{h}\ \mathsf{x}))\ (\lambda{=}\ \mathsf{p})$
$\quad = \{\mathsf{ap\ magic}\}$
$\mathsf{f}\text{-}\beta_0\ \mathsf{x} \cdot \mathsf{ap}\ \theta\ (\mathsf{ind\text{-}hyp}\ \mathsf{x}\ (\Box\text{-}\mathsf{lift}\ \mathsf{F}\ \mathsf{p}\ \mathsf{x}))$
$\quad = \{\mathsf{symmetry\ is\ involutive}\}$
$\mathsf{f}\text{-}\beta_0\ \mathsf{x} \cdot \mathsf{ap}\ \theta\ (\mathsf{ind\text{-}hyp}\ \mathsf{x}\ (\Box\text{-}\mathsf{lift}\ \mathsf{F}\ \mathsf{p}\ \mathsf{x})) \cdot\ !\ (\mathsf{g}\text{-}\beta_0\ \mathsf{x}) \cdot \mathsf{g}\text{-}\beta_0\ \mathsf{x}$
$\quad = \{\mathsf{computation\ rule\ for\ p}\}$
$\mathsf{p}\ (\mathsf{c}\ \mathsf{x}) \cdot \mathsf{g}\text{-}\beta_0\ \mathsf{x}$
$\quad = \{\mathsf{computation\ rule\ for}\ \lambda{=}\}$
$\mathsf{ap}\ (\lambda\ \mathsf{h} \to \mathsf{h}\ (\mathsf{c}\ \mathsf{x}))\ (\lambda{=}\ \mathsf{p}) \cdot \mathsf{g}\text{-}\beta_0\ \mathsf{x}$

We have now established initiality of (T, c).

6.2 Restricted 1-HITs

6.2.1 Existence

6.2.2 Uniqueness