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.:

$$\begin{array}{l}
\textbf{data}\ \mathsf{T}\ :\ \mathsf{Type}\ \textbf{where} \\
\quad \mathsf{c}_0\ :\ \mathsf{F}_0\ \mathsf{T}\ \to\ \mathsf{T} \\
\quad \mathsf{c}_1\ :\ \mathsf{F}_1\ \mathsf{T}\ \to\ \mathsf{T} \\
\qquad \vdots \\
\quad \mathsf{c}_\mathsf{k}\ :\ \mathsf{F}_\mathsf{k}\ \mathsf{T}\ \to\ \mathsf{T}
\end{array}$$

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

$$\begin{array}{l}
\textbf{data}\ \mathsf{T}\ :\ \mathsf{Type}\ \textbf{where} \\
\quad \mathsf{c}\ :\ \mathsf{F}\ \mathsf{T}\ \to\ \mathsf{T}
\end{array}$$

where $\mathsf{F}\ \mathsf{X}\ :\equiv\ \mathsf{F}_0\ \mathsf{X} + \mathsf{F}_1\ \mathsf{X} +\ \ldots\ + \mathsf{F}_\mathsf{k}\ \mathsf{X}$, 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:

$$\begin{array}{l}
\textbf{data}\ \mathsf{S}^1\ :\ \mathsf{Type}\ \textbf{where} \\
\quad \mathsf{base}\ :\ \mathsf{S}^1 \\
\quad \mathsf{loop}\ :\ \mathsf{base}\ =\ \mathsf{base}
\end{array}$$

There are two things that are different from our previous situation. Firstly, the result type of loop is not S^1, but a path space of S^1: 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 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 ... ck) → Gk T c0 ... ck x
```

We will refer to the $\mathsf{F_i}$ functors as *argument* functors and the $\mathsf{G_i}$ functors as *target* functors. The types of the argument functors are:

$$\begin{array}{lll}
\mathsf{F_0} : \mathsf{Type} & \to & \mathsf{Type} \\
\mathsf{F_1} : (\mathsf{F_0}, \mathsf{G_0})\ \text{-}\mathsf{alg} & \to & \mathsf{Type} \\
\mathsf{F_2} : (\mathsf{F_1}, \mathsf{G_1})\ \text{-}\mathsf{alg} & \to & \mathsf{Type} \\
\mathsf{F_{k+1}} : (\mathsf{F_k}, \mathsf{G_k})\ \text{-}\mathsf{alg} & \to & \mathsf{Type}
\end{array}$$

where $(\mathsf{F_0}, \mathsf{G_0})$ -alg) is the category whose objects are dependent dialgebras $(\mathsf{X} : \mathsf{Type}) \times (\theta : (\mathsf{x} : \mathsf{F_0}\ \mathsf{X}) \to \mathsf{G_0}\ \mathsf{X}\ \mathsf{x})$. The category $(\mathsf{F_{i+1}}, \mathsf{G_{i+1}})$ -alg has as objects: $(\mathsf{X} : (\mathsf{F_i}, \mathsf{G_i})\ \text{-}\mathsf{alg}) \times (\theta : (\mathsf{x} : \mathsf{F_{i+1}}\ \mathsf{X}) \to \mathsf{G_{i+1}}\ \mathsf{X}\ \mathsf{x})$. The target functors also take the value of the arguments as an argument, so they have the following types:

$$\begin{array}{llll}
\mathsf{G_0} & : \int \mathsf{Type} & \mathsf{F_0} & \to \mathsf{Type} \\
\mathsf{G_1} & : \int (\mathsf{F_0}, \mathsf{G_0})\ \text{-}\mathsf{alg}\ \mathsf{F_1} & & \to \mathsf{Type} \\
\mathsf{G_2} & : \int (\mathsf{F_1}, \mathsf{G_1})\ \text{-}\mathsf{alg}\ \mathsf{F_2} & & \to \mathsf{Type} \\
\mathsf{G_{k+1}} & : \int (\mathsf{F_k}, \mathsf{G_k})\ \text{-}\mathsf{alg}\ \mathsf{F}\ (\mathsf{kp1}) & & \to \mathsf{Type}
\end{array}$$

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

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

where $\mathbb{C} : \mathsf{Cat}$, $\mathsf{F} : \mathbb{C} \to \mathsf{Type}$ and $\mathsf{G} : \int \mathbb{C}\ \mathsf{F} \to \mathsf{Type}$, where $\mathbb{C}$ is Type or some category of dependent dialgebras, "containing" all the previous constructors.

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:

$$\mathsf{c} : (\mathsf{x} : \mathsf{F}\ \mathsf{X}) \to \mathsf{U}\ \mathsf{X}$$

where $\mathbb{C} : \mathsf{Cat}$, $\mathsf{F} : \mathbb{C} \to \mathsf{Type}$ and $\mathsf{U} : \mathbb{C} \to \mathsf{Type}$ its forgetful functor.

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

$$\mathsf{c} : (\mathsf{x} : \mathsf{F}\,\mathsf{X}) \to \mathsf{Eq}_0\,\mathsf{X}\,\mathsf{x}$$

where $\mathbb{C} : \mathsf{Cat}$, $\mathsf{F}, \mathsf{U} : \mathbb{C} \to \mathsf{Type}$, and $\mathsf{Eq}_0 : \int \mathbb{C}\ \mathsf{F} \to \mathsf{Type}$ is the functor:

$$\mathsf{Eq}_0\ (\mathsf{X}, \mathsf{x}) :\equiv (\mathsf{l}_0\,\mathsf{X}\,\mathsf{x} = \mathsf{r}_0\,\mathsf{X}\,\mathsf{x})$$

where $\mathsf{l}_0, \mathsf{r}_0 : \mathsf{F} \to \mathsf{U}$ 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 (k+1)-constructor, we need to give the natural transformations:

$$\begin{aligned}
\mathsf{l}_0, \mathsf{r}_0 &: \mathsf{F} \to \mathsf{U} \\
\mathsf{l}_1, \mathsf{r}_1 &: 1 \to \mathsf{Eq}_0 \\
\mathsf{l}_2, \mathsf{r}_2 &: 1 \to \mathsf{Eq}_1 \\
&\vdots \\
\mathsf{l}_k, \mathsf{r}_k &: 1 \to \mathsf{Eq}_{k-1}
\end{aligned}$$

where $\mathsf{Eq}_i\ \mathsf{X}\ \mathsf{x} :\equiv (\mathsf{l}_i\ \mathsf{X}\ \mathsf{x} = \mathsf{r}_i\ \mathsf{X}\ \mathsf{x})$. 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
```

$$
\begin{aligned}
&\mathsf{e\text{-}unit\text{-}l} : (\mathsf{x} : \mathsf{T}) \to \mathsf{e} \cdot \mathsf{x} = \mathsf{x}\\
&\mathsf{e\text{-}unit\text{-}r} : (\mathsf{x} : \mathsf{T}) \to \mathsf{x} \cdot \mathsf{e} = \mathsf{x}
\end{aligned}
$$

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 $\mathbb{C} : \mathsf{Cat}$. A *generalised container* is given by:

$$
\begin{aligned}
&\mathsf{S} : \mathsf{Type}\\
&\mathsf{P} : \mathsf{S} \to |\,\mathbb{C}\,|
\end{aligned}
$$

Its extension is then defined as:

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

with the action on morphisms defined as:

$$
\begin{aligned}
&[\![\, \mathsf{S} \triangleleft \mathsf{P} \,]\!] : \mathbb{C}\,(\mathsf{X}, \mathsf{Y}) \to [\![\, \mathsf{S} \triangleleft \mathsf{P} \,]\!]\, \mathsf{X} \to [\![\, \mathsf{S} \triangleleft \mathsf{P} \,]\!]\, \mathsf{Y}\\
&[\![\, \mathsf{S} \triangleleft \mathsf{P} \,]\!]\, \mathsf{f}\,(\mathsf{s}, \mathsf{t}) :\equiv (\mathsf{s}, \mathsf{f} \circ \mathsf{t})
\end{aligned}
$$

Just as with ordinary containers, we can define container morphisms for generalised containers in the standard way: for containers $\mathsf{S} \triangleleft \mathsf{P}$ and $\mathsf{T} \triangleleft \mathsf{Q}$, a container morphism $\mathsf{S} \triangleleft \mathsf{P} \to \mathsf{T} \triangleleft \mathsf{Q}$ is comprised of:

$$
\begin{aligned}
&\mathsf{f} : \mathsf{S} \to \mathsf{T}\\
&\mathsf{g} : (\mathsf{a} : \mathsf{S}) \to \mathbb{C}\,(\mathsf{Q}\,(\mathsf{f\,a}), \mathsf{P\,a})
\end{aligned}
$$

with its action of objects defined as:

$$
\begin{aligned}
&[\![\, (\mathsf{f}, \mathsf{g}) \,]\!] : (\mathsf{X} : |\,\mathbb{C}\,|) \to [\![\, \mathsf{S} \triangleleft \mathsf{P} \,]\!]\, \mathsf{X} \to [\![\, \mathsf{T} \triangleleft \mathsf{Q} \,]\!]\, \mathsf{X}\\
&[\![\, (\mathsf{f}, \mathsf{g}) \,]\!]\, \mathsf{X}\,(\mathsf{s}, \mathsf{t}) :\equiv (\mathsf{f\,s}, \mathsf{t} \circ (\mathsf{g\,s}))
\end{aligned}
$$

2.2 Higher inductive types as a sequence of monads

Given a strictly positive functor $\mathsf{F} : \mathsf{Type} \to \mathsf{Type}$, we can consider its category of algebras $\mathsf{F\text{-}alg}$. $\mathsf{F\text{-}alg}$ has a forgetful functor $\mathsf{U} : \mathsf{F\text{-}alg} \to \mathsf{Type}$, which has a left adjoint $\mathsf{L} : \mathsf{Type} \to \mathsf{F\text{-}alg}$ that maps a type to its free F-algebra. Note that plugging in the initial object in Type, i.e. the empty type into L gives us the initial object in $\mathsf{F\text{-}alg}$, as left adjoints preserve colimits. The monad UL is called the *free monad* of F, also denoted as $\mathsf{F}\,*$. This monad has the property that the category of F-algebras is equivalent to the category of monad algebras of $\mathsf{F}\,*$. Higher inductive types can be contrasted to ordinary inductive types as being characterised by a more general class of monads than just free monads.

Explain more of how we can then specify this

3 Restricted 1-HITs

Instead of considering a higher inductive type with arbitrarily many constructors, we can restrict ourselves to having two constructors: a 0-constructor followed by a 1-constructor. The data we need to supply for this is:

$$\begin{aligned} \mathsf{F}_0 &: \mathsf{Type} \to \mathsf{Type} \\ \mathsf{F}_1 &: \mathsf{F}_0\text{-}\mathsf{alg} \to \mathsf{Type} \end{aligned}$$

with the target of the 1-constructor given by:

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

for natural transformations $\mathsf{l}, \mathsf{r} : \mathsf{F}_1 \to \mathsf{U}$ where $\mathsf{U} : \mathsf{F}_0\text{-}\mathsf{alg} \to \mathsf{Type}$ the forgetful functor. If F_1 factors through U, i.e. the arguments of the 1-constructor do not refer to the 0-constructor, then can simplify the situation even further. Suppose $\mathsf{F}_1 = \mathsf{F}_1' \circ \mathsf{U}$ for some $\mathsf{F}_1' : \mathsf{Type} \to \mathsf{Type}$, natural transformations $\mathsf{F}_1 \to \mathsf{U}$ can be given by natural transformations $\mathsf{F}_1' \to \mathsf{F}_0\,*$, where $\mathsf{F}_0\,*$ is the free monad of F_0. Recall that U has a left adjoint L such that $\mathsf{F}_0\,* = \mathsf{UL}$. Since U has a left adjoint, it is representable, for any $\mathsf{X} : \mathsf{Type}$:

$$\begin{aligned} \mathsf{U}\,\mathsf{X} &= 1 \to \mathsf{U}\,\mathsf{X} \\ &= \mathsf{F}_0\text{-}\mathsf{alg}\,(\mathsf{L}_1, \mathsf{U}\,\mathsf{X}) \end{aligned}$$

If $\mathsf{F}_1' \equiv \mathsf{S} \lhd \mathsf{P}$, then we can characterise natural transformations $\mathsf{F}_1 \to \mathsf{U}$ as container morphisms. We can calculate the container representation of the composition $\mathsf{F}_1' \circ \mathsf{U}$ as follows: suppose $\mathsf{X} : \mathsf{Type}$, then:

$$\begin{aligned} \mathsf{F}_1'\,(\mathsf{U}\,\mathsf{X}) &= \mathsf{F}_1'\,(\mathsf{F}_0\text{-}\mathsf{alg}\,(\mathsf{L}_1, \mathsf{X})) \\ &= (\mathsf{s} : \mathsf{S}) \times (\mathsf{P}\,\mathsf{s} \to \mathsf{F}_0\text{-}\mathsf{alg}\,(\mathsf{L}_1, \mathsf{X})) \\ &= (\mathsf{s} : \mathsf{S}) \times (\mathsf{P}\,\mathsf{s} \to 1 \to \mathsf{U}\,\mathsf{X}) \\ &= (\mathsf{s} : \mathsf{S}) \times (\mathsf{P}\,\mathsf{s} \to \mathsf{U}\,\mathsf{X}) \\ &= (\mathsf{s} : \mathsf{S}) \times (\mathsf{F}_0\text{-}\mathsf{alg}\,(\mathsf{L}\,(\mathsf{P}\,\mathsf{s}), \mathsf{X})) \end{aligned}$$

So the container representation of $\mathsf{F}_1' \circ \mathsf{U}$ is $\mathsf{S} \lhd \mathsf{L} \circ \mathsf{P}$. Using this we can derive the following:

$$\begin{aligned} &\mathsf{F}_1 \to \mathsf{U} \\ &\quad = \{\text{definition of } \mathsf{F}_1\} \\ &\mathsf{F}_1' \circ \mathsf{U} \to \mathsf{U} \end{aligned}$$

$$\begin{array}{l}
\quad = \{\mathsf{container\ morphisms}\} \\
(\mathsf{s} : \mathsf{S}) \to \mathsf{U}\,(\mathsf{L}\,(\mathsf{P}\,\mathsf{s})) \\
\quad = \{\mathsf{definition\ of}\ \mathsf{F}\,{*}\} \\
(\mathsf{s} : \mathsf{S}) \to \mathsf{F}_0\,{*}\,(\mathsf{P}\,\mathsf{s}) \\
\quad = \{\mathsf{container\ morphisms}\} \\
\mathsf{F}_1' \to \mathsf{F}_0\,{*}
\end{array}$$

The data we need to describe a restricted 1-HIT consists of:

$$\begin{array}{l}
\mathsf{F}_0 : \mathsf{Type} \to \mathsf{Type} \\
\mathsf{F}_1 : \mathsf{Type} \to \mathsf{Type} \\
\mathsf{l}, \mathsf{r} : \mathsf{F}_1 \to \mathsf{F}_0\,{*}
\end{array}$$

where $\mathsf{F}_0, \mathsf{F}_1$ are given as container and l, r as container morphisms. Given these data, the restricted 1-HIT then looks as follows:

$$\begin{array}{l}
\textbf{data}\ \mathsf{T} : \mathsf{Type}\ \textbf{where} \\
\quad \mathsf{c}_0 : \mathsf{F}_0\,\mathsf{T} \to \mathsf{T} \\
\quad \mathsf{c}_1 : (\mathsf{x} : \mathsf{F}_1\,\mathsf{T}) \to \mathsf{c}_0\,{*}\,(\mathsf{l}\,\mathsf{x}) = \mathsf{c}_0\,{*}\,(\mathsf{r}\,\mathsf{x})
\end{array}$$

where $\mathsf{c}_0\,{*} : \mathsf{F}_0\,{*}\,\mathsf{T} \to \mathsf{T}$ is the lifting of the F_0-algebra $\mathsf{c}_0 : \mathsf{F}_0\,\mathsf{T} \to \mathsf{T}$ to a (monad) $\mathsf{F}_0\,{*}$-algebra.

The advantage of restricting ourselves to the above situation is that we can give the description of a higher inductive type using only ordinary containers and container morphisms. This avoids dealing with the coherence problems that stem from defining categories of algebras, needed to give the generalised containers for the constructors.

Many higher inductive types we see "in the wild" can be captured by this approach, such as the circle and n-truncation.

One limitation however is that it does not handle "nested" higher inductive types directly. The following example is not expressible, as the functor $||\,\_\,|| : \mathsf{Type} \to \mathsf{Type}$ is not a container:

$$\begin{array}{l}
\textbf{data}\ \mathsf{T} : \mathsf{Type}\ \textbf{where} \\
\quad \mathsf{c} : ||\,\mathsf{T}\,|| \to \mathsf{T}
\end{array}$$

4 Algebras

Before we set out to show that induction implies homotopy initiality and vice versa, we need to define exactly what we mean when we talk about these concepts. First we will define the relevant concepts for ordinary F-algebras and later give the definitions of algebras for restricted 1-HITs .

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 of algebra morphisms is defined as functions between the carriers such that it respects both algebra structures:

$$\begin{aligned}
&\mathsf{Alg\text{-}hom} \;:\; \mathsf{Alg} \;\to\; \mathsf{Alg} \;\to\; \mathsf{Type}\\
&\mathsf{Alg\text{-}hom}\;(\mathsf{X},\theta)\;(\mathsf{Y},\rho) \;:\equiv\\
&\qquad (\mathsf{f} \;:\; \mathsf{X} \;\to\; \mathsf{Y})\\
&\quad \times (\mathsf{f}\text{-}\beta_0 \;:\; (\mathsf{x} \;:\; \mathsf{F}\,\mathsf{X}) \;\to\; \mathsf{f}\;(\theta\,\mathsf{x}) \;=\; \rho\;(\mathsf{F}\,\mathsf{f}\,\mathsf{x}))
\end{aligned}$$

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

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.:

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

Note that this definition does not refer to how Alg and $\mathsf{Alg\text{-}hom}$ are defined exactly, so this definition also holds for the algebras for restricted 1-HITs .

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:

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

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:

$$\begin{aligned}
&\quad \mathsf{transport}\;(\lambda\,\mathsf{h}\;.\;(\mathsf{x} \;:\; \mathsf{F}\,\mathsf{X}) \to \mathsf{h}\;(\theta\,\mathsf{x}) \;=\; \rho\;(\mathsf{F}\,\mathsf{h}\,\mathsf{x}))\;\mathsf{p}\;\mathsf{f}\text{-}\beta_0 \;=\; \mathsf{g}\text{-}\beta_0\\
&\simeq \{\mathsf{transport\ over\ \Pi\text{-}types}\}\\
&\quad (\lambda\,\mathsf{x}\;.\;\mathsf{transport}\;(\lambda\,\mathsf{h}\;.\;\mathsf{h}\;(\theta\,\mathsf{x}) \;=\; \rho\;(\mathsf{F}\,\mathsf{h}\,\mathsf{x}))\;\mathsf{p}\;(\mathsf{f}\text{-}\beta_0\,\mathsf{x})) \;=\; \mathsf{g}\text{-}\beta_0\\
&\simeq \{\mathsf{function\ extensionality}\}\\
&\quad ((\mathsf{x} \;:\; \mathsf{A}) \to \mathsf{transport}\;(\lambda\,\mathsf{h}\;.\;\mathsf{h}\;(\theta\,\mathsf{x}) \;=\; \rho\;(\mathsf{F}\,\mathsf{h}\,\mathsf{x}))\;\mathsf{p}\;(\mathsf{f}\text{-}\beta_0\,\mathsf{x}) \;=\; \mathsf{g}\text{-}\beta_0\,\mathsf{x})\\
&\simeq \{\mathsf{transporting\ over\ equalities}\}
\end{aligned}$$

$$\begin{aligned}
&\ \ !\ (\mathsf{ap}\ (\lambda\ \mathsf{h}\ .\ \mathsf{h}\ (\theta\ \mathsf{x}))\ \mathsf{p}) \cdot \mathsf{f}\text{-}\beta_0\ \mathsf{x} \cdot \mathsf{ap}\ (\lambda\ \mathsf{h}\ .\ \rho\ (\mathsf{F}\ \mathsf{h}\ \mathsf{x}))\ \mathsf{p}\ =\ \mathsf{g}\text{-}\beta_0\ \mathsf{x}\\
\simeq&\ \{\mathsf{path\ algebra}\}\\
&\ \ \mathsf{f}\text{-}\beta_0\ \mathsf{x} \cdot \mathsf{ap}\ (\lambda\ \mathsf{h}\ .\ \rho\ (\mathsf{F}\ \mathsf{h}\ \mathsf{x}))\ \mathsf{p}\ =\ \mathsf{ap}\ (\lambda\ \mathsf{h}\ .\ \mathsf{h}\ (\theta\ \mathsf{x}))\ \mathsf{p} \cdot \mathsf{g}\text{-}\beta_0\ \mathsf{x}
\end{aligned}$$

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

$$\begin{tikzcd}
\mathsf{f}\ (\theta\ \mathsf{x}) \arrow[r, "\mathsf{f}\text{-}\beta_0\ \mathsf{x}"] \arrow[d, "\mathsf{ap}\ (\lambda\ \mathsf{h}\ .\ \mathsf{h}\ (\theta\ \mathsf{x}))\ \mathsf{p}"'] & \rho\ (\mathsf{F}\ \mathsf{f}\ \mathsf{x}) \arrow[d, "\mathsf{ap}\ (\lambda\ \mathsf{h}\ .\ \rho\ (\mathsf{F}\ \mathsf{h}\ \mathsf{x}))\ \mathsf{p}"] \\
\mathsf{g}\ (\theta\ \mathsf{x}) \arrow[r, "\mathsf{g}\text{-}\beta_0\ \mathsf{x}"] & \rho\ (\mathsf{F}\ \mathsf{g}\ \mathsf{x})
\end{tikzcd}$$

4.2 Algebras for restricted 1-HIT descriptions

For algebras of restricted 1-HITs , the structure starts out similar to the above with an added function for the 1-constructor.

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

Recall that θ_0 is an F_0-algebra and θ_1 is a dependent dialgebra by defining the target functor G:

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

action on morphisms of G

Its hom-types can be defined as follows:

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

4.2.1 Equality of algebra morphisms

As we saw previously, we also need to consider what equalities of algebra morphisms look like.

equality of algebra morphisms

$$\begin{tikzcd}
\mathsf{f}\ (\theta_0\ \mathsf{x}) \arrow[r, "\mathsf{f}\text{-}\beta_0\ \mathsf{x}"] \arrow[d, "\mathsf{ap}\ (\lambda\ \mathsf{h}\ .\ \mathsf{h}\ (\theta_0\ \mathsf{x}))\ \mathsf{p}"'] & \rho_0\ (\mathsf{F}_0\ \mathsf{f}\ \mathsf{x}) \arrow[d, "\mathsf{ap}\ (\lambda\ \mathsf{h}\ .\ \rho_0\ (\mathsf{F}\ \mathsf{h}\ \mathsf{x}))\ \mathsf{p}"] \\
\mathsf{g}\ (\theta_0\ \mathsf{x}) \arrow[r, "\mathsf{g}\text{-}\beta_0\ \mathsf{x}"] & \rho_0\ (\mathsf{F}_0\ \mathsf{g}\ \mathsf{x})
\end{tikzcd}$$

along with

$$\begin{tikzcd}
\mathsf{G}\ \mathsf{x}\ \mathsf{f}, \mathsf{f}_0\ (\theta_1\ \mathsf{x}) \arrow[r, "\mathsf{f}\text{-}\beta_1\ \mathsf{x}"] \arrow[d, "\mathsf{ap}\ (\lambda\ (\mathsf{h}, \mathsf{h}_0)\ .\ \mathsf{G}\ \mathsf{x}\ (\mathsf{h}, \mathsf{h}_0)\ (\theta_1\ \mathsf{x})\ (\mathsf{p}, \mathsf{p}_0)"'] & \rho_1\ (\mathsf{F}_1\ \mathsf{f}\ \mathsf{x}) \arrow[d, "\mathsf{ap}\ (\lambda\ (\mathsf{h}, \mathsf{h}_0)\ .\ \rho_1\ (\mathsf{F}\ \mathsf{h}\ \mathsf{x}))\ (\mathsf{p}, \mathsf{p}_0)"] \\
\mathsf{G}\ \mathsf{x}\ \mathsf{g}, \mathsf{g}_0\ (\theta_1\ \mathsf{x}) \arrow[r, "\mathsf{g}\text{-}\beta_1\ \mathsf{x}"] & \rho_1\ (\mathsf{F}_1\ \mathsf{g}\ \mathsf{x})
\end{tikzcd}$$

5 Induction

Now we know what the algebras for restricted 1-HITs look like, characterising restricted 1-HITs as an initial algebra tells us what the constructors look like and gives us a recursion principle with its computation rule. However, it does not immediately reveal to us what the induction principle looks like.

Let us first look at the induction principle for natural numbers:

```
Nat-ind :
      (P : Nat → Type)
      (m-zero : P zero)
      (m-succ : (n : Nat) → P n → P (succ n))
   → (n : Nat) → P n
```

In order to show that the predicate P holds for every n : Nat, we have to give a method for every constructor, which shows that P holds for that constructor, given that it holds for all its subterms.

Generalising this to an inductive type T, given by a container F : Type → Type and constructor c : F T → T, we get:

```
ind :
      (P : T → Type)
      (m : (x : F T) → □ F B x → B (c x))
   → (x : T) → P x
```

with computation rule:

```
ind-β :
      (P : T → Type)
```

```
     (m : (x : F T) → □ F B x → B (c x))
     (x : F T)
   → ind P m (c x) = m x (□-lift F B (ind P m) x)
```

The modality □ lifts the predicate P on T to a predicate on F T:

```
□ F P : F T → Type
□ F P (s, t) :≡ (p : Pos s) → B (t p)
```

where Pos is the position type of the container F. The type □ F B x can be interpreted as: the predicate P holds for all of the subterms of c x : T. □-lift lifts dependent functions:

```
□-lift : ((x : A) → B x) → (x : F A) → □ F B x
```

The induction principle gives us a dependent function (x : T) → P t for a family P : T → Type. Equivalently, we can ask for a section σ : B → T of a function p : T → B for some B : Type. For any algebra (B, θ), initiality gives us an algebra morphism $\sigma : (T, c) \to (B, \theta)$. We can ask ourselves what extra conditions we need to impose on the algebra (B, θ) in order for σ to be a section of p. If p turns out to be an algebra morphism $(B, \theta) \to (T, c)$, then the following diagram commutes:

$$\begin{tikzcd}
FT \arrow[r, "c"] \arrow[d, "F\sigma"'] & T \arrow[d, "\sigma"] \\
FB \arrow[r, "\theta"] \arrow[d, "Fp"'] & B \arrow[d, "p"] \\
FT \arrow[r, "c"] & T
\end{tikzcd}$$

$p \circ \sigma$ is an algebra morphism $(T, c) \to (T, c)$. Since id is also an algebra morphism $(T, c) \to (T, c)$, by uniqueness we can conclude that $p \circ \sigma = \mathrm{id}$, so σ is indeed a section.

By choosing $B :\equiv \Sigma\ T\ P$ and $p :\equiv \pi_0$, we can derive the induction principle mentioned above, observing that $F\ (\Sigma\ T\ P) \simeq \Sigma\ (FT)\ (\Box\ F\ P)$.

5.1 Induction principle for restricted 1-HITs

An induction principle for restricted 1-HITs can be derived using the method mentioned previously: the data for induction should be an algebra structure on $\Sigma\ T\ P$ such that π_0 is an algebra morphism $(\Sigma\ T\ P, \pi_0, G\ \pi_0) \to (T, c_0, c_1)$. We have to produce the methods:

$$\mathsf{m}_0 \;:\; \mathsf{F}_0\,(\Sigma\,\mathsf{T}\,\mathsf{P}) \;\to\; \Sigma\,\mathsf{T}\,\mathsf{P}$$
$$\mathsf{m}_1 \;:\; (\mathsf{x} \;:\; \mathsf{F}_1\,(\Sigma\,\mathsf{T}\,\mathsf{P})) \;\to\; \mathsf{G}\,(\Sigma\,\mathsf{T}\,\mathsf{P}, \mathsf{m}_0)\,\mathsf{x}$$

subject to the equations:

$$\mathsf{m}\text{-}\beta_0 \;:\; (\mathsf{x} \;:\; \mathsf{F}_0\,(\Sigma\,\mathsf{T}\,\mathsf{P})) \;\to\; \pi_0\,(\mathsf{m}_0\,\mathsf{x}) \;=\; \mathsf{c}_0\,(\mathsf{F}_0\,\pi_0\,\mathsf{x})$$
$$\mathsf{m}\text{-}\beta_1 \;:\; (\mathsf{x} \;:\; \mathsf{F}_1\,(\Sigma\,\mathsf{T}\,\mathsf{P})) \;\to\; \mathsf{G}\,(\pi_0, \mathsf{m}\text{-}\beta_0)\,(\mathsf{m}_1\,\mathsf{x}) \;=\; \mathsf{c}_1\,(\mathsf{F}_1\,\pi_0\,\mathsf{x})$$

From this we can make derive the methods for the induction principle:

$$\mathsf{m}_0 \;:\; (\mathsf{x} \;:\; \mathsf{F}_0\,\mathsf{T}) \;\to\; \Box\,\mathsf{F}_0\,\mathsf{P}\,\mathsf{x} \;\to\; \mathsf{P}\,(\mathsf{c}_0\,\mathsf{x})$$
$$\mathsf{m}_1 \;:\; (\mathsf{x} \;:\; \mathsf{F}_1\,\mathsf{T})\,(\mathsf{y} \;:\; \Box\,\mathsf{F}_1\,\mathsf{P}\,\mathsf{x}) \;\to\; \mathsf{m}_0 *^d (\mathsf{l}^d\,\mathsf{x}\,\mathsf{y}) \;=\; \mathsf{m}_0 *^d (\mathsf{r}^d\,\mathsf{x}\,\mathsf{y})\,[\mathsf{P} \downarrow \mathsf{c}_1\,\mathsf{x}]$$

where:

$$\begin{aligned}\mathsf{l}^d, \mathsf{r}^d \;:\; & \{\mathsf{A} \;:\; \mathsf{Type0}\}\,\{\mathsf{B} \;:\; \mathsf{A} \;\to\; \mathsf{Type0}\} \\ & \to\; (\mathsf{x} \;:\; \mathsf{F}_1\,\mathsf{A}) \;\to\; \Box\,\mathsf{F}_1\,\mathsf{B}\,\mathsf{x} \;\to\; \Sigma\,(\mathsf{F}_0\,*)\,(\Box\,(\mathsf{F}_0\,*)\,\mathsf{B})\end{aligned}$$

and

$$\mathsf{m}_0 *^d \;:\; (\mathsf{x} \;:\; \Sigma\,(\mathsf{F}_0\,*)) \times \Box\,(\mathsf{F}_0\,*)\,\mathsf{P}\,\mathsf{x} \;\to\; \mathsf{P}\,(\mathsf{c}_0\,(\pi_0\,\mathsf{x}))$$

are the liftings of their non-dependent counterparts.

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:

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

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 $(\mathsf{g}, \mathsf{g}_0) : (\mathsf{T}, \mathsf{c}) \rightarrow (\mathsf{X}, \theta)$, 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 $\lambda\, \mathsf{x}\, .\, \mathsf{f\,x} = \mathsf{g\,x}$. The induction step is then proven as follows:

```
step : (x : ⟦ F ⟧₀ T) → □ F (λ x . f x = g x) x → f=g-B (c x)
step x u :≡
  f (c x)
    ={ f₀ x }
  θ (⟦ F ⟧₁ f x)
    ={ ap θ (ind-hyp x u) }
  θ (⟦ F ⟧₁ g x)
    ={ ! (g₀ x) }
  g (c x) ■
```

where ind-hyp : (x : F T) → □ F (λ x . f x = g x) x → F f x = F g x. We can define the witness of $\mathsf{f} = \mathsf{g}$ as follows:

```
p : (x : T) → f x = g x
p :≡ ind (λ x . f x = g x) step
```

which comes with the computation rule:

```
p-β₀ : (x : FT) → p (c x) = f-β₀ x ▪ ap θ (ind-hyp x (□-lift F p x)) ▪ ! (g-β₀ x)
```

We now need to show that our witness p satisfies the "computation rule" (x : FT) → f-β_0 x ▪ ap (λ h . ρ (F h x)) p = ap (λ h . h (θ x)) p ▪ g-β_0 x. Let x : FT, then we can calculate:

```
f-β₀ x ▪ ap (λ h . θ (F h x)) (fun-ext p)
   = {ap magic}
f-β₀ x ▪ ap θ (ind-hyp x (□-lift F p x))
   = {symmetry is involutive}
f-β₀ x ▪ ap θ (ind-hyp x (□-lift F p x)) ▪ ! (g-β₀ x) ▪ g-β₀ x
   = {computation rule for p}
p (c x) ▪ g-β₀ x
   = {computation rule for fun-ext}
ap (λ h . h (c x)) (fun-ext p) ▪ g-β₀ x
```

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

6.2 Restricted 1-HITs

6.2.1 Existence

6.2.2 Uniqueness