Computer-generated proofs for the monoidal structure of the smash product

Guillaume Brunerie

November 8, 2018
HoTTEST
The smash product as a higher inductive type

Definition
Given two pointed types \((A, \star_A)\) and \((B, \star_B)\), their smash product \(A \land B\) is defined as the higher inductive type with constructors:

\[
\begin{align*}
\text{proj} : & A \times B \to A \land B, \\
\text{basel} : & A \land B, \\
\text{baser} : & A \land B, \\
\text{pushl} : & (a : A) \to \text{proj}(a, \star_B) = \text{basel}, \\
\text{pushr} : & (b : B) \to \text{proj}(\star_A, b) = \text{baser}.
\end{align*}
\]
Goal
We want to prove (in book HoTT) that the smash product is a 1-coherent symmetric monoidal product on pointed types.\(^1\)

This means that:

- The smash product is functorial (on pointed maps).
- There is a natural involution \(\sigma_{A,B} : A \land B \to B \land A\).
- There is a natural equivalence \(
\alpha_{A,B,C} : (A \land B) \land C \to A \land (B \land C)\).
- It satisfies the hexagon and pentagon coherences.
- It has a unit with a triangular coherence.

This is used in particular to prove that the cup product on cohomology is associative.

\(^1\)see pages 88 and 89 of my PhD thesis
Basic idea

All we have to do is to define various functions:

\[(x : A \land B) \rightarrow P(x) \quad (6 \text{ of them})\]
\[(x : (A \land B) \land C) \rightarrow P(x) \quad (4 \text{ of them})\]
\[(x : A \land (B \land C)) \rightarrow P(x) \quad (2 \text{ of them})\]
\[(x : ((A \land B) \land C) \land D) \rightarrow P(x) \quad (1 \text{ of them})\]

where \(P(x)\) is either constant or an equality \(f(x) = g(x)\).

We define them by (iterated) induction on the smash product.

- In the (iterated) proj case, we know what to do.
- In the other cases, we “just” need to do some complicated path algebra.
Recursion rule

Given a type \( C \), in order to define a map \( f : A \land B \rightarrow C \), we need to define five terms/functions \( f_{\text{proj}}, f_{\text{basel}}, f_{\text{baser}}, f_{\text{pushl}} \) and \( f_{\text{pushr}} \) such that:

\[
\begin{align*}
  f : A \land B & \rightarrow C \\
  f(\text{proj}(a, b)) & := f_{\text{proj}}(a, b) \quad (f_{\text{proj}} : A \times B \rightarrow C) \\
  f(\text{basel}) & := f_{\text{basel}} \quad (f_{\text{basel}} : C) \\
  f(\text{baser}) & := f_{\text{baser}} \quad (f_{\text{baser}} : C) \\
  \text{ap}_f(\text{pushl}(a)) & := f_{\text{pushl}}(a) \quad (f_{\text{pushl}} : (a : A) \rightarrow f_{\text{proj}}(a, \star B) = C \ f_{\text{basel}}) \\
  \text{ap}_f(\text{pushr}(b)) & := f_{\text{pushr}}(b) \quad (f_{\text{pushr}} : (b : B) \rightarrow f_{\text{proj}}(\star A, b) = C \ f_{\text{baser}})
\end{align*}
\]
Example 1: commutativity

\[ \sigma_{A,B} : A \land B \rightarrow B \land A \]

\[ \sigma_{A,B}(\text{proj}(a, b)) := \text{proj}(b, a) \]

\[ \sigma_{A,B}(\text{basel}) := \square^0 \]

\[ \sigma_{A,B}(\text{baser}) := \square^0 \]

\[ \text{ap}_{\sigma_{A,B}}(\text{pushl}(a)) := \square^1 \]

\[ \text{ap}_{\sigma_{A,B}}(\text{pushr}(b)) := \square^1 \]
Example 1: commutativity

\[ \sigma_{A,B} : A \land B \rightarrow B \land A \]

\[ \sigma_{A,B}(\text{proj}(a, b)) := \text{proj}(b, a) \]

\[ \sigma_{A,B}(\text{basel}) := \text{baser} \]

\[ \sigma_{A,B}(\text{baser}) := \text{basel} \]

\[ \text{ap}_{\sigma_{A,B}}(\text{pushl}(a)) := \text{pushr}(a) \]

\[ \text{ap}_{\sigma_{A,B}}(\text{pushr}(b)) := \text{pushl}(b) \]
Definition
Given two pointed types \((A, \star_A)\) and \((A', \star_{A'})\), a pointed map from \(A\) to \(A'\) is a pair \((f, \star_f)\) where

\[
f : A \to A' \\
\star_f : f(\star_A) = \star_{A'}
\]
Example 2: functoriality

We have two pointed maps $f : A \rightarrow A'$ and $g : B \rightarrow B'$.

$$(f \land g) : A \land B \rightarrow A' \land B'$$

$$(f \land g)(\text{proj}(a, b)) := \text{proj}(f(a), g(b))$$

$$(f \land g)(\text{basel}) := \text{basel}$$

$$(f \land g)(\text{baser}) := \text{baser}$$

$$\text{ap}_{f \land g}(\text{pushl}(a)) := 1$$

$$\text{ap}_{f \land g}(\text{pushr}(b)) := 1$$
Example 2: functoriality

We have two pointed maps \( f : A \to A' \) and \( g : B \to B' \).

\[
(f \land g) : A \land B \to A' \land B'
\]

\[
(f \land g)(\text{proj}(a, b)) := \text{proj}(f(a), g(b))
\]

\[
(f \land g)(\text{basel}) := \text{basel}
\]

\[
(f \land g)(\text{baser}) := \text{baser}
\]

\[
\text{ap}_{f \land g}(\text{pushl}(a)) := 1
\]

\[
\text{ap}_{f \land g}(\text{pushr}(b)) := 1
\]

The two holes have type

\[
\text{proj}(f(a), g(\ast_B)) = \text{basel} \quad \text{proj}(f(\ast_A), g(b)) = \text{baser}
\]
... with rewriting

We have

\[ \text{proj}(f(a), g(\star_B)) \]
\[ \leadsto \text{proj}(f(a), \star_{B'}) \text{ via } \star_g \text{ in the second argument of proj} \]
\[ \leadsto \text{basel} \text{ via } \text{pushl}(f(a)) \]

Therefore we can fill the first hole with

\[ \text{ap}_{\text{proj}(f(a),-)}(\star_g) \cdot \text{pushl}(f(a)) \]
... with rewriting

We have

\[ \text{proj}(f(a), g(\ast_B)) \]
\[ \leadsto \text{proj}(f(a), \ast_{B'}) \quad \text{via} \quad \ast_g \quad \text{in the second argument of proj} \]
\[ \leadsto \text{basel} \quad \text{via} \quad \text{pushl}(f(a)) \]

Therefore we can fill the first hole with

\[ \text{ap}_{\text{proj}(f(a), -)}(\ast_g) \cdot \text{pushl}(f(a)) \]

Similarly for the second hole:

\[ \text{proj}(f(\ast_A), g(b)) \]
\[ \leadsto \text{proj}(\ast_{A'}, g(b)) \quad \text{via} \quad \ast_f \quad \text{in the first argument of proj} \]
\[ \leadsto \text{baser} \quad \text{via} \quad \text{pushr}(g(b)) \]
Proof-relevant rewriting

\[ f(\star A) \rightsquigarrow \star A' \quad \text{via} \quad \star f \]

\[ \text{proj}(a, \star B) \rightsquigarrow \text{basel} \quad \text{via} \quad \text{pushl}(a) \]

\[ \text{proj}(\star A, b) \rightsquigarrow \text{baser} \quad \text{via} \quad \text{pushr}(b) \]

\[ \text{basel} \rightsquigarrow \text{proj}(\star A, \star B) \quad \text{via} \quad \text{pushl}(\star A) \]

\[ \text{baser} \rightsquigarrow \text{proj}(\star A, \star B) \quad \text{via} \quad \text{pushr}(\star B) \]

\[ \text{proj}(\star A, \star B) \not\rightsquigarrow \]

\[ f(u) \rightsquigarrow f(u') \quad \text{via} \quad \text{ap}_f(p) \]

(\text{if } u \rightsquigarrow u' \text{ via } p)

\[ u \rightsquigarrow u'' \quad \text{via} \quad p \cdot p' \]

(\text{if } u \rightsquigarrow u' \text{ via } p \text{ and } u' \rightsquigarrow u'' \text{ via } p')
We use squares and cubes in the sense of [LB15]\(^2\).

**Definition**

The type

\[
\text{Square} : \{A : \text{Type}\}\{a, b, c, d : A\}
\]

\[(p : a = b)(q : c = d)(r : a = c)(s : b = d) \to \text{Type}\]

is defined as the inductive family with one constructor

\[
\text{ids} : \text{Square}(\text{idp}, \text{idp}, \text{idp}, \text{idp})
\]
Application of a homotopy to a path

Given a dependent function (where \( g, h : A \rightarrow B \))

\[
f : (x : A) \rightarrow g(x) =_B h(x)
\]

and a path

\[
p : a =_A a'
\]

we have

\[
ap_f^+(p) : \text{Square}(ap_g(p), ap_h(p), f(a), f(a'))
\]
Induction rule (into an identity type)

Given a type $C$ and two functions $g, h : A \land B \rightarrow C$, in order to define a map

$$f : (x : A \land B) \rightarrow g(x) =_C h(x),$$

we need

$$f(\text{proj}(a, b)) : g(\text{proj}(a, b)) =_C h(\text{proj}(a, b))$$
$$f(\text{basel}) : g(\text{basel}) =_C h(\text{basel})$$
$$f(\text{baser}) : g(\text{baser}) =_C h(\text{baser})$$
$$\text{ap}_f^+ (\text{pushl}(a)) : \text{Square}(\text{ap}_g(\text{pushl}(a)), \text{ap}_h(\text{pushl}(a))),$$
$$\quad f(\text{proj}(a, \star_B)), f(\text{basel}))$$
$$\text{ap}_f^+ (\text{pushr}(b)) : \text{Square}(\text{ap}_g(\text{pushr}(b)), \text{ap}_h(\text{pushr}(b))),$$
$$\quad f(\text{proj}(\star_A, b)), f(\text{baser}))$$
Example 2: naturality of commutativity

We have two pointed maps \( f : A \to A' \) and \( g : B \to B' \).

\[
\sigma\text{-nat}_{f,g} : (x : A \land B) \to \sigma_{A',B'}((f \land g)(x)) = (g \land f)(\sigma_{A,B}(x))
\]

\[
\sigma\text{-nat}_{f,g}(\text{proj}(a, b)) := \text{idp}_{\text{proj}(g(b), f(a))}
\]

\[
\sigma\text{-nat}_{f,g}(\text{basel}) := \text{idp}_{\text{basel}}
\]

\[
\text{ap}^+_{\sigma\text{-nat}_{f,g}}(\text{pushl}(a)) := \Box^2 : \text{Square}(\text{ap}_{\sigma_{A',B'} \circ (f \land g)}(\text{pushl}(a)),
\]

\[
\text{ap}(g \land f) \circ \sigma_{A,B}(\text{pushl}(a)),
\]

\[
\text{idp}_{\text{proj}(g(\ast_B), f(a))},
\]

\[
\text{idp}_{\text{basel}})
\]

\[
\text{ap}^+_{\sigma\text{-nat}_{f,g}}(\text{pushr}(b)) := \Box^2 : [\ldots]
\]
More rewriting!

\[ \text{ap}_{\sigma_{A',B'} \circ (f \land g)}(\text{pushl}(a)) \]
More rewriting!

\[ \text{ap}_{\sigma_{A',B'}}(f \land g)(\text{pushl}(a)) \\sim \rightarrow \text{ap}_{\sigma_{A',B'}}(\text{ap}_{f \land g}(\text{pushl}(a))) \]
More rewriting!

\[ \text{ap}_{\sigma_{A'}, B'} (f \land g)(\text{pushl}(a)) \]

\[ \rightsquigarrow \text{ap}_{\sigma_{A'}, B'} (\text{ap}_{f \land g}(\text{pushl}(a))) \]

\[ \rightsquigarrow \text{ap}_{\sigma_{A'}, B'} (\text{ap}_{\text{proj}(f(a), -)}(\star g) \cdot \text{pushl}(f(a))) \]
More rewriting!

$$\text{ap}_{\sigma_{A'},B'}(f \land g)(\text{pushl}(a))$$

$$\leadsto \text{ap}_{\sigma_{A'},B'}(\text{ap}_{f \land g}(\text{pushl}(a)))$$

$$\leadsto \text{ap}_{\sigma_{A'},B'}(\text{ap}_{\text{proj}(f(a), -)}(\star g) \cdot \text{pushl}(f(a)))$$

$$\leadsto \text{ap}_{\sigma_{A'},B'}(\text{ap}_{\text{proj}(f(a), -)}(\star g) \cdot \text{ap}_{\sigma_{A'},B'}(\text{pushl}(f(a))))$$
More rewriting!

\[ \text{ap}_{\sigma_{A'},B'}(f \& g)(\text{pushl}(a)) \]
\[ \rightsquigarrow \text{ap}_{\sigma_{A'},B'}(\text{ap}_{f \& g}(\text{pushl}(a))) \]
\[ \rightsquigarrow \text{ap}_{\sigma_{A'},B'}(\text{ap}_{\text{proj}}(f(a),-)(\star g) \cdot \text{pushl}(f(a))) \]
\[ \rightsquigarrow \text{ap}_{\sigma_{A'},B'}(\text{ap}_{\text{proj}}(f(a),-)(\star g) \cdot \text{ap}_{\sigma_{A'},B'}(\text{pushl}(f(a)))) \]
\[ \rightsquigarrow \text{ap}_{\text{proj}}(-,f(a))(\star g) \cdot \text{pushr}(f(a)) \]
More rewriting!

\[
ap_{\sigma_{A',B'} \circ (f \wedge g)}(\text{pushl}(a))
\] \[\leadsto ap_{\sigma_{A',B'}}(ap_{f \wedge g}(\text{pushl}(a)))
\] \[\leadsto ap_{\sigma_{A',B'}}(ap_{\text{proj}(f(a),-)}(g) \cdot \text{pushl}(f(a)))
\] \[\leadsto ap_{\sigma_{A',B'}}(ap_{\text{proj}(f(a),-)}(g)) \cdot ap_{\sigma_{A',B'}}(\text{pushl}(f(a)))
\] \[\leadsto ap_{\text{proj}(-,f(a))}(g) \cdot \text{pushr}(f(a))
\]

\[
ap_{(g \wedge f) \circ \sigma_{A,B}}(\text{pushl}(a))
\] \[\leadsto ap_{g \wedge f}(ap_{\sigma_{A,B}}(\text{pushl}(a)))
\] \[\leadsto ap_{g \wedge f}(\text{pushr}(a))
\] \[\leadsto ap_{\text{proj}(-,f(a))}(g) \cdot \text{pushr}(f(a))
\]
More rewriting rules

\[\text{ap}_{\sigma_{A,B}}(\text{pushl}(a)) \rightsquigarrow \text{pushr}(a)\]  
and other \(\beta\)-reduction rules for HITs

\[\text{ap}_{\lambda x.x}(p) \rightsquigarrow p\]

\[\text{ap}_g(\text{ap}_f(p)) \rightsquigarrow \text{ap}_{g \circ f}(p)\]

\[\text{ap}_{g \circ f}(p) \rightsquigarrow \text{ap}_g(p') \quad \text{(if } \text{ap}_f(p) \rightsquigarrow p'\text{)}\]

\[\text{ap}_f(u \cdot v) \rightsquigarrow \text{ap}_f(u) \cdot \text{ap}_f(v)\]

\[u \cdot v \rightsquigarrow u' \cdot v' \quad \text{(if } u \rightsquigarrow u' \text{ and } v \rightsquigarrow v', \text{via horizontal composition)}\]
Example 4: associativity

\[ \alpha_{A,B,C} : (A \land B) \land C \rightarrow A \land (B \land C), \]

\[ \alpha_{A,B,C}(\text{proj}(x, c)) := \alpha_{A,B,C}^{\text{proj}}(x, c), \]

\[ \alpha_{A,B,C}(\text{basel}) := \blacksquare^0, \]

\[ \alpha_{A,B,C}(\text{baser}) := \blacksquare^0, \]

\[ \alpha_{A,B,C}(\text{pushl}(x)) := \alpha_{A,B,C}^{\text{pushl}}(x), \]

\[ \alpha_{A,B,C}(\text{pushr}(c)) := \blacksquare^1. \]
Example 4: associativity

\[ \alpha_{A,B,C} : (A \land B) \land C \rightarrow A \land (B \land C), \]

\[ \alpha_{A,B,C}(\text{proj}(x, c)) := \alpha_{A,B,C}^{\text{proj}}(x, c), \]

\[ \alpha_{A,B,C}(\text{basel}) := \square^0, \]

\[ \alpha_{A,B,C}(\text{baser}) := \square^0, \]

\[ \alpha_{A,B,C}(\text{pushl}(x)) := \alpha_{A,B,C}^{\text{pushl}}(x), \]

\[ \alpha_{A,B,C}(\text{pushr}(c)) := \square^1. \]

\[ \alpha_{A,B,C}^{\text{proj}} : A \land B \rightarrow C \rightarrow A \land (B \land C), \]

\[ \alpha_{A,B,C}^{\text{proj}}(\text{proj}(a, b), c) := \text{proj}(a, \text{proj}(b, c)), \]

\[ [\square^0 \ldots \square^0 \ldots \square^1 \ldots \square^1] \]
Example 4: associativity

\[ \alpha_{A,B,C} : (A \land B) \land C \rightarrow A \land (B \land C), \]

\[ \alpha_{A,B,C}(\text{proj}(x, c)) := \alpha_{A,B,C}^{\text{proj}}(x, c), \]

\[ \alpha_{A,B,C}(\text{basel}) := \mathbf{0}, \]

\[ \alpha_{A,B,C}(\text{baser}) := \mathbf{0}, \]

\[ \alpha_{A,B,C}(\text{pushl}(x)) := \alpha_{A,B,C}^{\text{pushl}}(x), \]

\[ \alpha_{A,B,C}(\text{pushr}(c)) := \mathbf{1}. \]

\[ \alpha_{A,B,C}^{\text{proj}} : A \land B \rightarrow C \rightarrow A \land (B \land C), \]

\[ \alpha_{A,B,C}^{\text{proj}}(\text{proj}(a, b), c) := \text{proj}(a, \text{proj}(b, c)), \]

\[ [\mathbf{0} \ldots \mathbf{0} \ldots \mathbf{1} \ldots \mathbf{1}] \]

\[ \alpha_{A,B,C}^{\text{pushl}} : (x : A \land B) \rightarrow \alpha_{A,B,C}^{\text{proj}}(x, \ast c) = \alpha_{A,B,C}(\text{basel}), \]

\[ [\mathbf{1} \ldots \mathbf{1} \ldots \mathbf{1} \ldots \mathbf{2} \ldots \mathbf{2}] \]
Hexagon

\[(A \land B) \land C \xrightarrow{\alpha_{A,B,C}} A \land (B \land C)\]

\[(B \land A) \land C \xrightarrow{\sigma_{A,B} \land \text{id}_C} (B \land C) \land A\]

\[(B \land (A \land C)) \xrightarrow{\alpha_{B,A,C}} B \land (C \land A)\]

\[B \land (A \land C) \xrightarrow{\text{id}_{B \land \sigma_{A,C}}} B \land (C \land A)\]
Example 5: hexagon

\[ \text{hexagon}_{A,B,C} : (x : (A \land B) \land C) \to \text{Id}(\ldots, \ldots) \]

\[ [\text{hexagon}_{A,B,C}^{\text{proj}} \ldots \blacksquare^1 \ldots \blacksquare^1 \ldots \text{hexagon}_{A,B,C}^{\text{pushl}} \ldots \blacksquare^2] \]
Example 5: hexagon

\[
\text{hexagon}_{A,B,C} : (x : (A \land B) \land C) \rightarrow \text{Id}(\ldots, \ldots)
\]

\[
[\text{hexagon}^{\text{proj}}_{A,B,C} \ldots \blacksquare^1 \ldots \blacksquare^1 \ldots \text{hexagon}^{\text{pushl}}_{A,B,C} \ldots \blacksquare^2]
\]

\[
\text{hexagon}^{\text{proj}}_{A,B,C} : (x : A \land B) \rightarrow C \rightarrow \text{Id}(\ldots, \ldots)
\]

\[
[\text{idp} \ldots \blacksquare^1 \ldots \blacksquare^1 \ldots \blacksquare^2 \ldots \blacksquare^2]
\]
Example 5: hexagon

\[ \text{hexagon}_{A,B,C} : (x : (A \land B) \land C) \rightarrow \text{Id}(\ldots, \ldots) \]

\[ \text{[hexagon}^{\text{proj}}_{A,B,C} \ldots \blacksquare^1 \ldots \blacksquare^1 \ldots \text{hexagon}^{\text{pushl}}_{A,B,C} \ldots \blacksquare^2 \] \]

\[ \text{hexagon}^{\text{proj}}_{A,B,C} : (x : A \land B) \rightarrow C \rightarrow \text{Id}(\ldots, \ldots) \]

\[ \text{[idp} \ldots \blacksquare^1 \ldots \blacksquare^1 \ldots \blacksquare^2 \ldots \blacksquare^2 \] \]

\[ \text{hexagon}^{\text{pushl}}_{A,B,C} : (x : A \land B) \rightarrow C \rightarrow \text{Square}(\ldots, \ldots, \ldots, \ldots) \]

\[ \text{[}\blacksquare^2 \ldots \blacksquare^2 \ldots \blacksquare^2 \ldots \blacktriangle^3 \ldots \blacktriangle^3] \]
Example 6: pentagon

\[ \text{pent} : (x : ((A \land B) \land C) \land D) \rightarrow \text{Id}(\ldots, \ldots) \]
Example 6: pentagon

\[
pent : (x : ((A \land B) \land C) \land D) \rightarrow \text{Id}(\ldots, \ldots)
\]

\[
pent : [pent^{\text{proj}}\ldots\blacksquare^1\ldots\blacksquare^1\ldots pent^{\text{pushl}}\ldots\blacksquare^2]
\]

\[
pent^{\text{proj}} : [pent^{\text{proj,proj}}\ldots\blacksquare^1\ldots\blacksquare^1\ldots pent^{\text{proj,pushl}}\ldots\blacksquare^2]
\]

\[
pent^{\text{proj,proj}} : [\text{idp}\ldots\blacksquare^1\ldots\blacksquare^1\ldots\blacksquare^2\ldots\blacksquare^2]
\]

\[
pent^{\text{proj,pushl}} : [\blacksquare^2\ldots\blacksquare^2\ldots\blacksquare^2\ldots\blacksquare^3\ldots\blacksquare^3]
\]

\[
pent^{\text{pushl}} : [pent^{\text{pushl,proj}}\ldots\blacksquare^2\ldots\blacksquare^2\ldots pent^{\text{pushl,pushl}}\ldots\blacksquare^3]
\]

\[
pent^{\text{pushl,proj}} : [\blacksquare^2\ldots\blacksquare^2\ldots\blacksquare^2\ldots\blacksquare^3\ldots\blacksquare^3]
\]

\[
pent^{\text{pushl,pushl}} : [\blacksquare^3\ldots\blacksquare^3\ldots\blacksquare^3\ldots\blacksquare^4\ldots\blacksquare^4]
\]
Cubical proof-relevant rewriting

Definition
Given \( f : A \rightarrow B \) and \( sq : \text{Square}_A(p, q, r, s) \), we have

\[
\text{ap}_f^2(sq) : \text{Square}_B(\text{ap}_f(p), \text{ap}_f(q), \text{ap}_f(r), \text{ap}_f(s)).
\]
Cubical proof-relevant rewriting

Definition
Given $f : A \to B$ and $sq : \text{Square}_A(p, q, r, s)$, we have

$$\text{ap}^2_f(sq) : \text{Square}_B(\text{ap}_f(p), \text{ap}_f(q), \text{ap}_f(r), \text{ap}_f(s)).$$

We want a reduction rule

$$\text{ap}^2_{\lambda x.x}(sq) \rightsquigarrow sq$$
Cubical proof-relevant rewriting

Definition
Given \( f : A \to B \) and \( sq : \text{Square}_A(p, q, r, s) \), we have

\[
ap^2_f(sq) : \text{Square}_B(ap_f(p), ap_f(q), ap_f(r), ap_f(s)).
\]

We want a reduction rule

\[
ap^2_{\lambda x. x}(sq) \rightsquigarrow sq
\]

... but the two sides don’t have the same type (!).
Cubical proof-relevant rewriting

Definition
Given \( f : A \rightarrow B \) and \( sq : \text{Square}_A(p, q, r, s) \), we have

\[
ap^2_f(sq) : \text{Square}_B(\text{ap}_f(p), \text{ap}_f(q), \text{ap}_f(r), \text{ap}_f(s)).
\]

We want a reduction rule

\[
ap^2_{\lambda x.x}(sq) \rightsquigarrow sq
\]

\[
\ldots \text{ but the two sides don’t have the same type (!).}
\]

Cubical proof-relevant rewriting:
If \( s \) and \( s' \) are two squares, we say

\[
s \rightsquigarrow s' \text{ via } c
\]

if \( c \) is a cube with \( s \) and \( s' \) as two of its opposite faces.
More variants of ap

<table>
<thead>
<tr>
<th>( f : A \to B )</th>
<th>( p : \text{Id}_A )</th>
<th>( p : \text{Square}_A )</th>
<th>( p : \text{Cube}_A )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f : A \to \text{Id}_B )</td>
<td>( \text{ap}_f(p) )</td>
<td>( \text{ap}_f^2(p) )</td>
<td></td>
</tr>
<tr>
<td>( f : A \to \text{Square}_B )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( f : A \to \text{Cube}_B )</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
More variants of ap

<table>
<thead>
<tr>
<th>( f : A \to B )</th>
<th>( p : \text{Id}_A )</th>
<th>( p : \text{Square}_A )</th>
<th>( p : \text{Cube}_A )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f : A \to \text{Id}_B )</td>
<td>( \text{ap}_f(p) )</td>
<td>( \text{ap}^2_f(p) )</td>
<td>( \text{ap}^3_f(p) )</td>
</tr>
<tr>
<td>( f : A \to \text{Square}_B )</td>
<td>( \text{ap}^+_f(p) )</td>
<td>( \text{ap}^{2,+}_f(p) )</td>
<td>( \text{ap}^{3,+}_f(p) )</td>
</tr>
<tr>
<td>( f : A \to \text{Cube}_B )</td>
<td>( \text{ap}^{++}_f(p) )</td>
<td>( \text{ap}^{2,++}_f(p) )</td>
<td></td>
</tr>
<tr>
<td>( f : A \to \text{Cube}_B )</td>
<td>( \text{ap}^{+++}_f(p) )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
More variants of $\text{ap}$

<table>
<thead>
<tr>
<th>$f : A \to B$</th>
<th>$p : \text{Id}_A$</th>
<th>$p : \text{Square}_A$</th>
<th>$p : \text{Cube}_A$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f : A \to \text{Id}_B$</td>
<td>$\text{ap}_f(p)$</td>
<td>$\text{ap}_f^2(p)$</td>
<td>$\text{ap}_f^3(p)$</td>
</tr>
<tr>
<td>$f : A \to \text{Square}_B$</td>
<td>$\text{ap}_f^{++}(p)$</td>
<td>$\text{ap}_f^{2,++}(p)$</td>
<td></td>
</tr>
<tr>
<td>$f : A \to \text{Cube}_B$</td>
<td>$\text{ap}_f^{+++}(p)$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

They interact in various ways, for instance

\[
\begin{align*}
\text{ap}_g^2(\text{ap}_f^+(p)) & \rightsquigarrow \text{ap}_{\text{ap}_g \circ f}^+(p) \\
\text{ap}_g^+(\text{ap}_f(p)) & \rightsquigarrow \text{ap}_{\text{ap}_g \circ f}^+(p) \\
\text{ap}_f^+(p \cdot q) & \rightsquigarrow \text{ap}_f^+(p) \Diamond \text{ap}_f^+(q) \\
\text{ap}_{\lambda x. p(x) \cdot q(x)}^+(r) & \rightsquigarrow \text{ap}_{\lambda x. p(x)}^+(r) \Diamond \text{ap}_{\lambda x. q(x)}^+(r)
\end{align*}
\]
Globular coherences

We can construct any map of the form:

\[
\text{coh} : (X : \text{Type})(a : X)
\]

\[
[\ldots]
\]

\[
(x_n : T_n)(p_n : x_n = u_n)
\]

\[
[\ldots]
\]

\[
\rightarrow T
\]

where \(T_n, u_n\) and \(T\) are built only from previous variables and other coherences, and \(T\) is an identity type.

**Idea:** path induction on all of the \(p_n\), then give \text{idp}.

**Use:** \(p_n\) represents a rewriting rule, and \(x_n\) the term being rewritten.
We also need to allow pairs of arguments of the form

\[(x_n : T_n)(p_n : \text{Square}(x_n, u_n, v_n, w_n))\]

\[(x_n : T_n)(p_n : \text{Cube}(x_n, u_n, v_n, w_n, r_n, s_n))\]

We can still construct all such coherences, using a generalized version of J where three sides of a square are fixed and one side is free.
Algorithm for building the proof

In order to fill a hole ($\blacksquare^1$, $\blacksquare^2$, $\blacksquare^3$ or $\blacksquare^4$) we proceed as follows. The variables are $\ell_1$ a list of terms and $\ell_2$ a list of pairs of terms.

- We start with $\ell_1$ consisting of all the faces (in every dimension) of the hole, and $\ell_2$ is empty.
- Take the first element $t$ of $\ell_1$.
- If it is the base point, or is already present in $\ell_2$, discard it.
- Otherwise, reduce it (it gives an $n$-cube $s$ which has $t$ as one of its faces), add $(t, s)$ to $\ell_2$ and all the other faces of $s$ to $\ell_1$.
- Repeat until $\ell_1$ is empty.
- Build a cubical coherence out of $\ell_2$.
- Use that coherence to fill the hole.
Example

We want to prove $\text{proj}(f(a), g(\star_B)) = \text{basel}$.

\[
\ell_1 = [\text{proj}(f(a), g(\star_B)), \text{basel}]
\]
\[
\ell_2 = []
\]

\[
\text{proj}(f(a), g(\star_B)) \leadsto \text{proj}(f(a), \star_{B'}) \text{ via } \text{ap}_{\text{proj}(f(a), -)}(\star_g)
\]

\[
\ell_1 = [\text{proj}(f(a), \star_{B'}), \text{basel}]
\]
\[
\ell_2 = [([\text{proj}(f(a), g(\star_B)), \text{ap}_{\text{proj}(f(a), -)}(\star_g)])]
\]
Example

\[ \ell_1 = [\text{proj}(f(a), \star_{B'}), \text{basel}] \]
\[ \ell_2 = [(\text{proj}(f(a), g(\star_B)), \text{ap}_{\text{proj}(f(a), \_)}(\star g))] \]

\[
\text{proj}(f(a), \star_{B'}) \rightsquigarrow \text{basel} \text{ via } \text{pushl}(f(a))
\]

\[ \ell_1 = [\text{basel}, \text{basel}] \]
\[ \ell_2 = [(\text{proj}(f(a), g(\star_B)), \text{ap}_{\text{proj}(f(a), \_)}(\star g)),
\text{proj}(f(a), \star_{B'}), \text{pushl}(f(a)))]\]
Example

\[ \ell_1 = [\text{basel}, \text{basel}] \]
\[ \ell_2 = [(\text{proj}(f(a), g(\star_B)), \text{ap}_{\text{proj}(f(a), -)}(\star g)), \]
\[ \quad (\text{proj}(f(a), \star_{B'}), \text{pushl}(f(a))))] \]

\[ \text{basel} \rightsquigarrow \text{proj}(\star_{A'}, \star_{B'}) \text{ via } \text{pushl}(\star_{A'}) \]

\[ \ell_1 = [\text{proj}(\star_{A'}, \star_{B'}), \text{basel}] \]
\[ \ell_2 = [(\text{proj}(f(a), g(\star_B)), \text{ap}_{\text{proj}(f(a), -)}(\star g)), \]
\[ \quad (\text{proj}(f(a), \star_{B'}), \text{pushl}(f(a))))] \]

\[ (\text{basel}, \text{pushl}(\star_{A'}))] \]

We’re done, as everything in \( \ell_1 \) is either in \( \ell_2 \) or the base point.
Example

\[ \ell_2 = \left[ (\text{proj}(f(a), g(\star B)), \text{ap}_{\text{proj}(f(a), -)}(\star g)), (\text{proj}(f(a), \star B'), \text{pushl}(f(a))) \right] \]

\[ \text{(basel, pushl}(\star A'))] \]

\[
\begin{align*}
\text{coh} : & \quad (X : \text{Type}) \quad (A' \land B') \\
& \quad (a : X) \quad (\text{proj}(\star A', \star B')) \\
& \quad (x_0 : X) \quad (\text{basel}) \\
& \quad (p_0 : a = x_0) \quad (\text{pushl}(\star A')) \\
& \quad (x_1 : X) \quad (\text{proj}(f(a), \star B')) \\
& \quad (p_1 : x_1 = x_0) \quad (\text{pushl}(f(a))) \\
& \quad (x_2 : X) \quad (\text{proj}(f(a), g(\star B))) \\
& \quad (p_2 : x_2 = x_1) \quad (\text{ap}_{\text{proj}(f(a), -)}(\star g)) \\
& \quad \rightarrow x_2 = x_0
\end{align*}
\]

The result is the desired term of type \( \text{proj}(f(a), g(\star B)) = \text{basel} \).
Metaprogramming

It seems possible to do it in theory, but it is so technical that we do not want to do it by hand.

Solution

Write a program which generates a formal proof for us!

The generated proof is written in the proof assistant Agda, and the generating program is also written in Agda, used as a programming language.

Workflow

$ agda --compile SmashGenerate.agda

# generate the executable

$ ./SmashGenerate > Result.agda

# generate the proof

$ agda Result.agda

# check the proof
Metaprogramming

It seems possible to do it in theory, but it is so technical that we do not want to do it by hand.

Solution
Write a program which generates a formal proof for us!
Metaprogramming

It seems possible to do it in theory, but it is so technical that we do not want to do it by hand.

Solution

Write a program which generates a formal proof for us!

The generated proof is written in the proof assistant Agda, and the generating program is also written in Agda, used as a programming language.
Metaprogramming

It seems possible to do it in theory, but it is so technical that we do not want to do it by hand.

Solution
Write a program which generates a formal proof for us!
The generated proof is written in the proof assistant Agda, and the generating program is also written in Agda, used as a programming language.

Workflow
$ agda --compile SmashGenerate.agda
       # generate the executable
$ ./SmashGenerate > Result.agda  # generate the proof
$ agda Result.agda                # check the proof
Results

The current version can prove almost everything except for the pentagon. In particular it can construct/prove

- $f \land g$, compatibility with identities
- $\sigma$, involutivity, naturality
- $\alpha, \alpha^{-1}$, inverses to each other, naturality (takes 10 minutes and 25 GB of memory)
- the hexagon (takes 7 minutes and 8 GB of memory)
Future directions

• Finish the pentagon and the few other things missing.
• Get a full meta-theoretic proof that it does work.
• Prove that the smash product is $\infty$-coherent (externally).
• Can this idea of higher dimensional rewriting be applied in other situations?
In topology, $A \wedge B$ is defined as a quotient.

- We identify points with each other, instead of adding paths between them.
- It is easy to define, e.g., $\alpha_{A,B,C} : (A \wedge B) \wedge C \rightarrow A \wedge (B \wedge C)$.
- The pentagon is trivial.
- It is not easy to prove that $\alpha_{A,B,C}$ is continuous!
The big picture

- There are some propositional equalities that we would like to pretend are reduction rules.
The big picture

• There are some propositional equalities that we would like to pretend are reduction rules.
• Cubical type theory does turn many of them into reduction rules, but we can’t really hope for, e.g., $f(*_A) \leadsto *_{A'}$ or $\text{proj}(a,*_B) \leadsto \text{base1}$ to ever be an actual reduction rule.
The big picture

- There are some propositional equalities that we would like to pretend are reduction rules.
- Cubical type theory does turn many of them into reduction rules, but we can’t really hope for, e.g., \( f(\star_A) \leadsto \star_{A'} \) or proj\((a, \star_B) \leadsto \text{base} \) to ever be an actual reduction rule.
- Can we find an automated way to handle such propositional reduction rules?
The big picture

• There are some propositional equalities that we would like to pretend are reduction rules.
• Cubical type theory does turn many of them into reduction rules, but we can’t really hope for, e.g., $f(\ast_A) \rightsquigarrow \ast_{A'}$ or $\text{proj}(a, \ast_B) \rightsquigarrow \text{base1}$ to ever be an actual reduction rule.
• Can we find an automated way to handle such propositional reduction rules?
• To a user of the proof assistant, it would look like things reduce, in reality the proof assistant is doing all the work behind the scenes.