Computer-generated proofs for the smash product

Guillaume Brunerie

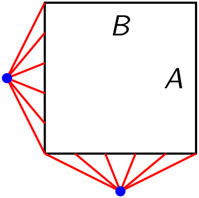
March 25, 2018
MURI Team meeting

The smash product

Definition

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

$$
\begin{aligned}
\texttt{proj} &: A \times B \to A \wedge B, \\
\texttt{basel} &: A \wedge B, \\
\texttt{gluel} &: (a : A) \to \texttt{proj}(a, \star_B) = \texttt{basel}, \\
\texttt{baser} &: A \wedge B, \\
\texttt{gluer} &: (b : B) \to \texttt{proj}(\star_A, b) = \texttt{baser}.
\end{aligned}
$$



1-coherent monoidality

Goal

We want a (formalized) proof of the fact 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 map $\sigma_{A,B} : A \wedge B \to B \wedge A$, which is natural and satisfies $\sigma_{B,A} \circ \sigma_{A,B} = \mathrm{id}_{A \wedge B}$.
- There is a map $\alpha_{A,B,C} : (A \wedge B) \wedge C \to A \wedge (B \wedge C)$, which is natural and has an inverse.
- It satisfies the pentagon and hexagon coherences.
- It has a unit with a triangular coherence.

[1] see pages 88 and 89 of my PhD thesis

Hexagon

$$\begin{tikzcd}
 & (A \wedge B) \wedge C \arrow[r, "\alpha_{A,B,C}"] \arrow[dl, "\sigma_{A,B} \wedge \mathrm{id}_C"'] & A \wedge (B \wedge C) \arrow[dr, "\sigma_{A,B \wedge C}"] & \\
(B \wedge A) \wedge C \arrow[dr, "\alpha_{B,A,C}"'] & & & (B \wedge C) \wedge A \arrow[dl, "\alpha_{B,C,A}"] \\
 & B \wedge (A \wedge C) \arrow[r, "\mathrm{id}_B \wedge \sigma_{A,C}"'] & B \wedge (C \wedge A) &
\end{tikzcd}$$

Basic idea

The basic idea is that all we have to do is to define various functions:

$$
\begin{aligned}
(x : A \wedge B) &\to P(x) \quad && \text{(6 of them)} \\
(x : (A \wedge B) \wedge C) &\to P(x) \quad && \text{(4 of them)} \\
(x : A \wedge (B \wedge C)) &\to P(x) \quad && \text{(2 of them)} \\
(x : ((A \wedge B) \wedge C) \wedge D) &\to P(x) \quad && \text{(1 of them)}
\end{aligned}
$$

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.

Example 1: commutativity

$$
\begin{aligned}
\sigma_{A,B} &: A \wedge B \to B \wedge A, \\
\sigma_{A,B}(\mathtt{proj}(a, b)) &:= \mathtt{proj}(b, a), \\
\sigma_{A,B}(\mathtt{basel}) &:= \mathtt{proj}(\star_B, \star_A), \\
\mathsf{ap}_{\sigma_{A,B}}(\mathtt{gluel}(a)) &:= \blacksquare_1 : \mathtt{proj}(\star_B, a) = \mathtt{proj}(\star_B, \star_A), \\
\sigma_{A,B}(\mathtt{baser}) &:= \mathtt{proj}(\star_B, \star_A), \\
\mathsf{ap}_{\sigma_{A,B}}(\mathtt{gluer}(b)) &:= \blacksquare_2 : \mathtt{proj}(b, \star_A) = \mathtt{proj}(\star_B, \star_A).
\end{aligned}
$$

We can fill the holes:

$$
\begin{aligned}
\blacksquare_1 &:= \mathtt{gluer}(a) \cdot \mathtt{gluer}(\star_A)^{-1} \\
\blacksquare_2 &:= \mathtt{gluel}(b) \cdot \mathtt{gluel}(\star_B)^{-1}
\end{aligned}
$$

□

Squares

We use squares and cubes in the sense of [LB15][2] .

Definition

The type

$$\mathtt{Square} : \{A : \mathtt{Type}\}\{a, b, c, d : A\}$$
$$(p : a = b)(q : c = d)(r : a = c)(s : b = d) \to \mathtt{Type}$$

is defined as the inductive family with one constructor

$$\mathtt{ids} : \mathtt{Square}(\mathtt{idp}, \mathtt{idp}, \mathtt{idp}, \mathtt{idp})$$

[2] D. Licata, G. Brunerie, *A Cubical Approach to Synthetic Homotopy Theory*, LICS 2015

Application of a homotopy to a path

Given two functions $f, g : A \to B$, a homotopy $h : (x : A) \to f(x) =_B g(x)$ and a path $p : a =_A a'$, then

$$\mathrm{ap}^{+}_{h}(p) : \texttt{Square}(\mathrm{ap}_f(p), \mathrm{ap}_g(p), h(a), h(a'))$$

$$\begin{tikzcd}
f(a) \arrow[r, dash, "h(a)"] \arrow[d, dash, "\mathrm{ap}_f(p)"'] & g(a) \arrow[d, dash, "\mathrm{ap}_g(p)"] \\
f(a') \arrow[r, dash, "h(a')"'] & g(a')
\end{tikzcd}$$

Example 2: involutivity of commutativity

$$
\begin{aligned}
\sigma\text{-}\mathtt{inv}_{A,B} &: (x : A \wedge B) \to \sigma_{B,A}(\sigma_{A,B}(x)) = x \\
\sigma\text{-}\mathtt{inv}_{A,B}(\mathtt{proj}(a,b)) &:= \mathtt{idp}_{\mathtt{proj}(a,b)} \\
\sigma\text{-}\mathtt{inv}_{A,B}(\mathtt{basel}) &:= \blacksquare_1 : \mathtt{proj}(\star_A, \star_B) = \mathtt{basel} \\
\mathtt{ap}^{+}_{\sigma\text{-}\mathtt{inv}_{A,B}}(\mathtt{gluel}(a)) &:= \blacksquare_2 : \mathtt{Square}(\mathtt{ap}_{\lambda x.\sigma_{B,A}(\sigma_{A,B}(x))}(\mathtt{gluel}(a)), \\
&\qquad\qquad\qquad \mathtt{ap}_{\lambda x.x}(\mathtt{gluel}(a)), \\
&\qquad\qquad\qquad \mathtt{idp}_{\mathtt{proj}(a,\star_B)}, \\
&\qquad\qquad\qquad \blacksquare_1) \\
\sigma\text{-}\mathtt{inv}_{A,B}(\mathtt{baser}) &:= \blacksquare_3 : \mathtt{proj}(\star_A, \star_B) = \mathtt{baser} \\
\mathtt{ap}^{+}_{\sigma_{A,B}}(\mathtt{gluer}(b)) &:= \blacksquare_4 : \mathtt{Square}([\dots])
\end{aligned}
$$

Reduction rules to fill the second hole

In order to fill $\blacksquare_2$, we need to use:

$$
\begin{aligned}
\mathsf{ap}_{\lambda x.x}(\texttt{gluel}(a)) &= \texttt{gluel}(a)\\
\mathsf{ap}_{\lambda x.\sigma_{B,A}(\sigma_{A,B}(x))}(\texttt{gluel}(a)) &= \mathsf{ap}_{\sigma_{B,A}}(\mathsf{ap}_{\sigma_{A,B}}(\texttt{gluel}(a)))\\
\mathsf{ap}_{\sigma_{A,B}}(\texttt{gluel}(a)) &= \texttt{gluer}(a)\cdot\texttt{gluer}(\star_A)^{-1}\\
\mathsf{ap}_{\sigma_{B,A}}(\texttt{gluer}(a)\cdot\texttt{gluer}(\star_A)^{-1}) &= \mathsf{ap}_{\sigma_{B,A}}(\texttt{gluer}(a))\cdot\mathsf{ap}_{\sigma_{B,A}}(\texttt{gluer}(\star_A))\\
\mathsf{ap}_{\sigma_{B,A}}(\texttt{gluer}(a)) &= \texttt{gluel}(a)\cdot\texttt{gluel}(\star_A)^{-1}\\
\mathsf{ap}_{\sigma_{B,A}}(\texttt{gluer}(\star_A)) &= \texttt{gluel}(\star_A)\cdot\texttt{gluel}(\star_A)^{-1}
\end{aligned}
$$

This is not enough, we then need to construct an element of type

$$
\begin{aligned}
&\texttt{Square}((\texttt{gluel}(a)\cdot\texttt{gluel}(\star_A)^{-1})\cdot(\texttt{gluel}(\star_A)\cdot\texttt{gluel}(\star_A)^{-1})^{-1},\\
&\qquad \texttt{gluel}(a), \texttt{idp}_{\texttt{proj}(a,\star_B)}, \texttt{gluel}(\star_A))
\end{aligned}
$$

End of the construction

We abstract over $\texttt{basel}$, $\texttt{proj}(\star_A, \star_B)$, $\texttt{proj}(a, \star_B)$, $\texttt{gluel}(a)$ and $\texttt{gluel}(\star_A)$. For arbitrary $X : \texttt{Type}$, $x, y, z : X$, $p : z = x$ and $q : y = x$, consider

$$\texttt{Square}((p \cdot q^{-1}) \cdot (q \cdot q^{-1})^{-1}, p, \texttt{idp}_z, q)$$

Finally, we apply path-induction on p and q, then return $\texttt{ids}$. $\square$

Example 3: associativity

$$
\begin{aligned}
\alpha_{A,B,C} &: (A \wedge B) \wedge C \to A \wedge (B \wedge C),\\
\alpha_{A,B,C}(\mathtt{proj}(x,c)) &:= \alpha^{\mathtt{proj}}_{A,B,C}(x,c),\\
\alpha_{A,B,C}(\mathtt{basel}) &:= \blacksquare,\\
\alpha_{A,B,C}(\mathtt{gluel}(x)) &:= \alpha^{\mathtt{gluel}}_{A,B,C}(x),\\
\alpha_{A,B,C}(\mathtt{baser}) &:= \blacksquare,\\
\alpha_{A,B,C}(\mathtt{gluer}(c)) &:= \blacksquare.
\end{aligned}
$$

Example 3: associativity

$$
\begin{aligned}
\alpha_{A,B,C} &: (A \wedge B) \wedge C \to A \wedge (B \wedge C), \\
\alpha_{A,B,C}(\mathtt{proj}(x, c)) &:= \alpha^{\mathtt{proj}}_{A,B,C}(x, c), \\
\alpha_{A,B,C}(\mathtt{basel}) &:= \blacksquare, \\
\alpha_{A,B,C}(\mathtt{gluel}(x)) &:= \alpha^{\mathtt{gluel}}_{A,B,C}(x), \\
\alpha_{A,B,C}(\mathtt{baser}) &:= \blacksquare, \\
\alpha_{A,B,C}(\mathtt{gluer}(c)) &:= \blacksquare.
\end{aligned}
$$

$$
\begin{aligned}
\alpha^{\mathtt{proj}}_{A,B,C} &: A \wedge B \to C \to A \wedge (B \wedge C), \\
\alpha^{\mathtt{proj}}_{A,B,C}(\mathtt{proj}(a, b), c) &:= \mathtt{proj}(a, \mathtt{proj}(b, c)), \\
&\quad [\ldots \blacksquare \ldots \blacksquare \ldots \blacksquare \ldots \blacksquare \ldots]
\end{aligned}
$$

Example 3: associativity

$$
\begin{aligned}
\alpha_{A,B,C} &: (A \wedge B) \wedge C \to A \wedge (B \wedge C), \\
\alpha_{A,B,C}(\mathtt{proj}(x, c)) &:= \alpha^{\mathtt{proj}}_{A,B,C}(x, c), \\
\alpha_{A,B,C}(\mathtt{basel}) &:= \blacksquare, \\
\alpha_{A,B,C}(\mathtt{gluel}(x)) &:= \alpha^{\mathtt{gluel}}_{A,B,C}(x), \\
\alpha_{A,B,C}(\mathtt{baser}) &:= \blacksquare, \\
\alpha_{A,B,C}(\mathtt{gluer}(c)) &:= \blacksquare.
\end{aligned}
$$

$$
\begin{aligned}
\alpha^{\mathtt{proj}}_{A,B,C} &: A \wedge B \to C \to A \wedge (B \wedge C), \\
\alpha^{\mathtt{proj}}_{A,B,C}(\mathtt{proj}(a, b), c) &:= \mathtt{proj}(a, \mathtt{proj}(b, c)), \\
&[\dots \blacksquare \dots \blacksquare \dots \blacksquare \dots \blacksquare \dots]
\end{aligned}
$$

$$
\begin{aligned}
\alpha^{\mathtt{gluel}}_{A,B,C} &: A \wedge B \to C \to [\dots] = [\dots], \\
&[\dots \blacksquare \dots \blacksquare \dots \blacksquare \dots \blacksquare \dots \blacksquare \dots]
\end{aligned}
$$

Irreducible elements of $A \wedge (B \wedge C)$

We may need to abstract over the following things in $A \wedge (B \wedge C)$. Will path induction work?

$\mathtt{proj}(a, \mathtt{basel})$
$\mathtt{proj}(\star_A, \mathtt{basel})$
$\mathtt{proj}(a, \mathtt{baser})$
$\mathtt{proj}(\star_A, \mathtt{baser})$
$\cancel{\mathtt{proj}(a, \mathtt{proj}(b, c))}$
$\mathtt{proj}(a, \mathtt{proj}(b, \star_C))$
$\mathtt{proj}(a, \mathtt{proj}(\star_B, c))$
$\mathtt{proj}(a, \mathtt{proj}(\star_B, \star_C))$
$\mathtt{proj}(\star_A, \mathtt{proj}(b, c))$
$\mathtt{proj}(\star_A, \mathtt{proj}(b, \star_C))$
$\mathtt{proj}(\star_A, \mathtt{proj}(\star_B, c))$
$\mathtt{proj}(\star_A, \mathtt{proj}(\star_B, \star_C))$

$\mathsf{ap}_{\mathtt{proj}(a,-)}(\mathtt{gluel}(b))$
$\mathsf{ap}_{\mathtt{proj}(a,-)}(\mathtt{gluel}(\star_B))$
$\mathsf{ap}_{\mathtt{proj}(\star_A,-)}(\mathtt{gluel}(b))$
$\mathsf{ap}_{\mathtt{proj}(\star_A,-)}(\mathtt{gluel}(\star_B))$
$\mathsf{ap}_{\mathtt{proj}(a,-)}(\mathtt{gluer}(c))$
$\mathsf{ap}_{\mathtt{proj}(a,-)}(\mathtt{gluer}(\star_C))$
$\mathsf{ap}_{\mathtt{proj}(\star_A,-)}(\mathtt{gluer}(c))$
$\mathsf{ap}_{\mathtt{proj}(\star_A,-)}(\mathtt{gluer}(\star_C))$
$\mathtt{basel}$
$\mathtt{baser}$

$\mathtt{gluel}(a)$
$\mathtt{gluel}(\star_A)$
$\mathtt{gluer}(\mathtt{basel})$
$\mathtt{gluer}(\mathtt{baser})$
$\mathtt{gluer}(\mathtt{proj}(b, c))$
$\mathtt{gluer}(\mathtt{proj}(b, \star_C))$
$\mathtt{gluer}(\mathtt{proj}(\star_B, c))$
$\mathtt{gluer}(\mathtt{proj}(\star_B, \star_C))$
$\mathsf{ap}^{+}_{\mathtt{gluer}}(\mathtt{gluel}(b))$
$\mathsf{ap}^{+}_{\mathtt{gluer}}(\mathtt{gluel}(\star_B))$
$\mathsf{ap}^{+}_{\mathtt{gluer}}(\mathtt{gluer}(c))$
$\mathsf{ap}^{+}_{\mathtt{gluer}}(\mathtt{gluer}(\star_C))$

Globular coherences (∞-groupoid structure on types)

We can construct any map of the form:

$$
\begin{aligned}
\texttt{coh} : (X : \texttt{Type})&(a : X) \\
&[\ldots] \\
&(x_n : T_n)(p_n : x_n = u_n) \qquad (\text{or } u_n = x_n) \\
&[\ldots] \\
&\to T
\end{aligned}
$$

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

Idea: induct on all p_n, then give `idp`. It works because such a coherence applied to only `idp`'s reduces (judgmentally!) to `idp`.

Cubical coherences

We actually also need to allow pairs of arguments of the form

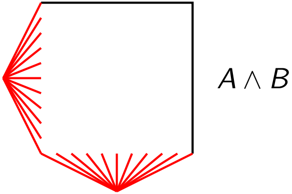
$$(x_n : T_n)(p_n : \mathtt{Square}(x_n, u_n, v_n, w_n))$$

with x_n being at any of the four positions, and similarly with cubes. . .

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.

Any such $(a : X)$ $[\ldots]$ $(x_n : [\ldots])(p_n : [\ldots])$ is called a (cubically) contractible context.

Intuition



(and imagine the corresponding picture for $A \wedge (B \wedge C)$)

Intuition

All the things that we may need to abstract over are in the red part, which is “contractible”.

More precise formulation

Sketch of definition (external to type theory)

Given a type A, a contractible system on A is a sequence of terms $(u_i : \Gamma_i \to T_i)$ where each T_i is either A, or an identity type of A, or a square type of A, such that for every finite family $(\gamma_k : \Gamma_{i_k})$, there exists a finite family $(\delta_j : \Gamma_{i'_j})$, such that

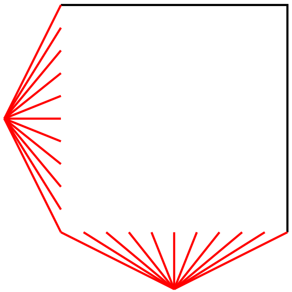
- every $u_{i_k}(\gamma_k)$ is one of the $u_{i'_j}(\delta_j)$,
- the family $u_{i'_j}(\delta_j)$ has the shape of a contractible context,
- all $u_{i'_j}(\delta_j)$ are judgmentally distinct.

Example

The following is a contractible system on $A \wedge B$:

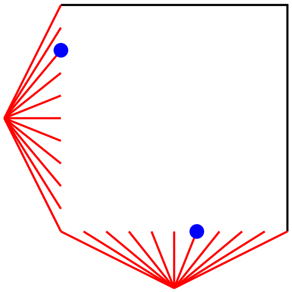
$$(\mathtt{basel}, \mathtt{baser}, \mathtt{gluel}, \mathtt{gluer}, \lambda a.\mathtt{proj}(a, \star_B), \lambda b.\mathtt{proj}(\star_A, b))$$

Main result



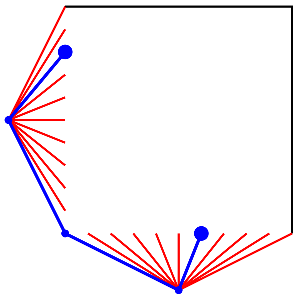
$A \wedge B$

Main result



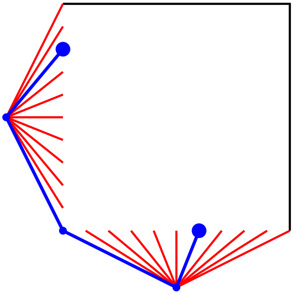
$A \wedge B$

Main result



$A \wedge B$

Main result



$A \wedge B$

Proposition (external)

Given a contractible system $\mathcal{C}_A$ on a type A, for any boundary landing in $\mathcal{C}_A$, there exists a term filling it.

Going back to associativity

In the definition of $\alpha_{A,B,C}$, for every ■ we needed to fill a boundary landing in $\mathcal{C}_{A\wedge(B\wedge C)}$. Therefore the previous proposition tells us (constructively) that there is a way to do it. $\square$

More things that we want

$$\alpha^{-1}_{A,B,C} : A \wedge (B \wedge C) \to (A \wedge B) \wedge C,$$

$$\alpha\text{-rinv}_{A,B,C} : (x : (A \wedge B) \wedge C) \to \alpha^{-1}_{A,B,C}(\alpha_{A,B,C}(x)) = x$$

$$\alpha\text{-linv}_{A,B,C} : (x : A \wedge (B \wedge C)) \to \alpha_{A,B,C}(\alpha^{-1}_{A,B,C}(x)) = x$$

$$\begin{aligned}\text{hexagon} : (x : (A \wedge B) \wedge C) \to (\mathrm{id}_B \wedge \sigma_{A,C})(\alpha_{B,A,C}((\sigma_{A,B} \wedge \mathrm{id}_C)(x))) \\ = \alpha_{B,C,A}(\sigma_{A,B\wedge C}(\alpha_{A,B,C}(x)))\end{aligned}$$

$$\begin{aligned}\text{pentagon} : (x : ((A \wedge B) \wedge C) \wedge D) \\ \to (\mathrm{id}_A \wedge \alpha_{B,C,D})(\alpha_{A,B\wedge C,D}((\alpha_{A,B,C} \wedge \mathrm{id}_D)(x)) \\ = \alpha_{A,B,C\wedge D}(\alpha_{A\wedge B,C,D}(x))\end{aligned}$$

Additional problems

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

$$\mathsf{ap}^2_f(sq) : \mathtt{Square}(\mathsf{ap}_f(p), \mathsf{ap}_f(q), \mathsf{ap}_f(r), \mathsf{ap}_f(s))$$

Question

Is there a term of type

$$\mathsf{ap}^2_{\lambda x.f(g(x))}(sq) = \mathsf{ap}^2_f(\mathsf{ap}^2_g(sq))?$$

Additional problems

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

$$\mathsf{ap}^2_f(sq) : \mathtt{Square}(\mathsf{ap}_f(p), \mathsf{ap}_f(q), \mathsf{ap}_f(r), \mathsf{ap}_f(s))$$

Question

Is there a term of type

$$\mathsf{ap}^2_{\lambda x.f(g(x))}(sq) = \mathsf{ap}^2_f(\mathsf{ap}^2_g(sq))?$$

This is not even well-typed!

Additional problems

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

$$\mathsf{ap}^2_f(sq) : \texttt{Square}(\mathsf{ap}_f(p), \mathsf{ap}_f(q), \mathsf{ap}_f(r), \mathsf{ap}_f(s))$$

Question

Is there a term of type

$$\mathsf{ap}^2_{\lambda x.f(g(x))}(sq) = \mathsf{ap}^2_f(\mathsf{ap}^2_g(sq))?$$

This is not even well-typed!

Solution

Instead we want

$$\texttt{Cube}(\mathsf{ap}^2_{\lambda x.f(g(x))}(sq), \mathsf{ap}^2_f(\mathsf{ap}^2_g(sq)), \mathsf{ap}\text{-}\circ_{f,g}(p), \mathsf{ap}\text{-}\circ_{f,g}(q),$$
$$\mathsf{ap}\text{-}\circ_{f,g}(r), \mathsf{ap}\text{-}\circ_{f,g}(s))$$

Additional problems (2)

Splitting the construction in two steps (do all "reduction rules", then solve the resulting problem) is a bad idea because we may need to prove coherences between those reduction rules and it's not clear where they would fit.

For instance there are many ways to reduce

$$\mathsf{ap}_{\lambda x.x}(\mathtt{gluel}(\star_A) \cdot \mathsf{ap}_{\lambda x.\sigma_{B,A}(\sigma_{A,B}(x))}(\mathtt{gluel}(\star_A)^{-1} \cdot \mathtt{gluer}(\star_B)))$$

Instead we need to combine both steps, by abstracting also over all the "reduction rules", and potentially over all the coherences needed.

The theory of contractible systems gets very messy.

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

Metaprogramming

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

Solution

Write a program which generates the Agda code for us!

Metaprogramming

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

Solution

Write a program which generates the Agda code for us!

Additional twist: the program is written in Agda as well, 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
```

(Actually the program constructs the coherences in a different way, using some form of higher-dimensional rewriting)

(demo)

Demo

(Actually the program constructs the coherences in a different way, using some form of higher-dimensional rewriting)

(demo)

The proof of the hexagon takes about 45 minutes and 45 GB of memory to type check, most of it used for constructing the coherences.

Using cubical type theory instead?

A lot of the trouble come from “reduction rules” that do not hold judgmentally.

Using cubical type theory instead?

A lot of the trouble come from “reduction rules” that do not hold judgmentally.

Idea

Use cubical type theory instead, where they do hold judgmentally.

Using cubical type theory instead?

A lot of the trouble come from “reduction rules” that do not hold judgmentally.

Idea

Use cubical type theory instead, where they do hold judgmentally.

Potential problem

We use the strict reduction rule of J in order to define the coherences, and the path type of cubical type theory has only weak J.

To be investigated. . .

Using reflection?

Crash course on reflection

Given A, we want to construct $a : A$ by "induction" on A. Define:

- an (inductive) type D,
- an interpretation function $[\![-]\!] : D \to \mathtt{Type}$,
- a function $\mathsf{solve} : (d : D) \to [\![d]\!]$,
- an element $d_A : D$ such that $[\![d_A]\!] \equiv A$.

Now set $a := \mathsf{solve}(d_A)$.

Using reflection?

Crash course on reflection

Given A, we want to construct $a : A$ by "induction" on A. Define:

- an (inductive) type D,
- an interpretation function $[\![-]\!] : D \to \mathtt{Type}$,
- a function $\mathsf{solve} : (d : D) \to [\![d]\!]$,
- an element $d_A : D$ such that $[\![d_A]\!] \equiv A$.

Now set $a := \mathsf{solve}(d_A)$.

In our case:

- D represents all ∞-groupoid coherences,
- $[\![d]\!]$ is the corresponding function type,
  $\to$ has the usual problems with an infinite tower of coherences
- solve is an (internal!) proof that every type is an ∞-groupoid

It may be possible to use reflection in a different way.

Future directions

- Finish the pentagon (either using hypercubes, or trying to make it globular) and the few other missing parts.
- Try to adapt the program to `cubicaltt` to compare.
- Develop the theory of contractible contexts further, to get a full human-readable proof.
- Prove that the smash product is ∞-coherent (externally).
- Apply the methodology of contractible contexts in other situations.