Short.

[DRAFT] https://github.com/1337777/cartier/blob/master/cartierSolution17.lp

How do you program a proof-assistant language without using variables names? Answer: you use a “context-extension” operation (ref. “categories with families”, “comprehension categories”, etc.) which is internalized (reflected/computational/strictified) into the language itself. So that syntactically there is only “one” (compound/telescope) variable in the context.

How do you program a higher-dimensional omega-categories proof-assistant language without using arbitrary-long (infinite) number of meta-grammar entities for 1-arrows, 2-arrows, 3-arrows, etc. ? Answer: you use a “comma/arrow-category” operation, together with the “context-extension” (now rebranded as “total category”) operation, to internalize/reflect back the comma/arrow dependent-category (fibration) as a base-category. So that syntactically a 2-arrow is simply an arrow in the comma/arrow category.

To make things precise, not only there should be a comma dependent-category construction if given a base (ordinary) category as input, but also there should be a “dependent comma/arrow” dependent-category (fibration) construction if given a dependent-category (fibration) $E \to B$ as input. Now such a dependent comma/arrow dependent-category F would be dependent/fibred over two bases: as $F \to E$ over the total category E of the dependent-category $E \to B$, and as $F \to \mathrm{arrow}(B)$ over the comma/arrow category $\mathrm{arrow}(B)$ of the base-category B. But this is a problem because possibly new meta-grammar entities are needed for 1-fibrations, 2-fibrations (over 2 bases), 3-fibrations, etc.

Herbelin solution.

This is essentially the problem that the existing attempts to globular, simplicial, cubical (omega-category) homotopy fail to address explicitly. Their “solutions” simply bypass this mathematics question and try to express things directly, declaratively using variables names, even using the so-called “dimension variables” names (and a constraint/boundary “solver”), with intricate meta-recursions over the dimension, in a very computer-sciencey style. Here is a research article closest to the solution (without the fancy “dimension variables”):

Hugo Herbelin; Ramkumar Ramachandra: “A parametricity-based formalization of semi-simplicial and semi-cubical sets”

This can be understood as an elaborate dependency/specification/constraint/boundary calculus, where a specification/dependency (“frame” for a “painting” of a cell) can be accumulated into telescopes of specifications, or can be finally discharged to specify/constraint a cell, and where a specification/dependency can be split as two sub-specifications (“restricted”) which are dispatched to the later “paintings” of cells.

So for a truncated cubical set $D : \mathrm{cSetTrunc}\_n$ representing a list $X\_0, X\_1, \ldots, X\_(n-1)$, of specified/dependent sets of cells, where $X\_(n-1) : \mathrm{frame}\_(n-1, n-1)(X\_0, X\_1, \ldots, X\_(n-2)) \to \mathrm{Set}$ and where $D.\mathrm{tl} = X\_(n-1)$; and for a frame $d : \mathrm{frame}\_(n,p)(D)$ representing a list of layers which specify/constraint the boundaries of subsequent layers and cell, one has mutually recursive definitions:

$$\mathrm{frame}\_(n,p)(D) \triangleq \Sigma\, d : (\ldots(\Sigma\, * \colon \mathrm{unit}.\, \mathrm{layer}\_(n,0)(D)(*))\ldots).\, \mathrm{layer}\_(n,p)(D)(d)$$

$$\mathrm{painting}\_(n,p)(D)(d) \triangleq \Sigma\, l\_p : \mathrm{layer}\_(n,p)(D)(d).\, (\ldots(\Sigma\, l\_n : \mathrm{layer}\_(n, n-1)(D)(d, l\_p, \ldots, l\_(n-1)).\, D.\mathrm{tl}\ (d, l\_p, \ldots, l\_n))\ldots)$$

$$\mathrm{layer}_{(n,p)(D)(d)} \triangleq \mathrm{painting}\_(n-1,p)(D.\mathrm{hd})(\mathrm{restrFrame}\_(\mathrm{left}, p)(d)) \times \mathrm{painting}\_(n-1,p)(D.\mathrm{hd})(\mathrm{restrFrame}\_(\mathrm{right}, p)(d))$$

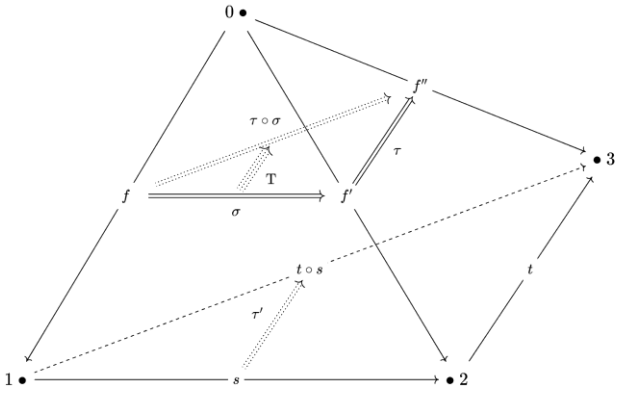
The problem with this solution is that each n-cell of the cubical set must be specified/"indexed" by a "fullframe" ($\triangleq \mathrm{frame\_}(n,n)$), that is, by all of its (n-1)-dim faces; instead of just by the usual target face (with fixed source face parameter) except the remainder of the faces being sigma-packed as components of/with the cell which can be later projected out.

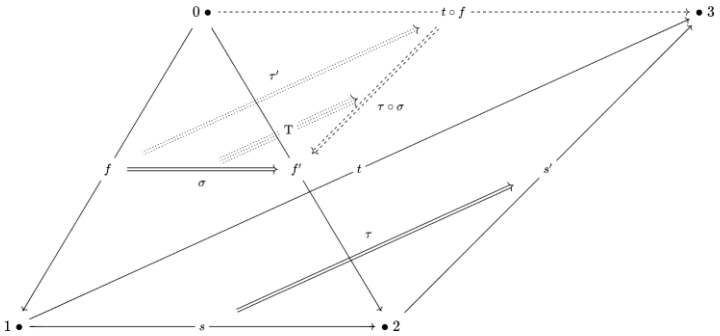
Mathematics solution.

But there is an alternative elementary solution which makes uses of the mathematics/algebra within the problem: the functoriality of the comma/arrow construction, that is if $E \to B$ then $\mathrm{arrow}(E) \to \mathrm{arrow}(B)$, and the observation that it is sufficient that an arrow (a cell/volume) be "pre-specified/indexed" only by its source-and-target, while the other faces of the arrow become "post-projections/components" out of such a specified arrow. That is, one of the legs of the would-be 2-fibration, say the leg $F \to E$, is regarded as an indexing $e : E \mapsto F(e) : \mathrm{Type}$, while the other leg $F \to \mathrm{arrow}(B)$ is regarded as an actual map/projection, indeed as a fibred map/projection from the fibration $(F \to E)$ to the fibration $(\mathrm{arrow}(B) \to B)$ over the map/projection $E \to B$. Also, there are versions where the roles of index or projection are switched between the legs of the 2-fibrations, and there are versions with contravariant fibrations and covariant fibrations.

For example, for the 3-dimensional tetrahedra $\mathrm{T} : V$ where $(V \to F \to E \to B)$ which witnesses the composition of two 2-dimensional triangles, then one (front) face σ is the source parameter (a fixed external parameter, not really a variable index which acts, used to define the arrow/slice category $V \coloneqq F(\sigma, -)$), and one (diagonal) side face $\xi : F$ is the target index used to pre-specify the volume $T : F(\sigma, \xi)$, and one (right) side face $\tau : \mathrm{arrow}(E)$ is a direct projection (via $V \to \mathrm{arrow}(E)$) from $T : V$ which had acted on the source face σ to output the composite target face $\xi \equiv \tau \circ \sigma$, and one (bottom) face $\tau' : \mathrm{arrow}(\mathrm{arrow}(B))$ is a functorial/recursive projection from the volume $T : V$ via the functoriality of the arrow construction $V \coloneqq \mathrm{arrow}(F) \to \mathrm{arrow}(\mathrm{arrow}(B))$ applied to the earlier/inner projection $F \to \mathrm{arrow}(B)$. More precisely:

```
constant symbol Commad_cov_catd [X : cat] [XX : catd X] [J : cat] [JJ : catd J] [R : mod J X] (RR : modd JJ R XX):

catd (Context_cat XX);

constant symbol Commad_cov_elimBase_funcd [X : cat] [XX : catd X] [J : cat] [JJ : catd J] [R : mod J X] (RR : modd JJ R
XX):

funcd (Commad_cov_catd RR) (Context_elimCat_func XX) (Comma_cov_catd R);

constant symbol Commad_baseCov_catd [X : cat] [XX : catd X] [J : cat] [JJ : catd J] [R : mod J X] (RR : modd JJ R XX):

catd (Context_cat (Comma_cov_catd R));

constant symbol Commad_baseCov_elimCov_funcd [X : cat] [XX : catd X] [J : cat] [JJ : catd J] [R : mod J X] (RR : modd JJ R
XX):

funcd (Commad_baseCov_catd RR) (Context_elimCat_func (Comma_cov_catd R)) XX;
```





Composition of cells

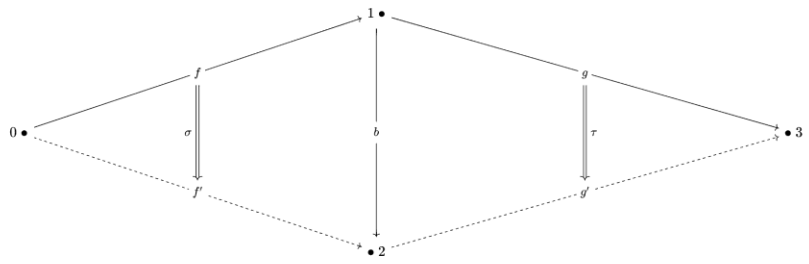
This action/composition/transport on a cell at its target or at its base are the more basic operations from which some of the usual operations are definable. For example, the whisker operation by a line is definable as the composition operation with a higher-dimensional composition-witness surface; and the simultaneous composition of multiple faces of an “open box” is defined by first factoring each extra face through a cartesian (composition-witness) face. The implementation of the “action/transport” for fibrations has an external formulation and an internalized formulation (via the comma/arrow category):

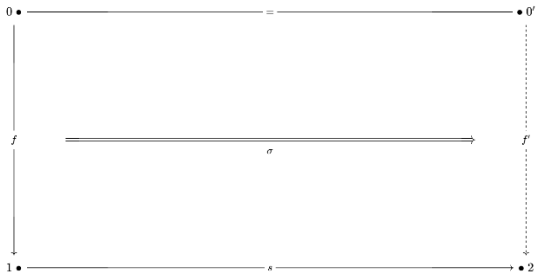
```
symbol Fibration_con_funcd : Π [I X X'  : cat] [x'x : func X X'] [H : func X I] [JJ : catd X] [F : func X' I] [II : catd
I] (II_isf : isFibration_con II), Π (FF : funcd JJ (x'x ∘> F) II) (f : hom H (Unit_mod Id_func F) x'x), funcd JJ H II;

injective symbol Comma_cov_elim_funcd : Π [I A J : cat] [F : func I A] [R : catd A] (R_isF : isFibration_cov R), Π (M :
func J A), funcd (Terminal_catd _) F R → funcd (Comma_cov_catd (Terminal_catd _) (Unit_mod F M)) M R;

constant symbol Comma_cov_elim_witness_funcd [X : cat] [A : catd X] (A_isF : isFibration_cov A) [J : cat] [R : func J X]
(RR : funcd (Terminal_catd _) R A) [I] (M : func I X):
funcd (Terminal_catd (Context_cat (Comma_cov_catd (Terminal_catd J) (Unit_mod R M))))
  (Context_intro_func (Comma_cov_elim_funcd A_isF M RR))
  (Commad_cov_catd (Unit_modd RR Id_funcd));
```

Stacking of cells

Besides these (“vertical”) composition/action operations, there must be a new “stacking along base” operation which generalizes the usual horizontal composition of 2-cells where now the two 2-cells share a common base 1-cell instead of a 0-cell (p.s. these are “relative dimensions”). This is the preliminary implementation (whose ability to effectively compute has not yet been tested):

```
constant symbol Commad_baseCov_stacking_funcd : Π [X : cat] [J : cat] [XX : catd X] [R : func J X]  (RR : funcd
(Terminal_catd _) R XX)  [T S : func J X] (g : hom Id_func (Unit_mod R T) Id_func) (b : funcd (Terminal_catd _) S
(Comma_cov_catd (Unit_mod R Id_func))),

(funcd (Terminal_catd _) (Context_intro_func b)
  (Product_catd (Commad_baseCov_catd (Unit_modd (Comma_con_intro_funcd ((Id_hom R ∘>'_ T) ∘' g)) Id_funcd))
    (Commad_baseCov_catd (Unit_modd RR Id_funcd)))) →
(funcd (Terminal_catd _) (Context_intro_func (Comma_cov_intro_funcd ( g '∘ ( R _'∘> Id_hom T) ))) (Commad_baseCov_catd
(Unit_modd RR Id_funcd)));
```





Cubical vs simplicial

For categorial (directed) homotopy, the “simplicial” and “cubical” are really just the same hybrid thing. A triangle of arrows is a square of arrows where one arrow is an identity/isomorphism/equality, and “cubical” means that only such (“marked”) composable cubes are considered. In a square, the source left arrow is a parameter, the acting botom arrow and top isomorphism are projections, and the target right arrow is an index (there is a version where projection-index are switched, and contravariant-covariant bi-fibrations become involved…). There is an implementation of the statement of external univalence and internalized univalence, should be definable/provable in the “cubical” case, even without access to “dimension variables” because this will be compensated by the fact that the setting here is double categorical with functors (out of omega-groupoids) instead of “single” objects within categories…

Site topology and sheafification, revisited (Concrete examples).

Often concrete examples of such (omega-)categories are more laborious to express, than the general theory. To illustrate this idea, here is an errata about a missing definition of “site topology” in an earlier implementation about sheaves and schemes; such a “site topology” concept is the (only) way to generate a concrete sheafification operation (Lawvere-Tierney operation). Refer to Maclane-Moerdijk. Sheaves in geometry and logic. (Chapter V, Section 1, Theorem 2 and Section 4, Theorem 1): Every (propositional-level) sheafification/closue j on the presheaf topos C^op -> Set defines a site topology J on a cate C.

FUNDAMENTAL FORMULA OF CATEGORIAL TOPOLOGY:

arrow f ∈ (sheafification-closure of sieve S) ⟷ (pullback sieve of S along f) ∈ site-topology

i.e. for all sieves S on an object F : C, for all arrows f in C with codomain F, then

f ∈ j(S) ⟷ f*(S) ∈ J

```
constant symbol site_topology : Π [S : cat] (S_site : site S) [I] [F : func I S],
  modd (Terminal_catd _) (Unit_mod Id_func F) (Terminal_catd _) → TYPE /*Prop*/;

constant symbol Maximal_site_topology : Π [S : cat] (S_site : site S) [I] [F : func I S],
  site_topology S_site (Terminal_modd (Unit_mod Id_func F));

constant symbol Site_topology_ssieve_homd : Π [S : cat] (S_site : site S) [I] [F : func I S][D] [K : func I D], Π (ff :
hom F (sieve S D) K), Π [G : func I S] (f : hom G (Unit_mod Id_func F) Id_func),
site_topology S_site (sieve_modd (pullback_sieve ff f))
→ /* ⟷ */ homd f (Fibre_elim_funcd _ G) (sieve_modd (ssieve_sieve S_site (ff '∘ sieve_ssieve S_site _)))
(Fibre_elim_funcd _ Id_func);
```