categorical model of dependent types

Categorical models of dependent types

Context

Type theory

Category theory

1. Idea

In the relation between type theory and category theory, dependent types are said to correspond to morphisms regarded as

indexed families. That is, if a type A corresponds to an object in some category, then a dependent type

$$(x:A) \vdash (B(x) \text{ type})$$

corresponds to a morphism $B \to A$ in that category. We think of this morphism as a bundle or fibration, whose fiber over $x:A$ is the type $B(x)$. We can then say that type forming operations such as dependent sum type and dependent product type correspond to category-theoretic operations of dependent sum and dependent product.

However, this correspondence is not quite precise; in the case of dependent types there are extra coherence issues. Substitution in type theory should correspond to pullback in category theory (but see at *substitution – Categorical semantics* for more); that is, given a term

$$(y:C) \vdash (f(y):A)$$

corresponding to a morphism $f:C \to A$, the substituted dependent type

$$(y:C) \vdash (B(f(y)) \text{ type})$$

should correspond to the pullback of $B \to A$ along f. However, substitution in type theory is strictly associative. That is, given also $g:D \to C$, the dependent type

$$(z:D) \vdash (B(f(g(z))) \text{ type})$$

is syntactically the same regardless of whether we obtain it by substituting $y \coloneqq g(z)$ into $B(f(y))$, or $x \coloneqq f(g(z))$ into $B(x)$. In category theory, however, pullback is not generally strictly associative, so there is a mismatch. Similarly, every type-forming operation must also be strictly preserved by substitition/pullback.

The way this is generally dealt with is to introduce a category-

theoretic structure which does have a “substitution” operation which is strictly associative, hence does correspond directly to type theory, and then show that any category can be “strictified” into such a stricter structure. Unfortunately, there are many different such structures which have been used, differing only slightly. On this page we define and compare them all.

One of these structures is called “contextual categories” (definition 2.16 below).

This and other kinds of categories-with-extra-structure may hence be thought of as stand-ins for the syntax of a type theory:

> Rather than constructing an interpretation of the syntax directly, we may work via the intermediary notion of contextual categories, a class of algebraic objects abstracting the key structure carried by the syntax. The plain definition of a contextual category corresponds to the structural core of the syntax; further syntactic rules (logical constructors, etc.) correspond to extra algebraic structure that contextual categories may carry. Essentially, contextual categories provide a completely equivalent alternative to the syntactic presentation of type theory.
>
> Why is this duplication of notions desirable? The trouble with the syntax is that it is mathematically tricky to handle. Any full presentation must account for (among other things) variable binding, capture-free substitution, and the possibility of multiple derivations of a judgement; and so a direct interpretation must deal with all of these, at the same time as tackling the details of the particular model in question. By passing to contextual categories, one deals with these subtleties and bureaucracy once and for all, and obtains a clear framework for subsequently constructing models. Conversely, why not work only with contextual categories, dispensing with syntax entirely?
>
> The trouble on this side is that working with higher-order logical structure in contextual categories quickly becomes unreadable. The reader preferring to take contextual categories as primary may regard the syntax as essentially a notation for working within them: a powerful, flexible,

> and intuitive notation, but one whose validity requires non-trivial work to establish. (The situation is comparable to that of string diagrams, as used in monoidal and more elaborately structured categories.)
>
> (from Kapulkin-Lumsdaine 12)

2. Definitions

In all the definitions, C will be a category. Generally, we will regard the objects of C as contexts in a type theory.

So far, we do not assume anything about C as a category. Usually, one at least wants C to have a terminal object, representing the empty context, although this is not always included in the definitions. The additional structures we impose on C below will imply in particular that certain pullbacks exist in C.

Sometimes, we want to consider C as a strict category, that is, we consider its objects up to equality rather than isomorphism. However, for most of the definitions below (until we get to contextual categories), it is still sensible to treat C in an ordinary category-theoretic way, with the strictness living in the additional structure.

All of this could be made more precise by assembling the structures considered below into categories, 2-categories, and/or strict 2-categories.

Comprehension categories

Definition 2.1. A ***comprehension category*** consists of a strictly commutative triangle of functors

$$\begin{array}{ccc} E & \to & C^I \\ & \searrow \quad \swarrow \text{cod} & \\ & C & \end{array}$$

where C^I is the arrow category of C and $\mathrm{cod}: C^I \to C$ denotes the codomain projection (which is a fibration if C has pullbacks), and such that

1. $E \to C$ is a Grothendieck fibration,
2. $E \to C^I$ takes cartesian morphisms in E to cartesian morphisms in C^I (i.e. to pullback squares in C).

Remark 2.2. In def 2.1 we do not ask that $C^I \to C$ be a fibration (that would require C to have all pullbacks), only that the *particular* morphisms in the image of E are cartesian.

Definition 2.3. A comprehension category (def. 2.1) is called

- ***full*** if $E \to C^I$ is a fully faithful functor.
- ***split*** if $E \to C$ is a split fibration.

In the latter case, we must consider E at least to be a strict category (that is, we consider its objects up to equality rather than isomorphism) for the notion to make sense.

Remark 2.4. The interpretation of def. 2.1 is as follows:

In a comprehension category, we may regard the objects of C as contexts, and the fiber E^Γ of $E \to C$ over an object Γ as the category of dependent types in context Γ. If the comprehension category is split (def. 2.3), then such dependent types have strictly associative substitution.

The functor $E \to C^I$ assigns to each "type A in context Γ" a new context which we think of as Γ *extended* by a fresh variable of type A, and write as $\Gamma.A$. This new context $\Gamma.A$ comes with a projection $\Gamma.A \to \Gamma$ (which forgets the fresh variable), and all substitutions in E are realized as pullbacks in C.

Display map categories

Definition 2.5. A ***display map category*** consists of a category C together with a class D of morphisms in C, called *display maps*, such that all pullbacks of display maps exist and are themselves display maps.

If C is a display map category, then by defining E to be the full subcategory of C^I whose objects are display maps, we obtain a full comprehension category (def. 2.3). Thus, we have:

Lemma 2.6. *Display map categories (def. 2.5) may be identified with those comprehension categories, def. 2.1, for which the functor $E \to C^I$ is the inclusion of a full subcategory.*

Remark 2.7. Working up to equivalence of categories, as is usual in category theory, it is natural to consider display map categories to also be equivalent to *full* comprehension categories, def. 2.3, (those where $E \to C^I$ is merely fully faithful).

However, this breaks down when we are interested in *split* comprehension categories, def. 2.3, for modeling substitution with strict associativity, since then E at least must be regarded as a strict category and considered up to isomorphism rather than equivalence. Thus, display map categories may be said to be equivalent to non-split full comprehension categories, but “split display map categories” are not equivalent to split full comprehension categories. (In fact, split display map categories are not very useful; usually in order to make pullbacks strictly associative we have to introduce extra “names” for the same objects.)

Categories with attributes

Definition 2.8. A ***category with attributes*** is a comprehension category, def. 2.1, for which $E \to C$ is a discrete fibration.

Remark 2.9. That is, in a category with attributes (def. 2.8) we demand only that each context comes with a set of dependent

types in that context, rather than a category of such. The intent is that the morphisms between two types in context Γ should be determined by the morphisms in C between the extended contexts over Γ.

Another way to convey this same intent with a comprehension category would be to ask that it be *full*, def. 2.3, i.e. that the functor $E \to C^I$ be fully faithful.

In fact, any category with attributes gives rise to a full comprehension category by factoring the functor $E \to C^I$ into a bijective on objects functor followed by a fully faithful functor. In this way, we obtain:

Lemma 2.10. *The category* **CwA** *of categories with attributes (def. 2.8) is equivalent to the category of* $\mathbf{CompCat}_{\text{full,split}}$ *of full split comprehension categories (def. 2.3).*

(These are, however, quite different as subcategories of **CompCat***.)*

Categories with families

A category with attributes specifies for each "context" only a set of "types" in that context. A comprehension category, by contrast, specifies a whole *category* of "types" in each context. If $A, B \in E^\Gamma$, then we may think of a morphism $f : A \to B$ in E^Γ as a term

$$(\vec{x} : \Gamma), (a : A(\vec{x})) \;\vdash\; (f(\vec{x}, a) : B(\vec{x})) \tag{1}$$

in type theory.
A *category with families* specifies instead, for each context and each type in that context, a set of "terms belonging to that type". These should be thought of as terms

$$(\vec{x} : \Gamma) \;\vdash\; (f(\vec{x}) : B(\vec{x})) \tag{2}$$

in type theory.

Remark 2.11. A term of the form (1) can equivalently be regarded as a term of the form (2) by replacing Γ by the extended context $\Gamma.A$, and B by its substitution along the projection $\Gamma.A \to \Gamma$.

The additional insight in the following definition is that if we expect all of these terms to be determined by the morphisms in C, as in a category with attributes or a full comprehension category, then instead of specifying the functor $E \to C^I$ and then asking either that it be fully faithful or that E be discrete (removing the information about extra morphisms in E), if we specify the terms of the form (2), then the functor $E \to C^I$ is determined by a universal property.

Let Fam denote the category of families of sets. Its objects are set-indexed families $(A_i)_{i \in I}$, and its morphisms $(A_i)_{i \in I} \to (B_j)_{j \in J}$ consist of a function $f : I \to J$ and functions $g_i : A_i \to B_{f(i)}$. We can equivalently, of course, regard this as the arrow category Set^I of Set, where $(B_j)_{j \in J}$ corresponds to the arrow $\coprod B \to J$.

Definition 2.12. A ***category with families*** is a category C together with

- A functor $F : C^{\mathrm{op}} \to \mathrm{Fam}$, written $F(\Gamma) = (\mathrm{Tm}(A))_{A \in \mathrm{Ty}(\Gamma)}$.
- For each $\Gamma \in C$ and $A \in \mathrm{Ty}(\Gamma)$, a representing object for the functor

$$(C/\Gamma)^{\mathrm{op}} \to \mathrm{Set}$$

$$(\Delta \xrightarrow{f} \Gamma) \mapsto \mathrm{Tm}(f^*(A))$$

Intuitively, $F(\Gamma)$ is the set of terms in the context Γ indexed by their type, and the representing object for the map $(C/\Gamma)^{\mathrm{op}} \to \mathrm{Set}$ is the context $\Gamma; A$.

We can then prove:

Lemma 2.13. *Categories with attributes, def. 2.8 are equivalent to categories with families, def. 2.12.*

Proof. Given a category with families, let $E \to C$ be the Grothendieck construction of the functor $\mathrm{Ty}: C^{\mathrm{op}} \to \mathrm{Set}$, and let $E \to C^I$ take each $A \in \mathrm{Ty}(\Gamma)$ to the above representing object. This is a category with attributes.

Conversely, given a category with attributes, let $\mathrm{Ty}: C^{\mathrm{op}} \to \mathrm{Set}$ be the functor corresponding to the discrete fibration $E \to C$, and for $A \in \mathrm{Ty}(\Gamma)$ let $\mathrm{Tm}(A)$ be the set of sections of the morphism in C that is the image of A in C^I. These constructions are inverses up to isomorphism. ∎

Natural Models

Steve Awodey (Awodey 2018) presented the following "natural model" of type theory as an alternative to categories with families.

Theorem 2.14. *If we modify Def. 2.12 by requiring only that the functors* $(\Delta \xrightarrow{f} \Gamma) \mapsto \mathrm{Tm}(f^*(A))$ *be representable (rather than equipped with representing objects), then it is equivalent to giving*

1. *a category C, together with*
2. *a morphism* $\mathrm{Tm} \to \mathrm{Ty}$ *in the category of presheaves on C which is a representable morphism.*

The category C models the category of contexts and substitutions, and the morphism $\mathrm{Tm} \to \mathrm{Ty}$ models the bundle of (context-dependent) terms over (context-dependent) types. The representability models the extension of a context with a new typed variable.

The condition of being a representable morphism can be reformulated in terms of representable profunctors as follows. A

natural model consists of

1. a category C,
2. a presheaf $\mathrm{Ty}\colon \hat{C}$ of types in context.
3. a presheaf $\mathrm{Tm}\colon \widehat{\int \mathrm{Ty}}$ of typed terms in context.
4. a functor $\mathrm{ext}\colon \widehat{\int \mathrm{Ty}} \to C$ of context extension with data that represents the profunctor $\int \mathrm{Ty}(\mathrm{ty} -, =) \odot C(-, \mathrm{ctx}(\mathrm{ty}(=)))\colon \int \mathrm{Ty} \nrightarrow C$ where $\mathrm{ty}\colon \int \mathrm{Tm} \to \int \mathrm{Ty}$ and $\mathrm{ctx}\colon \int \mathrm{Ty} \to C$ are the projections of the Grothendieck construction.

Writing the profunctor as P, it is equivalent to the following definition:

$$P(\Delta, (\Gamma, A)) = (\gamma\colon \Delta \to \Gamma) \times (a\colon \mathrm{Tm}(\Delta, A[\gamma]))$$

then the universal property of the context extension is that there is a natural isomorphism $\Delta \to \mathrm{ext}(\Gamma, A) \cong P(\Delta, (\Gamma, A))$

Contextual categories, or C-systems

Recall that

Definition 2.15. If C is a comprehension category (def. 2.1), $\Gamma \in C$ is a "context" and $A \in E^{\Gamma}$ is a "type" in context Γ, then we denote by $\Gamma . A$ the "extended context" in C (remark 2.4).

Definition 2.16. A ***contextual category*** (Cartmell 86, Streicher 91) or ***C-system*** (Voevodsky 15) is a category with attributes C (def. 2.8) together with a *length function* $\ell\colon \mathrm{ob}(C) \to \mathbb{N}$ such that

1. There is a unique object of length 0, which is a terminal object.
2. For any $\Gamma \in C$ and $A \in E^{\Gamma}$, we have $\ell(\Gamma . A) = \ell(\Gamma) + 1$.

3. For any $\Delta \in C$ with $\ell(\Delta) > 0$, there exists a unique $\Gamma \in C$ and $A \in E^\Gamma$ such that $\Delta = \Gamma . A$ (with notation as in def. 2.15).

Remark 2.17. Since def. 2.16 refers to equality of objects, a contextual category C must be a strict category.

Remark 2.18. The idea which distinguishes a contextual category is that "every context must be built out of types" in a unique way.

This makes for the closest match with type theory; in fact we have:

Theorem 2.19. *The category of contextual categories, def. 2.16, and (strictly) structure-preserving functors is equivalent to the category of dependent type theories and interpretations?.*

Remark 2.20. Since contextual categories are strict categories, the category of such is really a 1-category, or perhaps a strict 2-category.

Example 2.21. Given any category with attributes C, def. 2.8, possessing a terminal object, there is a canonical way to build a contextual category $\mathrm{cxt}(C)$, def. 2.16, from it.

1. Choose a terminal object $1 \in C$ (the resulting contextual category does not depend on this choice, up to isomorphism).
2. The objects of $\mathrm{cxt}(C)$ are the finite lists
$$(A_0, A_1, ..., A_n)$$
such that $A_0 \in E^1$ and $A_{k+1} \in E^{1 . A_0 . A_1 . \cdots . A_k}$ for all k.
3. The morphisms $(A_0, ..., A_n) \to (B_0, ..., B_m)$ in $\mathrm{cxt}(C)$ are the morphisms $1 . A_0 . A_1 . \cdots . A_n \to 1 . B_0 . B_1 . \cdots . B_m$ in C.

All the rest of the structure on $\mathrm{cxt}(C)$ is induced in an evident way from C.

3. Examples

Comprehension categories and display map categories are easy to

come by “in nature”, but it is more difficult to find examples of the “split” versions of the above structure (which are what is needed for modeling type theory). Here we summarize some basic known constructions.

However, first we should mention the examples that come from type theory itself.

Syntactic categories

Example 3.1. The syntactic category of any dependent type theory has all of the above structures. Its objects are contexts in the theory, and the types in context Γ form the set or category E^Γ. The strict associativity of substitution in type theory makes this fibration automatically split.

Splitting fibrations

There are standard constructions which can replace any Grothendieck fibration by an equivalent split fibration. Therefore,

Example 3.2. Given any comprehension category, def. 2.1,

1. we may replace it by a split comprehension category, def. 2.3,
2. then consider the underlying category with attributes, def. 2.8,
3. and finally pass to a contextual category by the construction in example 2.21.

Of course, comprehension categories are easy to come by; perhaps they arise most commonly as display map categories. For instance, if C has all pullbacks, then we can take all maps to be display maps. If C is a category of fibrant objects, we can take the fibrations to be the display maps.

So, for the record, we have in particular:

Example 3.3. For C a locally cartesian closed category C, it becomes a model for dependent type theory by regarding its codomain fibration $C^I \to C$ as a comprehension category, def. 2.1, and then strictifying that as in example 3.2.

Remark 3.4. It turns out that for modeling additional type-forming operations, however, not all splitting constructions are created equal.

Given $E \to C$, one classical construction (due to John Power) defines $E' \to C$, where an object of E' over $\Gamma \in C$ consists of a morphism $f : \Gamma \to \Delta$ in C along with an object A of E over Δ. Type-theoretically, we can regard (f, A) as a type A with a "delayed substitution" f. This produces a split fibration (the chosen cartesian arrows are given by composition of morphisms in C), but it seems impossible to define dependent sums and products on it in a strict way.

A better choice is a construction due to Benabou, which defines the objects of E' over $\Gamma \in C$ to be functors $C/\Gamma \to E$ over C which map every morphism of C/Γ to a cartesian arrow. Type-theoretically, we can think of such an object as a type together with *specified* compatible substitutions along any possible morphism. That type-formers may be extended in this case was proven by Martin Hofmann for dependent sums and dependent products and extensional identity types, and by Michael Warren in the case of intensional identity types (but not for the eliminator).

Universes

Suppose given a particular morphism $p : \widetilde{U} \to U$ in C. We can then define a category with attributes, def. 2.8, as follows: the discrete fibration $E \to C$ corresponds to the representable presheaf $C(-, U)$, and the functor $E \to C^I$ is defined by pullback of p. We are thus treating U as a "universe" of types. We may then of course pass to

a contextual category, via example 3.2.

Type-forming operations may be extended strictly in this case by performing them once in the "universal" case, then acting by composition. This construction is due to Voevodsky. It also meshes quite well with type theories that contain internal universes – a type of types– , and in particular for modeling the univalence axiom.

Particular important universes include:

- Any Grothendieck universe in Set, or more generally a universe in a topos. The resulting display maps are those with "U-small fibers".
- In the category Gpd, the groupoid of small groupoids. The resulting display maps are the split opfibrations with small fibers. Similarly, we can consider the groupoid of small sets, whose display maps are the discrete opfibrations with small fibers.
- In the category sSet of simplicial sets, there is a universal Kan fibration $p:\widetilde{U} \to U$ which classifies all Kan fibrations with small fibers.

Simple fibrations

Let C be any category with finite products, and define $E \to C$ to be the discrete fibration corresponding to the presheaf $C^{\mathrm{op}} \to \mathrm{Set}$ which is constant at $\mathrm{ob}(C)$. Thus, the objects of E are pairs (Γ, A) of objects of C, with the only morphisms being of the form $(\Gamma, A) \to (\Delta, A)$ induced by a morphism $\Gamma \to \Delta$ in C.

Define the functor $E \to C^I$ to take (Γ, A) to the projection $\Gamma \times A \to \Gamma$. It is straightforward to check that this defines a category with attributes. The corresponding (split) full comprehension category is called the *simple fibration* of C.

The dependent type theory which results from this structure “has no nontrivial dependency”. That is, whenever we have a dependent type $\Gamma \vdash (A \text{ type})$, it is already the case that A is a type in the empty context (that is, we have $\vdash (A \text{ type})$), and so it cannot depend nontrivially on Γ. In effect, it is not really a dependent type theory, but a *simple* (non-dependent) type theory — hence the name “simple fibration”.

4. Related pages

- categorical semantics
- relation between type theory and category theory
- Awodey's conjecture

5. References

Another overview can be found in the HoTT wiki.

A general overview may be found in

- Martin Hofmann, *Syntax and semantics of dependent types*, Semantics and logics of computation (Cambridge, 1995), Publ. Newton Inst., vol. 14, Cambridge Univ. Press, Cambridge, 1997, pp. 79–130

Comprehension categories are defined in

- Bart Jacobs, *Comprehension categories and the semantics of type dependency*, Theoret. Comput. Sci. 107 (1993), no. 2, 169–207

A correspondence with orthogonal factorization systems is discussed in

- Clemens Berger, Ralph M. Kaufmann, *Comprehensive factorisation systems* (pdf)

Display maps are discussed in

- Paul Taylor, *Practical Foundations of Mathematics*, section 8.3

Categories with attributes are discussed in

- John Cartmell, *Generalised algebraic theories and contextual categories*, Ph.D. thesis, Oxford, 1978 (GitHub LaTeXing project, organised by Peter LeFanu Lumsdaine. Currently only sections 1.0-1.4 are done)
- Eugenio Moggi, *A category-theoretic account of program modules*, Math. Structures Comput. Sci. 1 (1991), no. 1, 103–139
- Andrew M. Pitts, *Categorical logic*, Handbook of logic in computer science, Vol. 5, Handb. Log. Comput. Sci., vol. 5, Oxford Univ. Press, New York, 2000, pp. 39–128

Categories with families are defined in

- Peter Dybjer, *Internal type theory*, Types for proofs and programs (Torino, 1995), Lecture Notes in Comput. Sci., vol. 1158, Springer, Berlin, 1996, pp. 120–134, PDF

and shown to be equivalent to categories with attributes in

- Martin Hofmann, *Syntax and semantics of dependent types*, citeseer.

The formulation in terms of representable natural transformations is in

- Steve Awodey. (2018). *Natural models of homotopy type theory*, Mathematical Structures in Computer Science, 28(2), 241-286. PDF

A proof of initiality for dependent type theory is claimed in

- Simon Castellan, *Dependent type theory as the initial category with families*, 2014 (pdf)

This was formalized inside type theory with set quotients of higher inductive types in:

- Thorsten Altenkirch, Ambrus Kaposi, *Type Theory in Type Theory using Quotient Inductive Types*, (2015) (pdf), (formalisation in Agda).

Contextual categories were defined in

- John Cartmell, *Generalised algebraic theories and contextual categories*, Annals of Pure and Applied Logic Volume 32, 1986, Pages 209-243 (doi:10.1016/0168-0072(86)90053-9)
- Thomas Streicher, *Semantics of type theory*, Progress in Theoretical Computer Science, Birkhäuser Boston Inc., Boston, MA, 1991, Correctness, completeness and independence results.

Review includes

- Chris Kapulkin, Peter LeFanu Lumsdaine, section 1.2 and appendix B of *The Simplicial Model of Univalent Foundations (after Voevodsky)* (arXiv:1211.2851)

Contextual categories as models for homotopy type theory are discussed in

- Chris Kapulkin, Peter LeFanu Lumsdaine, *The homotopy theory of type theories* (arXiv:1610.00037)
- André Joyal, *Notes on Clans and Tribes* (arXiv:1710.10238)
- See also clan?.

Further discussion of contextual categories is in

- Vladimir Voevodsky, *A C-system defined by a universe category*, Theory Appl. Categ. 30 (2015), No. 37, 1181–1215, arXiv:1409.7925 (arXiv:1409.7925)
- Vladimir Voevodsky, *Martin-Löf identity types in the C-systems defined by a universe category* (arXiv:1505.06446)

- Vladimir Voevodsky, *Products of families of types in the C-systems defined by a universe category* (arXiv:1503.07072)
- Vladimir Voevodsky, *Subsystems and regular quotients of C-systems* (arXiv:1406.7413)

Strictification is discussed in

- Martin Hofmann, *On the interpretation of type theory in locally cartesian closed categories*
- Pierre-Louis Curien, Richard Garner, Martin Hofmann, *Revisiting the categorical interpretation of dependent type theory* (pdf)
- Peter LeFanu Lumsdaine, Michael Warren, *An overlooked coherence construction for dependent type theory*, CT2013
- Vladimir Voevodsky, Notes on type systems
- Steve Awodey. (2018). *Natural models of homotopy type theory,* Mathematical Structures in Computer Science, 28(2), 241-286. PDF

A comparison of various models, internally in type theory, is in

- Benedikt Ahrens, Peter LeFanu Lumsdaine, Vladimir Voevodsky, *Categorical structures for type theory in univalent foundations*, arxiv

Recent work on abstract definitions of (models of) type theory include:

- Valery Isaev, *Algebraic Presentations of Dependent Type Theories* arXiv
- Taichi Uemura, *A General Framework for the Semantics of Type Theory* arXiv

Last revised on April 23, 2019 at 09:09:15. See the history of this page for a list of all contributions to it.