Formalisation of Dependent Type Theory: The Example of CaTT

Thibaut Benjamin
Université Paris-Saclay, CEA, List, F-91120, Palaiseau, France

Abstract

We present the type theory CaTT, originally introduced by Finster and Mimram to describe globular weak ω-categories, formalise this theory in the language of homotopy type theory and discuss connections with the open problem internalising higher structures. Most of the studies about this type theory assume that it is well-formed and satisfy the usual syntactic properties that dependent type theories enjoy, without being completely clear and thorough about what these properties are exactly. We use our formalisation to list and formally prove all of these meta-properties, thus filling a gap in the foundational aspect. We discuss the aspects of the formalisation inherent to CaTT. We present the formalisation in a way that not only handles the type theory CaTT but also related type theories that share the same structure, and in particular we show that this formalisation provides a proper ground to the study of the theory MCaTT which describes the globular monoidal weak ω-categories. The article is accompanied by a development in the proof assistant Agda to check the formalisation that we present.

2012 ACM Subject Classification Theory of computation → Type theory; Theory of computation → Categorical semantics

Keywords and phrases Dependent type theory, homotopy type theory, higher categories, formalisation, Agda, proof assistant

Digital Object Identifier 10.4230/LIPIcs.TYPES.2021.2

Related Version *arXiv Version*: `https://arxiv.org/abs/2111.14736`

Supplementary Material *Software (Agda Source Code)*: `https://github.com/thibautbenjamin/catt-formalization`; archived at `swh:1:dir:db3f3e900b7541e07a7f19b1d895daa7fedbadd5`

Acknowledgements I want to thank Samuel Mimram and Eric Finster for their guidance in this project and the anonymous reviewers for their helpful comments.

1 Introduction

This article presents and formalises the foundations of the type theory CaTT introduced by Finster and Mimram [13], and similar type theories. CaTT is designed to encode a flavour of higher categorical structures called weak ω-categories, and its semantics has been proved [9] equivalent to a definition of weak ω-categories due to Maltsiniotis [22] based on an approach by Grothendieck [14]. Another example that fits in our framework is the theory MCaTT [8], which models monoidal weak ω-categories. We have formalised the work we present the proof-assistant Agda[1] in a fully proof-relevant way, without using Axiom K. we rely strongly on insights and ideas that emerged with Homotopy Type Theory (HoTT), and follow the homotopical interpretation of identity types. So we call this setting HoTT as well.

Although a few of the aforementioned articles primarily focus on CaTT, none of them give a complete foundation for it. Instead they simply assume that some syntactic meta-properties are satisfied. Far from being a shortcoming of those articles, this is common practice in the

[1] `https://github.com/thibautbenjamin/catt-formalization/tree/TYPES2021`

type theory community. Indeed low-level descriptions are lengthy and it is usually accepted that a full description satisfying the usual meta-theoretic properties is possible. As a result these articles rely on the reader's ability to infer and accept these foundations.

Moreover, the goal of formalising dependent type theory within itself, has been a long standing goal of dependent type theory [11], and is a hard and interesting problems. Important developments, such as the MetaCoq project [25], a formalisation of Coq written in Coq are being explored in this direction. More specifically in HoTT, it is a famous open problem to give an definition of HoTT within itself [24], and it is known to be closely linked with another open problem: The study of higher algebraic structures in HoTT.

We first present two main approaches to formalise dependent type theories and explore some connections with the definition of higher algebraic structures. We elaborate on the approach we chose and how it avoids the difficulty by restricting our focus to a purely syntactic definition. We then give a quick informal presentation of the theory CaTT and of a simpler type theory called GSeTT. Next, we discuss the formalisation of GSeTT, along with its meta-theoretic properties. Finally, we introduce our formalism of globular type theory, that models type theories similar to GSeTT, and show the meta-theoretic properties of interest. We show that CaTT and MCaTT are both instances of globular type theories and thus we get those properties for free. We present our formalisation in Agda pseudo-code with the convention that all the free variables are implicitly universally quantified, and provide some explanation for the syntax to help the reader navigate the article and the formalisation.

2 Structural foundation of dependent type theory

In this section, we give informal presentation of the type theories that we consider, and discuss the ways they can be formalised. The type theories we study are centred around four kinds of object, that we introduce here along with corresponding notations

$$\begin{array}{rlrl} \text{contexts:} & \Gamma, \Delta, \ldots & \text{terms:} & t, u, \ldots \\ \text{types:} & A, B, \ldots & \text{substitutions:} & \gamma, \delta, \ldots \end{array}$$

Each of these object is associated to a well-formedness judgement

$$\begin{array}{rlrl} \Gamma \text{ is a valid context:} & \Gamma \vdash & t \text{ is a term of type } A \text{ in } \Gamma\text{:} & \Gamma \vdash t : A \\ A \text{ is a valid type in } \Gamma\text{:} & \Gamma \vdash A & \gamma \text{ is a valid substitution from } \Delta \text{ to } \Gamma\text{:} & \Delta \vdash \gamma : \Gamma \end{array}$$

There are lots of flavours of type theories: dependent, linear (in the sense of linear logic), graded, or with refinement types... We consider are dependent type theories similar to Martin-Löf type theory without computation rules, that all respect the following rules.

$$\frac{}{\emptyset \vdash}(\text{EC}) \qquad \frac{\Gamma \vdash A}{\Gamma, x : A \vdash}(\text{CE}) \quad \text{Where } x \notin \text{Var}(\Gamma)$$

$$\frac{\Gamma \vdash \qquad (x : A) \in \Gamma}{\Gamma \vdash x : A}(\text{VAR})$$

$$\frac{\Delta \vdash}{\Delta \vdash \langle\rangle : \emptyset}(\text{ES}) \qquad \frac{\Delta \vdash \gamma : \Gamma \qquad \Gamma, x : A \vdash \qquad \Delta \vdash t : A[\gamma]}{\Delta \vdash \langle \gamma, x \mapsto t \rangle : (\Gamma, x : A)}(\text{SE})$$

Where $t[\gamma]$ is the application of the substitution γ to the term t and $\text{Var}(\Gamma)$ is the set of variables in Γ. For the sake of conciseness, we implicitly assume all type theories to satisfy those rules.

2.1 The typed syntax approach

An approach proposed by Dybjer [12] for internalising the semantics of dependent type theory is to define a typed syntax by induction induction. In this approach, every syntactic object must be well-formed by construction, it makes no sense to even consider an ill-formed object. For this reason we also refer to this approach as the *intrinsic type theory*.

Structure of the typed syntax. The intrinsic formalisation of dependent type theory relies on four inductive inductive types. They correspond to the four kinds of syntactic objects. We only give their signature in Agda pseudo-code to discuss specific points here. We use the keyword `data` to introduce mutually inductive types and the type `Set` is the type of types without any truncation assumption, this conflicts with the HoTT terminology.

```
data Ctx : Set                    data Tm : ∀ Γ → Ty Γ → Set
data Ty : Ctx → Set               data Sub : Ctx → Ctx → Set
```

In this framework the type `Ctx` is the type of well-formed contexts i.e., the type of all derivation trees of judgements of the form $\Gamma \vdash$, and similarly for the three other types. The type of types `Ty` depends on the type `Ctx`: It does not make sense to require a type to be well-formed, without a reference to the context in which this well-formedness is satisfied.

The semantics of the theory. This approach was originally proposed by Dybjer [12], in order to internalise a categorical device called category with families (CwF). It captures the algebraic structure of dependent type theory, and allows for studying its semantics. Defining the theory along its semantics is an important goal as it allows to study complex meta-theoretic properties, such as the independence of certain axioms by forcing or the initiality conjecture. This conjecture states that a theory should realise a CwF that is initial among a class of CwF that support the constructions of the theory. The typed syntax approach, provided that it can be developed completely, would provide a definition of a type theory that by construction enjoys the structure of a CwF and comes with a suitable notion of models by using the induction principle. Thus, it provides a unification between syntax and semantics, and gives both completely.

The dependency issue. An intrinsic approach presents hard challenges. For instance, we expect the action of substitutions on types and terms do be functorial:

$$A[\gamma][\delta] = A[\gamma \circ \delta] \qquad\qquad t[\gamma][\delta] = t[\gamma \circ \delta]$$

Within the typed syntax, the second of these equations is not even a valid statement. Indeed, suppose given the term `t : Tm` Γ `A` and two substitutions γ `: Sub` Δ Γ and δ `: Sub` Θ Δ then the term `t[`$\gamma \circ \delta$`]` is of type `Tm` Θ `A[`$\gamma \circ \delta$`]`, whereas the term `t[`γ`][`δ`]` is of type `Tm` Θ `A[`γ`][`δ`]`. Although these two types are equal by application of the first equality, they are not definitionally so. Thus even stating this equality, requires transporting one of the terms along the equality between the types. Proving later properties then requires a proper handling of these transports which becomes quickly prohibitive.

The coherence problem. On top the aforementioned technical issue, there is a more fundamental obstruction called the *coherence problem*: All the equalities, like the ones mentioned, have to be assumed as part of the structure in the form of term witnesses. But this is not enough, as equalities between those witnesses also need to be specified, and then equalities between their witnesses, and so on. This leads to a seemingly infinite amount of equations needed to present the theory. In such cases, it is sometimes possible to generate all of them with a finite scheme, but it is an open question to determine whether it is possible in this case. The same coherence issue also appears in a widely studied open problem consisting in defining higher algebraic structures in HoTT. In HoTT, each type is naturally endowed with a higher structure of groupoid, thanks to the identity types, and internalising a higher algebraic structure requires an additional higher structure. The coherence problem amounts to listing all the axioms of the latter in terms of the former, and ensuring that they play nicely together. A particularly studied instance of this is the definition of semi-simplicial types [16], which has been open since the birth of HoTT.

Workarounds for those problems. Several techniques to avoid the coherence problem while formalising either higher algebraic structures or dependent type theories have been considered. However, none of them solve the problem which is still open.

- In his work (anterior to HoTT) to formalise dependent type theory within itself [11], Chapman sidesteps the dependency by considering a heterogeneous equality predicate and mostly ignores the coherence problem. More recently, Lafont adapted this idea [2] and could avoid the coherence problem by requiring uniqueness of identity proofs (UIP) partially, which is not compatible with HoTT. This development is beyond the capabilities of Agda, and requires using the option `--no-termination-check` which deactivates the termination checker and is unsound in Agda.
- A technique proposed by Altenkirch, Capriotti and Kraus [2] is to use a 2-level type theory, where the equality is strict, but there exists a sub class of types called fibrant which also have a weak equality. Then one can restrict the focus only on fibrant types in the semantics to recover the desired behaviour. This lead to a lot of promising developments both for studying higher structures and formalising HoTT [18], but it sidesteps the coherence problem with a strict equality type, and thus is not internal to HoTT. This approach was also considered by Lafont in the same project[3].
- Another approach developped by Altenkirch and Kaposi [3] for formalising a dependent type theory consists in using quotient inductive inductive types (QIITs), a type theoretic construction that allows for set-truncated higher inductive types. The semantical interpretation of QIITs is very intricate, and from a practical perspective, it is really hard to develop such a method: Higher inductive types are not natively supported in Agda, and using them requires defining recursors manually and carefully avoiding Agda's pattern matching mechanism for those types, for which it is unsound.

Overall, the typed syntax approach presents an important challenge in the form of the coherence problem, so it does not translate easily in HoTT. The works we have presented provide workarounds by constraining the meta-theory further and adding some amount of proof-irrelevance, but this prevents the internalisation of the semantics.

2.2 The raw syntax approach

In this article, we take an alternate route to formalising dependent type theory, based on the separation of the syntax and the rules of the theory. To highlight the difference we call *raw syntax* the syntactic elements that are not tied to a derivation tree. Contrary to the typed

[2] `https://github.com/amblafont/omegatt-agda/tree/master`
[3] `https://github.com/amblafont/omegatt-agda/tree/2tt-fibrant`

syntax, raw syntax may contain ill-formed entities that do no correspond to any entity of the theory. We delegate the computational duty to the raw syntax, which completely sidesteps the coherence problem. We call this approach an *extrinsic* formalisation of a dependent type theory since it requires combining two separate ingredients, the raw syntax and the judgements. In this article we present a formalisation of the dependent type theory CaTT in an extrinsic way. Similar formalisations are being developped developed, notably by Finster[4] and Rice[5], but with the intent of formalising variations of CaTT with more intricate type-theoretic constructions. Moreover Lafont, Hirshowitz and Tabareau have also developed a formalisation of a type theory related to CaTT in a similar fashion in Coq[6] [19] with the goal of internalising that types are weak ω-groupoids [21, 27, 4].

Variables management. In order to compute operations on the raw syntax, we need the variables to be identifiable. To avoid the need of quotients, we develop a foundation in which the variables are natively normalised: Each variable name is uniquely determined. We use a variation on De Bruijn levels: The type of variables is the type of natural numbers $\mathbb{N}$ and we require contexts to enumerate their variables in increasing order. Since there is no variable binder in the theory CaTT, this suffices to determine all the variables.

Structure of the raw syntax. We now present the foundational structure of the dependent type theories that we are interested in. We first show the empty type theory: A theory with all the required structure but no types. This theory is completely vacuous: Its only context is empty, and no term or type is derivable in it. It is not part of our actual project and is not of any interest in itself, but we present it here to emphasize the structure of our type theories and factor out the common features of the theories we study. We first define the raw syntax: Contexts and substitutions may be built out of variables types and terms, and any variable is also a term. Type constructors and other term constructors may vary depending on the theory, so we do not include them in the empty type theory.

▶ **Definition 1.** *We define the raw syntax of the empty dependent type theory as a collection of four non-dependent types defined by mutual induction, representing respectively the (raw) contexts, substitutions, terms and types*

```
data Pre-Ctx : Set where
  ∅ : Pre-Ctx
  _·_#_ : Pre-Ctx → ℕ → Pre-Ty → Pre-Ctx
data Pre-Sub : Set where
  <> : Pre-Sub
  <_,_↦_> : Pre-Sub → ℕ → Pre-Tm → Pre-Sub
```

```
data Pre-Ty where
data Pre-Tm where
  Var : ℕ → Pre-Tm
```

The constructor `Var` produces an inhabitant of the type `Pre-Tm` from a variable (of type $\mathbb{N}$), and there is no constructor for the type `Pre-Ty` since there is no type. Those types do not need to be mutually inductive here, but we define them as such by anticipation with later theories. For all intents and purposes, we can think of contexts (resp. substitutions) as lists of pairs of the form (x, A) where x is a variable and A is a type (resp. lists of pairs (x, t) where x is a variable and t is a term, any variable is also a term). We denote $\ell\ \Gamma$ the length of a context Γ.

[4] https://github.com/ericfinster/catt.io/tree/master/agda
[5] https://github.com/alexarice/catt-agda
[6] https://github.com/amblafont/weak-cat-type/tree/untyped2tt

The action of substitutions. We define these operations on the raw syntax levels, and they are the reason why we need to introduce variable names. Those are functions that compute a new syntactic entity from a given one and a substitution to apply to it. These operations compute to normal forms, so when two applications are equal, the results of the computations are definitionnally so, thus we do not need equality witnesses and avoid the coherence problem at this level. The composition is an action of substitutions on substitutions.

▶ **Definition 2.** *We define the action of substitutions on types, terms and substitutions as the following mutually inductive operations*

```
_[_]T : Pre-Ty → Pre-Sub → Pre-Ty
_[_]t : Pre-Tm → Pre-Sub → Pre-Tm
_∘_ : Pre-Sub → Pre-Sub → Pre-Sub

() [ γ ]T

Var x [ <> ]t = Var x
Var x [ < γ , v ↦ t > ]t = if x ≡ v then t else ((Var x) [ γ ]t)

<> ∘ δ = <>
< γ , x ↦ t > ∘ δ = < γ ∘ δ , x ↦ t [ δ ]t >
```

Where `()[ γ ]T` represents the empty case, since there are no constructors for the type `Pre-Ty`. Again we use mutually inductive types for consistency with later developments.

Judgements and inference rules. We now select the well-formed syntactic entities from the syntax by introducing the judgements together with their inference rules. We define those as mutually inductive predicates over the raw syntax. We prove later (Theorems 17 and 25) that those types are actually propositions, their signature is given by

```
data _⊢C : Pre-Ctx → Set
data _⊢T_ : Pre-Ctx → Pre-Ty → Set
data _⊢t_#_ : Pre-Ctx → Pre-Tm → Pre-Ty → Set
data _⊢S_>_ : Pre-Ctx → Pre-Sub → Pre-Ctx → Set
```

The type $\Gamma \vdash_C$, for instance, is meant to represent the judgement $\Gamma \vdash$, an inhabitant of this type is built out of the type constructors, corresponding to inference rules. Hence an element of this type can be thought of as a derivation tree. Reasoning by induction on derivation trees, a common technique to prove meta-theoretic properties of dependent type theory, just translates to reasoning by induction on those four types. This discussion holds because there is no computation rules (i.e., rules postulating definitional equalities) in our theories: In the presence of such rules, one would need to consider higher inductive types, in order to preserve the correspondence between terms and derivation trees. The computation rules would then be higher order constructors.

▶ **Definition 3.** *We define the following inference rules for constructing contexts, substitutions and variable terms, which are the inference rules of the empty type theory (again, the type* $\_\vdash_T\_$ *has no constructor, the theory is vacuous).*

```
data _⊢C where
  ec : ∅ ⊢C
  cc : Γ ⊢C → Γ ⊢T A → x == ℓ Γ → (Γ · x # A) ⊢C
```

```
data _⊢T_ where
data _⊢t_#_ where
  var : Γ ⊢C → x # A ∈ Γ → Γ ⊢t (Var x) # A
data _⊢S_>_ where
  es : Δ ⊢C → Δ ⊢S <> > ∅
  sc : Δ ⊢S γ > Γ → (Γ · x # A) ⊢C → (Δ ⊢t t # (A [ γ ]T))
                                  → x == y → Δ ⊢S < γ , y ↦ t > > Γ · x # A
```

A term constructor of one of these types corresponds to a rule in the theory, so we have given the same name to the term constructors and the corresponding rules. In the rules `cc` and `sc`, we consider an equality on variables as part of the required data for a rule. This might seem odd at first, as we could instead inline this equality. However this lets us eliminate on the equality only when needed, which is important to avoid the axiom K. In the rule `cc`, the condition `x == ℓ Γ` enforces that contexts enumerate their variables in order.

Raw vs. typed syntax. Using a raw syntax approach has many advantages over a typed syntax. First, it can be completely formalised in a proof-relevant way and does not require any truncation. It also allows the use of the formalisation as a certified type checker, which is important for practical purposes. However, it focuses more narrowly on the syntax of the theory, and does not allow to internalise its algebraic properties (the CwF structure) nor its semantics. The transition from typed syntax is reminiscent of a more general construction to handle induction induction described by Kaposi, Kovács and Lafont [17].

Absence of semantics. This article focuses strongly on the syntax of the theory, and lacks semantics: We do not provide a way to show that a type models a theory. We conjecture that there should exist such a way, in the form of an internal notion of CwF, but that defining this notion would amount to solving the coherence problem. We do not address this question in this article. However we show that the syntax we provide realises an instance of this tentative notion of internal CwF, in which a lot of required equalities hold on the nose. Of course since this notion does not exist, this statement is imprecise, but we verify numerous properties that are expected to be part of this structure, such as weakening admissibility, the functoriality of the action of substitutions on types and terms... We also show the uniqueness of derivations, which is an important part of this tentative structure, as well as the decidability of type checking, which is relevant for practical applications and a result about a classifications of the types and terms of our theories. A similar approach was developed by Abel, Öman and Vezzosi [1], where they considered a more complex theory to prove weaker results, and give a similar discussion about the typed syntax in conclusion. We consider a simpler theory (in particular without computation rule), and are able to prove more results about it.

The duality theory/higher structure. We have discussed the coherence problem, and that it is the meeting point between internalising type theory and higher structures. Our work on CaTT illustrates the connection: The formalisation of CaTT can be seen as a formalisation of the internal language of weak ω-categories, since those two are the same [9]. More generally, we conjecture that the coherence problem appears in higher structures, like in type theory, at the level of the models and that in many case it is possible to define the internal language of a structure without running into it.

3 Introduction to the theory CaTT

We present here the type theory CaTT, focusing mostly on the syntactic aspects. We still provide some semantical intuition and we refer the reader to existing articles [13, 9] for more in-depth discussions about the semantics. In this section, we provide an informal presentation to serve as guideline for our foundations.

3.1 The theory GSeTT

We first introduce the theory GSeTT which is simpler than the theory CaTT and serves as a basis on which this theory relies. In the theory GSeTT there are no term constructors, hence the only terms are the variables. There are two type constructors, that we denote $\star$ and $\to$ and which are subject to the following introduction rules

$$\frac{\Gamma \vdash}{\Gamma \vdash \star}(\star\text{-INTRO}) \qquad\qquad \frac{\Gamma \vdash A \qquad \Gamma \vdash t : A \qquad \Gamma \vdash u : A}{\Gamma \vdash t \underset{A}{\to} u}(\to\text{-INTRO})$$

The contexts of the theory GSeTT can be represented by *globular sets*. Those are analogues to graphs, except that they are allowed to have arrows in every dimension (often called *cells*). A cell of dimension $n+1$ has for source and target a pair of cells of dimension n which are required to share the same source and target. In the type theory, this is imposed by rule ($\to$-INTRO) in which the terms t and u have to share the type A. We denote $\dim A$ the dimension of the cell represented by A. It can be computed as the number of iterated $\to$ needed to write it. We denote $\dim \Gamma$ the maximal dimension of the types that appear in Γ.

▶ **Example 4.** We illustrate the correspondence between contexts and finite globular sets with a few examples, using a diagrammatic representation of globular sets where we give the same name to variables and their corresponding cells.

$$\Gamma_c = (x : \star, y : \star, f : x \underset{\star}{\to} y, z : \star, h : y \underset{\star}{\to} z)$$

$$\begin{tikzcd} \overset{x}{\bullet} \arrow[r, "f"] & \overset{y}{\bullet} \arrow[r, "h"] & \overset{z}{\bullet} \end{tikzcd}$$

$$\Gamma_w = (x : \star, y : \star, f : x \underset{\star}{\to} y, g : x \underset{\star}{\to} y, \alpha : f \underset{\underset{\star}{x \to y}}{\to} g, z : \star, h : y \underset{\star}{\to} z)$$

$$\begin{tikzcd} \overset{x}{\bullet} \arrow[r, bend left, "f"] \arrow[r, bend right, "g"'] \arrow[r, phantom, "\Downarrow\alpha"] & \overset{y}{\bullet} \arrow[r, "h"] & \overset{z}{\bullet} \end{tikzcd}$$

$$\Gamma_{\circlearrowleft} = (x : \star, f : x \underset{\star}{\to} x)$$

$$\begin{tikzcd} \overset{x}{\bullet} \arrow[loop right, "f"] \end{tikzcd}$$

This correspondence is not an actual bijection: Several context may correspond to the same globular set if they only differ by reordering of the variables. One can account for this by considering the category of contexts with substitutions (called syntactic category). It is equivalent to the opposite of the category of finite globular sets [9, Th. 16]. A judgement $\Gamma \vdash x : A$ in the theory GSeTT corresponds to a well-defined cell x in the globular set corresponding to Γ, and the type A provides all the iterated sources and targets of x.

3.2 Ps-contexts

In the type theory CaTT there is an additional judgement on contexts, that recognises a special class of contexts called *ps-contexts*. We denote this judgement $\Gamma \vdash_{\mathsf{ps}}$, and we introduce it with the help of an auxiliary judgement $\Gamma \vdash_{\mathsf{ps}} x : A$. These two judgements are subject to the following derivation rules

$$\frac{}{x:\star \vdash_{\mathsf{ps}} x:\star}\,(\text{PSS}) \qquad \frac{\Gamma \vdash_{\mathsf{ps}} x : A}{\Gamma, y : A, f : x \underset{A}{\to} y \vdash_{\mathsf{ps}} f : x \underset{A}{\to} y}\,(\text{PSE})$$

$$\frac{\Gamma \vdash_{\mathsf{ps}} f : x \underset{A}{\to} y}{\Gamma \vdash_{\mathsf{ps}} y : A}\,(\text{PSD}) \qquad \frac{\Gamma \vdash_{\mathsf{ps}} x : \star}{\Gamma \vdash_{\mathsf{ps}}}\,(\text{PS})$$

The semantical intuition is that $\vdash_{\mathsf{ps}}$ characterises the contexts corresponding to *pasting schemes* [5, 22]. Those are the finite globular sets defining an essentially unique composition in weak ω-categories. Finster and Mimram have given an alternate characterisation of the ps-contexts [13] using a relation on the variables of a context, denoted $x \triangleleft y$.

▶ **Definition 5.** *The relation $\triangleleft$ in a context Γ, as the transitive closure of generated by*

$$x \triangleleft y \triangleleft z \text{ as soon as } \Gamma \vdash y : x \to z \text{ is derivable}$$

The authors have proved that a context is isomorphic to a ps-contexts if and only if this relation is a linear order on its variables.

Additionally, a ps-context Γ defines two subsets of its variables $\partial^-(\Gamma)$ and $\partial^+(\Gamma)$ respectively called the source and the target set, defined from a slightly more generic concept of i-source and i-target.

▶ **Definition 6.** *The i-*source *∂_i^- and the i-*target *∂_i^+ of a ps-context are given by induction*

$$\partial_i^-(x:\star) = (x:\star) \qquad \partial_i^-(\Gamma, y : A, f : x \to y) = \begin{cases} \partial_i^-(\Gamma) & \textit{if } \dim A \geq i \\ \partial_i^-(\Gamma), y : A, f : x \to y & \textit{otherwise} \end{cases}$$

$$\partial_i^+(x:\star) = (x:\star) \qquad \partial_i^+(\Gamma, y : A, f : x \to y) = \begin{cases} \partial_i^+(\Gamma) & \textit{if } \dim A > i \\ \mathrm{drop}(\partial_i^+(\Gamma)), y : A & \textit{if } \dim A = i \\ \partial_i^+(\Gamma), y : A, f : x \to y & \textit{otherwise} \end{cases}$$

where drop *is the list with its head removed. The* source *(resp. target) of a ps-context Γ is the set of all variables in its $(\dim \Gamma)$-source (resp. in its $(\dim \Gamma)$-target), i.e., $\partial^\pm(\Gamma) = \partial^\pm_{(\dim \Gamma)}(\Gamma)$.*

Semantically, those contain the variables in the source and target of the result of applying the essentially unique composition given by the pasting scheme corresponding to Γ.

▶ **Example 7.** The contexts Γ_c and Γ_w defined in Example 4 are ps-contexts, while the context $\Gamma_\circlearrowleft$ is not. We provide below the relation $\triangleleft$, and the source and target set variables.

$$\begin{array}{lclcc} \Gamma_c & : & x \triangleleft f \triangleleft y \triangleleft h \triangleleft z & \partial^-(\Gamma_c) = \{x\} & \partial^+(\Gamma_c) = \{z\} \\ \Gamma_w & : & x \triangleleft f \triangleleft \alpha \triangleleft g \triangleleft y \triangleleft h \triangleleft z & \partial^-(\Gamma_w) = \{x, f, y, h, z\} & \partial^+(\Gamma_w) = \{x, g, y, h, z\} \\ \Gamma_\circlearrowleft & : & x \underset{\triangleright}{\triangleleft} f & \multicolumn{2}{c}{\partial^-(\Gamma_\circlearrowleft) \text{ and } \partial^+(\Gamma_\circlearrowleft) \text{ undefined}} \end{array}$$

3.3 The type theory CaTT

The type theory CaTT is obtained from the type theory GSeTT by adding new term constructors that witness the structure of weak omega-categories. There are two of these constructors, op and coh, so a term is either a variable or of the form $\mathsf{op}_{\Gamma,A}[\gamma]$ or $\mathsf{coh}_{\Gamma,A}[\gamma]$, where in both two last cases, Γ is a ps-context, A is a type and γ is a substitution. These term constructors are subject to the following introduction rules.

$$\frac{\Gamma \vdash_{\mathsf{ps}} \qquad \Gamma \vdash t : A \qquad \Gamma \vdash u : A \qquad \Delta \vdash \gamma : \Gamma}{\Delta \vdash \mathsf{op}_{\Gamma, t \underset{A}{\to} u}[\gamma] : t[\gamma] \underset{A[\gamma]}{\to} u[\gamma]}\,(\textsc{op}) \qquad \frac{\Gamma \vdash_{\mathsf{ps}} \qquad \Gamma \vdash A \qquad \Delta \vdash \gamma : \Gamma}{\Delta \vdash \mathsf{coh}_{\Gamma, A}[\gamma] : A[\gamma]}\,(\textsc{coh})$$

Both these rules apply only under extra side-conditions. We denote (C_{op}) the side condition of (OP) and ($\mathrm{C}_{\mathsf{coh}}$) the one of (COH). Those side conditions are the following (where $\mathrm{Var}(A)$ denotes the set of variables needed to write the type A).

$$(\mathrm{C}_{\mathsf{op}}) : \begin{cases} \mathrm{Var}(t) \cup \mathrm{Var}(A) = \partial^-(\Gamma) \\ \mathrm{Var}(u) \cup \mathrm{Var}(A) = \partial^+(\Gamma) \end{cases} \qquad\qquad (\mathrm{C}_{\mathsf{coh}}) : \mathrm{Var}(A) = \mathrm{Var}(\Gamma)$$

Recall that a ps-context is meant to represent an essentially unique composition in a weak ω-category. The rules (OP) and (COH) enforce this condition, in a weak sense analogue to requiring the type of composition to be contractible in HoTT. More specifically, the rule (OP) asserts that in a context Δ, for every situation described by a ps-context Γ (as witnessed by γ), there exists a term witnessing the existence of the composition of this situation. The rule (COH) imposes that any two such compositions are related by a higher cell. Indeed in this rule the type A is necessarily of the form $a \to b$, where a and b represent two ways of composing the ps-context. The role of the side condition is to prevent the composition to apply partially.

▶ **Example 8.** Consider the context Γ_c introduced in Example 4, we have established in Example 7 that it is a ps-context, with $\partial^-(\Gamma_c) = \{x\}$ and $\partial^+(\Gamma_c) = \{z\}$. The type $x \underset{\star}{\to} z$ thus satisfies the condition (C_{op}). This shows that for every context Δ with a substitution $\Delta \vdash \gamma : \Gamma_c$, we can define the term `comp` $\gamma = \mathsf{op}_{\Gamma_c, x \underset{\star}{\to} z}[\gamma]$, and we have a derivation of $\Delta \vdash$ `comp` $\gamma : x[\gamma] \to z[\gamma]$. The data of a substitution $\Delta \vdash \gamma : \Gamma_c$ is equivalent to two composable arrows in Δ, and the semantics is that the term `comp` γ witnesses the composition of those arrows. Additional examples are presented in [13].

4 Formalisation and properties of the theory GSeTT

From now on, we use the extrinsic formalisation of dependent type theory, and present our type theories in a similar way to the empty type theory. We first define GSeTT by specifying two constructors for the type `Pre-Ty`, corresponding to the two type constructors of GSeTT. We then prove the inversion and weakening lemmas, and show that this theory has a structure of a category with families, with particular contexts classifying its types and terms, that type checking is decidable, and the derivation trees are necessarily unique.

4.1 Formal presentation of the theory GSeTT

Recall that the type theory GSeTT has only two type constructors $\star$ and $\to$, so we introduce two constructors to the type `Pre-Ty`. Note that this addition changes the entire raw syntax, since the other types `Pre-Ctx`, `Pre-Tm` and `Pre-Sub` are all mutually inductively defined with `Pre-Ty`. For simplicity purposes, we only present here the parts whose definition changes, but these changes actually propagate to the entire raw syntax.

▶ **Definition 9.** *The raw syntax of the type theory GSeTT is defined the same way as the raw syntax of the empty type theory, replacing the type* `Pre-Ty` *with the type*

```
 -- Pre-Ctx, Pre-Sub and Pre-Tm defined like in the empty TT
data Pre-Ty where
  ∗ : Pre-Ty
  _⇒[_]_ : Pre-Tm → Pre-Ty → Pre-Tm → Pre-Ty
```

Additionally, we introduce the action of substitution to be defined the same way on raw terms and substitutions as in the empty type theory, and we define it on raw types by

```
 -- _[_]t and _∘_ defined like in the empty TT
∗ [ γ ]T = ∗
(t ⇒[ A ] u)[ γ ]T = (t [ γ ]t) ⇒[ A [ γ ]T ] (u [ γ ]t)
```

We also specify the judgements of the theory, again those are defined in the same way for the empty type theory, except for the type judgement $\_\vdash_T\_$. Since these judgements are mutually inductive, a change here again propagates to all the judgements of the theory.

▶ **Definition 10.** *The type theory GSeTT is obtained from its raw syntax by adding the judgements defined in the same way as for the empty type theory, except for the following*

```
 -- _⊢C _⊢t_#_ and _⊢S_>_ defined like in the empty TT
data _⊢T_ where
  ob : ∀ Γ → Γ ⊢C → Γ ⊢T ∗
  ar : ∀ Γ A t u → Γ ⊢t t # A → Γ ⊢t u # A → Γ ⊢T t ⇒[ A ] u
```

We now present a lot of meta-properties that are required to study the semantics of the theory GSeTT. All of these properties are proved by induction on the derivation trees. A lot of these proofs are straightforward and we just give a discussion on the non-trivial proof techniques that come up. We refer the reader to the Agda implementation provided as supplementary material for the complete proofs of all of these properties.

4.2 Structure of the dependent type theory

First we show a few properties about the theory GSeTT, ensuring that the raw syntax together with a typing rule describe a dependent type theory with all the expected structure. Although these results are used a lot to study the semantics of the theory [13, 9], they are generally admitted and proving them requires as much specificity on the foundational aspects as we have provided here.

Weakening and judgement preservation. The following proposition states that GSeTT supports the weakening for types, terms and substitutions, as well as inversion.

▶ **Proposition 11.** *The theory GSeTT supports weakening for types, terms and substitutions.*

```
wkT : Γ ⊢T A → Γ · y # B ⊢C → Γ · y # B ⊢T A
wkt : Γ ⊢t t # A → Γ · y # B ⊢C → Γ · y # B ⊢t t # A
wkS : Δ ⊢S γ > Γ → Δ · y # B ⊢C → Δ · y # B ⊢S γ > Γ
```

Moreover, this theory staisfies inversion: Any sub-term of a derivable term is itself derivable. More precisely, we have the following (c.f. admitted results [13, Lemma 6] and [9, Lemma 6])

```
Γ⊢A→Γ⊢ : Γ ⊢T A → Γ ⊢C                  Γ,x:A⊢→Γ⊢ : Γ · x # A ⊢C → Γ ⊢C
Γ⊢t:A→Γ⊢ : Γ ⊢t t # A → Γ ⊢C            Γ⊢t:A→Γ⊢A : Γ ⊢t t # A → Γ ⊢T A
Δ⊢γ:Γ→Γ⊢ : Δ ⊢S γ > Γ → Γ ⊢C            Γ⊢s : Γ ⊢T t ⇒[ A ] u → Γ ⊢t t # A
Δ⊢γ:Γ→Δ⊢ : Δ ⊢S γ > Γ → Δ ⊢C            Γ⊢t : Γ ⊢T t ⇒[ A ] u → Γ ⊢t u # A
```

Structure of category with families. The study of the semantics relies on the structure of CwF of the theory. We can prove by induction on the derivation trees all the ingredients to show that GSeTT defines a CwF (at least set-theoretically). We do not show the final statement, as it requires developing a lot of category theory in Agda. The four following propositions show the structure of CwF of the theory GSeTT. We present them in the order in which we prove them, and they correspond to [9, Propositions 8, 9 and 10], where they are used without proof. We expect these results to be also necessary for a hypothetical internal notion of CwF.

▶ **Proposition 12.** *The action of a derivable substitution on a derivable type (resp. term) is again a derivable type (resp. term)*

```
[]T : Γ ⊢T A → Δ ⊢S γ > Γ → Δ ⊢T (A [ γ ]T)
[]t : Γ ⊢t t # A → Δ ⊢S γ > Γ → Δ ⊢t (t [ γ ]t) # (A [ γ ]T)
```

▶ **Proposition 13.** *In the theory GSeTT, there is an identity substitution defined by*

```
Pre-id : ∀ (Γ : Pre-Ctx) → Pre-Sub
Pre-id ∅ = <>
Pre-id (Γ · x # A) = < Pre-id Γ , x ↦ Var x >
```

It acts trivially on types, terms and substitutions on a raw syntax level and it is derivable.

```
[id]T : (A [ Pre-id Γ ]T) == A          ∘-right-unit : (γ ∘ Pre-id Δ) == γ
[id]t : (t [ Pre-id Γ ]t) == t          Γ⊢id:Γ : Γ ⊢C → Γ ⊢S Pre-id Γ > Γ
```

▶ **Proposition 14.** *The action of substitution is compatible with the composition of substitution*

```
[∘]T : Γ ⊢T A → Δ ⊢S γ > Γ → Θ ⊢S δ > Δ
    → ((A [ γ ]T) [ δ ]T) == (A [ γ ∘ δ ]T)
[∘]t : Γ ⊢t t # A → Δ ⊢S γ > Γ → Θ ⊢S δ > Δ
    → ((t [ γ ]t) [ δ ]t) == (t [ γ ∘ δ ]t)
```

▶ **Proposition 15.** *The composition of substitutions preserves the derivability of substitutions. Moreover, it is associative, and the identity is the left unit of the composition.*

```
∘-adm : Δ ⊢S γ > Γ → Θ ⊢S δ > Δ → Θ ⊢S (γ ∘ δ) > Γ a : Δ ⊢S γ > Γ
      → Θ ⊢S δ > Δ → Ξ ⊢S θ > Θ → (γ ∘ δ) ∘ θ == γ ∘ (δ ∘ θ)
∘-left-unit : Δ ⊢S γ > Γ → (Pre-id Γ ∘ γ) == γ
```

The result in Proposition 14 and 15 may not hold at the raw syntax level. For instance `id (∅ · 0 # ∗ · y # ∗) ∘ < x ↦ x >` computes to `< x ↦ x , y ↦ y>`.

4.3 Proof-theoretic considerations

We also prove notions that are more proof-theoretic about the theory. We also expect the uniqueness of derivation tree to be important for an hypothetical notion of internal CwF.

Decidability of type checking. In Martin-Löf type theory, we express that a type `A` is decidable, by exhibiting an inhabitant of the type `dec A = A + ¬ A`.

▶ **Theorem 16.** *Type checking in the theory GSeTT is a decidable problem*

```
dec-⊢C : ∀ Γ → dec (Γ ⊢C)          dec-⊢t : ∀ Γ A t → dec (Γ ⊢t t # A)
dec-⊢T : ∀ Γ A → dec (Γ ⊢T A)      dec-⊢S : ∀ Δ Γ γ → dec (Δ ⊢S γ > Γ)
```

We prove this theorem by mutual induction of the derivation tree, however the structure of the induction is quite complicated and showing that it is well-founded is a hard problem. This is where the use of a proof assistant like Agda becomes extremely useful, since its termination checker is able to verify this automatically. Proving this theorem in Agda amounts to implementing a certified type checker for the theory GSeTT.

Uniqueness of derivation trees. We express uniqueness in the language of HoTT, by defining contractible types (i.e., types a unique inhabitant) and proposition types (i.e., types that are either empty or contractible).

```
is-contr A = Σ A (λ x → ((y : A) → x == y))
is-prop A = ∀ (x y : A) → is-contr (x == y)
```

▶ **Theorem 17.** *In the theory GSeTT, every derivable judgement has a unique derivation (stated without proof [9, Lemma 7])*

```
is-prop-⊢C : is-prop (Γ ⊢C)          is-prop-⊢t : is-prop (Γ ⊢t t # A)
is-prop-⊢T : is-prop (Γ ⊢T A)        is-prop-⊢S : is-prop (Δ ⊢S γ > Γ)
```

We again prove by mutual induction, and the proof is fairly straightforward. The absence of computation rule is key for this proof to be simple. Computation rules, that could be represented as type constructors performing homotopy coherent quotients would make such a result much more technical to prove. We can recover the typed syntax from the raw syntax and the judgements, by considering dependent pairs of an element of the raw syntax together with its judgement as follows. For instance for contexts, we define the type `Ctx = Σ Pre-Ctx (λ Γ → Γ ⊢C)`, and similarly for types, terms and substitutions, we define the types `Ty Γ`, `Tm Γ A` and `Sub Δ Γ`.

4.4 Familial representability of types

We now define the disks and sphere contexts in GSeTT, which are families of contexts that play an important in the understanding of the semantics of the theory [9].

▶ **Definition 18.** *For every number* `n`*, we define a type* $\Rightarrow_u$ `n` *(u stands for "universal") and two contexts* `Pre-𝕊 n` *and* `Pre-𝔻 n` *by mutual induction in the raw syntax as follows*

```
⇒u O = ∗
⇒u (S n) = Var (2 n) ⇒[ ⇒u n ] Var (2 n + 1)
Pre-𝕊 O = ∅
Pre-𝕊 (S n) = (Pre-𝔻 n) · ℓ (Pre-𝔻 n) # ⇒u n
Pre-𝔻 n = (Pre-𝕊 n) · ℓ (Pre-𝕊 n) # ⇒u n
```

▶ **Proposition 19.** *The disk and sphere contexts are valid contexts in the theory GSeTT, and the type* $\Rightarrow_u$ *is derivable in the sphere context*

```
𝕊⊢ : Pre-𝕊 n ⊢C          𝔻⊢ : Pre-𝔻 n ⊢C          𝕊⊢⇒ : Pre-𝕊 n ⊢T ⇒u n
```

The sphere contexts play a particular role in the theory since they classify the types in a context: types in a context are equivalent to substitution from that context to a disk context. This is a result that we call familial representability of types [9], and that we formally prove in our foundational framework, using the definition of equivalence `is-equiv` usual to HoTT.

▶ **Theorem 20.** *For every context* Γ*, and any derivable substitution from* Γ *to a sphere, we define a derivable type in* Γ *by applying the substitution on the type* $\Rightarrow_u$ _*. The resulting map defines an equivalence*

```
Ty-n : ∀ Γ → Σ ℕ (λ n → Sub Γ (Pre-𝕊 n)) → Ty Γ
Ty-n Γ (n , (γ , Γ⊢γ:Sn) ) = ((⇒u n)[ γ ]T) , ([]T (𝕊⊢⇒ n) Γ⊢γ:Sn)

Ty-classifier : ∀ Γ → is-equiv (Ty-n Γ)
```

This result is substantially harder to prove formally than the previously mentioned ones, and relies on the uniqueness of derivation trees.

5 Formalisation and properties of the theory CaTT

In this section, we introduce our notion of globular type theory, and formalise it. We show that under suitable conditions, these theories satisfy the same good properties than GSeTT. We then construct CaTT and MCaTT as particular examples.

5.1 Globular type theories

Globular type theories are dependent type theories that have the same type structure as the theory GSeTT, but have term constructors. In order to describe not only the type theory CaTT, but also other dependent type theories, we define these term constructors generically. To this end, we assume a type `I`, which serves as an index to all the term constructors.

▶ **Definition 21.** *The raw syntax of a globular type theory is defined by the four mutually inductive types.*

```
 -- Pre-Ctx, Pre-Ty and Pre-Sub defined like in GSeTT
data Pre-Tm where
  Var : ℕ → Pre-Tm
  Tm-c : ∀ (i : I) → Pre-Sub → Pre-Tm
```

▶ **Definition 22.** *The action of substitution on the raw syntax is computed the same way as the action of substitutions on the raw syntax of the theory GSeTT, except on terms where it is defined by*

```
 -- _[_]T and _∘_ defined like in GSeTT
t [ <> ]t = t
Var x [ < δ , v ↦ t > ]t = if x ≡ v then t else ((Var x) [ δ ]t)
Tm-c i γ [ δ ]t = Tm-c i (γ ∘ δ)
```

Inference rules of globular type theories. We give a presentation of a generic form for the introduction of the indexed term constructors in globular type theories. To achieve this, we parameterise the rules, in such a way that every term constructor corresponds to its own introduction rule. We allow to have term constructors in the pre-syntax that do not correspond to any derivable term, if the rule is inapplicable. From now on, we assume that the type `I` has decidable equality, that is, we have a term

```
eqdecI : ∀ (x y : I) → dec (x == y)
```

In order to parameterise the rules, we suppose that for every inhabitant `i` of the type `I`, there exists a context `Ci i` in the raw syntax of GSeTT and a type `Ti i` in the raw syntax of the globular type theory. Moreover, we assume that the context `Ci i` is derivable in the theory GSeTT.

▶ **Definition 23.** *A globular type theory is a theory obtained from its syntax by imposing the same judgement rules as in the theory GSeTT for contexts, types and substitutions, and imposing for terms*

```
 -- _⊢C , _⊢T_ and _⊢S_>_ same as GSeTT
data _⊢t_#_ where
  var : Γ ⊢C → (x , A) ∈ Γ → Γ ⊢t (Var x) # A
  tm : Ci i ⊢T Ti i → Δ ⊢S γ > Ci i → Δ ⊢t Tm-c i γ # (Ti i [ γ ]T)
```

Note that again, the judgements of the theory are defined mutually inductively, and this change propagates to the other types. Here we use an explicit equality `A == Ti i [ γ ]T` instead of inlining the equality so that we have fine control over when it is eliminated, this allows us to stay compatible with avoiding the use of axiom K.

Properties of globular type theories. Most of the meta-theoretic properties extend from the theory GSeTT to any globular type theory, but there can be some difficulties in doing so.

▶ **Proposition 24.** *Every globular type theory satisfy all the results presented in Propositions 11, 12, 13, 14 and 15*

In this case, these results are a bit more involved to prove, because of the added dependency of terms on substitutions. Many results that could be proven separately in the case of GSeTT now depend on each other and have to be proven by mutual induction. Again, termination checking is not trivial, this is one instance where using Agda is a strong benefit.

▶ **Theorem 25** (c.f. Theorem 17)**.** *In any globular type theory, every derivable judgement has a single derivation tree.*

Definition 18 of the disks and sphere contexts also makes sense in any globular type theory. We also call those the disk and sphere contexts in the raw syntax of the globular theory.

▶ **Theorem 26** (c.f. Proposition 19 and Theorem 20)**.** *The disk and sphere context define valid contexts in any globular type theory, and the sphere contexts classify the types: There is an equivalence between the derivable types in a context and the substitutions from that context to a sphere context.*

Decidability of type checking. The decidability of type checking is a result that does not generalise as well to any globular type theory, because the generic form we have given for the rules is too permissive. Trying to reproduce the proof of GSeTT yields a proof whose termination cannot be checked by Agda: There is not a variant that decreases along the rules. And indeed, one may devise a globular type theory for which type checking is not decidable. But we can restrict our attention further, and consider theories satisfying an extra hypothesis

```
wfI : ∀ i → Ci i ⊢T Ti i → dimC (Ci i) ≤ dim (Ti i)
```

where `dim` is the dimension of a type (i.e., the number of iterated arrows it is built with) and `dimC` is the dimension of a context (i.e., the maximal dimension among the types it contains). The theory CaTT satisfies this hypothesis.

▶ **Theorem 27** (cf Theorem 16). *For every globular type theory satisfying the hypothesis* `wfI`*, the type checking is decidable.*

Proving this by induction is fairly straightforward, but ensuring that the induction is well-formed is quite involved. There is no obvious decreasing variant, so one needs to keep track of both the dimension and the number of nested term constructors in a precise way. This argument is non-trivial and the use of a termination checker like Agda's is extremely valuable.

5.2 Ps-contexts and the theory CaTT

We leverage the definition of globular type theory to formalise and prove some meta-theoretic properties of the theory CaTT. To this end, we define a particular type `J` to index the term constructors, as well as the contexts `Ci j` and the types `Ti j` to define the inference rules.

Ps-contexts. In our formalism, there is no difference between the term constructors op and coh, both of them are of the form `Tm-c`. If anything, formally, op and coh correspond to families of term constructors and not term constructors. One of the ingredients to index these families are the ps-contexts that we formally define here.

▶ **Definition 28.** *We define the judgements* _⊢ps *and* _⊢_#_ *over the raw syntax of the type theory GSeTT as the following inductive types (where we denote* `l` *for* ℓ Γ*)*

```
data _⊢ps_#_ : Pre-Ctx → ℕ → Pre-Ty → Set where
  pss : (nil · 0 # ∗) ⊢ps 0 # ∗
  psd : Γ ⊢ps f # (Var x ⇒[ A ] Var y) → Γ ⊢ps y # A
  pse : Γ ⊢ps x # A → ((Γ · l # A) · S l # Var x ⇒[ A ] Var l) ⊢ps
      S l # Var x ⇒[ A ] Var l

data _⊢ps : Pre-Ctx → Set where ps :
  Γ ⊢ps x # ∗ → Γ ⊢ps
```

▶ **Proposition 29.** *The ps-contexts are valid contexts of the theory GSeTT.*

```
Γ⊢ps→Γ⊢ : Γ ⊢ps → Γ ⊢
```

The relation ◁**.** To work with ps-contexts formally, we define the relation ◁ introduced by Finster and Mimram [13]. The purpose of this relation is to perform inductive reasoning.

▶ **Definition 30.** *Given a contest* Γ*, we define a generating relation* Γ ,_◁₀_*, together with its transitive closure* Γ ,_◁_ *as follows*

```
data _,_◁₀_ Γ x y : Set where
  ◁∂⁻ : Γ ⊢t (Var y) # (Var x ⇒[ A ] Var z) → Γ , x ◁₀ y
  ◁∂⁺ : Γ ⊢t (Var x) # (Var z ⇒[ A ] Var y) → Γ , x ◁₀ y

data _,_◁_ Γ x y : Set where
  gen : Γ , x ◁₀ y → Γ , x ◁ y
  ◁T : Γ , x ◁ z → Γ , z ◁₀ y → Γ , x ◁ y
```

▶ **Proposition 31.** *The ps-contexts are linear for the relation* _,_◁_*, i.e., whenever* Γ *is a ps-context, the relation* Γ , _ ◁ _ *defines a linear order on the variables of* Γ*.*

```
ps-◁-linear : ∀ Γ → Γ ⊢ps → ◁-linear Γ
```

The proof of this proposition in [13] relies on semantic consideration and the link between GSeTT and globular sets. In our approach, we instead give a purely syntactic proof of this result. This makes it very technical. The main ingredient of the proof is a subtle invariant, stating that whenever we have `Γ ⊢ps x # A` and `Γ , x ◁ y`, then necessarily `y` is an iterated target of `x` in `Γ`.

▶ **Theorem 32.** *The judgement* `_⊢ps` *is decidable, and any two derivation of the same judgement of this form are equal.*

```
is-prop-⊢ps : ∀ Γ → is-prop (Γ ⊢ps)        dec-⊢ps : ∀ Γ → dec (Γ ⊢ps)
```

These results are proven by induction on the derivation trees, but they are not straightforward. For instance in the case of the uniqueness, a derivation of `Γ ⊢ps` necessarily comes from a derivation of the form `Γ ⊢ps x # ∗` and by induction this derivation is necessarily unique. But the hard part is to prove that there can only be a unique `x` such that `Γ ⊢ps x # ∗`. Using the ◁-linearity, we can prove a more general lemma: if we have a `Γ ⊢ps x # A` and `Γ ⊢ps y # A` with `A` and `B` two types of the same dimension, then `x == y`. The decidability also presents a difficulty: The rule `psd` contains variables in its premises that are not bound in its conclusion. However, these variables belong to the context, so we can solve this issue by enumeration of the variables and ◁-linearity.

Index of term constructors. With the ps-contexts, we can define the index type for the term constructors in the theory CaTT. Recall that the term constructors in this theory are defined by $\mathsf{op}_{\Gamma,A}$ and $\mathsf{coh}_{\Gamma,A}$, where Γ is a ps-context and A is a type. In our informal presentation, we also required the side conditions (C_{op}) and ($\mathrm{C}_{\mathsf{coh}}$) in the derivation rules. For convenience, we integrate these side conditions in the index type in our formalisation, and define `J` to be the type of pairs of the form (Γ, A), where Γ is a ps-context and A is a type satisfying either (C_{op}) or ($\mathrm{C}_{\mathsf{coh}}$). The conditions ($\mathrm{C}_{\mathsf{op}}$) and ($\mathrm{C}_{\mathsf{coh}}$) are a bit subtle to formalise in HoTT, because one variable may appear several times in the same term, so there may be several witnesses that a term contains all the desired variables. But the intended semantics is propositional. To solve this issue, we propositionally truncate the type witnessing that a variable appears in type. We realise the truncation by using the type `set` of sets of numbers, for which membership is a proposition. We define the type `A ⊂ B` of witnesses that a set `A` is included in a set `B`, as well as the type `A` $\overset{o}{=}$ `B = (A ⊂ B) × (B ⊂ A)` of set equality. These two types are propositions since we are only manipulating finite subsets of ℕ.

▶ **Definition 33.** *We define the sets of source and target variables of a ps-context. First we define the i-sources and i-targets by induction on the judgement* `Γ ⊢ps x # A` *as lists.*

```
srcᵢ-var i pss = if i ≡ 0 then nil else (nil :: 0)
srcᵢ-var i (psd Γ⊢psx) = srcᵢ-var i Γ⊢psx
srcᵢ-var i (pse Γ = Γ A = A Γ⊢psx) with dec-≤ i (S (dim A))
... | inl i≤SdimA = srcᵢ-var i Γ⊢psx
... | inr SdimA<i = (srcᵢ-var i Γ⊢psx :: (ℓ Γ)) :: (S (ℓ Γ))

tgtᵢ-var i pss = if i ≡ 0 then nil else (nil :: 0)
tgtᵢ-var i (psd Γ⊢psx) = tgtᵢ-var i Γ⊢psx
tgtᵢ-var i (pse Γ = Γ A = A Γ⊢psx) with dec-≤ i (S (dim A))
... | inl i≤SdimA = if i ≡ S (dim A) then drop(tgtᵢ-var i Γ⊢psx) :: (ℓ Γ)
                                        else tgtᵢ-var i Γ⊢psx
... | inr SdimA<i = (tgtᵢ-var i Γ⊢psx :: (ℓ Γ)) :: (S (ℓ Γ))
```

Here the `with` *construction matches on the decidability of the order in* $\mathbb{N}$*. For instances matching on* `dec-≤ i n` *produces two cases of type* `i ≤ n` *and* `n < i`*. Moreover* `drop` *takes a list and removes its head. The source and target sets of a ps-context are the sets corresponding to the source and target lists in maximal dimension.*

```
src-var (Γ , ps Γ⊢psx) = set-of-list (srcᵢ-var (dimC Γ) Γ⊢psx)
tgt-var (Γ , ps Γ⊢psx) = set-of-list (tgtᵢ-var (dimC Γ) Γ⊢psx)
```

▶ **Definition 34.** *We define the type* `_is-full-in_`*, witnessing whether either the condition* $(\mathrm{C_{op}})$ *or* $(\mathrm{C_{coh}})$ *is satisfied, as follows*

```
data _is-full-in_ where
  Cop : (src-var Γ) ≗ ((varT A) ∪-set (vart t))
     → (tgt-var Γ) ≗ ((varT A) ∪-set (vart u)) → (t ⇒[ A ] u) is-full-in Γ
  Ccoh : (varC (fst Γ)) ≗ (varT A) → A is-full-in Γ
```

where `varT` *(resp.* `vart`*,* `varC`*) is the set of variables associated to a type (resp. to a term, to a context).*

▶ **Definition 35.** *The type* `J = Σ (ps-ctx × Ty) λ {(Γ , A) → A is-full-in Γ}` *is the index type for the term constructors of the theory CaTT. It is the types of pairs* (Γ, A) *where* Γ *is a ps-context and* A *is a raw type satisfying* $(\mathrm{C_{op}})$ *or* $(\mathrm{C_{coh}})$*.*

The raw syntax of CaTT is the raw syntax of the globular type theory indexed by `J`.

The type theory CaTT. We define the dependent type theory CaTT by adding rules to the raw syntax. It suffices to define a context `Ci i` and a type `Ti i` for every `i` of type `J`.

▶ **Definition 36.** *Considering a term* `(((Γ , Γ⊢ps), A), A-full)` *of type* `J`*, we pose*

```
Ci (((Γ , Γ⊢ps) , A) , A-full) = (Γ , Γ⊢ps→Γ⊢ Γ⊢ps)
Ti (((Γ , Γ⊢ps) , A) , A-full) = A
```

The theory CaTT is the globular type theory obtained from these assignations.

▶ **Proposition 37.** `J` *has decidable equality and satisfies the well-foundedness condition*

```
eqdecJ : ∀ (x y : J) → dec (x == y)
wfJ : ∀ j → Ci j ⊢T Ti j → dimC (Ci j) ≤ dim (Ti j)
```

Since this definition realises CaTT as a particular case of a globular type theory, it enjoys all the properties that we have already proved for them. In particular we have already proved

▶ **Theorem 38.** *In the theory CaTT the following statements hold.*

- *The theory support weakening and derivability is preserved by the inference rules.*
- *The theory CaTT defines a category with families.*
- *Every derivable judgement in CaTT has a unique derivation tree.*
- *The sphere contexts in CaTT classify the types.*
- *Type checking is decidable in the theory CaTT.*

5.3 The theory MCaTT

In addition to CaTT, we have also defined a dependent type theory to describe monoidal weak ω-categories, that we call MCaTT. There are actually two slightly different but equivalent formulations for this theory [8] and [7], and only the second one is a globular type theory. Thanks to the generic indexing mechanism that we have provided, we were able to formalise this theory as well, straightforwardly following the presentation given in [7]. Our understanding of the semantics relies on translations back and forth between CaTT and MCaTT. Such translations are defined by induction on the syntax, formalising them and proving their correctness is a technical challenge that we leave for future works.

6 Conclusion and further work

We have presented a full formalisation of the foundational aspects of the dependent type theory CaTT. Although this dependent type theory is quite simple, in that it does not have computation rules, proving formally all the relevant aspects that we expected turned out to be a substantial amount of work with highly non-trivial challenges to solve. In particular for some of the aspects such as the decidability of type checking, the use of a proof assistant such as Agda appears almost mandatory given the subtlety of the arguments. The notion of globular type theory is a limited attempt at a framework to carry out this work once in a generic enough setting that it can be used for CaTT and MCaTT, and it shows how a careful indexing of term constructors allows for some genericity while still being able to retain enough precision to show meaningful properties. It would be valuable to connect this notion with the work of Leena-Subramaniam [20], and a reasonable conjecture is that it corresponds to finitary monads on globular sets. More general approaches are being developed by Bauer, Haselwarter and Lumsdaine [6] and their development with Petkovic of the proof assistant Andromeda, Umeura [26] and Gylterud with the Myott project[7] [15]. Ultimately it would be valuable to have a formalised definitive framework in which all those meta-theoretic properties are proved once and for all.

The formalisation that we have presented, and in particular the proof for the decidability of type checking constitutes a verified implementation of a type checker for the theory CaTT. We have developed regular implementation of such a type checker[8] in OCaml. However, to improve the user experience, we have defined some meta-operations (called suspension and functorialization) on the syntax of the theory, that we proved correct manually [10, 7]. To avoid relying on the correctness of our implementation, the software computes the result of these operations and checks them like any user inputs. This leads to inefficiency in the implementation, and is not very satisfying. A better practice would be to define and prove formally those meta-operations, and then export the results to executable code in order to have a natively certified, but computationally light implementation of these meta-operations.

The work we have presented also shows once again the connection between the problem of internalising higher structures and internalising dependent type theory in HoTT. The meet point is the coherence issue, and we have avoided it here by only focusing solely on the syntax. A natural, but much harder follow up for this work would be to formalise the semantics of this theory. There is already some progress made in this direction by Uemura [23] with the definition of ∞-CwF, and we conjecture that internalizing this notion in HoTT would allow

[7] https://git.app.uib.no/Hakon.Gylterud/myott
[8] https://github.com/thibautbenjamin/catt

us to define an internal notion of models of CaTT, which could provide a suitable definition of weak ω-categories internally to HoTT. In general, giving a formulation of higher categorical results in terms of a syntax and a dependent type theory allows to perform some amount of reasoning that can be formalised within a proof assistant. We believe that it constitutes an asset for higher category theory, where the complexity of the theory itself quickly becomes a meaningful obstacle for any non-trivial exploration by hand of the theories.

References

1 Andreas Abel, Joakim Öhman, and Andrea Vezzosi. Decidability of conversion for type theory in type theory. *Proc. ACM Program. Lang.*, 2(POPL), December 2017. doi:10.1145/3158111.

2 Thorsten Altenkirch, Paolo Capriotti, and Nicolai Kraus. Extending homotopy type theory with strict equality. *CoRR*, 2016. arXiv:1604.03799.

3 Thorsten Altenkirch and Ambrus Kaposi. Type theory in type theory using quotient inductive types. *ACM SIGPLAN Notices*, 51(1):18–29, 2016. doi:10.1145/2837614.2837638.

4 Thorsten Altenkirch and Ondrej Rypacek. A syntactical approach to weak omega-groupoids. In *26th Computer Science Logic (CSL'12)*, volume 16 of *LIPIcs*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2012. doi:10.4230/LIPIcs.CSL.2012.16.

5 Michael A Batanin. Monoidal globular categories as a natural environment for the theory of weak n-categories. *Advances in Mathematics*, 136(1):39–103, 1998. doi:10.1006/aima.1998.1724.

6 Andrej Bauer, Philipp G Haselwarter, and Peter LeFanu Lumsdaine. A general definition of dependent type theories. *CoRR*, 2020. arXiv:2009.05539.

7 Thibaut Benjamin. *A type theoretic approach to weak ω-categories and related higher structures.* PhD thesis, Institut Polytechnique de Paris, 2020.

8 Thibaut Benjamin. Monoidal weak ω-categories as models of a type theory. *CoRR*, abs/2111.14208, 2021. arXiv:2111.14208.

9 Thibaut Benjamin, Eric Finster, and Samuel Mimram. Globular weak ω-categories as models of a type theory. *CoRR*, 2021. arXiv:2106.04475.

10 Thibaut Benjamin and Samuel Mimram. Suspension et Fonctorialité: Deux Opérations Implicites Utiles en CaTT. In *Journées Francophones des Langages Applicatifs*, 2019.

11 James Chapman. Type theory should eat itself. *Electronic notes in theoretical computer science*, 228:21–36, 2009. doi:10.1016/j.entcs.2008.12.114.

12 Peter Dybjer. Internal Type Theory. In *Types for Proofs and Programs. TYPES 1995*, pages 120–134. Springer, Berlin, Heidelberg, 1996. doi:10.1007/3-540-61780-9_66.

13 Eric Finster and Samuel Mimram. A Type-Theoretical Definition of Weak ω-Categories. In *2017 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–12, 2017. doi:10.1109/LICS.2017.8005124.

14 Alexander Grothendieck. Pursuing stacks. Unpublished manuscript, 1983.

15 Hakon Gylterud. Defining and relating theories. Presentation HoTT Electronic Seminar, 2021.

16 Hugo Herbelin. A dependently-typed construction of semi-simplicial types. *Mathematical Structures in Computer Science*, 25(5):1116–1131, 2015. doi:10.1017/S0960129514000528.

17 Ambrus Kaposi, András Kovács, and Ambroise Lafont. For finitary induction-induction, induction is enough. In *TYPES 2019*, volume 175 of *LIPIcs*, pages 6:1–6:30. Schloss Dagstuhl–Leibniz-Zentrum für Informatik GmbH, 2020. doi:10.4230/LIPIcs.TYPES.2019.6.

18 Nicolai Kraus. Internal ∞-categorical models of dependent type theory: Towards 2ltt eating hott. In *36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–14. IEEE, 2021. doi:10.1109/LICS52264.2021.9470667.

19 Ambroise Lafont, Tom Hirschowitz, and Nicolas Tabareau. Types are weak omega-groupoids, in Coq. *TYPES 2018*, 2018.

20 Chaitanya Leena-Subramaniam. *From dependent type theory to higher algebraic structures.* PhD thesis, Université Paris 7, 2021. arXiv:2110.02804.

21 Peter LeFanu Lumsdaine. Weak ω-categories from intensional type theory. In *TLCA*, pages 172–187. Springer, 2009. `doi:10.1007/978-3-642-02273-9_14`.

22 Georges Maltsiniotis. Grothendieck ∞-groupoids, and still another definition of ∞-categories. *CoRR*, 2010. `arXiv:1009.2331`.

23 Hoang Kim Nguyen and Taichi Uemura. ∞-type theories. In *abstract presented at the online workshop HoTT/UF'20*, 2020.

24 Mike Shulman. Homotopy type theory should eat itself (but so far, it's too big to swallow), 2014. URL: `https://homotopytypetheory.org/2014/03/03/hott-should-eat-itself/`.

25 Matthieu Sozeau, Abhishek Anand, Simon Boulier, Cyril Cohen, Yannick Forster, Fabian Kunze, Gregory Malecha, Nicolas Tabareau, and Théo Winterhalter. The MetaCoq project. *Journal of Automated Reasoning*, 64(5):947–999, 2020. `doi:10.1007/s10817-019-09540-0`.

26 Taichi Uemura. A general framework for the semantics of type theory. *CoRR*, 2019. `arXiv:1904.04097`.

27 Benno Van Den Berg and Richard Garner. Types are weak ω-groupoids. *Proceedings of the London Mathematical Society*, 102(2):370–394, 2011. `doi:10.1112/plms/pdq026`.