Gradualizing the Calculus of Inductive Constructions

Meven Lennon-Bertrand, Kenji Maillard, Nicolas Tabareau, Éric Tanter

Gradualizing the Calculus of Inductive Constructions

MEVEN LENNON-BERTRAND, Gallinette Project-Team, Inria, France
KENJI MAILLARD, Gallinette Project-Team, Inria, France
NICOLAS TABAREAU, Gallinette Project-Team, Inria, France
ÉRIC TANTER, PLEIAD Lab, Computer Science Department (DCC), University of Chile, Chile

We investigate gradual variations on the Calculus of Inductive Construction (CIC) for swifter prototyping with imprecise types and terms. We observe, with a no-go theorem, a crucial tradeoff between graduality and the key properties of normalization and closure of universes under dependent product that CIC enjoys. Beyond this Fire Triangle of Graduality, we explore the gradualization of CIC with three different compromises, each relaxing one edge of the Fire Triangle. We develop a parametrized presentation of Gradual CIC (GCIC) that encompasses all three variations, and develop their metatheory. We first present a bidirectional elaboration of GCIC to a dependently-typed cast calculus, CastCIC, which elucidates the interrelation between typing, conversion, and the gradual guarantees. We use a syntactic model of CastCIC to inform the design of a safe, confluent reduction, and establish, when applicable, normalization. We study the static and dynamic gradual guarantees as well as the stronger notion of graduality with embedding-projection pairs formulated by New and Ahmed, using appropriate semantic model constructions. This work informs and paves the way towards the development of malleable proof assistants and dependently-typed programming languages.

CCS Concepts: • **Theory of computation → Type theory**; **Type structures**; **Program reasoning**.

Additional Key Words and Phrases: Gradual typing, proof assistants, dependent types

1 INTRODUCTION

Gradual typing arose as an approach to selectively and soundly relax static type checking by endowing programmers with imprecise types [Siek and Taha 2006; Siek et al. 2015]. Optimistically well-typed programs are safeguarded by runtime checks that detect violations of statically-expressed assumptions. A gradual version of the simply-typed lambda calculus (STLC) enjoys such expressiveness that it can embed the untyped lambda calculus. This means that gradually-typed languages tend to accommodate at least two kinds of effects, non-termination and runtime errors. The smoothness of the static-to-dynamic checking spectrum afforded by gradual languages is usually captured by (static and dynamic) gradual guarantees which stipulate that typing and reduction are monotone with respect to precision [Siek et al. 2015].

Originally formulated in terms of simple types, the extension of gradual typing to a wide variety of typing disciplines has been an extremely active topic of research, both in theory and in practice. As part of this quest towards more sophisticated type disciplines, gradual typing was bound to meet with full-blown dependent types. This encounter saw various premises in a variety of approaches to integrate (some form of) dynamic checking with (some form of) dependent types [Dagand et al. 2018; Knowles and Flanagan 2010; Lehmann and Tanter 2017; Ou et al. 2004; Tanter and Tabareau 2015; Wadler and Findler 2009]. Naturally, the highly-expressive setting of dependent types, in which terms and types are not distinct and computation happens as part of typing, raises a lot of

*This work is partially funded by ANID FONDECYT Regular Project 1190058, and Inria Équipe Associée GECO.

To appear in ACM Transactions in Programming Languages and Systems, 2022, doi:10.1145/3495528

Authors' addresses: Meven Lennon-Bertrand, Gallinette Project-Team, Inria, Nantes, France; Kenji Maillard, Gallinette Project-Team, Inria, Nantes, France; Nicolas Tabareau, Gallinette Project-Team, Inria, Nantes, France; Éric Tanter, PLEIAD Lab, Computer Science Department (DCC), University of Chile, Santiago, Chile.

subtle challenges for gradualization. In the most elaborate effort to date, Eremondi et al. [2019] present a gradual dependently-typed programming language, GDTL, which can be seen as an effort to gradualize a two-phase programming language such as Idris [Brady 2013]. A key idea of GDTL is to adopt an approximate form of computation at compile-time, called *approximate normalization*, which ensures termination and totality of typing, while adopting a standard gradual reduction semantics with errors and non-termination at runtime. The metatheory of GDTL however still needs to be extended to account for inductive types.

This paper addresses the open challenge of gradualizing a full-blown dependent type theory, namely the Calculus of Inductive Constructions (hereafter, CIC) [Coquand and Huet 1988; Paulin-Mohring 2015], identifying and addressing the corresponding metatheoretic challenges. In doing so, we build upon several threads of prior work in the type theory and gradual typing literature: syntactic models of type theories to justify extensions of CIC [Boulier et al. 2017], in particular the exceptional type theory of Pédrot and Tabareau [2018], an effective re-characterization of the dynamic gradual guarantee as *graduality* with embedding-projection pairs [New and Ahmed 2018], as well as the work on GDTL [Eremondi et al. 2019].

Motivation. We believe that studying the gradualization of a full-blown dependent type theory like CIC is in and of itself an important scientific endeavor, which is very likely to inform the gradual typing research community in its drive towards supporting ever more challenging typing disciplines. In this light, the aim of this paper is not to put forth a unique design or solution, but to explore the space of possibilities. Nor is this paper about a concrete implementation of gradual CIC and an evaluation of its applicability; these are challenging perspectives of their own, which first require the theoretical landscape to be unveiled.

This being said, as Eremondi et al. [2019], we can highlight a number of practical motivating scenarios for gradualizing CIC, anticipating what could be achieved in a hypothetical gradual version of Coq, for instance.

Example 1 (Smoother development with indexed types). CIC, which underpins languages and proof assistants such as Coq, Agda and Idris, among others, is a very powerful system to program in, but at the same time extremely demanding. Mixing programs and their specifications is attractive but challenging.

Consider the classical example of length-indexed lists, of type vec A n as defined in Coq:[1]

```
Inductive vec (A : □) : ℕ → □ :=
| nil  : vec A 0
| cons : A → forall n : ℕ, vec A n → vec A (S n).
```

Indexing the inductive type by its length allows us to define a *total* head function, which can only be applied to non-empty lists:

```
head : forall A n, vec A (S n) → A
```

Developing functions over such structures can be tricky. For instance, what type should the `filter` function be given?

```
filter : forall A n (f : A → 𝔹), vec A n → vec A …
```

The size of the resulting list depends on how many elements in the list actually match the given predicate f! Dealing with this level of intricate specification can (and does) scare programmers away from mixing programs and specifications. The truth is that many libraries, such as Math-Comp [Mahboubi and Tassi 2008], give up on mixing programs and specifications even for simple

[1]We use the notation $\square_i$ for the predicative universe of types Type_i, and omit the universe level i when not required.

structures such as these, which are instead dealt with as ML-like lists with extrinsically-established properties. This tells a lot about the current intricacies of dependently-typed programming.

Instead of avoiding the obstacle altogether, gradual dependent types provide a uniform and flexible mechanism to a tailored adoption of dependencies. For instance, one could give `filter` the following gradual type, which makes use of the *unknown term* ? in an index position:

$$\texttt{filter} : \texttt{forall}\ \texttt{A}\ \texttt{n}\ (\texttt{f} : \texttt{A} \to \mathbb{B}),\ \texttt{vec}\ \texttt{A}\ \texttt{n} \to \texttt{vec}\ \texttt{A}\ ?$$

This imprecise type means that uses of `filter` will be optimistically accepted by the typechecker, although subject to associated checks during reduction. For instance:

$$\texttt{head}\ \mathbb{N}\ ?\ (\texttt{filter}\ \mathbb{N}\ 4\ \texttt{even}\ [\ 0\ ;\ 1\ ;\ 2\ ;\ 3\ ])$$

typechecks, and is successfully convertible to 0, while:

$$\texttt{head}\ \mathbb{N}\ ?\ (\texttt{filter}\ \mathbb{N}\ 2\ \texttt{even}\ [\ 1\ ;\ 3\ ])$$

typechecks but fails upon reduction, when discovering that the assumption that the argument to head is non-empty is in fact incorrect.

Example 2 (Defining general recursive functions). Another challenge of working in CIC is to convince the type checker that recursive definitions are well founded. This can either require tight syntactic restrictions, or sophisticated arguments involving accessibility predicates. At any given stage of a development, one might not be in a position to follow any of these. In such cases, a workaround is to adopt the "fuel pattern", *i.e.,* parametrizing a function with a clearly syntactically decreasing argument in order to please the typechecker, and to use an arbitrary initial fuel value. In practice, one sometimes requires a simpler way to unplug termination checking, and for that purpose, many proof assistants support external commands or parameters to deactivate termination checking.[2]

Because the use of the unknown type allows the definition of fix-point combinators [Eremondi et al. 2019; Siek and Taha 2006], one can use this added expressiveness to bypass termination checking locally. This just means that the external facilities provided by specific proof assistant implementations now become internalized in the language.

Example 3 (Large elimination, gradually). One of the argued benefit of dynamically-typed languages, which is accommodated by gradual typing, is the ability to define functions that can return values of different types depending on their inputs, such as:

```
def foo(n)(m) { if (n > m) then m + 1 else m > 0 }
```

In a gradually-typed language, one can give this function the type ?, or even $\mathbb{N} \to \mathbb{N} \to ?$ in order to enforce proper argument types, and remain flexible in the treatment of the returned value. Of course, one knows very well that in a dependently-typed language, with large elimination, we can simply give `foo` the dependent type:

$$\texttt{foo} : \texttt{forall}\ (\texttt{n}\ \texttt{m} : \mathbb{N}),\ \texttt{if}\ (\texttt{n} > \texttt{m})\ \texttt{then}\ \mathbb{N}\ \texttt{else}\ \mathbb{B}$$

Lifting the term-level comparison `n > m` to the type level is extremely expressive, but hard to work with as well, both for the implementer of the function and its clients.

In a gradual dependently-typed setting, one can explore the whole spectrum of type-level precision for such a function, starting from the least precise to the most precise, for instance:

$\texttt{foo} : ?$
$\texttt{foo} : \mathbb{N} \to \mathbb{N} \to ?$
$\texttt{foo} : \mathbb{N} \to \mathbb{N} \to \texttt{if}\ ?\ \texttt{then}\ \mathbb{N}\ \texttt{else}\ ?$
$\texttt{foo} : \texttt{forall}\ (\texttt{n}\ \texttt{m} : \mathbb{N}),\ \texttt{if}\ (\texttt{n} > \texttt{m})\ \texttt{then}\ \mathbb{N}\ \texttt{else}\ ?$

[2] such as `Unset Guard Checking` in Coq, or `{-# TERMINATING #-}` in Agda.

```
foo : forall (n m : ℕ), if (n > m) then ℕ else 𝔹
```

At each stage from top to bottom, there is less flexibility (but more guarantees!) for both the implementer of `foo` and its clients. The gradual guarantee ensures that if the function is actually faithful to the most precise type then giving it any of the less precise types above does not introduce any new failure [Siek et al. 2015].

Example 4 (Gradually refining specifications). Let us come back to the `filter` function from Example 1. Its fully-precise type requires appealing to a type-level function that counts the number of elements in the list that satisfy the predicate (notice the dependency to the input vector `v`):

```
filter : forall A n (f : A → 𝔹) (v : vec A n), vec A (count_if A n f v)
```

Anticipating the need for this function, a gradual specification could adopt the above signature for `filter` but leave `count_if` unspecified:

```
Definition count_if A n (f : A → 𝔹) (v: vec A n) : ℕ := ?.
```

This situation does not affect the behavior of the program compared to leaving the return type index unknown. More interestingly, one could immediately define the base case, which trivially specifies that there are no matching elements in an empty vector:

```
Definition count_if A n (f : A → 𝔹) (v : vec A n) : ℕ :=
  match v with
  | nil _ _ ⇒ 0
  | cons _ _ _ ⇒ ?
  end.
```

This slight increment in precision provides a little more static checking, for instance:

```
head ℕ ? (filter ℕ 4 even [])
```

does not typecheck, instead of failing during reduction.

Again, the gradual guarantee ensures that such incremental refinements in precision towards the proper fully-precise version do not introduce spurious errors. Note that this is in stark contrast with the use of axioms (which will be discussed in more depth in § 2). Indeed, replacing correct code with an axiom can simply break typing! For instance, with the following definitions:

```
Axiom to_be_done : ℕ.
Definition count_if A n (f : A → 𝔹) (v: vec A n) : ℕ := to_be_done.
```

the definition of `filter` does not typecheck anymore, as the axiom at the type-level is not convertible to any given value.

Note: Gradual programs or proofs? When adapting the ideas of gradual typing to a dependent type theory, one might expect to deal with programs rather than proofs. This observation is however misleading: from the point of view of the Curry-Howard correspondence, proofs and programs are intrinsically related, so that gradualizing the latter begs for a gradualization of the former. The examples above illustrate mixed programs and specifications, which naturally also appeal to proofs: dealing with indexed types typically requires exhibiting equality proofs to rewrite terms. Moreover, there are settings in which one must consider computationally-relevant proofs, such as constructive algebra and analysis, homotopy type theory, etc. In such settings, using axioms to bypass unwanted proofs breaks reduction, and because typing requires reduction, the use of axioms can simply prevent typing, as illustrated in Example 4.

Contribution. This article reports on the following contributions:

- We analyze, from a type theoretic point of view, the fundamental tradeoffs involved in gradualizing a dependent type theory such as CIC (§2), and establish a no-go theorem, the Fire Triangle of Graduality, which does apply to CIC. In essence, this result tells us that a gradual type theory[3] cannot satisfy at the same time normalization, graduality, and conservativity with respect to CIC. We explain each property and carefully analyze what it means in the type theoretic setting.
- We present an approach to gradualizing CIC (§3), parametrized by two knobs for controlling universe constraints on the dependent function space, resulting in three meaningful variants of Gradual CIC (GCIC), that reflect distinct resolutions of the Fire Triangle of Graduality. Each variant sacrifices one key property.
- We give a bidirectional and mutually-recursive elaboration of GCIC to a dependently-typed cast calculus CastCIC (§5). This elaboration is based on a bidirectional presentation of CIC, which has been recently studied in details by Lennon-Bertrand [2021], and of which we give a comprehensive summary in §4. Like GCIC, CastCIC is parametrized, and encompasses three variants. We develop the metatheory of GCIC, CastCIC and elaboration. In particular, we prove type safety for all variants, as well as the gradual guarantees and normalization, each for two of the three variants.
- To further develop the metatheory of CastCIC, we appeal to various models (§6). First, to prove strong normalization of two CastCIC variants, we provide a syntactic model of CastCIC with a translation to CIC extended with induction-recursion [Dybjer and Setzer 2003; Ghani et al. 2015; Martin-Löf 1996]. Second, to prove the stronger notion of graduality with embedding-projection pairs [New and Ahmed 2018] for a normalizing variant, we provide a model of CastCIC that captures the notion of monotonicity with respect to precision. Finally, we present an extension of Scott's model based on ω-complete partial orders [Scott 1976] to prove graduality for the variant with divergence.
- We describe how to handle indexed inductive types in GCIC, either directly or via different encodings, under some constraints on indices (§7).

We then elucidate the current limitations of this work regarding three important features of CIC—impredicativity, η-equality and propositional equality (§8). We finally discuss related work (§9) and conclude (§10). Some detailed proofs are omitted from the main text and can be found in appendix.

2 FUNDAMENTAL TRADEOFFS IN GRADUAL DEPENDENT TYPE THEORY

Before exposing a specific approach to gradualizing CIC, we present a general analysis of the main properties at stake and tensions that arise when gradualizing a dependent type theory.

We start by recalling two cornerstones of type theory, namely progress and normalization, and allude to the need to reconsider them carefully in a gradual setting (§2.1). We explain why the obvious approach based on axioms is unsatisfying (§2.2), as well as why simply using a type theory with exceptions [Pédrot and Tabareau 2018] is not enough either (§2.3). We then turn to the gradual approach, recalling its essential properties in the simply-typed setting (§2.4), and revisiting them in the context of a dependent type theory (§2.5). This finally leads us to establish a fundamental impossibility in the gradualization of CIC, which means that at least one of the desired properties has to be sacrificed (§2.6).

[3]Note that we sometimes use "dependent type theory" in order to differentiate from the Gradual Type Theory of New et al. [2019], which is simply typed. But by default, in this article, the expression "type theory" is used to refer to a type theory with full dependent types, such as CIC.

2.1 Safety and Normalization, Endangered

As a well-behaved typed programming language, CIC enjoys (type) **Safety** ($\mathcal{S}$), meaning that well-typed closed terms cannot get stuck, *i.e.,* the normal forms of closed terms of a given type are exactly the canonical forms of that type. In CIC, a closed canonical form is a term whose typing derivation ends with an introduction rule, *i.e.,* a λ-abstraction for a function type, and a constructor for an inductive type. For instance, any closed term of type $\mathbb{B}$ is convertible (and reduces) to either `true` or `false`. Note that an open term can reduce to an open canonical form called a *neutral term*, such as `not x`.

As a logically consistent type theory, CIC enjoys (strong) **Normalization** ($\mathcal{N}$), meaning that any term is convertible to its (unique) normal form. $\mathcal{N}$ together with $\mathcal{S}$ imply *canonicity*: any closed term of a given type *must* reduce to a canonical form of that type. When applied to the empty type `False`, canonicity ensures *logical consistency*: because there is no canonical form for `False`, there is no closed proof of `False`. Note that $\mathcal{N}$ also has an important consequence in CIC. Indeed, in this system, conversion—which coarsely means syntactic equality up-to reduction—is used in the type-checking algorithm. $\mathcal{N}$ ensures that one can devise a sound and complete decision procedure (a.k.a. a reduction strategy) in order to decide conversion, and hence, typing.

In the gradual setting, the two cornerstones $\mathcal{S}$ and $\mathcal{N}$ must be considered with care. First, any closed term can be ascribed the unknown type **?** first and then any other type: for instance, $0 :: \mathbf{?} :: \mathbb{B}$ is a well-typed closed term of type $\mathbb{B}$.[4] However, such a term cannot possibly reduce to either `true` or `false`, so some concessions must be made with respect to safety—at least, the notion of canonical forms must be extended.

Second, $\mathcal{N}$ is endangered. The quintessential example of non-termination in the untyped lambda calculus is the term $\Omega := \delta\ \delta$ where $\delta := (\lambda\ x.\ x\ x)$. In the simply-typed lambda calculus (hereafter STLC), as in CIC, *self-applications* like $\delta\ \delta$ and $x\ x$ are ill-typed. However, when introducing gradual types, one usually expects to accommodate such idioms, and therefore in a standard gradually-typed calculus such as GTLC [Siek and Taha 2006], a variant of Ω that uses $(\lambda\ x : ?.\ x\ x)$ for δ is well-typed and diverges, that is, admits no normal form. The reason is that the argument type of δ, the unknown type **?**, is *consistent* with the type of δ itself, $\mathbf{?} \to \mathbf{?}$, and at runtime, nothing prevents reduction from going on forever. Therefore, if one aims at ensuring $\mathcal{N}$ in a gradual setting, some care must be taken to restrict expressiveness.

2.2 The Axiomatic Approach

Let us first address the elephant in the room: why would one want to gradualize CIC instead of simply postulating an axiom for any term (be it a program or a proof) that one does not feel like providing (yet)?

Indeed, we can augment CIC with a general-purpose wildcard axiom `ax`:

```
Axiom ax : forall A, A.
```

The resulting theory, called CIC+`ax`, has an obvious practical benefit: we can use (`ax A`), hereafter noted ax_A, as a wildcard whenever we are asked to exhibit an inhabitant of some type `A` and we do not (yet) want to. This is exactly what admitted definitions are in Coq, for instance, and they do play an important practical role at some stages of any Coq development.

However, we cannot use the axiom ax_A in any meaningful way as a value *at the type level*. For instance, going back to Example 1, one might be tempted to give to the `filter` function on vectors the type `forall A n (f : A → ` $\mathbb{B}$`), vec A n → vec A ` $\mathsf{ax}_\mathbb{N}$, in order to avoid the complications related to specifying the size of the vector produced by `filter`. The problem is that the term:

[4]We write $a :: A$ for a type ascription, which is syntactic sugar for $(\lambda x : A.x)\ a$ [Siek and Taha 2006]; in other systems, it can be taken as a primitive notion [Garcia et al. 2016].

$$\mathsf{head}\ \mathbb{N}\ \mathsf{ax}_{\mathbb{N}}\ (\mathtt{filter}\ \mathbb{N}\ 4\ \mathsf{even}\ [\ 0\ ;\ 1\ ;\ 2\ ;\ 3\ ])$$

does not typecheck because the type of the filtering expression, $\mathtt{vec}\ \mathtt{A}\ \mathsf{ax}_{\mathbb{N}}$, is not convertible to $\mathtt{vec}\ \mathtt{A}\ (\mathtt{S}\ \mathsf{ax}_{\mathbb{N}})$, as required by the domain type of $\mathsf{head}\ \mathbb{N}\ \mathsf{ax}_{\mathbb{N}}$.

So the axiomatic approach is not useful for making dependently-typed programming any more pleasing. That is, using axioms goes in total opposition to the gradual typing criteria [Siek et al. 2015] when it comes to the smoothness of the static-to-dynamic checking spectrum: given a well-typed term, making it "less precise" by using axioms for some subterms actually results in programs that do not typecheck or reduce anymore.

Because CIC+ax amounts to working in CIC with an initial context extended with ax, this theory satisfies normalization ($\mathcal{N}$) as much as CIC, so conversion remains decidable. However, CIC+ax lacks a satisfying notion of safety because there is an *infinite* number of open canonical normal forms (more adequately called *stuck terms*) that inhabit any type A. For instance, in $\mathbb{B}$, we not only have the normal forms true, false, and $\mathsf{ax}_{\mathbb{B}}$, but an infinite number of terms stuck on eliminations of ax, such as match ax_{A} with ... or $\mathsf{ax}_{\mathbb{N}\to\mathbb{B}}\ 1$.

2.3 The Exceptional Approach

Pédrot and Tabareau [2018] present the exceptional type theory ExTT, demonstrating that it is possible to extend a type theory with a wildcard term while enjoying a satisfying notion of safety, which coincides with that of programming languages with exceptions.

ExTT is essentially CIC+err, that is, it extends CIC with an indexed error term $\mathsf{err}_{\mathsf{A}}$ that can inhabit any type A. But instead of being treated as a computational black box like ax_{A}, $\mathsf{err}_{\mathsf{A}}$ is endowed with computational content emulating exceptions in programming languages, which propagate instead of being stuck. For instance, in ExTT we have the following conversion:

$$\mathsf{match}\ \mathsf{err}_{\mathbb{B}}\ \mathsf{return}\ \mathbb{N}\ \mathsf{with}\ |\ \mathsf{true} \to 0\ |\ \mathsf{false} \to 1\ \mathsf{end} \quad \equiv \quad \mathsf{err}_{\mathbb{N}}$$

Notably, such exceptions are *call-by-name exceptions*, so one can only discriminate exceptions on positive types (*i.e.,* inductive types), not on negative types (*i.e.,* function types). In particular, in ExTT, $\mathsf{err}_{\mathsf{A}\to\mathsf{B}}$ and $\lambda\ \_\ :\ \mathsf{A} \Rightarrow \mathsf{err}_{\mathsf{B}}$ are convertible, and the latter is considered to be in normal form. So $\mathsf{err}_{\mathsf{A}}$ is a normal form of A only if A is a positive type.

ExTT has a number of interesting properties: it is normalizing ($\mathcal{N}$) and safe ($\mathcal{S}$), taking $\mathsf{err}_{\mathsf{A}}$ into account as usual in programming languages where exceptions are possible outcomes of computation: the normal forms of closed terms of a positive type (*e.g.,* $\mathbb{B}$) are either the constructors of that type (*e.g.,* true and false) or err at that type (*e.g.,* $\mathsf{err}_{\mathbb{B}}$). As a consequence, ExTT does not satisfy full canonicity, but it does satisfy a weaker form of it. In particular, ExTT enjoys (weak) logical consistency: any closed proof of False is convertible to $\mathsf{err}_{\mathsf{False}}$, which is discriminable at False. It has been shown that we can still reason soundly in an exceptional type theory, either using a parametricity requirement [Pédrot and Tabareau 2018], or more flexibly, using different universe hierarchies [Pédrot et al. 2019].

It is also important to highlight that this weak form of logical consistency is the *most* one can expect in a theory with effects. Indeed, Pédrot and Tabareau [2020] have shown that it is not possible to define a type theory with full dependent elimination that has observable effects (from which exceptions are a particular case) and at the same time validates traditional canonicity. Settling for less, as explained in §2.2 for the axiomatic approach, leads to an infinite number of stuck terms, even in the case of booleans, which is in opposition to the type safety criterion of gradual languages, which only accounts for runtime type errors.

Unfortunately, while ExTT solves the safety issue of the axiomatic approach, it still suffers from the same limitation as the axiomatic approach regarding type-level computation. Indeed, even though we can use $\mathtt{err}_{\mathsf{A}}$ to inhabit any type, we cannot use it in any meaningful way as a value at the type level. The term:

$$\mathtt{head}\ \mathbb{N}\ \mathtt{err}_{\mathbb{N}}\ (\mathtt{filter}\ \mathbb{N}\ 4\ \mathtt{even}\ [\ 0\ ;\ 1\ ;\ 2\ ;\ 3\ ])$$

does not typecheck, because $\mathtt{vec}\ \mathsf{A}\ \mathtt{err}_{\mathbb{N}}$ is still not convertible to $\mathtt{vec}\ \mathsf{A}\ (\mathsf{S}\ \mathtt{err}_{\mathbb{N}})$. The reason is that $\mathtt{err}_{\mathbb{N}}$ behaves like an extra constructor to $\mathbb{N}$, so $\mathsf{S}\ \mathtt{err}_{\mathbb{N}}$ is itself a normal form, and normal forms with different head constructors (S and $\mathtt{err}_{\mathbb{N}}$) are not convertible.

2.4 The Gradual Approach: Simple Types

Before going on with our exploration of the fundamental challenges in gradual dependent type theory, we review some key concepts and expected properties in the context of simple types [Garcia et al. 2016; New and Ahmed 2018; Siek et al. 2015].

Static semantics. Gradually-typed languages introduce the unknown type, written **?**, which is used to indicate the lack of static typing information [Siek and Taha 2006]. One can understand such an unknown type in terms of an *abstraction* of the set of possible types that it stands for [Garcia et al. 2016]. This interpretation provides a naive but natural understanding of the meaning of partially-specified types, for instance $\mathbb{B} \rightarrow ?$ denotes the set of all function types with $\mathbb{B}$ as domain. Given imprecise types, a gradual type system relaxes all type predicates and functions in order to optimistically account for occurrences of **?**. In a simple type system, the predicate on types is equality, whose relaxed counterpart is called *consistency*.[5] For instance, given a function `f` of type $\mathbb{B} \rightarrow ?$, the expression `(f true) + 1` is well-typed because `f` could *plausibly* return a number, given that its codomain is **?**, which is consistent with $\mathbb{N}$.

Note that there are other ways to consider imprecise types, for instance by restricting the unknown type to denote base types (in which case **?** would not be consistent with any function type), or to only allow imprecision in certain parts of the syntax of types, such as effects [Bañados Schwerter et al. 2016], security labels [Fennell and Thiemann 2013; Toro et al. 2018], annotations [Thiemann and Fennell 2014], or only at the top-level [Bierman et al. 2010]. Here, we do not consider these specialized approaches, which have benefits and challenges of their own, and stick to the mainstream setting of gradual typing in which the unknown type is consistent with any type and can occur anywhere in the syntax of types.

Dynamic semantics. Having optimistically relaxed typing based on consistency, a gradual language must detect inconsistencies at runtime if it is to satisfy safety ($\mathcal{S}$), which therefore has to be formulated in a way that encompasses runtime errors. For instance, if the function `f` above returns `false`, then an error must be raised to avoid reducing to `false + 1`—a closed stuck term, denoting a violation of safety. The traditional approach to do so is to avoid giving a direct reduction semantics to gradual programs, and instead, to elaborate them to an intermediate language with runtime casts, in which casts between inconsistent types raise errors [Siek and Taha 2006]. Alternatively—and equivalently from a semantics point of view—one can define the reduction of gradual programs directly on gradual typing derivations augmented with evidence about consistency judgments, and report errors when transitivity of such judgments is unjustified [Garcia et al. 2016]. There are many ways to realize each of these approaches, which vary in terms of efficiency and eagerness of checking [Bañados Schwerter et al. 2020; Herman et al. 2010; Siek et al. 2009; Siek and Wadler 2010; Tobin-Hochstadt and Felleisen 2008; Toro and Tanter 2020].

Conservativity. A first important property of a gradual language is that it is a *conservative extension* of a related static typing discipline: the gradual and static systems should coincide on

[5]Not to be confused with logical consistency!

static terms. This property is hereafter called **Conservativity** ($\mathcal{C}$), and parametrized with the considered static system. For instance, we write that GTLC satisfies $\mathcal{C}_{/\mathsf{STLC}}$. Technically, Siek and Taha [2006] prove that typing and reduction of GTLC and STLC coincide on their common set of terms (*i.e.,* terms that are fully precise). An important aspect of $\mathcal{C}$ is that the type formation rules and typing rules themselves are also preserved, modulo the presence of ? as a new type and the adequate lifting of predicates and functions [Garcia et al. 2016]. While this aspect is often left implicit, it ensures that the gradual type system does not behave in ad hoc ways on imprecise terms.

Note that, despite its many issues, CIC+ax (§2.2) satisfies $\mathcal{C}_{/\mathsf{CIC}}$: all pure (*i.e.,* axiom-free) CIC terms behave as they would in CIC. More precisely, two CIC terms are convertible in CIC+ax iff they are convertible in CIC. Importantly, this does not mean that CIC+ax is a conservative extension of CIC *as a logic*—which it clearly is not!

Gradual guarantees. The early accounts of gradual typing emphasized consistency as the central idea. However, Siek et al. [2015] observed that this characterization left too many possibilities for the impact of type information on program behavior, compared to what was originally intended [Siek and Taha 2006]. Consequently, Siek et al. [2015] brought forth *type precision* (denoted $\sqsubseteq$) as the key notion, from which consistency can be derived: two types A and B are consistent if and only if there exists T such that $\mathsf{T} \sqsubseteq \mathsf{A}$ and $\mathsf{T} \sqsubseteq \mathsf{B}$. The unknown type ? is the most imprecise type of all, *i.e.,* $\mathsf{T} \sqsubseteq ?$ for any T. Precision is a preorder that can be used to capture the intended *monotonicity* of the static-to-dynamic spectrum afforded by gradual typing. The static and dynamic *gradual guarantees* specify that typing and reduction should be *monotone with respect to precision*: losing precision should not introduce new static or dynamic errors. These properties require precision to be extended from types to terms. Siek et al. [2015] present a natural extension that is purely syntactic: a term is more precise than another if they are syntactically equal except for their type annotations, which can be more precise in the former.

The *static gradual guarantee* (SGG) ensures that imprecision does not break typeability:

Definition 1 (SGG). *If* $\mathsf{t} \sqsubseteq \mathsf{u}$ *and* $\mathsf{t} : \mathsf{T}$, *then* $\mathsf{u} : \mathsf{U}$ *for some* U *such that* $\mathsf{T} \sqsubseteq \mathsf{U}$.

The SGG captures the intuition that "sprinkling ? over a term" maintains its typeability. As such, the notion of precision $\sqsubseteq$ used to formulate the SGG is inherently syntactic, over as-yet-untyped terms: typeability is the *consequence* of the SGG theorem.

The *dynamic gradual guarantee* (DGG) is the key result that bridges the syntactic notion of precision to reduction: if $\mathsf{t} \sqsubseteq \mathsf{u}$ and t reduces to some value v, then u reduces to some value $\mathsf{v'}$ such that $\mathsf{v} \sqsubseteq \mathsf{v'}$; and if t diverges, then so does u. This property entails that $\mathsf{t} \sqsubseteq \mathsf{u}$ means that t may error more than u, but otherwise they should behave the same. Instead of the original formulation of the DGG by Siek et al. [2015], New and Ahmed [2018] appeal to the semantic notion of *observational error-approximation* to capture the relation between two terms that are contextually equivalent except that the left-hand side term may fail more:[6]

Definition 2 (Observational error-approximation). *A term* $\Gamma \vdash t : A$ *observationally error-approximates a term* $\Gamma \vdash u : A$, *noted* $t \preccurlyeq^{obs} u$, *if for all boolean-valued observation contexts* $\mathcal{C} : (\Gamma \vdash A) \Rightarrow (\vdash \mathbb{B})$ *closing over all free variables, either*

- $\mathcal{C}[t]$ *and* $\mathcal{C}[u]$ *both diverge.*
- *Otherwise if* $\mathcal{C}[u] \rightsquigarrow^* \mathsf{err}_{\mathbb{B}}$, *then* $\mathcal{C}[t] \rightsquigarrow^* \mathsf{err}_{\mathbb{B}}$.

Using this semantic notion, the DGG simply states that term precision implies observational error-approximation:

Definition 3 (DGG). *If* $\mathsf{t} \sqsubseteq \mathsf{u}$ *then* $\mathsf{t} \preccurlyeq^{obs} \mathsf{u}$.

[6]Observational error-approximation does not mention the case where $\mathcal{C}[t]$ reduces to `true` or `false` but the quantification over all contexts ensures that, in that case, $\mathcal{C}[u]$ must reduce to the same value.

While often implicit, it is important to highlight that the DGG is relative to both the notion of precision $\sqsubseteq$ and the notion of observations $\preccurlyeq^{obs}$. Indeed, it is possible to study alternative notions of precisions beyond the natural definition stated by Siek et al. [2015]. For instance, following the Abstracting Gradual Typing methodology [Garcia et al. 2016], precision follows from the definition of gradual types as a concretization to sets of static types. This opens the door to justifying alternative precisions, *e.g.,* by considering that the unknown type only stands for specific static types, such as base types. Additionally, variants of precision have been studied in more challenging typing disciplines where the natural definition seems incompatible with the DGG, see *e.g.,* [Igarashi et al. 2017]. As we will soon see below, it can also be necessary in certain situations to consider another notion of observations.

Graduality. As we have seen, the DGG is relative to a notion of precision, but what should this relation be? To go beyond a syntactic axiomatic definition of precision, New and Ahmed [2018] characterize the good dynamic behavior of precision: the runtime checking mechanism used to define a gradual language, such as casting, should only perform typechecking, and not otherwise affect behavior. Specifically, they mandate that precision gives rise to *embedding-projection pairs* (ep-pairs): the cast induced by two types related by precision forms an adjunction, which induces a retraction. In particular, going to a less precise type and back is the identity: for any term a of type A, and given $\mathsf{A} \sqsubseteq \mathsf{B}$, then $\mathsf{a} :: \mathsf{B} :: \mathsf{A}$ should be observationally equivalent to a (recall from Footnote 4 that $::$ is a type ascription). For instance, $1 :: \mathbf{?} :: \mathbb{N}$ should be equivalent to 1. Dually, when gaining precision, there is the potential for errors: given a term b of type B, $\mathsf{b} :: \mathsf{A} :: \mathsf{B}$ may fail. By considering error as the least precise term, this can be stated as $\mathsf{b} :: \mathsf{A} :: \mathsf{B} \sqsubseteq \mathsf{b}$. For instance, with the imprecise successor function $\mathsf{f} := \lambda\, \mathsf{n}{:}\mathbf{?} \Rightarrow (\mathsf{S}\ \mathsf{n}) :: \mathbf{?}$ of type $\mathbf{?}{\to}\mathbf{?}$, we have $\mathsf{f} :: \mathbb{N}{\to}\mathbb{B} :: \mathbf{?}{\to}\mathbf{?} \sqsubseteq \mathsf{f}$, because the ascribed function will fail when applied.

Technically, the adjunction part states that if we have $\mathsf{A} \sqsubseteq \mathsf{B}$, a term a of type A, and a term b of type B, then $\mathsf{a} \sqsubseteq \mathsf{b} :: \mathsf{A} \Leftrightarrow \mathsf{a} :: \mathsf{B} \sqsubseteq \mathsf{b}$. The retraction part further states that t is not only more precise than $\mathsf{t} :: \mathsf{B} :: \mathsf{A}$ (which is given by the unit of the adjunction) but is *equi-precise* to it, noted $\mathsf{t} \sqsupseteq\sqsubseteq \mathsf{t} :: \mathsf{B} :: \mathsf{A}$. Because the DGG dictates that precision implies observational error-approximation, equi-precision implies observational equivalence, and so losing and recovering precision must produce a term that is observationally equivalent to the original one.

A couple of additional observations need to be made here, as they will play a major role in the development of this article:

- These two approaches to characterizing gradual typing highlight the need to distinguish syntactic from semantic notions of precision. Indeed, with the usual *syntactic* precision from Siek et al. [2015], one cannot derive the ep-pair property, in particular the equi-precision stated above. This is why New and Ahmed [2018] introduce a *semantic* precision, defined on well-typed terms. This semantic precision serves as a proxy between the syntactic precision and the desired observational error-approximation.
- A type-based semantic precision cannot be used for the SGG. Indeed, this theorem (not addressed by New and Ahmed [2018]) requires a syntactic notion of precision that *predates* typing: well-typedness of the less precise term is the *consequence* of the theorem. Therefore a full study of a gradual language that covers SGG, DGG, and embedding-projection pairs needs to consider both syntactic and semantic notions of precision.
- The embedding-projection property does not *per se* imply the DGG: one could pick precision to be the universal relation, which trivially induces ep-pairs, but does not imply observational error-approximation. It appears that, in the simply-typed setting considered in prior work, the DGG implies the embedding-projection property. In fact, New and Ahmed [2018] essentially advocate ep-pairs as an elegant and compositional proof technique to establish

the DGG. But as we uncover later in this article, it turns out that in certain settings—and in particular dependent types—the embedding-projection property imposes *more* desirable constraints on the behavior of casts than the DGG alone.

In this paper, we use the term **Graduality** ($\mathcal{G}$) for the DGG established with respect to a notion of precision that also induces embedding-projection pairs.

2.5 The Gradual Approach: Dependent Types

Extending the gradual approach to a setting with full dependent types requires reconsidering several aspects.

Newcomers: the unknown term and the error type. In the simply-typed setting, there is a clear stratification: ? is at the type level, `err` is at the term level. Likewise, *type precision*, with ? as greatest element, is separate from *term precision*, with `err` as least element. In the absence of a type/term syntactic distinction as in CIC, this stratification is untenable:

- Because types permeate terms, ? is no longer only the unknown type, but it also acts as the "unknown term". In particular, this makes it possible to consider unknown indices for types, as in Example 1. More precisely, there is a family of unknown terms $?_{\mathsf{A}}$, indexed by their type A. The traditional unknown type is just $?_{\square}$, the unknown of the universe $\square$.
- Dually, because terms permeate types, we also have the "error type", $\mathtt{err}_{\square}$. We have to deal with errors in types.
- Precision must be unified as a single preorder, with ? at the top and `err` at the bottom. The most imprecise term of all is $?_{?_{\square}}$ (? for short)—more exactly, there is one such term per type universe. At the bottom, $\mathtt{err}_{\mathsf{A}}$ is the most precise term of type A.

Revisiting safety. The notion of closed canonical forms used to characterize legitimate normal forms via safety ($\mathcal{S}$) needs to be extended not only with errors as in the simply-typed setting, but also with unknown terms. Indeed, as there is an unknown term $?_{\mathsf{A}}$ inhabiting any type A, we have one new canonical form for each type A. In particular, $?_{\mathbb{B}}$ cannot possibly reduce to either `true` or `false` or $\mathtt{err}_{\mathbb{B}}$, because doing so would collapse the precision order. Therefore, $?_{\mathsf{A}}$ should propagate computationally, like $\mathtt{err}_{\mathsf{A}}$ (§ 2.3).

The difference between errors and unknown terms is rather on their static interpretation. In essence, the unknown term $?_{\mathsf{A}}$ is a dual form of exceptions: it propagates, but is optimistically comparable, *i.e.,* consistent with, any other term of type A. Conversely, $\mathtt{err}_{\mathsf{A}}$ should not be consistent with any term of type A. Going back to the issues we identified with the axiomatic (§ 2.2) and exceptional (§ 2.3) approaches when dealing with type-level computation, the term:

$$\mathsf{head}\ \mathbb{N}\ ?_{\mathbb{N}}\ (\mathtt{filter}\ \mathbb{N}\ 4\ \mathsf{even}\ [\ 0\ ;\ 1\ ;\ 2\ ;\ 3\ ])$$

now typechecks: $\mathsf{vec}\ \mathsf{A}\ ?_{\mathbb{N}}$ can be deemed consistent with $\mathsf{vec}\ \mathsf{A}\ (\mathsf{S}\ ?_{\mathbb{N}})$, because $\mathsf{S}\ ?_{\mathbb{N}}$ is consistent with $?_{\mathbb{N}}$. This newly-brought flexibility is the key to support the different scenarios from the introduction. So let us now turn to the question of how to integrate consistency in a dependently-typed setting.

Relaxing conversion. In the simply-typed setting, consistency is a relaxing of syntactic type equality to account for imprecision. In a dependent type theory, there is a more powerful notion than syntactic equality to compare types, namely *conversion* (§ 2.1): if $\mathsf{t}:\mathsf{T}$ and $\mathsf{T} \rightsquigarrow \mathsf{U}$, then $\mathsf{t}:\mathsf{U}$. For instance, a term of type T can be used as a function as soon as T is *convertible* to the type `forall` (a:A), B for some types A and B. The proper notion to relax in the gradual dependently-typed setting is therefore conversion, not syntactic equality.

Garcia et al. [2016] give a general framework for gradual typing that explains how to relax any static type predicate to account for imprecision: for a binary type predicate P, its consistent lifting Q(A,B) holds iff there exist static types A' and B' in the denotation (*concretization* in abstract interpretation parlance) of A and B, respectively, such that P(A',B'). As observed by Castagna et al. [2019], when applied to equality, this defines consistency as a unification problem. Therefore, the consistent lifting of conversion ought to be that two terms t and u are consistently convertible iff they denote some static terms t' and u' such that $t' \rightsquigarrow u'$. This property is essentially higher-order unification, which is undecidable.

It is therefore necessary to adopt some approximation of consistent conversion (hereafter called consistency for short) in order to be able to implement a gradual dependent type theory. And there lies a great challenge: because of the absence of stratification between typing and reduction, the static gradual guarantee (SGG) already demands monotonicity for conversion, a demand very close to that of the DGG.[7]

Dealing with neutrals. Prior work on gradual typing usually only considers reduction on closed terms in order to establish results about the dynamics, such as the DGG. But in dependent type theory, conversion must operate on *open* terms, yielding *neutral* terms such as $1 :: X :: \mathbb{N}$ where X is a type variable, or x+1 where x is of type $\mathbb{N}$ or $?_\square$. Such neutral terms cannot reduce further, and can occur in both terms and types. Depending on the upcoming substitutions, neutrals can fail or not. For instance, in $1 :: X :: \mathbb{N}$, if $?_\square$ is substituted for X, the term reduces to 1, but fails if $\mathbb{B}$ is substituted instead.

Importantly, less precise variants of neutrals can reduce *more*. For instance, both $1 :: ?_\square :: \mathbb{N}$ and $?_\mathbb{N}+1$ are less precise than the neutrals above, but do evaluate further (typically, to 1 and to $?_\mathbb{N}$, respectively). This interaction between neutrals, reduction, and precision spices up the goal of establishing DGG and $\mathcal{G}$. In particular, this re-enforces the need to consider semantic precision, because a syntactic precision is likely not to be stable by reduction: $1 :: X :: \mathbb{N} \sqsubseteq 1 :: ? :: \mathbb{N}$ is obvious syntactically, but $1 :: X :: \mathbb{N} \sqsubseteq 1$ is not.

DGG vs Graduality. In a dependently-typed setting, it is possible to satisfy the DGG while not satisfying the embedding-projection pairs requirement of $\mathcal{G}$. To see why, consider a system in which any term of type A that is not fully-precise immediately reduces to $?_A$. This system would satisfy $\mathcal{C}$, $\mathcal{S}$, $\mathcal{N}$, and …the DGG. Recall that the DGG only requires reduction to be monotone with respect to precision, so using the most imprecise term $?_A$ as a universal redux is surely valid. This collapse of the DGG is impossible in the simply-typed setting because there is no unknown term: it is only possible when $?_A$ exists *as a term*. It is therefore possible to satisfy the DGG while being useless when *computing* with imprecise terms. Conversely, the degenerate system breaks the embedding-projection requirement of graduality stated by New and Ahmed [2018]. For instance, $1 :: ?_\square :: \mathbb{N}$ would be convertible to $?_\mathbb{N}$, which is *not* observationally equivalent to 1. Therefore, the embedding-projection requirement of graduality goes beyond the DGG in a way that is critical in a dependent type theory, where it captures both the smoothness of the static-to-dynamic checking spectrum, and the proper computational content of valid uses of imprecision.

Observational refinement. Let us come back to the notion of observational error-approximation used in the simply-typed setting to state the DGG. New and Ahmed [2018] justify this notion because in "gradual typing we are not particularly interested in when one program diverges more than another, but rather when it produces more type errors." This point of view is adequate in the simply-typed setting because the addition of casts may only produce more type errors; in particular,

[7]In a dependently-typed programming language with separate typing and execution phases, this demand of the SGG is called the *normalization gradual guarantee* by Eremondi et al. [2019].

adding casts can never lead to divergence when the original term does not diverge itself. Therefore, in that setting, the definition of error-approximation includes equi-divergence. The situation in the dependent setting is however more complicated, if the theory admits divergence. There exist non-gradual dependently-typed programming languages that admit divergence (*e.g.,* Dependent Haskell [Eisenberg 2016], Idris [Brady 2013]); we will also present one such theory in this article.

In a gradual dependent type theory that admits divergence, a *diverging term* is more precise than the *unknown term* ?. Because the unknown term in itself does not diverge, this breaks the left-to-right implication of equi-divergence. Note that this argument does not rely on any specific definition of precision, just on the fact that the unknown term is the most imprecise term (at its type). Additionally, an error at a *diverging type* X may be ascribed to $?_\square$ then back to X. Evaluating this roundtrip requires evaluating X itself, which makes the less precise term diverge. This breaks the right-to-left implication of equi-divergence.

To summarize, the way to understand these counterexamples is that in a dependent and non-terminating setting, the motto of graduality ought to be adjusted: more precise programs produce more type errors *or diverge more*. This leads to the following definition of *observational refinement*.

Definition 4 (Observational refinement). *A term $\Gamma \vdash t : A$ observationally refines a term $\Gamma \vdash u : A$, noted $t \sqsubseteq^{obs} u$ if for all boolean-valued observation context $C : (\Gamma \vdash A) \Rightarrow (\vdash \mathbb{B})$ closing over all free variables, if $C[u] \rightsquigarrow^* \mathtt{err}_{\mathbb{B}}$ or diverges, then either $C[t] \rightsquigarrow^* \mathtt{err}_{\mathbb{B}}$ or $C[t]$ diverges.*

In this definition, errors and divergence are collapsed. Thus, in a gradual dependent theory that admits divergence, equi-refinement does not imply observational equivalence, because one term might diverge while the other reduces to an error. Of course, if the gradual dependent theory is strongly normalizing, then both notions $\preccurlyeq^{obs}$ (Definition 2) and $\sqsubseteq^{obs}$ (Definition 4) coincide.

2.6 The Fire Triangle of Graduality

To sum up, we have seen four important properties that can be expected from a gradual type theory: safety ($\mathcal{S}$), conservativity with respect to a theory X ($\mathcal{C}_{/X}$), graduality ($\mathcal{G}$), and normalization ($\mathcal{N}$). Any type theory ought to satisfy at least $\mathcal{S}$. Unfortunately, we now show that mixing the three other properties $\mathcal{C}$, $\mathcal{G}$ and $\mathcal{N}$ is impossible for STLC, as well as for CIC.

Preliminary: regular reduction. To derive this general impossibility result, by relying only on the properties and without committing to a specific language or theory, we need to assume that the reduction system used to decide conversion is regular, in that it only looks at the weak head normal form of subterms for reduction rules, and does not magically shortcut reduction, for instance based on the specific syntax of inner terms. As an example, β-reduction is not allowed to look into the body of the lambda term to decide how to proceed.

This property is satisfied in all actual systems we know of, but formally stating it in full generality, in particular without devoting to a particular syntax, is beyond the scope of this paper. Fortunately, in the following, we need only rely on a much weaker hypothesis, which is a slight strengthening of the retraction hypothesis of $\mathcal{G}$. Recall that retraction says that when $\mathsf{A} \sqsubseteq \mathsf{B}$, any term $\mathtt{t}$ of type A is equi-precise to $\mathtt{t} :: \mathsf{B} :: \mathsf{A}$. We additionally require that for any context C, if $\mathsf{C}[\mathtt{t}]$ reduces at least k steps, then $\mathsf{C}[\mathtt{t} :: \mathsf{B} :: \mathsf{A}]$ also reduces at least k steps. Intuitively, this means that the reduction of $\mathsf{C}[\mathtt{t} :: \mathsf{B} :: \mathsf{A}]$, while free to decide when to get rid of the embedding-to-B-projection-to-A, cannot use it to avoid reducing $\mathtt{t}$. This property is true in all gradual languages, where type information at runtime is used only as a monitor.

Gradualizing STLC. Let us first consider the case of STLC. We show that Ω is *necessarily* a well-typed diverging term in any gradualization of STLC that satisfies the other properties.

Theorem 5 (Fire Triangle of Graduality for STLC). *Suppose a gradual type theory that satisfies properties $\mathcal{C}_{/\mathsf{STLC}}$ and $\mathcal{G}$. Then $\mathcal{N}$ cannot hold.*

Proof. We pose $\Omega := \delta\ (\delta :: ?)$ with $\delta := \lambda\, x : ?.\ (x :: ? \to ?)\ x$ and show that it must necessarily be a well-typed diverging term. Because the unknown type $?$ is consistent with any type (§2.4) and $? \to ?$ is a valid type (by $\mathcal{C}_{/\mathsf{STLC}}$), the self-applications in Ω are well-typed, δ has type $? \to ?$, and Ω has type $?$. Now, we remark that $\Omega = C[\delta]$ with $C[\cdot] = [\cdot]\ (\delta :: ?)$.

We show by induction on k that Ω reduces at least k steps, the initial case being trivial. Suppose that Ω reduces at least k steps. By maximality of $?$ with respect to precision, we have that $? \to ? \sqsubseteq ?$, so we can apply the strengthening of $\mathcal{G}$ applied to δ, which tells us that $C[\delta :: ? :: ? \to ?]$ reduces at least k steps because $C[\delta]$ reduces at least k steps. But by β-reduction, we have that Ω reduces in one step to $C[\delta :: ? :: ? \to ?]$. So Ω reduces at least $k + 1$ steps.

This means that Ω diverges, which is a violation of $\mathcal{N}$. □

This result could be extended to all terms of the untyped lambda calculus, not only Ω, in order to obtain the embedding theorem of GTLC [Siek et al. 2015]. Therefore, the embedding theorem is not an independent property, but rather a consequence of $\mathcal{C}$ and $\mathcal{G}$—that is why we have not included it as such in our overview of the gradual approach (§2.4).

Gradualizing CIC. We can now prove the same impossibility theorem for CIC, by reducing it to the case of STLC. Therefore this theorem can be proven for type theories others than CIC, as soon as they faithfully embed STLC.

Theorem 6 (Fire Triangle of Graduality for CIC). *A gradual dependent type theory cannot simultaneously satisfy properties $\mathcal{C}_{/\mathsf{CIC}}$, $\mathcal{G}$ and $\mathcal{N}$.*

Proof. We show that a gradual dependent type theory satisfying $\mathcal{C}_{/\mathsf{CIC}}$ and $\mathcal{G}$ must contain a diverging term, thus contravening $\mathcal{N}$. The typing rules of CIC contain the typing rules of STLC, using only one universe $\square_0$, where the function type is interpreted using the dependent product and the notions of reduction coincide, so CIC embeds STLC; a well-known result on PTS [Barendregt 1991]. This means that $\mathcal{C}_{/\mathsf{CIC}}$ implies $\mathcal{C}_{/\mathsf{STLC}}$. Additionally, $\mathcal{G}$ can be specialized to the simply-typed fragment of the theory, by setting the unknown type $?$ to be $?_{\square_0}$. Therefore, we can apply Theorem 5 and we get a well-typed term that diverges, finishing the proof. □

The Fire Triangle in practice. In non-dependent settings, all gradual languages where $?$ is universal admit non-termination and therefore compromise $\mathcal{N}$. Garcia and Tanter [2020] discuss the possibility to gradualize STLC without admitting non-termination, for instance by considering that $?$ is not universal and denotes only base types (in such a system, $? \to ? \not\sqsubseteq ?$, so the argument with Ω is invalid). Without sacrificing the universal unknown type, one could design a variant of GTLC that uses some mechanism to detect divergence, such as termination contracts [Nguyen et al. 2019]. This would yield a language that certainly satisfies $\mathcal{N}$, but it would break $\mathcal{G}$. Indeed, because the contract system is necessarily over-approximating in order to be sound (and actually imply $\mathcal{N}$), there are effectively-terminating programs with imprecise variants that yield termination contract errors.

To date, the only related work that considers the gradualization of full dependent types with $?$ as both a term and a type, is the work on GDTL [Eremondi et al. 2019]. GDTL is a programming language with a clear separation between the typing and execution phases, like Idris [Brady 2013]. GDTL adopts a different strategy in each phase: for typing, it uses Approximate Normalization (AN), which always produces $?_A$ as a result of going through imprecision and back. This means that conversion is both total and decidable (satisfies $\mathcal{N}$), but it breaks $\mathcal{G}$ for the same reason as the degenerate system we discussed in §2.5 (notice that the example uses a gain of precision from the unknown type to $\mathbb{N}$, so the example behaves just the same with AN). In such a phased setting, the lack of computational content of AN is not critical, because it only means that typing becomes overly optimistic. To execute programs, GDTL relies on standard GTLC-like reduction semantics, which is computationally precise, but does not satisfy $\mathcal{N}$.

3 GCIC: OVERALL APPROACH, MAIN CHALLENGES AND RESULTS

Given the Fire Triangle of Graduality (Theorem 6), we know that gradualizing CIC implies making some compromise. Instead of focusing on one possible compromise, this work develops three novel solutions, each compromising one specific property ($\mathcal{N}$, $\mathcal{G}$, or $\mathcal{C}_{/\mathrm{CIC}}$), and does so in a common parametrized framework, GCIC.

This section gives an informal, non-technical overview of our approach to gradualizing CIC, highlighting the main challenges and results. As such, it serves as a gentle roadmap to the following sections, which are rather dense and technical.

3.1 GCIC: 3-in-1

To explore the spectrum of possibilities enabled by the Fire Triangle of Graduality, we develop a general approach to gradualizing CIC, and use it to define three theories, corresponding to different resolutions of the triangular tension between normalization ($\mathcal{N}$), graduality ($\mathcal{G}$) and conservativity with respect to CIC ($\mathcal{C}_{/\mathrm{CIC}}$).

The crux of our approach is to recognize that, while there is not much to vary within STLC itself to address the tension of the Fire Triangle of Graduality, there are several variants of CIC that can be considered by changing the hierarchy of universes and its impact on typing—after all, CIC is but a particular Pure Type System (PTS) [Barendregt 1991].

In particular, we consider a parametrized version of a gradual CIC, called GCIC, with two parameters (Fig. 3):

- The first parameter characterizes how the universe level of a Π type is determined in typing rules: either as taking the *maximum* of the levels of the involved types, as in standard CIC, or as the *successor* of that maximum. The latter option yields a variant of CIC that we call $\mathrm{CIC}^{\uparrow}$ (read "CIC-shift"). $\mathrm{CIC}^{\uparrow}$ is a subset of CIC, with a stricter constraint on universe levels. In particular $\mathrm{CIC}^{\uparrow}$ loses the closure of universes under dependent product that CIC enjoys. As a consequence, some well-typed CIC terms are not well-typed in $\mathrm{CIC}^{\uparrow}$.[8]
- The second parameter is the dynamic counterpart of the first parameter: its role is to enforce that universe levels are coherent through type casts during the reduction of casts. Note that we only allow this reduction parameter to be loose (*i.e.,* using maximum) if the typing parameter is also loose. Indeed, letting the typing parameter be strict (*i.e.,* using successor) while the reduction parameter is loose breaks subject reduction, and hence $\mathcal{S}$.

Based on these parameters, this work develops the following three variants of GCIC, whose properties are summarized in Table 1 with pointers to the respective theorems—because GCIC is one common parametrized framework, we are able to establish most properties for all variants at once:

(1) $\mathrm{GCIC}^{\mathcal{G}}$**: a theory that satisfies both $\mathcal{C}_{/\mathrm{CIC}}$ and $\mathcal{G}$, but sacrifices $\mathcal{N}$.** This theory is a rather direct application of the principles discussed in §2 by extending CIC with errors and unknown terms, and changing conversion with consistency. This results in a theory that is not normalizing.

[8] A minimal example of a well-typed CIC term that is ill typed in $\mathrm{CIC}^{\uparrow}$ is $\mathtt{narrow} : \mathbb{N} \to \square$, where $\mathtt{narrow}\ \mathtt{n}$ is the type of functions that accept $\mathtt{n}$ arguments. Such dependent arities violate the universe constraint of $\mathrm{CIC}^{\uparrow}$.

	$\mathcal{S}$	$\mathcal{N}$	$\mathcal{C}_{/X}$	$\mathcal{G}$	SGG	DGG
$\mathsf{GCIC}^{\mathcal{G}}$	✓(Th. 8)	✗	CIC (Th. 23)	✓(Th. 34)	✓(Th. 24)	✓(Th. 25)
$\mathsf{GCIC}^{\uparrow}$	✓(idem)	✓(Th. 9 & 26)	$\mathsf{CIC}^{\uparrow}$ (idem)	✓(Th. 32)	✓(idem)	✓(Th. 25)
$\mathsf{GCIC}^{\mathcal{N}}$	✓(idem)	✓(idem)	CIC (idem)	✗	✗	✗

$\mathcal{S}$: safety, $\mathcal{N}$: normalization, $\mathcal{C}_{/X}$: conservativity wrt theory X, $\mathcal{G}$: graduality (DGG + ep-pairs), SGG: static gradual guarantee, DGG: dynamic gradual guarantee

Table 1. GCIC variants and their properties

(2) $\mathsf{GCIC}^{\uparrow}$**: a theory that satisfies both $\mathcal{N}$ and $\mathcal{G}$, and supports $\mathcal{C}$ with respect to** $\mathsf{CIC}^{\uparrow}$**.** This theory uses the universe hierarchy at the *typing level* to detect the potential non-termination induced by the use of consistency instead of conversion. This theory simultaneously satisfies $\mathcal{G}$, $\mathcal{N}$ and $\mathcal{C}_{/\mathsf{CIC}^{\uparrow}}$.
(3) $\mathsf{GCIC}^{\mathcal{N}}$**: a theory that satisfies both $\mathcal{C}_{/\mathsf{CIC}}$ and $\mathcal{N}$, but does not fully validate $\mathcal{G}$.** This theory uses the universe hierarchy at the *computational level* to detect potential divergence. Such runtime check failures invalidate the DGG for some terms, and hence $\mathcal{G}$, as well as the SGG.

Practical implications of GCIC *variants.* Regarding the examples from § 1, all three variants of GCIC support the exploration of the type-level precision spectrum for the functions described in Examples 1, 3 and 4. In particular, we can define `filter` by giving it the imprecise type `forall A n (f : A` $\to \mathbb{B}$), `vec A n` $\to$ `vec A` $?_{\mathbb{N}}$ in order to bypass the difficulty of precisely characterizing the size of the output vector. Any invalid optimistic assumption is detected during reduction and reported as an error.

Unsurprisingly, the semantic differences between the three GCIC variants crisply manifest in the treatment of potential non-termination (Example 2), more specifically, *self application*. Let us come back to the term Ω used in the proof of Theorem 6. In all three variants, this term is well-typed. In $\mathsf{GCIC}^{\mathcal{G}}$, it reduces forever, as it would in the untyped lambda calculus. In that sense, $\mathsf{GCIC}^{\mathcal{G}}$ can embed the untyped lambda calculus just as GTLC [Siek et al. 2015]. In $\mathsf{GCIC}^{\mathcal{N}}$, this term fails at runtime because of the strict universe check in the reduction of casts, which breaks graduality because $?_{\square_i} \to ?_{\square_i} \sqsubseteq ?_{\square_i}$ tells us that the upcast-downcast coming from an ep-pair should not fail. A description of the reductions in $\mathsf{GCIC}^{\mathcal{G}}$ and in $\mathsf{GCIC}^{\mathcal{N}}$ is given in full details in §5.3. In $\mathsf{GCIC}^{\uparrow}$, Ω fails in the same way as in $\mathsf{GCIC}^{\mathcal{N}}$, but this does not break graduality because of the shifted universe level on Π types. A consequence of this stricter typing rule is that in $\mathsf{GCIC}^{\uparrow}$, $?_{\square_i} \to ?_{\square_i} \sqsubseteq ?_{\square_j}$ for any $j > i$, but $?_{\square_i} \to ?_{\square_i} \not\sqsubseteq ?_{\square_i}$. Therefore, the casts performed in Ω do not come from an ep-pair anymore and can legitimately fail.

Another scenario where the differences in semantics manifest is functions with *dependent arities*. For instance, the well-known C function `printf` can be embedded in a well-typed fashion in CIC: it takes as first argument a format string and computes from it both the type and *number* of later arguments. This function brings out the limitation of $\mathsf{GCIC}^{\uparrow}$: since the format string can specify an arbitrary number of arguments, we need as many $\to$, and `printf` cannot typecheck in a theory where universes are not closed under function spaces. In $\mathsf{GCIC}^{\mathcal{N}}$, `printf` typechecks but the same problem will appear dynamically when casting `printf` to ? and back to its original type: the result will be a function that works only on format strings specifying no more arguments than the universe level at which it has been typechecked. Note that this constitutes an example of violation of graduality for $\mathsf{GCIC}^{\mathcal{N}}$, even of the dynamic gradual guarantee. Finally, in $\mathsf{GCIC}^{\mathcal{G}}$ the function can be gradualized as much as one wants, without surprises.

Which variant to pick? As explained in the introduction, the aim of this paper is to shed light on the design space of gradual dependent type theories, not to advocate for one specific design. We believe the appropriate choice depends on the specific goals of the language designer, or perhaps more pertinently, on the specific goals of a given project, at a specific point in time.

The key characteristics of each variant are:

- $\mathsf{GCIC}^{\mathcal{G}}$ favors flexibility over decidability of type-checking. While this might appear heretical in the context of proof assistants, this choice has been embraced by practical languages such as Dependent Haskell [Eisenberg 2016], a dependently-typed Haskell where both divergence and runtime errors can happen at the type level. The pragmatic argument is simplicity: by letting programmers be responsible, there is no need for termination checking techniques and other restrictions.
- $\mathsf{GCIC}^{\uparrow}$ is theoretically pleasing as it enjoys both normalization and graduality. In practice, though, the fact that it is not conservative wrt full CIC means that one would not be able to simply import existing libraries as soon as they fall outside of the $\mathsf{CIC}^{\uparrow}$ subset. In $\mathsf{GCIC}^{\uparrow}$, the introduction of **?** should be done with an appropriate understanding of universe levels. This might not be a problem for advanced programmers, but would surely be harder to grasp for beginners.
- $\mathsf{GCIC}^{\mathcal{N}}$ is normalizing and able to import existing libraries without restrictions, at the expense of some surprises on the graduality front. Programmers would have to be willing to accept that they cannot just sprinkle **?** as they see fit without further consideration, as any dangerous usage of imprecision will be flagged during conversion.

In the same way that systems like Coq, Agda or Idris support different ways to customize their semantics (such as allowing `Type-in-Type`, switching off termination checking, using the `partial`/`total` compiler flags)—and of course, many programming languages implementations supporting some sort of customization, GHC being a salient representative—one can imagine a flexible realization of GCIC that give users the control over the two parameters we identify in this work, and therefore have access to all three GCIC variants. Considering the inherent tension captured by the Fire Triangle of Graduality, such a pragmatic approach might be the most judicious choice, making it possible to gather experience and empirical evidence about the pros and cons of each in a variety of concrete scenarios.

3.2 Typing, Cast Insertion, and Conversion

As explained in §2.4, in a gradual language, whenever we reclaim precision, we might be wrong and need to fail in order to preserve safety ($\mathcal{S}$). In a simply-typed setting, the standard approach is to define typing on the gradual source language, and then to translate terms via a type-directed cast insertion to a target cast calculus, *i.e.,* a language with explicit runtime type checks, needed for a well-behaved reduction [Siek and Taha 2006] . For instance, in a call-by-value language, the upcast (loss of precision) $\langle ? \Leftarrow \mathbb{N} \rangle\ 10$ is considered a (tagged) value, and the downcast (gain of precision) $\langle \mathbb{N} \Leftarrow ? \rangle\ v$ reduces successfully if v is such a tagged natural number, or to an error otherwise.

We follow a similar approach for GCIC, which is elaborated in a type-directed manner to a second calculus, named CastCIC (§5.1). The interplay between typing and cast insertion is however more subtle in the context of a dependent type theory. Because typing needs computation, and reduction is only meaningful in the target language, CastCIC is used *as part of the typed elaboration* in order to compare types (§5.2). This means that GCIC has no typing on its own, independent of its elaboration to the cast calculus.[9]

[9]This is similar to what happens in practice in proof assistants such as Coq [The Coq Development Team 2020, Core language], where terms input by the user in the Gallina language are first elaborated in order to add implicit arguments,

In order to satisfy conservativity with respect to CIC ($\mathcal{C}_{/\mathrm{CIC}}$), ascriptions in GCIC are required to satisfy consistency: for instance, $\mathtt{true} :: ? :: \mathbb{N}$ is well-typed by consistency (twice), but $\mathtt{true} :: \mathbb{N}$ is ill typed. Such ascriptions in CastCIC are realized by casts. For instance $0 :: ? :: \mathbb{B}$ in GCIC elaborates (modulo sugar and reduction) to $\langle \mathbb{B} \Leftarrow ?_{\square} \rangle \langle ?_{\square} \Leftarrow \mathbb{N} \rangle\, 0$ in CastCIC. A major difference between ascriptions in GCIC and casts in CastCIC is that casts are not required to satisfy consistency: a cast between any two types is well-typed, although of course it might produce an error.

Finally, standard presentations of CIC use a standalone conversion rule, as usual in declarative presentations of type systems. To gradualize CIC, we have to move to a more algorithmic presentation in order to forbid transitivity, otherwise all terms would be well-typed by way of a transitive step through ?. But $\mathcal{C}_{/\mathrm{CIC}}$ demands that only terms with explicitly-ascribed imprecision enjoy its flexibility. This observation is standard in the gradual typing literature [Garcia et al. 2016; Siek and Taha 2006, 2007]. As in prior work on gradual dependent types [Eremondi et al. 2019], we adopt a bidirectional presentation of typing for CIC (§4), which allows us to avoid accidental transitivity and directly derive a deterministic typing algorithm for GCIC.

3.3 Realizing a Dependent Cast Calculus: CastCIC

To inform the design and justify the reduction rules provided for CastCIC, we build a syntactic model of CastCIC by translation to CIC augmented with induction-recursion [Dybjer and Setzer 2003; Ghani et al. 2015; Martin-Löf 1996] (§6.1). From a type theory point of view, what makes CastCIC peculiar is first of all the possibility of having *errors* (both “pessimistic” as `err` and “optimistic” as ?), and the necessity to do *intensional type analysis* in order to resolve casts. For the former, we build upon the work of Pédrot and Tabareau [2018] on the exceptional type theory ExTT. For the latter, we reuse the technique of Boulier et al. [2017] to account for `typerec`, an elimination principle for the universe $\square$, which requires induction-recursion to be implemented.

We call the syntactic model of CastCIC the *discrete model*, in contrast with a semantic model motivated in the next subsection. The discrete model of CastCIC captures the intuition that the unknown type is inhabited by “hiding” the underlying type of the injected term. In other words, $?_{\square_i}$ behaves as a dependent sum $\Sigma\ \mathsf{A}{:}\square_i.\ \mathsf{A}$. Projecting out of the unknown type is realized through type analysis (`typerec`), and may fail (with an error in the ExTT sense). Note that here, we provide a particular interpretation of the unknown term in the universe, which is legitimized by an observation made by Pédrot and Tabareau [2018]: ExTT does not constrain in any way the definition of exceptions in the universe. The syntactic model of CastCIC allows us to establish that the reduction semantics enjoys strong normalization ($\mathcal{N}$), for the two variants $\mathrm{CastCIC}^{\mathcal{N}}$ and $\mathrm{CastCIC}^{\uparrow}$. Together with safety ($\mathcal{S}$), this gives us weak logical consistency for $\mathrm{CastCIC}^{\mathcal{N}}$ and $\mathrm{CastCIC}^{\uparrow}$.

3.4 Precisions and Properties

As explained earlier (§2.5), we need two different notions of precision to deal with SGG and $\mathcal{G}$. At the source level (GCIC), we introduce a notion of *syntactic precision* that captures the intuition of a more imprecise term as “the same term with subterms and/or annotated types replaced by ?”, and is defined without any assumption of typing. In CastCIC, we define a notion of *structural precision*, which is mostly syntactic except that, in order to account for cast insertion during elaboration, it tolerates precision-preserving casts (for instance, $\langle A \Leftarrow A \rangle\, t$ is related to t by structural precision). Armed with these two notions of precision, we prove *elaboration graduality* (Theorem 24), which is the equivalent of SGG in our setting: if a term t of GCIC elaborates to a term t' of CastCIC, then a

coercions, etc. The computation steps required by conversion are performed on the elaborated terms, never on the raw input syntax.

term u less syntactically precise than t in GCIC elaborates to a term u' less structurally precise than t' in CastCIC.

Because DGG is about the behavior of terms, it is technically stated and proven for CastCIC. We show in §5.5 that DGG can be proven for CastCIC (in its variants CastCIC$^{\mathcal{G}}$ and CastCIC$^{\uparrow}$) on the structural precision. However, as explained in §2.4, we cannot expect to prove $\mathcal{G}$ for these CastCIC variants with respect to structural precision directly. In order to overcome this problem, we build an alternative model of CastCIC called the *monotone model* (§6.2 to 6.5). This model endows types with the structure of an ordered set, or poset. In the monotone model, we can reason about the semantic notion of *propositional precision* and prove that it gives rise to embedding-projection pairs [New and Ahmed 2018], thereby establishing $\mathcal{G}$ for CastCIC$^{\uparrow}$ (Theorem 32). The monotone model only works for a normalizing gradual type theory, thus we then establish $\mathcal{G}$ for CastCIC$^{\mathcal{G}}$ using a variant of the monotone model based on Scott's model [Scott 1976] of the untyped λ-calculus using ω-complete partial orders (§6.7).

4 PRELIMINARIES: BIDIRECTIONAL CIC

We develop GCIC on top of a bidirectional version of CIC, whose presentation was folklore among type theory specialists [McBride 2019], and that has recently been studied in details by Lennon-Bertrand [2021]. As explained before, this bidirectional presentation is mainly useful to avoid multiple uses of a standalone conversion rule during typing, which becomes crucial to preserve $C_{/\mathrm{CIC}}$ in a gradual setting where conversion is replaced by consistency, which is not transitive. We give here a comprehensive summary of the bidirectional version of CIC that will help the reader follow the presentation of GCIC in §5.

Syntax. Our syntax for CIC terms, featuring a predicative universe hierarchy $\square_i$, is the following (in Backus-Naur form):

$$\mathrm{Term}_{\mathrm{CIC}} \ni t ::= x \mid \square_i \mid t\ t \mid \lambda\, x : t.t \mid \Pi\, x : t.t \mid I@\{i\}(\mathbf{t}) \mid c@\{i\}(\mathbf{t}, \mathbf{t}) \mid \mathtt{ind}_I(t, z.t, f.\mathbf{y}.\mathbf{t}) \qquad \text{(Syntax of CIC)}$$

We reserve letters x, y, z to denote variables. Other lower-case and upper-case Roman letters are used to represent terms, with the latter used to emphasize that the considered terms should be thought of as types (although the difference does not occur at a syntactic level in this presentation). Finally Greek capital letters are for contexts (lists of declarations of the form $x : T$). We also use bold letters $\mathbf{X}$ to denote sequences of objects $X_1, \ldots, X_n$ and $t[\mathbf{a}/\mathbf{y}]$ for the simultaneous substitution of $\mathbf{a}$ for $\mathbf{y}$. We present generic inductive types I with constructors c, although we restrict to well-formed (and in particular, strictly positive) ones to preserve normalization, following [Giménez 1998]. At this point we consider only inductive types without indices; we consider indexed inductive types in §7. Inductive types are formally annotated with a universe level @{i}, controlling the level of its parameters: for instance List@{i}(A) expects A to be a type in $\square_i$. This level is omitted when inessential. An inductive type at level i with parameters $\mathbf{a}$ is noted $I@\{i\}(\mathbf{a})$, and we use $\mathbf{Params}(I, i)$ to denote the types of those parameters. The well-formedness condition on inductives in particular enforces that the k-th parameter $\mathbf{Params}_k(I, i)$ only contains $k-1$ variables, corresponding to the previous $k-1$ parameters. Thus if $\mathbf{a}$ is a list of terms of the same length as $\mathbf{Params}(I, i)$ we denote as $\mathbf{Params}(I, i)[\mathbf{a}]$ the list where in parameter type $\mathbf{Params}_k(I, i)$, the $k-1$ first elements of $\mathbf{a}$ have been substituted for the $k-1$ free variables. Similarly $c_k^I@\{i\}(\mathbf{a}, \mathbf{b})$ denotes the k-th constructor of the inductive I, taking parameters $\mathbf{a}$ and arguments $\mathbf{b}$. Again, the type of parameters is denoted $\mathbf{Params}(I, i)$, and the type of the arguments $\mathbf{Args}(I, i, c_k)$. Similarly as for parameters, we also use $\mathbf{Args}(I, i, c_k)[\mathbf{a}, \mathbf{b}]$ for the list where in the m-th argument type $\mathbf{a}$ have been substituted for parameter variables, and the first $m-1$ elements of $\mathbf{b}$ for argument variables.

The inductive eliminator $\mathtt{ind}_I(s, z.P, f.\mathbf{y}.\mathbf{t})$ corresponds to a fixpoint immediately followed by a match. In Coq, one would write it

$$\mathtt{fix}\ f\ s := \mathtt{match}\ s\ \mathtt{as}\ z\ \mathtt{return}\ P\ \mathtt{with} \mid c_1\ \mathbf{y} \Rightarrow t_1 \ldots \mid c_n\ \mathbf{y} \Rightarrow t_n\ \mathtt{end}$$

In particular, the return predicate P has access to an extra bound variable z for the scrutinee, and similarly the branches t_k are given access to variables f and $\mathbf{y}$, corresponding respectively to the recursive function and the arguments of the corresponding constructor. Describing the exact guard condition to ensure termination is outside the scope of this presentation, again see [Giménez 1998]. We implicitly assume in the rest of this paper that every fixpoint is guarded.

Bidirectional Typing. In the usual, declarative, presentation of CIC, conversion between types is allowed at any stage of a typing derivation through a free-standing conversion rule. However, when conversion is replaced by a non-transitive relation of consistency, this free-standing rule is much too permissive and would violate $\mathcal{C}_{/\mathrm{CIC}}$. Indeed, as every type should be consistent with the unknown type $?_\square$, using such a rule twice in a row makes it possible to change the type of a typable term to any arbitrary type: if $\Gamma \vdash t : T$, because $T \sim ?_\square$ and $?_\square \sim S$, we could derive $\Gamma \vdash t : S$. This in turn would allow typeability of any term, including fully-precise terms, which is in contradiction with $\mathcal{C}_{/\mathrm{CIC}}$.

Thus, we rely on a bidirectional presentation of CIC typing, presented in Fig. 1, where the usual judgment $\Gamma \vdash t : T$ is decomposed into several mutually-defined judgments. The difference between the judgments lies in the role of the type: in the *inference* judgment $\Gamma \vdash t \triangleright T$, the type is considered an output, whereas in the *checking* judgment $\Gamma \vdash t \triangleleft T$, the type is instead seen as an input. Conversion can then be restricted to specific positions, namely to mediate between inference and checking judgments (see Check), and can thus never appear twice in a row.

Additionally, in the framework of an elaboration procedure, it is interesting to make a clear distinction between the subject of the rule (*i.e.,* the object that is to be elaborated), inputs that can be used for this elaboration, and outputs that must be constructed during the elaboration. In the context checking judgment $\vdash \Gamma$, Γ is the subject of the judgment. In all the other judgments, the subject is the term, the context is an input, and the type is either an input or an output, as we just explained.

An important discipline, that goes with this distinction, is that judgments should ensure that outputs are well-formed, under the hypothesis that the inputs are. All rules are built to ensure this invariant. This distinction between inputs, subject and output, and the associated discipline, are inspired by McBride [2018, 2019]. This is also the reason why no rule for term elaboration re-checks the context, as it is an input that is assumed to be well-formed. Hence, most properties we state in an open context involve an explicit hypothesis that the involved context is well-formed.

Constrained Inference. Apart from inference and checking, we also use a set of *constrained inference* judgments $\Gamma \vdash t \blacktriangleright_\bullet T$, with the same modes as inference. These judgments infer the type T but under some constraint $\bullet$: for instance that it should be a universe at some level ($\bullet = \square$), a Π-type ($\bullet = \Pi$), or an instance of an inductive I ($\bullet = \mathrm{I}$). Constrained inference judgments come from a close analysis of typing algorithms, such as the one of Coq, where in some places, an intermediate judgment between inference and checking happens: inference is performed, but then the type is reduced to expose its head constructor, which is imposed to be a specific one. A stereotypical example is App: one starts by inferring a type for t, but want it to be a Π-type so that its domain can be used to check u. To the best of our knowledge, these judgments have never been formally described elsewhere. Instead, in the rare bidirectional presentations of CIC, they are inlined in some way, as they only amount to some reduction. However, this is no longer true in a gradual setting: $?$ introduces an alternative, valid solution to the constrained inference, as a term of type $?$ can be

$\boxed{\vdash \Gamma}$

$$\frac{}{\vdash \cdot}\textsc{Empty} \qquad \frac{\vdash \Gamma \qquad \Gamma \vdash T \blacktriangleright_{\square} \square_i}{\vdash \Gamma, x : T}\textsc{Concat}$$

$\boxed{\Gamma \vdash t \triangleright T}$

$$\frac{}{\Gamma \vdash \square_i \triangleright \square_{i+1}}\textsc{Univ} \qquad \frac{(x : T) \in \Gamma}{\Gamma \vdash x \triangleright T}\textsc{Var} \qquad \frac{\Gamma \vdash A \blacktriangleright_{\square} \square_j \qquad \Gamma, x : A \vdash B \blacktriangleright_{\square} \square_i}{\Gamma \vdash \Pi x : A.B \triangleright \square_{\max(i,j)}}\textsc{Prod}$$

$$\frac{\Gamma \vdash A \blacktriangleright_{\square} \square_i \qquad \Gamma, x : A \vdash t \triangleright B}{\Gamma \vdash \lambda x : A.t \triangleright \Pi x : A.B}\textsc{Abs} \qquad \frac{\Gamma \vdash t \blacktriangleright_{\Pi} \Pi x : A.B \qquad \Gamma \vdash u \triangleleft A}{\Gamma \vdash t\ u \triangleright B[x/u]}\textsc{App}$$

$$\frac{\Gamma \vdash a_k \triangleleft \mathbf{Params}_k(I,i)[\mathbf{a}]}{\Gamma \vdash I@\{i\}(\mathbf{a}) \triangleright \square_i}\textsc{Ind} \qquad \frac{\Gamma \vdash a_k \triangleleft \mathbf{Params}_k(I,i)[\mathbf{a}] \qquad \Gamma \vdash b_m \triangleleft \mathbf{Args}_m(I,i,c)[\mathbf{a},\mathbf{b}]}{\Gamma \vdash c^I@\{i\}(\mathbf{a},\mathbf{b}) \triangleright I@\{i\}(\mathbf{a})}\textsc{Cons}$$

$$\frac{\begin{array}{c}\Gamma \vdash s \blacktriangleright_{I} I@\{i\}(\mathbf{a}) \qquad \Gamma, z : I(\mathbf{a}) \vdash P \blacktriangleright_{\square} \square_j \\ \Gamma, f : (\Pi z : I@\{i\}(\mathbf{a}).P), \mathbf{y} : \mathbf{Args}(I,i,c_k)[\mathbf{a},\mathbf{y}] \vdash t_k \triangleleft P[c_k^I@\{i\}(\mathbf{a},\mathbf{y})/z]\end{array}}{\Gamma \vdash \mathtt{ind}_I(s, z.P, f.\mathbf{y}.\mathbf{t}) \triangleright P[s/z]}\textsc{Fix}$$

$\boxed{\Gamma \vdash t \triangleleft T}$

$$\frac{\Gamma \vdash t \triangleright T' \qquad T' \equiv T}{\Gamma \vdash t \triangleleft T}\textsc{Check}$$

$\boxed{\Gamma \vdash t \blacktriangleright_{\bullet} T}$

$$\frac{\Gamma \vdash t \triangleright T \qquad T \rightsquigarrow^* \Pi x : A.B}{\Gamma \vdash t \blacktriangleright_{\Pi} \Pi x : A.B}\textsc{Prod-Inf} \qquad \frac{\Gamma \vdash t \triangleright T \qquad T \rightsquigarrow^* I@\{i\}(\mathbf{a})}{\Gamma \vdash t \blacktriangleright_{I} I@\{i\}(\mathbf{a})}\textsc{Ind-Inf}$$

$$\frac{\Gamma \vdash t \triangleright T \qquad T \rightsquigarrow^* \square_i}{\Gamma \vdash t \blacktriangleright_{\square} \square_i}\textsc{Univ-Inf}$$

$\boxed{t \rightsquigarrow u}$ (congruence rules omitted)

$$(\lambda x : A.t)\ u \rightsquigarrow t[u/x] \qquad \mathtt{ind}_I(c_k(\mathbf{a},\mathbf{b}), z.P, f.\mathbf{y}.\mathbf{t}) \rightsquigarrow t_k[\lambda x : I(\mathbf{a}).\ \mathtt{ind}_I(x, z.P, f.\mathbf{y}.\mathbf{t})/f][\mathbf{b}/\mathbf{y}]$$

$\boxed{t \equiv u}$

$$t \equiv u \quad := \quad \exists v\ v', t \rightsquigarrow^* v \land u \rightsquigarrow^* v' \land v =_\alpha v'$$

where $=_\alpha$ denotes syntactic equality up-to renaming

Fig. 1. CIC: Bidirectional typing

used where a term with a Π-type is expected. Thus, we will need multiple rules for constrained inference, which is why we make it explicit already at this stage.

Reduction. From here on, we impose no reduction strategy by default, and use $\rightsquigarrow$ and the unqualified word "reduction" for *full* reduction, *i.e.*, reduction that can be performed at an arbitrary place in a term, and $\rightsquigarrow^*$ for its reflexive, transitive closure. Most of the properties would however

$s_\Pi(i,j) := \max(i,j)$	$c_\Pi(i) := i$	($\mathrm{GCIC}^{\mathcal{G}}$-$\mathrm{CastCIC}^{\mathcal{G}}$)
$s_\Pi(i,j) := \max(i,j)$	$c_\Pi(i) := i-1$	($\mathrm{GCIC}^{\mathcal{N}}$-$\mathrm{CastCIC}^{\mathcal{N}}$)
$s_\Pi(i,j) := \max(i,j)+1$	$c_\Pi(i) := i-1$	($\mathrm{GCIC}^{\uparrow}$-$\mathrm{CastCIC}^{\uparrow}$)

Fig. 2. Universe parameters

carry over if we fixed *weak-head* reduction instead, and we sketch at the end of some proofs how they would carry over to such a fixed strategy. As uniqueness of inferred types and elaborated terms becomes stronger with a deterministic reduction strategy, we discuss weak-head reduction specifically in that case.

Finally, we observe that the equivalence of this bidirectional formulation with standard CIC relies on the transitivity of conversion; this has been very recently spelled out in details and formalized by Lennon-Bertrand [2021]. However, in the gradual setting, this property does not hold. This is precisely the point of using a bidirectional formulation: since consistency is not a transitive relation, a standard presentation of typing is not appropriate.

5 FROM GCIC TO CastCIC

We now present the elaboration from the source gradual system GCIC to the cast calculus CastCIC. We start with CastCIC, describing its typing, reduction and metatheoretical properties (§ 5.1). We next describe GCIC and its elaboration to CastCIC, along with few direct properties (§ 5.2). This elaboration is mainly an extension of the bidirectional CIC presented in the previous section. We illustrate the semantics of the different GCIC variants by considering the Ω term (§ 5.3). We finally expose technical properties of the reduction of CastCIC (§ 5.4) used to prove the most important theorems on elaboration: conservativity over CIC or $\mathrm{CIC}^{\uparrow}$, as well as the gradual guarantees (§ 5.5).

5.1 CastCIC

Syntax. The syntax of CastCIC[10] extends that of CIC (§ 4) with three new term constructors: the unknown term $?_T$ and dynamic error $\mathtt{err}_T$ of type T, as well as the cast $\langle T \Leftarrow S\rangle\, t$ of a term t of type S to type T

$$\mathrm{Term}_{\mathrm{CastCIC}} \ni t ::= \cdots \mid ?_t \mid \mathtt{err}_t \mid \langle t \Leftarrow t\rangle\, t \qquad \text{(Syntax of CastCIC)}$$

with casts associating to the right: $\langle S' \Leftarrow S\rangle \langle T \Leftarrow T'\rangle\, t$ is $\langle S' \Leftarrow S\rangle\, (\langle T' \Leftarrow T\rangle\, t)$. We also compress successive ones in the following way: $\langle T'' \Leftarrow T' \Leftarrow T\rangle\, t$ is shorthand for $\langle T'' \Leftarrow T'\rangle \langle T' \Leftarrow T\rangle\, t$. The unknown term and dynamic error both behave as exceptions as defined in ExTT [Pédrot and Tabareau 2018]. Casts keep track of the use of consistency during elaboration, implementing a form of runtime type-checking, raising the error $\mathtt{err}_T$ in case of a type mismatch. We call *static* the terms of CastCIC that do not use any of these new constructors—static CastCIC terms correspond to CIC terms.

Universe parameters. CastCIC is parametrized by two functions, described in Fig. 2, to account for the three different variants of GCIC we consider (§ 3.1). The first function s_Π computes the level of the universe of a dependent product, given the levels of its domain and codomain (see the updated Prod rule in Fig. 3). The second function c_Π controls the universe level in the reduction of a cast between $? \to ?$ and $?$ (see Fig. 5).

[10]Written using a blue color.

$\boxed{\Gamma \vdash t \triangleright T}$

$$\dots \qquad \frac{\Gamma \vdash A \blacktriangleright_{\square} \square_j \qquad \Gamma, x : A \vdash B \blacktriangleright_{\square} \square_i}{\Gamma \vdash \Pi x : A.B \triangleright \square_{s_\Pi(i,j)}}\textsc{Prod} \qquad \dots$$

$$\frac{\Gamma \vdash T \blacktriangleright_{\square} \square_i}{\Gamma \vdash ?_T \triangleright T}\textsc{Unk} \qquad \frac{\Gamma \vdash T \blacktriangleright_{\square} \square_i}{\Gamma \vdash \mathtt{err}_T \triangleright T}\textsc{Err} \qquad \frac{\Gamma \vdash A \blacktriangleright_{\square} \square_i \qquad \Gamma \vdash B \blacktriangleright_{\square} \square_j \qquad \Gamma \vdash t \triangleleft A}{\Gamma \vdash \langle B \Leftarrow A \rangle\, t \triangleright B}\textsc{Cast}$$

Fig. 3. CastCIC: Bidirectional typing (extending CIC Fig. 1, replacing Prod)

$\boxed{\text{Head, head} : \text{Type}_{\text{CastCIC}} \to \text{Head and germ} : \text{Head} \to \text{Type}_{\text{CastCIC}}}$

$$\text{Head} \ni h ::= \square_i \mid \Pi \mid I$$

$$\text{head}\,(\Pi x : A.B) := \Pi \qquad \text{head}\,(\square_i) := \square_i \qquad \text{head}\,(I(\mathbf{a})) := I$$

$$\text{germ}_i\ \square_j := \begin{cases} \square_j & \text{if } j < i \\ \mathtt{err}_{\square_i} & \text{if } j \geq i \end{cases} \qquad \text{germ}_i\ I := I(?_{\mathbf{Params}(I,i)})$$

$$\text{germ}_i\ \Pi := \begin{cases} ?_{\square_{c_\Pi(i)}} \to ?_{\square_{c_\Pi(i)}} & \text{if } c_\Pi(i) \geq 0 \\ \mathtt{err}_{\square_i} & \text{if } c_\Pi(i) < 0 \end{cases}$$

Fig. 4. Head constructor and germ

Typing. Fig. 3 gives the typing rules for the three new primitives of CastCIC. Apart from the modified Prod rule, which uses the s_Π parameter, all other typing rules are exactly the same as in CIC. When disambiguation is needed, we note this typing judgment as $\vdash_{\mathtt{cast}}$. The typing rules Unk and Err say that both $?_T$ and $\mathtt{err}_T$ infer T when T is a type. Note that in CastCIC, as is sometimes the case in cast calculi [New and Ahmed 2018; Siek and Wadler 2010], no consistency premise is required for a cast to be well-typed. Here, consistency only plays a role in GCIC, but disappears after elaboration. Instead, we rely on the usual conversion, defined as in CIC as the existence of α-equal reducts for the reduction described hereafter. The Cast rule only ensures that both the source and target of the cast are indeed types, and that the casted term indeed has the source type.

Reduction. The typing rules provide little insight on the new primitives; the interesting part really lie in their reduction behavior. The reduction rules of CastCIC are given in Fig. 5 (congruence rules omitted). Reduction relies on two auxiliary functions relating head constructors $h \in$ Head (Fig. 4) to those terms that start with either Π, $\square$ or I, the set of which we call $\text{Type}_{\text{CastCIC}}$. The first is the function head, which returns the head constructor of a type. In the other direction, the germ[11] function $\text{germ}_i\ h$ constructs the least precise type with head h at level i. In the case where no such type exists (*e.g.*, when $c_\Pi(i) < 0$), this least precise type is the error.

[11]The germ function corresponds to an abstraction function as in AGT [Garcia et al. 2016], if one interprets the head h as the set of all types whose head type constructor is h. Wadler and Findler [2009] christened the corresponding notion a *ground type*, later reused in the gradual typing literature. This terminology however clashes with its prior use in denotational semantics [Levy 2004]: there a ground type is a first-order datatype. Note that Siek and Taha [2006] also call ground types the base types of the language, such as $\mathbb{B}$ and $\mathbb{N}$. We therefore prefer the less overloaded term *germ*, used by analogy with the geometrical notion of the *germ of a section* [MacLane and Moerdijk 1992]: the germ of a head constructor represents an equivalence class of types that are locally the same.

$\boxed{t \rightsquigarrow t}$

Propagation rules for ? and err

Prod-Unk : $?_{\Pi(x:A).B} \rightsquigarrow \lambda(x:A).?_B$

Prod-Err : $\mathtt{err}_{\Pi(x:A).B} \rightsquigarrow \lambda(x:A).\,\mathtt{err}_B$

Match-Unk : $\mathtt{ind}_I(?_{I(\mathbf{a})}, z.P, f.\mathbf{y}.\mathbf{t}) \rightsquigarrow ?_{P[?_{I(\mathbf{a})}/z]}$

Match-Err : $\mathtt{ind}_I(\mathtt{err}_{I(\mathbf{a})}, z.P, f.\mathbf{y}.\mathbf{t}) \rightsquigarrow \mathtt{err}_{P[\mathtt{err}_{I(\mathbf{a})}/z]}$

Ind-Unk : $\langle I(\mathbf{a}') \Leftarrow I(\mathbf{a}'')\rangle\, ?_{I(\mathbf{a})} \rightsquigarrow ?_{I(\mathbf{a}')}$

Ind-Err : $\langle I(\mathbf{a}') \Leftarrow I(\mathbf{a}'')\rangle\, \mathtt{err}_{I(\mathbf{a})} \rightsquigarrow \mathtt{err}_{I(\mathbf{a}')}$

Down-Unk : $\langle X \Leftarrow ?_{\square}\rangle\, ?_{?_{\square}} \rightsquigarrow ?_X$

Down-Err : $\langle X \Leftarrow ?_{\square}\rangle\, \mathtt{err}_{?_{\square}} \rightsquigarrow \mathtt{err}_X$

Reduction rules for cast

Prod-Prod : $\langle \Pi(y:A_2).B_2 \Leftarrow \Pi(x:A_1).B_1\rangle\, (\lambda\, x : A.t) \rightsquigarrow$

$$\lambda\, y : A_2.\langle B_2 \Leftarrow B_1[\langle A_1 \Leftarrow A_2\rangle\, y/x]\rangle\, (t[\langle A \Leftarrow A_2\rangle\, y/x])$$

Univ-Univ : $\langle \square_i \Leftarrow \square_i\rangle\, A \rightsquigarrow A$

Ind-Ind : $\langle I(\mathbf{a}_2) \Leftarrow I(\mathbf{a}_1)\rangle\, c(\mathbf{a}, b_1, \dots, b_n) \rightsquigarrow c(\mathbf{a}', \mathbf{b}'_1, \dots, \mathbf{b}'_n)$

with $\mathbf{b}'_k := \langle \mathbf{Args}_k(I,i,c)[\mathbf{a}',\mathbf{b}'] \Leftarrow \mathbf{Args}_k(I,i,c)[\mathbf{a},\mathbf{b}]\rangle\, b_k$

Head-Err : $\langle T' \Leftarrow T\rangle\, t \rightsquigarrow \mathtt{err}_{T'}$ when $T, T' \in \mathrm{Type}_{\mathrm{CastCIC}}$ and $\mathrm{head}\, T \neq \mathrm{head}\, T'$

Dom-Err : $\langle T \Leftarrow \mathtt{err}_{\square}\rangle\, t \rightsquigarrow \mathtt{err}_T$

Codom-Err : $\langle \mathtt{err}_{\square} \Leftarrow T\rangle\, t \rightsquigarrow \mathtt{err}_{\mathtt{err}_{\square}}$ when $T \in \mathrm{Type}_{\mathrm{CastCIC}}$

Prod-Germ : $\langle ?_{\square_i} \Leftarrow \Pi\, x : A.B\rangle\, f \rightsquigarrow \langle ?_{\square_i} \Leftarrow \mathrm{germ}_i\, \Pi \Leftarrow \Pi\, x : A.B\rangle\, f$

when $\Pi\, x : A.B \neq \mathrm{germ}_j\, \Pi$ for $j \geq i$

Ind-Germ : $\langle ?_{\square_i} \Leftarrow I(\mathbf{a})\rangle\, t \rightsquigarrow \langle ?_{\square_i} \Leftarrow \mathrm{germ}_i\, I \Leftarrow I(\mathbf{a})\rangle\, t$ when $I(\mathbf{a}) \neq \mathrm{germ}_j\, I$ for $j \geq i$

Up-Down : $\langle X \Leftarrow ?_{\square_i} \Leftarrow \mathrm{germ}_i\, h\rangle\, t \rightsquigarrow \langle X \Leftarrow \mathrm{germ}_i\, h\rangle\, t$ when $\mathrm{germ}_i\, h \neq \mathtt{err}_{\square_i}$

Size-Err : $\langle ?_{\square_i} \Leftarrow A\rangle\, t \rightsquigarrow \mathtt{err}_{?_{\square_i}}$ when $\min\{j \mid \exists h \in \mathrm{Head}, \mathrm{germ}_j\, h = A\} > i$

Fig. 5. CastCIC: Reduction rules (extending Fig. 1, congruence rules omitted)

The design of the reduction rules is mostly dictated by the discrete and monotone models of CastCIC presented later in §6. Nevertheless, we now provide some intuition about their meaning. Let us start with rules Prod-Unk, Prod-Err, Match-Unk and Match-Err. These rules specify the exception-like propagation behavior of both ? and `err` at product and inductive types. Rules Ind-Unk and Ind-Err similarly propagate ? and `err` when cast between the same inductive type, and rules Down-Unk and Down-Err do the same from the unknown type to any type X.

Next are rules Prod-Prod, Ind-Ind and Univ-Univ, which correspond to success cases of dynamic checks, where the cast is between types with the same head. In that case, casts are either completely erased when possible, or propagated. As usual in gradual typing, directly inspired by higher-order contracts [Findler and Felleisen 2002], Prod-Prod distributes the function cast in two casts, one for the argument and one for the body; note the substitution in the source codomain in order to account for dependency. Also, because constructors and inductives are fully-applied, this Prod-Prod rule cannot be blocked on a partially-applied constructor or inductive. Regarding inductive types, the restriction to reduce only on constructors means that a cast between $\mathbb{N}$ and $\mathbb{N}$ is blocked until its argument term is a constructor, rather than disappearing right away as for the universe. We follow this somewhat non-optimal strategy to be consistent between inductive types, because for more complex inductive types such as lists, the propagation of casts on subterms cannot be avoided.

On the contrary, rule Head-Err specifies failure of a dynamic check when the considered types have different heads. Similarly, rules Dom-Err, Codom-Err specify that cast to or from the error type is always an error.

Finally, there are specific rules pertaining to casts to and from ?, showcasing its behaviour as a universal type. Rules Prod-Germ and Ind-Germ decompose an upcast into ? as an upcast to a germ followed by an upcast from the germ to ?. This decomposition of an upcast to ? into a series of "atomic" upcasts from a germ to ? is a consequence of the way the cast operation is implemented in §6, but similar decompositions appear e.g. in Siek et al. [2015], where the equivalent of our germs are called ground types. The side conditions guarantee that this rule is used when no other applies. Rule Up-Down erases the succession of an upcast to ? and a downcast from it. Note that in this rule the upcast $\langle ?_{\square_i} \Leftarrow \mathrm{germ}_h\ i \rangle\, t$ works like a constructor for $?_{\square_i}$ and $\langle X \Leftarrow ?_{\square_i} \rangle$ as a destructor—a view reflected by the canonical and neutral forms of Fig. 7 for $?_{\square}$.[12] Finally, rule Size-Err corresponds to a peculiar kind of error, which only happens due to the presence of a type hierarchy: $?_{\square_i}$ is only universal with respect to types at level i, and so a type might be of a level too high to fit into it. To detect such a case, we check whether A is a germ for a level that is below i, and when not throw an error.

Meta-Theoretical Properties. The typing and reduction rules just given ensure two of the meta-theoretical properties introduced in §2: $\mathcal{S}$ for the three variants of CastCIC, as well as $\mathcal{N}$ for CastCIC$^{\mathcal{N}}$ and CastCIC$^{\uparrow}$. Before turning to these properties, let us establish a crucial lemma, namely the confluence of the rewriting system induced by reduction.

Lemma 7 (Confluence of CastCIC). *If t and u are related by the symmetric, reflexive, transitive closure of $\rightsquigarrow$, then there exists s such that $t \rightsquigarrow^* s$ and $u \rightsquigarrow^* s$.*

Proof. We extend the notion of parallel reduction ($\Rrightarrow$) for CIC from [Sozeau et al. 2020] to account for our additional reduction rules and show that the triangle property—the existence, for any term t, of an optimal reduced term $\rho(t)$ in one step (Fig. 6a)—still holds. From the triangle property, it

[12] In a simply-typed language such as GTLC [Siek et al. 2015], where there are no neutrals at the type level, casts from a germ/ground type to the unknown type are usually interpreted as tagged values [Siek and Taha 2006]. Here, these correspond exactly to the canonical forms of $?_{\square}$, but we also have to account for the many neutral forms that appear in open contexts.

(a) The triangle property. (b) The triangle property implies confluence.

Fig. 6. Representation of the triangle property (left) and its consequence on confluence (right).

is easy to deduce confluence of parallel reduction in one step (Fig. 6b), which implies confluence because parallel reduction is between one-step reduction and iterated reductions. This proof method is basically an extension of the Tait-Martin Löf criterion on parallel reduction [Barendregt 1984; Takahashi 1995]. □

Let us now turn to $\mathcal{S}$, which we prove using the standard progress and subject reduction properties [Wright and Felleisen 1994]. Progress describes a set of canonical forms, asserting that all terms that do not belong to such canonical forms are not in normal form, *i.e.,* can take at least one reduction step. Fig. 7 provides the definition of canonical forms, considering head reduction.

As standard in dependent type theories, we distinguish between canonical forms and neutral terms. Neutral terms correspond to (blocked) destructors, waiting for a substitution to happen, while other canonical forms correspond to constructors. Additionally, the notion of neutral terms naturally induces a weak-head reduction strategy that consists in either applying a top-level reduction or reducing the (only) argument of the top-level destructor that is in a neutral position.

The canonical forms for plain CIC are given by the first three lines of Fig. 7. The added rules deal with errors, unknown terms and casts. First, an error $\mathtt{err}_t$ or an unknown term $?_t$ is neutral when t is neutral, and is canonical only when t is $\square$ or $I(\mathbf{a})$, but not a Π-type. This is because exception-like terms reduce on Π-types [Pédrot and Tabareau 2018]. Second, there is an additional specific form of canonical inhabitants of $?_\square$: these are upcasts from a germ, which can be seen as a term tagged with the head constructor of its type, in a matter reminiscent of actual implementations of dynamic typing using type tags. As we explained when presenting Fig. 5, these canonical forms work as constructors for $?_\square$. Finally, the cast operation behaves as a destructor on the universe $\square$—as if it were an inductive type of usual CIC. This destructor first scrutinizes the source type of the cast. This is why the cast is neutral as soon as its source type is neutral. When the source type reduces to a head constructor, there are two possibilities. Either that constructor is $?_\square$, in which case the cast scrutinizes its argument to be a canonical form $\langle ?_\square \Leftarrow t \rangle\, \mathrm{germ}_i\ h$ and is neutral when this is not the case. In all other cases, it first scrutinizes the target type, so the cast is neutral when the target type is neutral. Finally, when both types have head constructors, the cast might still need its argument to be either a λ-abstraction or an inductive constructor to reduce.

Equipped with the notion of canonical forms, we can state $\mathcal{S}$ for CastCIC:

Theorem 8 (Safety of the three variants of CastCIC ($\mathcal{S}$)). CastCIC *enjoys:*

Progress: *if* t *is a well-typed term of* CastCIC*, then either* canonical t *or there is some* t' *such that* $t \rightsquigarrow t'$*.*

Subject reduction: *if* $\Gamma \vdash_{\mathsf{cast}} t \triangleright A$ *and* $t \rightsquigarrow t'$ *then* $\Gamma \vdash_{\mathsf{cast}} t' \triangleleft A$*.*

Thus CastCIC *enjoys* $\mathcal{S}$*.*

Proof. **Progress:** The proof is by induction on the typing derivation of t. As standard, we show that in all cases, either a reduction on a subterm happens, t itself reduces because some canonical form was not neutral and creates a redex, or t is neutral.

$\boxed{\text{canonical } t, \text{neutral } t}$

$$\frac{}{\text{canonical } \lambda\, x : A.t} \qquad \frac{}{\text{canonical } c(\mathbf{a}, \mathbf{b})} \qquad \frac{}{\text{canonical } \Pi\, x : A.B}$$

$$\frac{}{\text{canonical } \square} \qquad \frac{}{\text{canonical } I(\mathbf{a})} \qquad \frac{\text{neutral } t}{\text{canonical } t}$$

$$\frac{}{\text{neutral } x} \qquad \frac{\text{neutral } t}{\text{neutral } t\ u} \qquad \frac{\text{neutral } t}{\text{neutral } \mathtt{ind}_I(t, z.P, \mathbf{f}.\mathbf{y}.\mathbf{b})}$$

$$\frac{T \in \{\, \square, I(\mathbf{a}), ?_\square, \mathtt{err}_\square \,\}}{\text{canonical } ?_T} \qquad \frac{\text{neutral } t}{\text{neutral } ?_t} \qquad \frac{T \in \{\, \square, I(\mathbf{a}), ?_\square, \mathtt{err}_\square \,\}}{\text{canonical } \mathtt{err}_T} \qquad \frac{\text{neutral } t}{\text{neutral } \mathtt{err}_t}$$

$$\frac{}{\text{canonical} \langle ?_{\square_i} \Leftarrow \mathrm{germ}_i\ h \rangle\ t} \qquad \frac{\text{neutral } S}{\text{neutral} \langle T \Leftarrow S \rangle\ t} \qquad \frac{\text{neutral } t}{\text{neutral} \langle T \Leftarrow ?_\square \rangle\ t}$$

$$\frac{\text{neutral } T}{\text{neutral} \langle T \Leftarrow \square \rangle\ t} \qquad \frac{\text{neutral } T}{\text{neutral} \langle T \Leftarrow \Pi\, x : A.B \rangle\ t} \qquad \frac{\text{neutral } t}{\text{neutral} \langle \Pi\, x : A'.B' \Leftarrow \Pi\, x : A.B \rangle\ t}$$

$$\frac{\text{neutral } T}{\text{neutral} \langle T \Leftarrow I(\mathbf{a}) \rangle\ t} \qquad \frac{\text{neutral } t}{\text{neutral} \langle I(\mathbf{a}') \Leftarrow I(\mathbf{a}) \rangle\ t}$$

Fig. 7. Head neutral and canonical forms for CastCIC

Subject reduction: Subject reduction can be derived from the injectivity of type constructors, which is a direct consequence of confluence. See [Sozeau et al. 2020] for a detailed account of this result in the simpler setting of CIC.

□

We now establish normalization of $\mathsf{CastCIC}^{\mathcal{N}}$ and $\mathsf{CastCIC}^{\uparrow}$, although the proof below relies on the discrete model defined in §6.1.

Theorem 9 (Normalization of $\mathsf{CastCIC}^{\mathcal{N}}$ and $\mathsf{CastCIC}^{\uparrow}$ ($\mathcal{N}$)). *Every reduction path for a well-typed term in $\mathsf{CastCIC}^{\mathcal{N}}$ or $\mathsf{CastCIC}^{\uparrow}$ is finite.*

Proof. The translation induced by the discrete model presented in §6.1 maps each reduction step to at least one step (Theorem 26). So strong normalization holds because the target calculus of the translation is normalizing. □

5.2 Elaboration from GCIC to CastCIC

Now that CastCIC has been described, we move on to GCIC. The typing judgment of GCIC is *defined* by an elaboration judgment from GCIC to CastCIC, based upon Fig. 1, augmenting all judgments with an extra output: the elaborated CastCIC term. This definition of typing using elaboration is required because of the intricate interdependency between typing and reduction exposed in §3.

$$\boxed{t \sim_\alpha t}$$

$$\frac{}{x \sim_\alpha x} \qquad \frac{}{\square_i \sim_\alpha \square_i} \qquad \frac{A \sim_\alpha A' \qquad t \sim_\alpha t'}{\lambda x : A.t \sim_\alpha \lambda x : A'.t'} \qquad \frac{A \sim_\alpha A' \qquad B \sim_\alpha B'}{\Pi x : A.B \sim_\alpha \Pi x : A'.B'} \qquad \frac{t \sim_\alpha t' \qquad u \sim_\alpha u'}{t\, u \sim_\alpha t'\, u'}$$

$$\frac{\mathbf{a} \sim_\alpha \mathbf{a}'}{I(\mathbf{a}) \sim_\alpha I(\mathbf{a}')} \qquad \frac{\mathbf{a} \sim_\alpha \mathbf{a}' \qquad \mathbf{b} \sim_\alpha \mathbf{b}'}{c_k(\mathbf{a}, \mathbf{b}) \sim_\alpha c_k(\mathbf{a}', \mathbf{b}')} \qquad \frac{s \sim_\alpha s' \qquad P \sim_\alpha P' \qquad \mathbf{t} \sim_\alpha \mathbf{t}'}{\mathtt{ind}_I(s, z.P, f.\mathbf{y}.\mathbf{t}) \sim_\alpha \mathtt{ind}_I(s', z.P', f.\mathbf{y}.\mathbf{t}')}$$

$$\frac{t \sim_\alpha t'}{t \sim_\alpha \langle B' \Leftarrow A' \rangle\, t'} \qquad \frac{t \sim_\alpha t'}{\langle B \Leftarrow A \rangle\, t \sim_\alpha t'} \qquad \frac{}{t \sim_\alpha ?_{T'}} \qquad \frac{}{?_T \sim_\alpha t}$$

Fig. 8. CastCIC: α-consistency

Syntax. The syntax of GCIC[13] extends that of CIC with a single new term constructor $?_{@\{i\}}$, where i is a universe level. From a user perspective, one is not given direct access to the failure and cast primitives, those only arise through uses of $?$.

Consistent conversion. Before we can describe typing, we should focus on conversion. Indeed, to account for the imprecision introduced by $?$, elaboration employs *consistent conversion* to compare CastCIC terms rather than usual conversion relation.

Definition 5 (Consistent conversion). *Two* CastCIC *terms are* α-consistent, *written* $\sim_\alpha$, *if they are in the relation defined by the inductive rules of Fig. 8.*

Two terms are consistently convertible, *or simply* consistent, *noted* $s \sim t$, *if and only if there exists* s' *and* t' *such that* $s \leadsto^* s'$, $t \leadsto^* t'$ *and* $s' \sim_\alpha t'$.

Thus α-consistency is an extension of α-equality that takes imprecision into account. Apart from the standard rules making $?$ consistent with any term, α-consistency optimistically ignores casts, and does not consider errors to be consistent with themselves. The first point is to prevent casts inserted by the elaboration from disrupting valid conversions, typically between static terms. The second is guided by the idea that if errors are encountered at elaboration already, the term cannot be well behaved, so it must be rejected as early as possible and we should avoid typing it. The consistency relation is then built upon α-consistency in a way totally similar to how conversion in Figs. 1 and 5 is built upon α-equality. Also note that this formulation of consistent conversion makes no assumption of normalization, and is therefore usable as such in the non-normalizing GCIC$^{\mathcal{G}}$.

An important property of consistent conversion, and a necessary condition for the conservativity of GCIC with respect to CIC ($\mathcal{C}_{/\text{CIC}}$), is that it corresponds to conversion on static terms.

Proposition 10 (Properties of consistent conversion).

(1) Two static terms are consistently convertible if and only if they are convertible in CIC.
(2) If s *and* t *have a normal form, then* $s \sim t$ *is decidable.*

Proof. *(1)* First remark that α-consistency between static terms corresponds to α-equality of terms. Thus, and because the reduction of static terms in CastCIC is the same as the reduction of CIC, two consistent static terms must reduce to α-equal terms, which in turn implies that they are convertible. Conversely two convertible terms of CIC have a common reduct, which is α-consistent with itself.

[13] We use green for terms of GCIC. To maintain a distinction in the absence of colors, we also use tildes ($\tilde{t}$) for terms in GCIC in expressions mixing both source and target terms.

$\boxed{\Gamma \vdash \tilde{t} \rightsquigarrow t \triangleright T}$

$$\frac{(x : T) \in \Gamma}{\Gamma \vdash x \rightsquigarrow x \triangleright T}\textsc{Var} \qquad \frac{}{\Gamma \vdash \square_i \rightsquigarrow \square_i \triangleright \square_{i+1}}\textsc{Univ}$$

$$\frac{\Gamma \vdash \tilde{A} \rightsquigarrow A \blacktriangleright_{\square} \square_i \qquad \Gamma, x : A \vdash \tilde{B} \rightsquigarrow B \blacktriangleright_{\square} \square_j}{\Gamma \vdash \Pi x : \tilde{A}.\tilde{B} \rightsquigarrow \Pi x : A.B \triangleright \square_{s_\Pi(i,j)}}\textsc{Prod} \qquad \frac{\Gamma \vdash \tilde{A} \rightsquigarrow A \blacktriangleright_{\square} \square_i \qquad \Gamma, x : A \vdash \tilde{t} \rightsquigarrow t \triangleright B}{\Gamma \vdash \lambda x : \tilde{A}.\tilde{t} \rightsquigarrow \lambda x : A.t \triangleright \Pi x : A.B}\textsc{Abs}$$

$$\frac{\Gamma \vdash \tilde{t} \rightsquigarrow t \blacktriangleright_{\Pi} \Pi x : A.B \qquad \Gamma \vdash \tilde{u} \triangleleft A \rightsquigarrow u}{\Gamma \vdash \tilde{t}\ \tilde{u} \rightsquigarrow t\ u \triangleright B[u/x]}\textsc{App} \qquad \frac{}{\Gamma \vdash ?_{@\{i\}} \rightsquigarrow ?_{?_{\square_i}} \triangleright ?_{\square_i}}\textsc{Unk}$$

$$\frac{\Gamma \vdash \tilde{a}_k \triangleleft \mathbf{Params}_k(I,i)[\mathbf{a}] \rightsquigarrow a_k}{\Gamma \vdash I_{@\{i\}}(\tilde{\mathbf{a}}) \rightsquigarrow I_{@\{i\}}(\mathbf{a}) \triangleright \square_i}\textsc{Ind}$$

$$\frac{\Gamma \vdash \tilde{a}_k \triangleleft \mathbf{Params}_k(I,i)[\mathbf{a}] \rightsquigarrow a_k \qquad \Gamma \vdash \tilde{b}_m \triangleleft \mathbf{Args}_m(I,i,c)[\mathbf{a},\mathbf{b}] \rightsquigarrow b_m}{\Gamma \vdash c_{k@\{i\}}(\tilde{\mathbf{a}},\tilde{\mathbf{b}}) \rightsquigarrow c(\mathbf{a},\mathbf{b}) \triangleright I(\mathbf{a})}\textsc{Cons}$$

$$\frac{\begin{array}{c}\Gamma \vdash \tilde{s} \rightsquigarrow s \blacktriangleright_{I} I(\mathbf{a}) \qquad \Gamma, z : I(\mathbf{a}) \vdash \tilde{P} \rightsquigarrow P \blacktriangleright_{\square} \square_i \\ \Gamma, f : (\Pi z : I(\mathbf{a}), P), \mathbf{y} : \mathbf{Args}(I,i,c_k)[\mathbf{a},\mathbf{y}] \vdash \tilde{t}_k \triangleleft P[c_k(\mathbf{a},\mathbf{y})/z] \rightsquigarrow t_k\end{array}}{\Gamma \vdash \mathtt{ind}_I(\tilde{s}, z.\tilde{P}, f.\mathbf{y}.\tilde{\mathbf{t}}) \rightsquigarrow \mathtt{ind}_I(s, z.P, \mathbf{f}.\mathbf{y}.\mathbf{t}) \triangleright P[s/z]}\textsc{Fix}$$

$\boxed{\Gamma \vdash \tilde{t} \triangleleft T \rightsquigarrow t}$

$$\frac{\Gamma \vdash \tilde{t} \rightsquigarrow t \triangleright T \qquad T \sim S}{\Gamma \vdash \tilde{t} \triangleleft S \rightsquigarrow \langle S \Leftarrow T \rangle\, t}\textsc{Check}$$

$\boxed{\Gamma \vdash \tilde{t} \rightsquigarrow t \blacktriangleright_{\bullet} T}$

$$\frac{\Gamma \vdash \tilde{t} \rightsquigarrow t \triangleright T \qquad T \rightsquigarrow^* \square_i}{\Gamma \vdash \tilde{t} \rightsquigarrow t \blacktriangleright_{\square} \square_i}\textsc{Inf-Unk} \qquad \frac{\Gamma \vdash \tilde{t} \rightsquigarrow t \triangleright T \qquad T \rightsquigarrow^* ?_{\square_{i+1}}}{\Gamma \vdash \tilde{t} \rightsquigarrow \langle \square_i \Leftarrow T \rangle\, t \blacktriangleright_{\square} \square_i}\textsc{Inf-Univ?}$$

$$\frac{\Gamma \vdash \tilde{t} \rightsquigarrow t \triangleright T \qquad T \rightsquigarrow^* \Pi x : A.B}{\Gamma \vdash \tilde{t} \rightsquigarrow t \blacktriangleright_{\Pi} \Pi x : A.B}\textsc{Inf-Prod} \qquad \frac{\Gamma \vdash \tilde{t} \rightsquigarrow t \triangleright T \qquad T \rightsquigarrow^* ?_{\square_i} \qquad c_\Pi(i) \geq 0}{\Gamma \vdash \tilde{t} \rightsquigarrow \langle \mathrm{germ}_i\ \Pi \Leftarrow T \rangle\, t \blacktriangleright_{\Pi} \mathrm{germ}_i\ \Pi}\textsc{Inf-Prod?}$$

$$\frac{\Gamma \vdash \tilde{t} \rightsquigarrow t \triangleright T \qquad T \rightsquigarrow^* I(\mathbf{a})}{\Gamma \vdash \tilde{t} \rightsquigarrow t \blacktriangleright_{I} I(\mathbf{a})}\textsc{Inf-Ind} \qquad \frac{\Gamma \vdash \tilde{t} \rightsquigarrow t \triangleright T \qquad T \rightsquigarrow^* ?_{\square_i}}{\Gamma \vdash \tilde{t} \rightsquigarrow \langle \mathrm{germ}_i\ I \Leftarrow T \rangle\, t \blacktriangleright_{I} \mathrm{germ}_i\ I}\textsc{Inf-Ind?}$$

Fig. 9. Type-directed elaboration from GCIC to CastCIC

(2) If s and t are normalizing, they have a finite number of reducts, thus to decide their consistency it is sufficient to check each pair of reducts for the decidable α-consistency. Comparing normal forms is not enough, because a term $t$might be stuck because of a cast while another one s can be α-consistent with it and reduce further, so that the normal form of t and s are not α-consistent while t and s are consistent. $\square$

Elaboration. Elaboration from GCIC to CastCIC is given in Fig. 9, closely following the bidirectional presentation of CIC (Fig. 1) for most rules, simply carrying around the extra elaborated terms. Note that only the subject of the judgment is a source term in GCIC; other inputs (that have already been elaborated), as well as outputs (that are to be constructed), are target terms in CastCIC. Let us comment a bit on the specific modifications and additions compared to Fig. 1.

The most salient feature of elaboration is the insertion of casts that mediate between merely consistent but not convertible types. They of course are needed in the rule CHECK where the terms are compared using consistency. But this is not enough: casts also appear in the newly-introduced rules INF-UNIV? INF-PROD? and INF-IND? for constrained inference, where the type $?_{\square_i}$ is replaced by the least precise type of the appropriate universe level having the constrained head constructor, which is exactly what the germ function gives us. Note that in the case of INF-UNIV? we could have replaced $\square_i$ with $\mathrm{germ}_{i+1}\ \square_i$ to make for a presentation similar to the other two rules. The role of these three rules is to ensure that a term of type $?_{\square_i}$ can be used as a function, or as a scrutinee of a match, by giving a way to derive constrained inference for such a term.

It is interesting to observe that the rules for constrained elaboration in a gradual setting bear a close resemblance with those described by Cimini and Siek [2016, Section 3.3], where a matching operator is introduced to verify that an output type can fit into a certain type constructor—either by having that type constructor as head symbol or by virtue of being ?. Such a form of matching was already present in our static, bidirectional system, because of the presence of reduction in types. In a way, both Cimini and Siek [2016] and Lennon-Bertrand [2021] have the same need of separating the inferred type from operations on it to recover its head constructor, and our mixing of both computation and gradual typing makes that need even clearer.

Rule UNK also deserves some explanation: $?@\{i\}$ is elaborated to $?_{?_{\square_i}}$, the least precise term of the least precise type of the whole universe $\square_i$. This avoids unneeded type annotations on $?$ in GCIC. Instead, the context is responsible for inserting the appropriate cast, *e.g.*, $? :: T$ elaborates to a term reducing to $?_T$. We do not drop annotations altogether because of an important property on which bidirectional CIC is built: any well-formed term should *infer* a type, not just check. Thus, we must be able to infer a type for $?$. The obvious choice is to have $?$ infer $?$, but this $?$ is a term of CastCIC, and thus needs a type index. Because this $?$ is used as a type, this index must be $\square$, and the universe level of the source $?$ is there to give us the level of this $\square$. In a real system, this should be handled by *typical ambiguity*,[14] alleviating the user from the need to give any annotations when using $?$.

Direct properties. As the elaboration rules are completely syntax-directed, they immediately translate to an algorithm for elaboration. Coupled with decidability of consistency (Prop. 10), this makes elaboration decidable whenever $\rightsquigarrow^*$ is normalizing; when $\rightsquigarrow^*$ is not normalizing, the elaboration algorithm might diverge, resulting in only semi-decidability of typing (as in, for instance, Dependent Haskell [Eisenberg 2016]).

THEOREM 11 (Decidability of elaboration). *The relations of inference, checking and partial inference of Fig. 9 are decidable in* $\mathrm{GCIC}^{\mathcal{N}}$ *and* $\mathrm{GCIC}^{\uparrow}$. *They are semi-decidable in* $\mathrm{GCIC}^{\mathcal{G}}$.

Let us now establish two important properties of elaboration that we can prove at this stage: elaboration is *correct*, insofar as it produces well-typed CastCIC terms, and functional, in the sense that a given GCIC term can be elaborated to at most one CastCIC term up to conversion.

THEOREM 12 (Correctness of elaboration). *The elaboration produces well-typed terms in a well-formed context. Namely, given* Γ *such that* $\vdash_{\mathsf{cast}} \Gamma$, *we have that:*

- *if* $\Gamma \vdash \tilde{t} \rightsquigarrow t \triangleright T$, *then* $\Gamma \vdash_{\mathsf{cast}} t \triangleright T$;
- *if* $\Gamma \vdash \tilde{t} \rightsquigarrow t \blacktriangleright_{\bullet} T$ *then* $\Gamma \vdash_{\mathsf{cast}} t \blacktriangleright_{\bullet} T$ *(with* $\bullet$ *denoting the same index in both derivations);*
- *if* $\Gamma \vdash \tilde{t} \triangleleft T \rightsquigarrow t$ *and* $\Gamma \vdash_{\mathsf{cast}} T \blacktriangleright_{\square} \square_i$, *then* $\Gamma \vdash_{\mathsf{cast}} t \triangleleft T$.

[14] Typical ambiguity [Harper and Pollack 1991] is the possibility to avoid giving explicit universe levels, letting the system decide whether a consistent assignment of levels can be found. In Coq, for instance, one almost never has to be explicit about universe levels when writing Type.

Proof. The proof is by induction on the elaboration derivation, mutually with similar properties for all typing judgments. In particular, for checking, we have an extra hypothesis that the given type is well-formed, as it is an input that should already have been typed.

Because the bidirectional typing rules of CIC are very similar to the GCIC-to-CastCIC elaboration rules, the induction is mostly routine. Let us point however that the careful design of the bidirectional rules already in CIC regarding the input/output separation is important here. Indeed, we have that inputs to the successive premises of a rule are always well-formed, either as inputs to the conclusion, or thanks to previous premises. In particular, all context extensions are valid, *i.e.,* $\Gamma, x : A$ is used only when $\Gamma \vdash A \blacktriangleright_{\square} \square_i$, and similarly only well-formed types are used for checking. This ensures that we can always use the induction hypothesis.

The only novel points to consider are the rules where a cast is inserted. For these, we rely on the validity property (an inferred type is always well-typed itself) to ensure that the domain of inserted casts is well-typed, and thus that the casts can be typed. □

Because of the absence of a fixed, deterministic reduction strategy, the elaborated term is not unique. Indeed, since a type can be reduced to multiple product types in rule § 5.2, a term can infer multiple, different types, and since those appear later on in casts, the elaborated terms can differ by having different, albeit convertible, types in their casts. We thus state two theorems: one is uniqueness up to conversion, in case full reduction is used. The second is a strengthening if a weak-head reduction strategy is imposed for reduction.

Theorem 13 (Uniqueness of elaboration—Full reduction). *Elaborated terms are convertible:*

- *if* $\Gamma \vdash \tilde{t} \rightsquigarrow t \triangleright T$ *and* $\Gamma \vdash \tilde{t} \rightsquigarrow t' \triangleright T'$, *then* $t \equiv t'$ *and* $T \equiv T'$;
- *if* $\Gamma \vdash \tilde{t} \rightsquigarrow t \blacktriangleright_{\bullet} T$ *and* $\Gamma \vdash \tilde{t} \rightsquigarrow t' \blacktriangleright_{\bullet} T'$ *then* $t \equiv t'$ *and* $T \equiv T'$;
- *if* $\Gamma \vdash \tilde{t} \triangleleft T \rightsquigarrow t$ *and* $\Gamma \vdash \tilde{t} \triangleleft T \rightsquigarrow t'$ *then* $t \equiv t'$.

(Recall that conversion $\equiv$ in CastCIC is defined (similarly as in CIC) as the existence of α-equal reducts for the reduction given in Fig. 5.)

Theorem 14 (Uniqueness of elaboration—Weak-head reduction). *If in Fig. 9,* $\rightsquigarrow^*$ *is replaced by weak-head reduction, then elaborated terms are unique:*

- *given* Γ *and* $\tilde{t}$, *there is at most one* t *and one* T *such that* $\Gamma \vdash \tilde{t} \rightsquigarrow t \triangleright T$;
- *given* Γ *and* $\tilde{t}$, *there is at most one* t *and one* T *such that* $\Gamma \vdash \tilde{t} \rightsquigarrow t \blacktriangleright_{\bullet} T$;
- *given* Γ, $\tilde{t}$ *and* T, *there is at most one* t *such that* $\Gamma \vdash \tilde{t} \triangleleft T \rightsquigarrow t$.

Proof. Like for Theorem 12, those are proven mutually by induction on the typing derivation.

The main argument is that there is always at most one rule that can apply to get a typing conclusion for a given term. This is true for all inference statements because there is exactly one inference rule for each term constructor, and for checking because there is only one rule to derive checking. In those cases simply combining the hypothesis of uniqueness is enough.

For $\blacktriangleright_{\Pi}$, by confluence of CastCIC the inferred type cannot at the same time reduce to $?_{\square}$ and $\Pi x : A.B$, because those do not have a common reduct. Thus, only one of the two rules Inf-Prod and Inf-Prod? can apply. It is enough to conclude for Theorem 13, because reducts of convertible types are still convertible. For Theorem 14 the deterministic reduction strategy ensures that the inferred type is indeed unique, rather than unique up to conversion. The reasoning is similar for the other constrained inference judgments. □

5.3 Illustration: Back to Omega

Now that GCIC has been entirely presented, let us come back to the important example of Ω, and explain in detail the behavior described in § 3.1 for the three GCIC variants.

Recall that Ω is the term $\delta\ \delta$, with $\delta := \lambda\, x : ?\text{@\{i+1\}}.x\ x$. We leave out the casts present in §2 and 3, knowing that they will be introduced by elaboration. We also use $?$ at level $i+1$, because $?\text{@\{i+1\}}$, when elaborated as a type, becomes $T := \langle \square_i \Leftarrow ?_{\square_{i+1}} \rangle\, ?_{?_{\square_{i+1}}}$, such that $T \rightsquigarrow^* ?_{\square_i}$. For the rest of this section, we write $?_j$ instead of $?_{\square_j}$ to avoid stacked indices and ease readability.

If $i = 0$ the elaboration of δ (and thus of Ω) fails in GCIC$^{\uparrow}$ and GCIC$^{\mathcal{N}}$, because the inferred type for x is T, which reduces to $?_0$. Then, because $c_\Pi(0) = -1 < 0$ in both GCIC$^{\uparrow}$ and GCIC$^{\mathcal{N}}$, rule Inf-Prod? does not apply and δ is deemed ill-typed, as is Ω.

Otherwise, if $i > 0$ or we are considering GCIC$^{\mathcal{G}}$, δ can be elaborated, and we have

$$\cdot \vdash \delta \rightsquigarrow \lambda\, x : T.\, \left(\langle \mathrm{germ}_i\ \Pi \Leftarrow T \rangle\, x\right)\ \left(\langle ?_{c_\Pi(i)} \Leftarrow T \rangle\, x\right) \triangleright T \to ?_{c_\Pi(i)}$$

From this, we get that Ω also elaborates, namely (with δ' the elaboration of δ above)

$$\cdot \vdash \Omega \rightsquigarrow \delta'\ \left(\langle T \Leftarrow T \to ?_{c_\Pi(i)} \rangle\, \delta'\right) \triangleright ?_{c_\Pi(i)}$$

Let us now look at the reduction behavior of this elaborated term Ω' in the three systems: it reduces seamlessly when $c_\Pi(i) = i$ (GCIC$^{\mathcal{G}}$/CastCIC$^{\mathcal{G}}$), while having $c_\Pi(i) < i$ makes it fail (GCIC$^{\uparrow}$/CastCIC$^{\uparrow}$ and GCIC$^{\mathcal{N}}$/CastCIC$^{\mathcal{N}}$). The reduction of Ω' in CastCIC$^{\mathcal{G}}$ is as follows:

$$\begin{array}{lll}
\Omega' & \rightsquigarrow^* & (\lambda\, x : ?_i.\, (\langle ?_i \to ?_i \Leftarrow T \rangle\, x)\ (\langle ?_i \Leftarrow T \rangle\, x))\ (\langle T \Leftarrow T \to ?_i \rangle\, \delta') \\
& \rightsquigarrow^* & (\lambda\, x : ?_i.\, (\langle ?_i \to ?_i \Leftarrow ?_i \rangle\, x)\ (\langle ?_i \Leftarrow ?_i \rangle\, x))\, (\langle ?_i \Leftarrow ?_i \to ?_i \rangle\, \delta') \\
& \rightsquigarrow^* & (\langle ?_i \to ?_i \Leftarrow ?_i \Leftarrow ?_i \to ?_i \rangle\, \delta')\ (\langle ?_i \Leftarrow ?_i \Leftarrow ?_i \to ?_i \rangle\, \delta') \\
& \rightsquigarrow^* & (\langle ?_i \to ?_i \Leftarrow ?_i \to ?_i \rangle\, \delta')\ (\langle ?_i \Leftarrow ?_i \to ?_i \rangle\, \delta') \\
& \rightsquigarrow^* & \left(\lambda\, x : ?_i.\, \boxed{\langle ?_i \Leftarrow ?_i \rangle} \left((\langle ?_i \to ?_i \Leftarrow ?_i \rangle\, x)\ (\ \boxed{\langle ?_i \Leftarrow ?_i \rangle}\ x)\right)\right)\ (\langle ?_i \Leftarrow ?_i \to ?_i \rangle\, \delta')
\end{array}$$

The first step is the identity, simply replacing $\Omega', c_\Pi(i)$ and the first occurrence of δ' by their definitions. The second reduces T to $?_i$. In the third, the casted δ' is substituted for x by a β step. Casts are finally simplified using Up-Down and Prod-Prod. At that point, the reduction has almost looped back to the second step, apart from the casts $\boxed{\langle ?_i \Leftarrow ?_i \rangle}$ in the first occurrence of δ', which will simply accumulate through reduction, but without hindering divergence.

On the contrary, the normalizing variants have $c_\Pi(i) < i$, and thus share the following reduction path:

$$\begin{array}{lll}
\Omega' & \rightsquigarrow^* & (\langle ?_{i-1} \to ?_{i-1} \Leftarrow ?_i \Leftarrow ?_i \to ?_{i-1} \rangle\, \delta')\ \left(\boxed{\langle ?_{i-1} \Leftarrow ?_i \Leftarrow ?_i \to ?_{i-1} \rangle}\ \delta'\right) \\
& \rightsquigarrow^* & (\langle ?_{i-1} \to ?_{i-1} \Leftarrow ?_i \Leftarrow ?_i \to ?_{i-1} \rangle\, \delta')\, (\langle ?_{i-1} \Leftarrow ?_{i-1} \to ?_{i-1} \Leftarrow ?_i \to ?_{i-1} \rangle\, \delta') \\
& \rightsquigarrow^* & (\langle ?_{i-1} \to ?_{i-1} \Leftarrow ?_i \Leftarrow ?_i \to ?_{i-1} \rangle\, \delta')\ \mathtt{err}_{?_{i-1}} \\
& \rightsquigarrow^* & (\langle ?_{i-1} \to ?_{i-1} \Leftarrow ?_{i-1} \to ?_{i-1} \Leftarrow ?_i \to ?_{i-1} \rangle\, \delta')\ \mathtt{err}_{?_{i-1}} \\
& \rightsquigarrow^* & (\lambda\, x : ?_{i-1}.\langle ?_{i-1} \Leftarrow ?_{i-1} \Leftarrow ?_{i-1} \rangle\, ((\langle ?_{i-1} \to ?_{i-1} \Leftarrow ?_i \rangle\, x')\ (\langle ?_{i-1} \Leftarrow ?_i \rangle\, x')))\ \mathtt{err}_{?_{i-1}} \\
& & \text{where } x' \text{ is } \langle ?_i \Leftarrow ?_{i-1} \Leftarrow ?_{i-1} \rangle\, x \\
& \rightsquigarrow^* & \langle ?_{i-1} \Leftarrow ?_{i-1} \Leftarrow ?_{i-1} \rangle\, (\mathtt{err}_{?_{i-1} \to ?_{i-1}}\ \mathtt{err}_{?_{i-1}}) \\
& \rightsquigarrow^* & \mathtt{err}_{?_{i-1}}
\end{array}$$

The first step corresponds to the first three above, the only difference being the value of $c_\Pi(i)$. The reductions however differ in the next step because $?_i \to ?_{i-1} \neq \mathrm{germ}_i\ \Pi$, so Prod-Germ applies before Up-Down. For the third step, note that $?_{i-1} \to ?_{i-1} = \mathrm{germ}_i\ \Pi$, so that Down-Err applies in the rightmost sequence of casts. The last three steps of reduction then propagate the error by first using Prod-Germ, Up-Down and Prod-Prod, then the β rule, and finally Down-Err, Prod-Err and a last β step. At a high-level, the error can be seen as a dynamic universe inconsistency, triggered by the invalid downcast $\boxed{\langle ?_{i-1} \Leftarrow ?_i \rangle}$ highlighted on the first line.

5.4 Precision is a simulation for reduction

Establishing the graduality of elaboration—the formulation of the static gradual guarantee (SGG) in our setting—is no small feat, as it requires properties about computations in CastCIC that amount to the *dynamic* gradual guarantee (DGG). Indeed, to handle the typing rules for checking and constrained inference, it is necessary to know how consistency and reduction evolve as a type becomes less precise. As already explained in § 3.4, we cannot directly prove graduality for a syntactic notion of precision. However, we can still show that this relation is a simulation for reduction. While weaker than graduality, this property implies the DGG and suffices to conclude that graduality of elaboration holds. The purpose of this section is to establish it. Our proof is partly inspired by the proof of DGG by Siek et al. [2015].[15] We however had to adapt to the much higher complexity of CIC compared to STLC. In particular, the presence of computation in the domain and codomain of casts is quite subtle to tame, as we must in general reduce types in a cast before we can reduce the cast itself.[16]

Technically, we need to distinguish between two notions of precision, one for GCIC and one for CastCIC: (i) *syntactic precision* on terms in GCIC, which corresponds to the usual syntactic precision of gradual typing [Siek et al. 2015], (ii) *structural precision* on terms in CastCIC, which corresponds to syntactic precision together with a proper account of casts. In this section, we concentrate on properties of structural precision in CastCIC. We only state and discuss the various lemmas and theorems on a high level, and refer the reader to Appendix B.2 for the detailed proofs.

Structural precision for CastCIC*.* As emphasized already, the key property we want to establish is that precision is a simulation for reduction, *i.e.*, that less precise terms reduce at least as well as more precise ones. This property guides the quite involved definition we are about to give for structural precision: it is rigid enough to give the induction hypotheses needed to prove simulation, while being lax enough to be a consequence of syntactic precision after elaboration, which is the key point to establish elaboration graduality (Theorem 24), our equivalent of the static gradual guarantee.

Similarly to $\sim_\alpha$, precision can ignore some casts, in order to handle casts that might appear or disappear in one term but not the other during reduction. But in order to control what casts can be ignored, we impose some restriction on the types involved. In particular, we want to ensure that ignored casts would not have raised an error: *e.g.*, we want to prevent $0 \sqsubseteq_\alpha \langle \mathbb{B} \Leftarrow \mathbb{N} \rangle\, 0$. Thus the definition of structural precision relies on typing, and to do this we need to record the contexts of the two compared terms. We do so by using double-struck letters to denote contexts where each variable is given two types, writing $\mathbb{\Gamma}, x : A \mid A'$ for context extensions. We use $\mathbb{\Gamma}_i$ for projections, *i.e.*, $(\mathbb{\Gamma}, x : A \mid A')_1 := \mathbb{\Gamma}_1, x : A$, and write $\Gamma \mid \Gamma'$ for the converse pairing operation.

Definition 6 (Structural and definitional precision in CastCIC).

Structural precision, *denoted* $\mathbb{\Gamma} \vdash t \sqsubseteq_\alpha t'$*, is defined in Fig. 10, mutually with* definitional precision, *denoted* $\mathbb{\Gamma} \vdash t \sqsubseteq_\leadsto t'$*, which is its closure by reduction. We write* $\Gamma \sqsubseteq_\alpha \Gamma'$ *and* $\Gamma \sqsubseteq_\leadsto \Gamma'$ *for the pointwise extensions of those to contexts.*

Although $\mathbb{\Gamma} \vdash t \sqsubseteq_\leadsto t'$ is defined in a stepwise way, it is equivalent to the existence of s and s' such that $t \leadsto^* s$, $t' \leadsto^* s'$ and $\mathbb{\Gamma} \vdash s \sqsubseteq_\alpha s'$. The situation is the same as for consistency (resp. conversion), which is the closure by reduction of α-consistency (resp. α-equality). However, here definitional precision is also used in the definition of structural precision, in order to permit computation in

[15] Lemma 7 in Siek et al. [2015] is similar to our Theorem 20, and Fig. 10 draws from their Fig. 9, especially for Cast-R and Cast-L. Also, while we do not make them explicit here, Lemmas 8, 10 and 11 also appear in our proofs.

[16] Thus, while Lemmas 17 and 18 correspond roughly to Lemma 9 in Siek et al. [2015], Lemmas 15 and 16 are completely novel.

$$\boxed{\Gamma \vdash t \sqsubseteq_\alpha t'}$$

$$\frac{}{\Gamma \vdash \square_i \sqsubseteq_\alpha \square_i}\textsc{Diag-Univ} \qquad \frac{\Gamma \vdash A \sqsubseteq_\alpha A' \qquad \Gamma, x : A \mid A' \vdash B \sqsubseteq_\alpha B'}{\Gamma \vdash \Pi x : A.B \sqsubseteq_\alpha \Pi x : A'.B'}\textsc{Diag-Prod}$$

$$\frac{\Gamma \vdash A \sqsubseteq_{\leadsto} A' \qquad \Gamma, x : A \mid A' \vdash t \sqsubseteq_\alpha t'}{\Gamma \vdash \lambda x : A.t \sqsubseteq_\alpha \lambda x : A'.t'}\textsc{Diag-Abs} \qquad \frac{\Gamma \vdash t \sqsubseteq_\alpha t' \qquad \Gamma \vdash u \sqsubseteq_\alpha u'}{\Gamma \vdash t\ u \sqsubseteq_\alpha t'\ u'}\textsc{Diag-App}$$

$$\frac{}{\Gamma \vdash x \sqsubseteq_\alpha x}\textsc{Diag-Var} \qquad \frac{\Gamma \vdash A \sqsubseteq_\alpha A' \qquad \Gamma \vdash B \sqsubseteq_\alpha B' \qquad \Gamma \vdash t \sqsubseteq_\alpha t'}{\Gamma \vdash \langle B \Leftarrow A \rangle\, t \sqsubseteq_\alpha \langle B' \Leftarrow A' \rangle\, t'}\textsc{Diag-Cast}$$

$$\frac{\Gamma \vdash \mathbf{a} \sqsubseteq_\alpha \mathbf{a}' \qquad i = i'}{\Gamma \vdash I@\{i\}(\mathbf{a}) \sqsubseteq_\alpha I@\{i'\}(\mathbf{a}')}\textsc{Diag-Ind} \qquad \frac{\Gamma \vdash \mathbf{a} \sqsubseteq_\alpha \mathbf{a}' \qquad \Gamma \vdash \mathbf{b} \sqsubseteq_\alpha \mathbf{b}' \qquad i = i'}{\Gamma \vdash c@\{i\}(\mathbf{a}, \mathbf{b}) \sqsubseteq_\alpha c@\{i'\}(\mathbf{a}, \mathbf{b})}\textsc{Diag-Cons}$$

$$\frac{\begin{array}{c}\Gamma \vdash s \sqsubseteq_\alpha s' \qquad \Gamma_1 \vdash s \blacktriangleright_{\mathrm{I}} I(\mathbf{a}) \qquad \Gamma_2 \vdash s' \blacktriangleright_{\mathrm{I}} I(\mathbf{a}') \qquad \Gamma, z : I(\mathbf{a}) \mid I(\mathbf{a}') \vdash P \sqsubseteq_\alpha P' \\ \Gamma, f : (\Pi z : I(\mathbf{a}), P) \mid (\Pi z : I(\mathbf{a}'), P'), \mathbf{y} : \mathbf{Y_k}[\mathbf{a}/\mathbf{x}] \mid \mathbf{Y_k}[\mathbf{a}'/\mathbf{x}] \vdash t_k \sqsubseteq_\alpha t'_k\end{array}}{\Gamma \vdash \mathtt{ind}_I(s, z.P, \mathbf{f.y.t}) \sqsubseteq_\alpha \mathtt{ind}_I(s', z.P', \mathbf{f.y.t}')}\textsc{Diag-Fix}$$

$$\frac{\Gamma_1 \vdash t \triangleright T \qquad \Gamma \vdash T \sqsubseteq_{\leadsto} A' \qquad \Gamma \vdash T \sqsubseteq_{\leadsto} B' \qquad \Gamma \vdash t \sqsubseteq_\alpha t'}{\Gamma \vdash t \sqsubseteq_\alpha \langle B' \Leftarrow A' \rangle\, t'}\textsc{Cast-R}$$

$$\frac{\Gamma_2 \vdash t' \triangleright T' \qquad \Gamma \vdash A \sqsubseteq_{\leadsto} T' \qquad \Gamma \vdash B \sqsubseteq_{\leadsto} T' \qquad \Gamma \vdash t \sqsubseteq_\alpha t'}{\Gamma \vdash \langle B \Leftarrow A \rangle\, t \sqsubseteq_\alpha t'}\textsc{Cast-L}$$

$$\frac{\Gamma_1 \vdash t \triangleright T \qquad \Gamma \vdash T \sqsubseteq_{\leadsto} T'}{\Gamma \vdash t \sqsubseteq_\alpha ?_{T'}}\textsc{Unk} \qquad \frac{\Gamma_1 \vdash A \blacktriangleright_{\square} \square_i \qquad i \leq j}{\Gamma \vdash A \sqsubseteq_\alpha ?_{\square_j}}\textsc{Unk-Univ}$$

$$\frac{\Gamma_2 \vdash t' \triangleright T' \qquad \Gamma \vdash T \sqsubseteq_{\leadsto} T'}{\Gamma \vdash \mathtt{err}_T \sqsubseteq_\alpha t'}\textsc{Err}$$

$$\frac{\Gamma_1 \vdash t' \blacktriangleright_{\Pi} \Pi x : A'.B' \qquad \Gamma \vdash \Pi x : A.B \sqsubseteq_{\leadsto} \Pi x : A'.B'}{\Gamma \vdash \lambda x : A.\, \mathtt{err}_B \sqsubseteq_\alpha t'}\textsc{Err-Lambda}$$

$$\boxed{\Gamma \vdash t \sqsubseteq_{\leadsto} t'}$$

$$\frac{\Gamma \vdash t \sqsubseteq_\alpha t'}{\Gamma \vdash t \sqsubseteq_{\leadsto} t'} \qquad \frac{\Gamma \vdash s \sqsubseteq_{\leadsto} t' \qquad t \leadsto s}{\Gamma \vdash t \sqsubseteq_{\leadsto} t'} \qquad \frac{\Gamma \vdash t \sqsubseteq_{\leadsto} s' \qquad t' \leadsto s'}{\Gamma \vdash t \sqsubseteq_{\leadsto} t'}$$

Fig. 10. Structural precision in CastCIC

types—recall that in a dependently-typed setting the two types involved in a cast may need to reduce before the cast itself can reduce—and thus the two notions are mutually defined.

Let us now explain the rules defining structural precision. Diagonal rules are completely structural, apart from the Diag-Fix rule, where typing assumptions provide us with the contexts needed to compare the predicates. More interesting are the non-diagonal rules. First, $?_T$ is greater than any term of the "right type". This incorporates loss of precision (rule Unk), and accommodates for a small bit of cumulativity (rule Unk-Univ). This is needed because of technical reasons linked with possibility to form products between types at different levels. On the contrary, the error is smaller than any term (rule Err), even in its extended form on Π-types (rule Err-Lambda), with a typing premise similar to that of rule Unk. Finally, casts on the right-hand side can be ignored as long as they are performed on types that are *less* precise than the type of the term on the left (rule Cast-R). Dually, casts on the left-hand side can be ignored as long as they are performed on types that are *more* precise than the type of the term on the right (rule Cast-L).

Catch-up lemmas. The fact that structural precision is a simulation relies on a series of lemmas that all have the same form: under the assumption that a term t' is less precise than a term t with a known head ($\square$, Π, I, λ or c), the term t' can be reduced to a term that either has the same head, or is some $?$. We call these *catch-up* lemmas, as they enable the less precise term to catch up to the more precise one whose head is already known. Their aim is to ensure that casts appearing in a less precise term never block reduction, as they can always be reduced away.

The lemmas are established in a descending fashion: first, on the universe in Lemma 15, then on other types in Lemma 16, and finally on terms, namely on λ-abstractions in Lemma 17 and inductive constructors in Lemma 18. Each time, the previously proven catch-up lemmas are used to reduce types in casts appearing in the less precise term, apart from Lemma 15, where the induction hypothesis of the lemma being proven is used instead.

Lemma 15 (Universe catch-up).
Under the hypothesis that $\mathbb{\Gamma}_1 \sqsubseteq_\alpha \mathbb{\Gamma}_2$, if $\mathbb{\Gamma} \vdash \square_i \sqsubseteq_{\leadsto} T'$ and $\mathbb{\Gamma}_2 \vdash T' \blacktriangleright_\square \square_j$, either $T' \leadsto^* ?_{\square_j}$ with $i < j$, or $T' \leadsto^* \square_i$.

Lemma 16 (Types catchup). *Under the hypothesis that $\mathbb{\Gamma}_1 \sqsubseteq_\alpha \mathbb{\Gamma}_2$, we have the following:*

- *if $\mathbb{\Gamma} \vdash ?_{\square_i} \sqsubseteq_\alpha T'$ and $\mathbb{\Gamma}_2 \vdash T' \blacktriangleright_\square \square_j$, then $T' \leadsto^* ?_{\square_j}$ and $i \leq j$;*
- *if $\mathbb{\Gamma} \vdash \Pi x : A.B \sqsubseteq_\alpha T'$, $\mathbb{\Gamma}_1 \vdash \Pi x : A.B \triangleright \square_i$ and $\mathbb{\Gamma}_2 \vdash T' \blacktriangleright_\square \square_j$ then either $T' \leadsto^* ?_{\square_j}$ and $i \leq j$, or $T' \leadsto^* \Pi x : A'.B'$ for some A' and B' such that $\mathbb{\Gamma} \vdash \Pi x : A.B \sqsubseteq_\alpha \Pi x : A'.B'$;*
- *if $\mathbb{\Gamma} \vdash I(\mathbf{a}) \sqsubseteq_\alpha T'$, $\mathbb{\Gamma}_1 \vdash I(\mathbf{a}) \triangleright \square_i$ and $\mathbb{\Gamma}_2 \vdash T' \blacktriangleright_\square \square_j$ then either $T' \leadsto^* ?_{\square_j}$ and $i \leq j$, or $T' \leadsto^* I(\mathbf{a}')$ for some $\mathbf{a}'$ such that $\mathbb{\Gamma} \vdash I(\mathbf{a}) \sqsubseteq_\alpha I(\mathbf{a}')$.*

Lemma 17 (λ-abstraction catch-up).
If $\mathbb{\Gamma} \vdash \lambda x : A.t \sqsubseteq_\alpha s'$, where t is not an error, $\mathbb{\Gamma}_1 \vdash \lambda x : A.t \triangleright \Pi x : A.B$ and $\mathbb{\Gamma}_2 \vdash s' \blacktriangleright_\Pi \Pi x : A'.B'$, then $s' \leadsto^* \lambda x : A'.t'$ with $\mathbb{\Gamma} \vdash \lambda x : A.t \sqsubseteq_\alpha \lambda x : A'.t'$.

This holds in CastCIC$^{\mathcal{G}}$, CastCIC$^{\uparrow}$, *and for terms without $?$ in* CastCIC$^{\mathcal{N}}$.

Lemma 18 (Constructors and inductive error catch-up).
If $\mathbb{\Gamma} \vdash c(\mathbf{a}, \mathbf{b}) \sqsubseteq_\alpha s'$, $\mathbb{\Gamma}_1 \vdash c(\mathbf{a}, \mathbf{b}) \triangleright I(\mathbf{a})$ and $\mathbb{\Gamma}_2 \vdash s' \blacktriangleright_I I(\mathbf{a}')$, then either $s' \leadsto^* ?_{I(\mathbf{a}')}$ or $s' \leadsto^* c(\mathbf{a}', \mathbf{b}')$ with $\mathbb{\Gamma} \vdash c(\mathbf{a}, \mathbf{b}) \sqsubseteq_\alpha c(\mathbf{a}', \mathbf{b}')$.
Similarly, if $\mathbb{\Gamma} \vdash ?_{I(\mathbf{a})} \sqsubseteq_\alpha s'$, $\mathbb{\Gamma}_1 \vdash ?_{I(\mathbf{a})} \triangleright I(\mathbf{a})$ and $\mathbb{\Gamma}_2 \vdash s' \blacktriangleright_I I(\mathbf{a}')$, then $s' \leadsto^* ?_{I(\mathbf{a}')}$ with $\mathbb{\Gamma} \vdash I(\mathbf{a}) \sqsubseteq_{\leadsto} I(\mathbf{a}')$.

Note that for Lemma 18, we need to deal with unknown terms specifically, which is not necessary for Lemma 17 because the unknown term in a Π-type reduces to a λ-abstraction.

Lemma 17 deserves a more extensive discussion, because it is the critical point where the difference between the three variants of CastCIC manifests. In fact, it does not hold in full generality for CastCIC$^{\mathcal{N}}$. Indeed, the fact that $i \leq c_\Pi(s_\Pi(i, j))$ and $j \leq c_\Pi(s_\Pi(i, j))$ is used crucially to ensure that casting from a Π-type into $?$ and back does not reduce to an error, given the restrictions on types in Cast-R. This is the manifestation in the reduction of the embedding-projection property [New and Ahmed 2018]. In CastCIC$^{\mathcal{N}}$ it holds only if one restricts to terms without $?$, where such casts never happen. This is important with regard to conservativity, as elaboration produces terms with casts but without $?$, and Lemma 17 ensures that for those precision is still a simulation, even in CastCIC$^{\mathcal{N}}$.

Example 19 (Catch-up of λ-abstraction). The following term t_i illustrates these differences

$$t_i := \langle \mathbb{N} \to \mathbb{N} \Leftarrow ?_{\square_i} \Leftarrow \mathbb{N} \to \mathbb{N} \rangle\, \lambda\, x : \mathbb{N}.\mathsf{suc}(x)$$

where $\mathbb{N}$ is taken at the lowest level, *i.e.*, to mean $\mathbb{N}@\{0\}$. Such terms appear naturally whenever a loss of precision happens on a function, for instance when elaborating a term like $((\lambda\, x : \mathbb{N}.\mathsf{suc}(x)) :: ?)\ 0$. Now this term t_i always reduces to

$$\langle \mathbb{N} \to \mathbb{N} \Leftarrow \mathrm{germ}_i\ \Pi \Leftarrow ?_{\square_i} \Leftarrow \mathrm{germ}_i\ \Pi \Leftarrow \mathbb{N} \to \mathbb{N} \rangle\, \lambda\, x : \mathbb{N}.\mathsf{suc}(x)$$

and at this point the difference kicks in: if $\mathrm{germ}_i\ \Pi$ is $\mathtt{err}_{?_{\square_i}}$ (*i.e.*, if $c_\Pi(i) < 0$) then the whole term reduces to $\mathtt{err}_{\mathbb{N}\to\mathbb{N}}$. Otherwise, further reductions finally give

$$\lambda\, x : \mathbb{N}.\mathsf{suc}\,(\langle \mathbb{N} \Leftarrow \mathbb{N} \Leftarrow \mathbb{N} \rangle\, x)$$

Although the body is blocked by the variable x, applying the function to 0 would reduce to 1 as expected. Let us compare what happens in the three systems.

In all of them, if $i \geq 1$, we have $\vdash \lambda\, x : \mathbb{N}.\mathsf{suc}(x) \sqsubseteq_\alpha t_i$ via repeated uses of Cast-R since $\vdash \mathbb{N} \rightsquigarrow \mathbb{N} \triangleright \square_{s_\Pi(0,0)}$ and $s_\Pi(0,0) \leq 1 \leq i$. Moreover, also $0 \leq i-1 \leq c_\Pi(i)$ and so the reduction is errorless. Thus Lemma 17 holds in all three systems when $i \geq 1$.

The difference appears in the specific case where $i = 0$. In $\mathsf{CastCIC}^{\mathcal{G}}$ and $\mathsf{CastCIC}^{\mathcal{N}}$, we still have $\vdash \lambda\, x : \mathbb{N}.\mathsf{suc}(x) \sqsubseteq_\alpha t_0$, since $s_\Pi(0,0) = 0 \leq i$. In the former, $c_\Pi(0) = 0$ so t_0 reduces safely and Lemma 17 holds. In the latter, however, $c_\Pi(0) = -1$, and so t_0 errors even if it is less precise than an errorless term—Lemma 17 does not hold in that case. Finally, in $\mathsf{CastCIC}^{\uparrow}$, t_0 errors since again $c_\Pi(0) = -1$. However, because $s_\Pi(0,0) = 1$, t_0 is not less precise than $\lambda\, x : \mathbb{N}.\mathsf{suc}(x)$ thanks to the typing restriction in Cast-R, so this error does not contradict Lemma 17.

Note that in an actual implementation with typical ambiguity (Footnote 14), the case where $i = 0$ would most likely not manifest: elaborating $((\lambda\, x : \mathbb{N}.\mathsf{suc}(x)) :: ?)\ 0$ would produce a fresh level that could be chosen high enough so as to prevent the error we just described. Only more involved situations like that of Ω (§ 5.3) would actually exhibit failures due to universe levels, which are precisely those unavoidable to ensure normalization.

Simulation. We finally come to the main property of this section, the advertised simulation. Remark that the simulation property needs to be stated (and proven) mutually for structural and definitional precision, but it is really informative only for structural precision (definitional precision is somehow a simulation by construction).

Theorem 20 (Precision is a simulation for reduction). *Let $\mathbb{\Gamma}_1 \sqsubseteq_{\rightsquigarrow} \mathbb{\Gamma}_2$, $\mathbb{\Gamma}_1 \vdash t \triangleright T$, $\mathbb{\Gamma}_2 \vdash u \triangleright U$ and $t \rightsquigarrow^* t'$. Then*

- *if $\mathbb{\Gamma} \vdash t \sqsubseteq_\alpha u$ then there exists u' such that $u \rightsquigarrow^* u'$ and $\mathbb{\Gamma} \vdash t' \sqsubseteq_\alpha u'$;*
- *if $\mathbb{\Gamma} \vdash t \sqsubseteq_{\rightsquigarrow} u$ then $\mathbb{\Gamma} \vdash t' \sqsubseteq_{\rightsquigarrow} u$.*

This holds in $\mathsf{CastCIC}^{\mathcal{G}}$, $\mathsf{CastCIC}^{\uparrow}$ and for terms without $?$ in $\mathsf{CastCIC}^{\mathcal{N}}$.

Proof sketch. The case of definitional precision follows by confluence of reduction. For the case of structural precision, the hardest point is to simulate β and ι redexes—terms of the shape $\mathtt{ind}_I(c(\mathbf{a}), z.P, f.\mathbf{y}.\mathbf{t})$. This is where we use Lemmas 17 and 18, to show that similar reductions can also happen in t'. We must also put some care into handling the premises of precision where typing is involved. In particular, subject reduction is needed to relate the types inferred after reduction to the type inferred before, and the mutual induction hypothesis on $\sqsubseteq_{\rightsquigarrow}$ is used to conclude that the premises holding on t still hold on t'. Finally, the restriction to terms without $?$ in $\mathsf{CastCIC}^{\mathcal{N}}$ similar to Lemma 17 appears again when treating Up-Down, where having $c_\Pi(s_\Pi(i,i)) = i$ is required. □

From this theorem, we get as direct corollaries the following properties, that are required to handle reduction (Corollary 21) and consistency (Corollary 22) in elaboration. Again those corollaries hold in $\mathrm{GCIC}^{\mathcal{G}}$, $\mathrm{GCIC}^{\uparrow}$ and for terms in $\mathrm{GCIC}^{\mathcal{N}}$ containing no $?$.

Corollary 21 (Monotonicity of reduction to type constructor). *Let $\mathbb{\Gamma}$, T and T' be such that $\mathbb{\Gamma}_1 \vdash T \blacktriangleright_{\square} \square_i$, $\mathbb{\Gamma}_2 \vdash T' \blacktriangleright_{\square} \square_j$, $\mathbb{\Gamma} \vdash T \sqsubseteq_\alpha T'$. Then*

- *if $T \rightsquigarrow^* ?_{\square_i}$ then $T' \rightsquigarrow^* ?_{\square_j}$ with $i \leq j$;*
- *if $T \rightsquigarrow^* \square_{i-1}$ then either $T' \rightsquigarrow^* ?_{\square_j}$ with $i \leq j$, or $T' \rightsquigarrow^* \square_{i-1}$;*
- *if $T \rightsquigarrow^* \Pi x : A.B$ then either $T' \rightsquigarrow^* ?_{\square_j}$ with $i \leq j$, or $T' \rightsquigarrow^* \Pi x : A'.B'$ and $\mathbb{\Gamma} \vdash \Pi x : A.B \sqsubseteq_\alpha \Pi x : A'.B'$;*
- *if $T \rightsquigarrow^* I(\mathbf{a})$ then either $T' \rightsquigarrow^* ?_{\square_j}$ with $i \leq j$, or $T' \rightsquigarrow^* I(\mathbf{a}')$ and $\mathbb{\Gamma} \vdash I(\mathbf{a}) \sqsubseteq_\alpha I(\mathbf{a}')$.*

Proof. It suffices to simulate the reductions of T by using Theorem 20, and then use Lemmas 15 and 16 to conclude. Note that head reductions are simulated using head reductions in Theorem 20, and the reductions of Lemmas 15 and 16 are also head reductions. Thus the corollary still holds when fixing weak-head reduction as a reduction strategy. □

Corollary 22 (Monotonicity of consistency). *If $\mathbb{\Gamma} \vdash T \sqsubseteq_\alpha T'$, $\mathbb{\Gamma} \vdash S \sqsubseteq_\alpha S'$ and $T \sim S$ then $T' \sim S'$.*

Proof. By definition of $\sim$, we get some U and V such that $T \rightsquigarrow^* U$ and $S \rightsquigarrow^* V$, and $U \sim_\alpha V$. By Theorem 20, we can simulate these reductions to get some U' and V' such that $T' \rightsquigarrow^* U'$ and $S' \rightsquigarrow^* V'$, and also $\mathbb{\Gamma}_1 \vdash U \sqsubseteq_\alpha U'$ and $\mathbb{\Gamma}_1 \vdash V \sqsubseteq_\alpha V'$. Thus we only need to show that α-consistency is monotone with respect to structural precision, which is direct by induction on structural precision. □

5.5 Properties of GCIC

We now have enough technical tools to prove most of the properties of GCIC. We state those theorems in an empty context in this section to make them more readable, but they are of course corollaries of similar statements including contexts, proven by mutual induction. The complete statements and proofs can be found in Appendix B.3.

Conservativity with respect to CIC. Elaboration systematically inserts casts during checking, thus even static terms are not elaborated to themselves. Therefore we use a (partial) erasure function ε that translates terms of CastCIC to terms of CIC by erasing all casts. We also introduce the notion of erasability, characterizing terms that contain "harmless" casts, such that in particular the elaboration of a static term is always erasable.

Definition 7 (Equiprecision). *Two terms s and t are equiprecise in a context $\mathbb{\Gamma}$, denoted $\mathbb{\Gamma} \vdash s \sqsupseteq\sqsubseteq_\alpha t$ if both $\mathbb{\Gamma} \vdash s \sqsubseteq_\alpha t$ and $\mathbb{\Gamma} \vdash t \sqsubseteq_\alpha s$.*

Definition 8 (Erasure, erasability). *Erasure ε is a partial function from the syntax of* CastCIC *to the syntax of* CIC*, which is undefined on $?$ and $\mathtt{err}$, is such that $\varepsilon(\langle B \Leftarrow A \rangle\, t) = \varepsilon(t)$, and is a congruence for all other term constructors.*

Given a context $\mathbb{\Gamma}$ we say that a term well-typed in $\mathbb{\Gamma}_1$ t is erasable if $\varepsilon(t)$ is defined, well-typed in $\mathbb{\Gamma}_2$, and equiprecise to t in $\mathbb{\Gamma}$. Similarly a context Γ is called erasable if it is pointwise erasable. When Γ is erasable, we say that a term t is erasable in Γ to mean that it is erasable in $\Gamma \mid \varepsilon(\Gamma)$.

Conservativity holds in all three systems, typeability being of course taken into the corresponding variant of CIC: full CIC for $\mathrm{GCIC}^{\mathcal{G}}$ and $\mathrm{GCIC}^{\mathcal{N}}$, and $\mathrm{CIC}^{\uparrow}$ for $\mathrm{GCIC}^{\uparrow}$.

Theorem 23 (Conservativity). *Let $\tilde{t}$ be a static term (*i.e.*, is a term of* GCIC *that is also a term of* CIC*). If $\vdash_{\mathrm{CIC}} \tilde{t} \triangleright T$ for some type T, then there exists t and T' such that $\vdash \tilde{t} \rightsquigarrow t \triangleright T'$, and moreover $\varepsilon(t) = \tilde{t}$ and $\varepsilon(T') = T$. Conversely if $\vdash \tilde{t} \rightsquigarrow t \triangleright T$ for some t and T, then $\vdash_{\mathrm{CIC}} \tilde{t} \triangleright \varepsilon(T)$.*

$$\frac{}{x \sqsubseteq^{G}_{\alpha} x} \qquad \frac{}{\square_i \sqsubseteq^{G}_{\alpha} \square_i} \qquad \frac{A \sqsubseteq^{G}_{\alpha} A' \quad B \sqsubseteq^{G}_{\alpha} B'}{\Pi x : A.B \sqsubseteq^{G}_{\alpha} \Pi x : A'.B'} \qquad \frac{A \sqsubseteq^{G}_{\alpha} A' \quad t \sqsubseteq^{G}_{\alpha} t'}{\lambda x : A.t \sqsubseteq^{G}_{\alpha} \lambda x : A.t}$$

$$\frac{t \sqsubseteq^{G}_{\alpha} t' \quad u \sqsubseteq^{G}_{\alpha} u'}{t\ u \sqsubseteq^{G}_{\alpha} t'\ u'} \qquad \frac{\mathbf{a} \sqsubseteq^{G}_{\alpha} \mathbf{a}'}{I(\mathbf{a}) \sqsubseteq^{G}_{\alpha} I(\mathbf{a}')} \qquad \frac{\mathbf{a} \sqsubseteq^{G}_{\alpha} \mathbf{a}' \quad \mathbf{b} \sqsubseteq^{G}_{\alpha} \mathbf{b}'}{c(\mathbf{a},\mathbf{b}) \sqsubseteq^{G}_{\alpha} c(\mathbf{a}',\mathbf{b}')}$$

$$\frac{a \sqsubseteq^{G}_{\alpha} a' \quad P \sqsubseteq^{G}_{\alpha} P' \quad \mathbf{t} \sqsubseteq^{G}_{\alpha} \mathbf{t}'}{\mathtt{ind}_I(a, z.P, \mathbf{f.y.t}) \sqsubseteq^{G}_{\alpha} \mathtt{ind}_I(a', z.P', \mathbf{f.y.t}')} \qquad \frac{}{t \sqsubseteq^{G}_{\alpha} ?}$$

Fig. 11. Syntactic precision for GCIC

Proof sketch. Because t is static, its typing derivation in GCIC can only use rules that have a counterpart in CIC, and conversely all rules of CIC have a counterpart in GCIC. The only difference is about the reduction/conversion side conditions, which are used on elaborated types in GCIC, rather than their non-elaborated counterparts in CIC.

Thus, the main difficulty is to ensure that the extra casts inserted by elaboration do not alter reduction. For this we maintain the property that all terms t considered in CastCIC are erasable, and in particular that any static term t that elaborates to some t is such that $\varepsilon(t) = t$. From the simulation property of structural precision (Theorem 20), we get that an erasable term t has the same reduction behavior as its erasure, *i.e.*, if $t \rightsquigarrow^* s$ then $\varepsilon(t) \rightsquigarrow^* s'$ with s' and s equiprecise, and conversely if $\varepsilon(t) \rightsquigarrow^* s'$ then $t \rightsquigarrow^* s$ with s' and s equiprecise. Using that property, we prove that constraint reductions ($\blacktriangleright_{\Pi}$, $\blacktriangleright_{\square}$ and $\blacktriangleright_{\mathrm{I}}$) in CastCIC and CIC behave the same on static terms. □

Elaboration Graduality. Next, we turn to elaboration graduality, the equivalent of the static gradual guarantee (SGG) of Siek et al. [2015] in our setting. We state it with respect to a notion of precision for terms in GCIC, *syntactic precision* $\sqsubseteq^{G}_{\alpha}$, defined in Fig. 11. Syntactic precision is the usual and expected source-level notion of precision in gradual languages: it is generated by a single non-trivial rule $t \sqsubseteq^{G}_{\alpha} ?_{@\{i\}}$, and congruence rules for all term formers.

In contrast with the simply-typed setting, the presence of multiple unknown types $?$, one for each universe level i, requires an additional hypothesis relating elaboration and precision. We say that two judgments $\tilde{t} \sqsubseteq^{G}_{\alpha} ?_{@\{i\}}$ and $\Gamma \vdash \tilde{t} \rightsquigarrow t \triangleright T$ are *universe adequate* if the universe level j given by the well-formedness judgment $\Gamma \vdash T \blacktriangleright_{\square} \square_j$ induced by correction of the elaboration satisfies $i = j$. More generally, $\tilde{t} \sqsubseteq^{G}_{\alpha} \tilde{s}$ and $\vdash \tilde{t} \rightsquigarrow t \triangleright T$ are *universe adequate* if for any subterm $\tilde{t}_0$ of $\tilde{t}$ inducing judgments $\tilde{t}_0 \sqsubseteq^{G}_{\alpha} ?_{@\{i\}}$ and $\Gamma_0 \vdash \tilde{t}_0 \rightsquigarrow t \triangleright T$, those are universe adequate. Note that this extraneous technical assumption on universe levels is not needed if we use typical ambiguity (Footnote 14), since universe levels are not given explicitly.

Theorem 24 (Elaboration Graduality / Static Gradual Guarantee). *In* GCIC$^{\mathcal{G}}$ *and* GCIC$^{\uparrow}$, *if* $\tilde{t} \sqsubseteq^{G}_{\alpha} \tilde{s}$ *and* $\vdash \tilde{t} \rightsquigarrow t \triangleright T$ *are universe adequate, then* $\vdash \tilde{s} \rightsquigarrow s \triangleright S$ *for some* s *and* S *such that* $\vdash t \sqsubseteq_{\alpha} s$ *and* $\vdash T \sqsubseteq_{\alpha} S$.

Proof sketch. The proof is by induction on the elaboration derivation for $\tilde{t}$. All cases for inference consist in a straightforward combination of the hypotheses, with the universe adequacy hypothesis used in the case where $\tilde{s}$ is $?_{@\{i\}}$. Here again the technical difficulties arise in the rules involving reduction. This is where Corollary 21 is useful, proving that the less structurally precise term obtained by induction in a constrained inference reduces to a less precise type. Thus either the same rule can still be used, or one has to trade a Inf-Unk, Inf-Prod or Inf-Ind rule respectively for a Inf-Univ?, Inf-Prod? or Inf-Ind? rule in case the less precise type is some $?_{\square_i}$ and the more precise type is not. Similarly, Corollary 22 proves that in the checking rule the less precise types are still consistent. Note that again, because Corollary 21 holds when restricted to weak-head reduction, elaboration graduality also holds when fixing a weak-head strategy in Fig. 9. □

Dynamic Gradual Guarantee. Following Siek et al. [2015], using the fact that structural precision is a simulation (Theorem 20), we can prove the DGG for $\mathsf{CastCIC}^{\mathcal{G}}$ and $\mathsf{CastCIC}^{\uparrow}$ (stated using the notion of observational refinement $\sqsubseteq^{obs}$ from Definition 4).

Theorem 25 (Dynamic Gradual Guarantee for $\mathsf{CastCIC}^{\mathcal{G}}$ and $\mathsf{CastCIC}^{\uparrow}$). *Suppose that* $\Gamma \vdash t \triangleright A$ *and* $\Gamma \vdash u \triangleright A$. *If moreover* $\Gamma \mid \Gamma \vdash t \sqsubseteq_{\alpha} u$ *then* $t \sqsubseteq^{obs} u$.

Proof. Let $C : (\Gamma \vdash A) \Rightarrow (\vdash \mathbb{B})$ closing over all free variables. By the diagonal rules of structural precision, we have $\Gamma \mid \Gamma \vdash C[t] \sqsubseteq_{\alpha} C[u]$. By progress (Theorem 8), $C[t]$ either reduces to $\mathtt{true}$, $\mathtt{false}$, $?_{\mathbb{B}}$, $\mathtt{err}_{\mathbb{B}}$ or diverges, and similarly for $C[u]$. If $C[t]$ diverges or reduces to $\mathtt{err}_{\mathbb{B}}$, we are done. If it reduces to either $\mathtt{true}$, $\mathtt{false}$ or $?_{\mathbb{B}}$, then by the catch-up Lemma 18, $C[u]$ either reduces to the same value, or to $?_{\mathbb{B}}$. In particular, it cannot diverge or reduce to an error. □

Note that Example 19 provides a counter-example to this theorem for $\mathsf{CastCIC}^{\mathcal{N}}$, by choosing the context $\mathtt{ind}_{\mathbb{N}}(\bullet\ 0, z.\mathbb{B}, f.\mathtt{true}, f.n.\mathtt{true})$, because in that context the function $\lambda\, x : \mathbb{N}.\mathtt{suc}(x)$ reduces to $\mathtt{true}$ while the less precise casted function reduces to $\mathtt{err}_{\mathbb{B}}$.

As observed in §2.4, graduality—and in particular the fact that precision induces ep-pairs—is inherently semantic, and thus cannot rely on the syntactic precision $\sqsubseteq^{\mathrm{G}}_{\alpha}$ introduced in this section. Therefore, we defer the proof of $\mathcal{G}$ for $\mathsf{CastCIC}^{\uparrow}$ and $\mathsf{CastCIC}^{\mathcal{G}}$ to the next section, where the semantic notion of *propositional precision* is introduced.

6 REALIZING CastCIC AND GRADUALITY

To prove normalization of $\mathsf{CastCIC}^{\mathcal{N}}$ and $\mathsf{CastCIC}^{\uparrow}$, we now build a model of both theories with a simple implementation of casts using case-analysis on types as well as exceptions, yielding the *discrete model*, allowing us to reduce the normalization of both theories to the normalization of the target theory (§6.1).

Then, to prove graduality of $\mathsf{CastCIC}^{\uparrow}$, we build a more elaborate *monotone model* inducing a precision relation well-behaved with respect to conversion. Following generalities about the interpretation of CIC's types as posets in §6.2, we describe the construction of a monotone unknown type $\ddot{?}$ in §6.3 and a hierarchy of universes in §6.4 and put these pieces together in §6.5, culminating in a proof of graduality for $\mathsf{CastCIC}^{\uparrow}$ (§6.6). In both the discrete and monotone case, the parameters $c_{\Pi}(-)$ and $s_{\Pi}(-,-)$ appear when building the hierarchy of universes and tying the knot with the unknown type.

Finally, to deduce graduality for the non-terminating variant, $\mathsf{CastCIC}^{\mathcal{G}}$, we describe at the end of this section a model based on ω-complete partial orders, extending the seminal model of Scott [1976] for λ-calulus to $\mathsf{CastCIC}^{\mathcal{G}}$ (§6.7).

The discrete model embeds into a variant of CIC extended with induction-recursion [Dybjer and Setzer 2003], noted $\mathsf{CIC}^{\mathsf{IR}}$, and the monotone model into a variant that additionally features quotients (and hence also function extensionality [Shulman 2011]), noted $\mathsf{CIC}^{\mathsf{IR}}_{\mathsf{QIT}}$.

Formalization in Agda. We use Agda [Norell 2009] as a practical tool to typecheck the components of the models and assume that Agda satisfies standard metatheoretical properties, namely subject reduction and strong normalization.

The correspondence between the notions developed in the following sections and the formal development in Agda [Lennon-Bertrand et al. 2020] is as follows. The formalization covers most component of the discrete (`DiscreteModelPartial.agda`) and monotone model (`UnivPartial.agda`) in a partial (non-normalizing) setting and only the discrete model is proved to be normalizing assuming normalization of the type theory implemented by Agda (no escape hatch to termination checking is used in `DiscreteModelTotal`). The main definitions surrounding posets can be found in `Poset.agda`: top and bottom elements (called `Initial` and `Final` in the formalization), embedding-projection pairs (called `Distr`) as well as the notions corresponding to indexed families of posets (`IndexedPoset`, together with `IndexedDistr`). It is then proved that we endow can the translation of each type formers from CastCIC with a poset structure: natural numbers in `nat.agda`, booleans in `bool.agda`, dependent product in `pi.agda`. The definition of the monotone unknown type $\ddot{?}$ is defined in the subdirectory `Unknown/`. It is more involved since we need to use a quotient (that we axiomatize together with a rewriting rule in `Unknown/Quotient.agda`). Finally, all these building blocks are put together when assembling the inductive-recursive hierarchies of universes (`UnivPartial.agda`, `DiscreteModelPartial.agda` and `DiscreteModelTotal.agda`).

6.1 Discrete Model of CastCIC

The discrete model explains away the new term formers of CastCIC (Syntax of CastCIC) by a translation into CIC using two important ingredients from the literature:

- Exceptions, following the approach of ExTT [Pédrot and Tabareau 2018]: each inductive type is extended with two new constructors, one for ? and one for `err`. As alluded to early on (§ 2.5), both ? and `err` are exceptional terms in their propagation semantics, and only differ in their static interpretation: $?_{\mathsf{A}}$ is consistent with any other term of type A, while $\mathtt{err}_{\mathsf{A}}$ is not consistent with any such term.
- Case analysis on types [Boulier et al. 2017] to define the cast operator. The essence of the translation is to interpret types as *codes* when they are seen as terms, and as *the semantics of those codes* when they are seen as types. This allows us to get the standard interpretation for a term inhabiting a type, but at the same time, it allows functions taking terms in the universe $\square_i$ to perform a case analysis on the code of the type, because this time, the type is seen as a term in $\square_i$.

The latter ingredient for intensional type analysis requires the target theory of the translation to be an extension of CIC with induction-recursion [Dybjer and Setzer 2003], noted $\mathsf{CIC}^{\mathsf{IR}}$. We write $\rightsquigarrow_{\mathrm{IR}}$ and $\vdash_{\mathrm{IR}}$ to denote the reduction and typing judgments of $\mathsf{CIC}^{\mathsf{IR}}$, respectively.

Inductive types. Following the general pattern of ExTT, we interpret each inductive type I by an inductive type $\ddot{I}$ featuring all constructors of I and extended with two new constructors $\top_{\ddot{I}}$ and $\bot_{\ddot{I}}$, corresponding respectively to $?_I$ and $\mathtt{err}_I$ of CastCIC. The constructors $\top_{\ddot{I}}$ and $\bot_{\ddot{I}}$ of $\ddot{I}$ are called *exceptional* by opposition to the other constructors that we call *non-exceptional*. For instance, the inductive type used to interpret natural numbers, $\ddot{\mathbb{N}}$, thus has 4 constructors: the non-exceptional constructors 0 and `suc`, and the exceptional constructors $\top_{\ddot{\mathbb{N}}}$, $\bot_{\ddot{\mathbb{N}}}$. In the rest of this section, we only illustrate inductive types on natural numbers.

Universe and type-case. Case analysis on types is obtained through an explicit inductive-recursive description of the universes [Martin-Löf 1984; McBride 2010] to build a type of codes $\square_i$ described in Fig. 12. Codes are noted with $\widehat{\cdot}$ and the universe type contains codes for dependent product ($\widehat{\Pi}$), universes ($\widehat{\square}_j$), inductive types (*e.g.*, $\widehat{\mathbb{N}}$) as well as $\widehat{?}$ for the unknown type and $\widehat{\mathtt{err}}$ for the error type. The main subtlety here is that the code $\widehat{\Pi}\, A\, B$ is at level $s_\Pi(i, j)$ when A is at i and B is a family at j, emulating the rule of Fig. 3. Accompanying the inductive definition of $\square_i$, the recursively defined decoding function El provides a semantics for these codes. The semantics of $\widehat{\Pi}$ is given by the dependent product in the target theory, applying El on the domain and the codomain of the code. The semantics of $\widehat{\square}_j$ is precisely the type of codes $\square_j$. The semantics of $\widehat{\mathbb{N}}$ is given by the extended natural numbers $\ddot{\mathbb{N}}$, explained above.

$$\frac{A \in \square_i \qquad B \in \mathrm{El}\,A \to \square_j}{\widehat{\Pi}\,A\,B \in \square_{s_\Pi(i,j)}} \qquad \frac{j < i}{\widehat{\square}_j \in \square_i} \qquad \widehat{\mathbb{N}} \in \square_i \qquad \widehat{?}_i \in \square_i \qquad \widehat{\mathtt{err}_i} \in \square_i$$

$$\mathrm{El}\,(\widehat{\Pi}\,A\,B) = \Pi(a : \mathrm{El}\,A)\ \mathrm{El}(B\,a) \qquad \mathrm{El}\,\widehat{\square}_j = \square_j \qquad \mathrm{El}\,\widehat{\mathbb{N}} = \ddot{\mathbb{N}} \qquad \mathrm{El}\,\widehat{?}_i = \ddot{\Sigma}\ \mathrm{Head}_i\ \mathrm{germ}_i \qquad \mathrm{El}\,\widehat{\mathtt{err}_i} = \mathtt{unit}$$

Fig. 12. Inductive-recursive encoding of the discrete universe hierarchy

$$\begin{array}{lllll} ?_{\widehat{\Pi}\,A\,B} := \lambda\,x : \mathrm{El}\,A.?_{\mathrm{El}\,(B\,x)} & ?_{\widehat{\square}_j} := \widehat{?}_j & ?_{\widehat{\mathbb{N}}} := \top_{\ddot{\mathbb{N}}} & ?_{\widehat{?}_j} := \top_{\mathrm{El}\,\widehat{\square}_j} & ?_{\widehat{\mathtt{err}_j}} := () \\ \mathtt{err}_{\widehat{\Pi}\,A\,B} := \lambda\,x : \mathrm{El}\,A.\,\mathtt{err}_{\mathrm{El}\,(B\,x)} & \mathtt{err}_{\widehat{\square}_j} := \widehat{\mathtt{err}_j} & \mathtt{err}_{\widehat{\mathbb{N}}} := \bot_{\ddot{\mathbb{N}}} & \mathtt{err}_{\widehat{?}_j} := \bot_{\mathrm{El}\,\widehat{\square}_j} & \mathtt{err}_{\widehat{\mathtt{err}_j}} := () \end{array}$$

Fig. 13. Realization of exceptions

Intuitively, the semantics of $\widehat{?}_i$ is that an inhabitant of the unknown type corresponds to a pair of a type and an inhabitant of that type. More precisely, we first define a notion of germ for codes where we stratify the head constructors Head (see Fig. 4) according to the universe level i, e.g. $\widehat{\mathrm{germ}}_i\,\Pi := \widehat{\Pi}\,\widehat{\square}_{c_\Pi(i)}\ (\lambda(x : \square_{c_\Pi(i)}).\widehat{\square}_{c_\Pi(i)})$ when $c_\Pi(i) \geq 0$, and its decoding to types $\mathrm{germ}_i\ h := \mathrm{El}\,(\widehat{\mathrm{germ}}_i\,h)$. The unknown type $\widehat{?}_i$ is then decoded to the extended dependent sum $\ddot{\Sigma}\ \mathrm{Head}_i\ \mathrm{germ}_i$ whose elements are either:

- one of the two freely added constructors $\top_{\ddot{\Sigma}}, \bot_{\ddot{\Sigma}}$ following the interpretation scheme of inductive types;
- or a dependent pair $(h; t)$ of a head constructor $h \in \mathrm{Head}_i$ together with an element $t \in \mathrm{germ}_i\ h$.

Finally, the error type $\widehat{\mathtt{err}_i}$ is decoded to the unit type $\mathtt{unit}$ containing a unique element $()$.

Variants of CastCIC. Crucially, the code for Π-types (Fig. 12) depends on the choice made for $s_\Pi(i, j)$. Observe that for the choice of parameters corresponding to $\mathrm{CastCIC}^{\mathcal{G}}$, the inductive-recursive definition of $\square_i$ is ill-founded since $c_\Pi(s_\Pi(i, i)) = s_\Pi(i, i)$. We can thus inject $\mathrm{germ}_{s_\Pi(i,i)}\Pi = \mathrm{El}\,\widehat{?}s_\Pi(i, i) \to \mathrm{El}\,\widehat{?}s_\Pi(i, i)$ into $\mathrm{El}\,\widehat{?}s_\Pi(i, i)$ and project back in the other direction, exhibiting an embedding-retraction suitable to interpret the untyped λ-calculus and hence Ω.[17]

In order to maintain normalization, the construction of the unknown type and the universe therefore needs to be stratified, which is possible when $c_\Pi(s_\Pi(i, i)) < s_\Pi(i, i)$. This strict inequality occurs for both $\mathrm{CastCIC}^{\mathcal{N}}$ and $\mathrm{CastCIC}^{\uparrow}$. We then proceed by strong induction on the universe level, and note that thanks to the level gap, the decoding $\mathrm{El}\,\widehat{?}_i$ of the unknown type at a level i can be defined solely from the data of smaller universes available by inductive hypothesis, without any reference to $\square_i$. We can then define the rest of the universe $\square_i$ and the decoding function El at level i in a well-founded manner, validating the strict positivity criterion of Agda's termination checker.

Exceptions. The definition of exceptions $?_A, \mathtt{err}_A\ :\ \mathrm{El}\ A$ at an arbitrary code A then follows by case analysis on the code, as shown in Fig. 13. On the code for the universe, $\widehat{\square}_j$, we directly use the code for the unknown and the error types respectively. On codes that have an inductive interpretation—$\widehat{\mathbb{N}}$, $\widehat{?}_i$—we use the two added constructors. On the code for dependent functions,

[17]In the Agda implementation, we deactivate the termination checker on the definition of the universe for the model interpreting $\mathrm{CastCIC}^{\mathcal{G}}$, thus effectively working in a partial, inconsistent type theory.

$$
\begin{array}{lcll}
\mathtt{cast}\ (\widehat{\Pi}\, A^{\mathrm{d}}\, A^{\mathrm{c}})\ (\widehat{\Pi}\, B^{\mathrm{d}}\, B^{\mathrm{c}})\ f & := & \lambda\, b : \mathrm{El}\, B^{\mathrm{d}}.\ \mathrm{let}\ a = \mathtt{cast}\ B^{\mathrm{d}}\ A^{\mathrm{d}}\ b\ \mathrm{in}\ \mathtt{cast}\ (A^{\mathrm{c}}\ a)\ (B^{\mathrm{c}}\ b)\ (f\ a) & \\
\mathtt{cast}\ (\widehat{\Pi}\, A^{\mathrm{d}}\, A^{\mathrm{c}})\ \widehat{?}_i\ f & := & (\Pi; \mathtt{cast}\ (\widehat{\Pi}\, A^{\mathrm{d}}\, A^{\mathrm{c}})\ (\widehat{\mathrm{germ}}_i\, \Pi)\ f) & \text{if } \widehat{\mathrm{germ}}_i\, \Pi \neq \widehat{\mathtt{err}} \\
\mathtt{cast}\ (\widehat{\Pi}\, A^{\mathrm{d}}\, A^{\mathrm{c}})\ X\ f & := & \mathtt{err}_X \quad \text{otherwise} &
\end{array}
$$

$$
\begin{array}{lcl}
\mathtt{cast}\ \widehat{\mathbb{N}}\ \widehat{\mathbb{N}}\ n & := & n \\
\mathtt{cast}\ \widehat{\mathbb{N}}\ \widehat{?}_i\ n & := & (\mathbb{N}; n) \\
\mathtt{cast}\ \widehat{\mathbb{N}}\ X\ n & := & \mathtt{err}_X \\
\\
\mathtt{cast}\ \widehat{\mathtt{err}}_i\ Z\ () & := & \mathtt{err}_Z
\end{array}
\qquad
\begin{array}{lcll}
\mathtt{cast}\ \widehat{\square}_j\ \widehat{\square}_j\ A & := & A & \\
\mathtt{cast}\ \widehat{\square}_j\ \widehat{?}_i\ A & := & (\square_j; A) & \text{if } j < i \\
\mathtt{cast}\ \widehat{\square}_j\ X\ A & := & \mathtt{err}_X & \text{otherwise} \\
\\
\mathtt{cast}\ \widehat{?}_i\ Z\ (c; x) & := & \mathtt{cast}\ (\widehat{\mathrm{germ}}_i\ c)\ Z\ x & \\
\mathtt{cast}\ \widehat{?}_i\ Z\ \top_{\mathrm{El}\,\widehat{?}_i} & := & ?_Z & \\
\mathtt{cast}\ \widehat{?}_i\ Z\ \bot_{\mathrm{El}\,\widehat{?}_i} & := & \mathtt{err}_Z &
\end{array}
$$

Fig. 14. Definition of cast (discrete model)

$$
\begin{array}{lcl}
[\![\cdot]\!] & := & \cdot \\
[\![A]\!] & := & \mathrm{El}\,[A] \\
\\
[x] & := & x \\
[\square_i] & := & \widehat{\square}_i \\
[\Pi\, x : A.B] & := & \widehat{\Pi}\ [A]\ (\lambda\, x : [\![A]\!].[B]) \\
[t\ u] & := & [t]\ [u] \\
[\lambda\, x : A.t] & := & \lambda\, x : [\![A]\!].[t]
\end{array}
\qquad
\begin{array}{lcl}
[\![\Gamma, x : A]\!] & := & [\![\Gamma]\!], x : [\![A]\!] \\
[\mathbb{N}] & := & \widehat{\mathbb{N}} \\
[0] & := & 0 \\
[\mathtt{suc}] & := & \mathtt{suc} \\
[\mathtt{ind}_{\mathbb{N}}]\ P\ h_0\ h_{\mathtt{suc}} & := & \mathtt{ind}_{\ddot{\mathbb{N}}}\ P\ h_0\ h_{\mathtt{suc}}\ ?_{P\,?_{\ddot{\mathbb{N}}}}\ \mathtt{err}_{(P\,\mathtt{err}_{\ddot{\mathbb{N}}})} \\
[?_A] & := & ?_{[A]} \\
[\mathtt{err}_A] & := & \mathtt{err}_{[A]} \\
[\langle B \Leftarrow A\rangle\ t] & := & \mathtt{cast}\ [A]\ [B]\ [t]
\end{array}
$$

Fig. 15. Discrete translation from CastCIC to $\mathrm{CIC}^{\mathrm{IR}}$

exceptions are defined by re-raising the exception at the codomain in a pointwise fashion. Finally, on the error type $\widehat{\mathtt{err}}$, exceptions are degenerated and forced to take the only value $() : \mathtt{unit}$[18] of its interpretation as a type.

Casts. Equipped with exceptions and type analysis, we define $\mathtt{cast} : \Pi(A : \square_i)(B : \square_j).A \to B$ by induction on the universe levels and case analysis on the codes of the types A and B (Fig. 14). In the total setting (when $c_\Pi(s_\Pi(i,i)) < s_\Pi(i,i)$), the definition of cast is well-founded: each recursive call happens either at a strictly smaller universe (the two cases for $\widehat{\Pi}$) or on a strict subterm of the term being cast (case of inductives, *i.e.,* $\widehat{\mathbb{N}}$ and $\widehat{?}$). Note that each of the defining equations of cast corresponds straightforwardly to a reduction rule of Fig. 5.

Discrete translation. We can finally define the discrete syntactic model of CastCIC in $\mathrm{CIC}^{\mathrm{IR}}$ (Fig. 15). The translations $[-]$ and $[\![-]\!]$ are defined by induction on the syntax of terms and types. A type A is translated to its corresponding code $[A]$ in $\square_i$ when seen as a term, and is translated to the interpretation of this code $[\![A]\!] := \mathrm{El}\,[A]$ when seen as a type. $?_A$ and $\mathtt{err}_A$ are directly translated using the exceptions defined in Fig. 13. The following theorem shows that the translation is a syntactic model in the sense of Boulier et al. [2017].

[18] This definition is indeed uniform if unit is seen as the record type with no projection.

$$0 \sqsubseteq^{\widehat{\mathbb{N}}} 0 \qquad \bot_{\ddot{\mathbb{N}}} \sqsubseteq^{\widehat{\mathbb{N}}} n \qquad \frac{n \sqsubseteq^{\widehat{\mathbb{N}}} m}{\mathsf{suc}\, n \sqsubseteq^{\widehat{\mathbb{N}}} \mathsf{suc}\, m} \qquad \frac{n \sqsubseteq^{\widehat{\mathbb{N}}} \top_{\ddot{\mathbb{N}}}}{\mathsf{suc}\, n \sqsubseteq^{\widehat{\mathbb{N}}} \top_{\ddot{\mathbb{N}}}}$$

$$0 \sqsubseteq^{\widehat{\mathbb{N}}} \top_{\ddot{\mathbb{N}}} \qquad \top_{\ddot{\mathbb{N}}} \sqsubseteq^{\widehat{\mathbb{N}}} \top_{\ddot{\mathbb{N}}}$$

Fig. 16. Order structure on extended natural numbers

Theorem 26 (Discrete syntactic model). *The translation defined in Fig. 15 preserves conversion and typing derivations:*

(1) if $\Gamma \vdash_{\mathsf{cast}} t \rightsquigarrow u$ *then* $[\![\Gamma]\!] \vdash_{\mathrm{IR}} [t] \rightsquigarrow^{+}_{\mathrm{IR}} [u]$, *in particular* $[\![\Gamma]\!] \vdash_{\mathrm{IR}} [t] \equiv [u]$,
(2) if $\Gamma \vdash_{\mathsf{cast}} t : A$ *then* $[\![\Gamma]\!] \vdash_{\mathrm{IR}} [t] : [\![A]\!]$.

Proof. (1) All reduction rules from CIC are preserved without a change so that we only need to be concerned with the reduction rules involving exceptions or casts. A careful inspection shows that these reductions are preserved too once we observe that the terms of the shape $\langle ?_{\square_i} \Leftarrow \mathrm{germ}_i\ h \rangle\, t$ that are stuck in CastCIC are in one-to-one correspondence with the one-step reduced form of its translation $(h; [t]) : \ddot{\Sigma}\ \mathrm{Head}_i\ \mathrm{germ}_i$. (2) Proved by a direct induction on the typing derivation of $\Gamma \vdash_{\mathsf{cast}} t : A$, using the fact that exceptions and casts are well-typed—that is $\vdash_{\mathrm{IR}} ? : \Pi(A : \square_i)\, \mathrm{El}\, A$, $\vdash_{\mathrm{IR}} \mathtt{err} : \Pi(A : \square_i)\, \mathrm{El}\, A$, and $\vdash_{\mathrm{IR}} \mathtt{cast} : \Pi(A : \square_i)(B : \square_i) \mathrm{El}\, A \rightarrow \mathrm{El}\, B$—and relying on assertion (1) to handle the conversion rule. ◻

As explained in Theorem 9, Theorem 26 implies in particular that $\mathrm{CastCIC}^{\uparrow}$ and $\mathrm{CastCIC}^{\mathcal{N}}$ are strongly normalizing.

6.2 Poset-Based Models of Dependent Type Theory

The simplicity of the discrete model comes at the price of an inherent inability to characterize which casts are guaranteed to succeed, *i.e.*, a graduality theorem. To overcome this limitation, we develop a monotone model on top of the discrete model where, by construction, each type A comes equipped with an order structure $\sqsubseteq^A$—a reflexive, transitive, antisymmetric and proof-irrelevant relation—modelling precision between terms. In particular, the exceptions $\mathtt{err}_A$ and $?_A$ correspond respectively to the smallest and greatest element of A for this order. We note $\square^{\leq}$ for a universe of types equipped with the structure of a poset together with smallest and greatest elements. Each term and type constructor is enforced to be monotone with respect to these orders, providing a strong form of graduality. This implies in particular that such a model cannot be defined for $\mathrm{CastCIC}^{\mathcal{N}}$ because this type theory lacks graduality, as shown by Example 19.

As an illustration, the order on extended natural numbers (Fig. 16) makes $\bot_{\ddot{\mathbb{N}}}$ the smallest element and $\top_{\ddot{\mathbb{N}}}$ the biggest element.[19] The standard natural numbers – 0 or $\mathsf{suc}\, n$ for a standard natural number n – then stand between failure and indeterminacy, but are never related to each other by precision. Indeed, in order to ensure conservativity with respect to CIC, $\sqsubseteq^{\widehat{\mathbb{N}}}$ must coincide with CIC's conversion on static closed natural numbers.

Beyond the precision order on types, the nature of dependency forces us to spell out what the precision between types entails. Following the analysis of New and Ahmed [2018], a relation $A \sqsubseteq B$ between types should induce an embedding-projection pair (ep-pair): a pair of an *upcast* $\uparrow\ : A \rightarrow B$ and a *downcast* $\downarrow\ : B \rightarrow A$ satisfying a handful of properties with gradual guarantees as a corollary.

Definition 9 (Embedding-projection pairs). *An ep-pair* $d : A \lhd B$ *between posets* $A, B : \square^{\leq}$ *consists of*

[19] We abusively note $\ddot{\mathbb{N}}$ for both the poset and its carrier to avoid introducing too many notations.

- *an underlying relation $d \subseteq A \times B$ such that*
$$a' \sqsubseteq^A a \;\wedge\; d(a,b) \;\wedge\; b \sqsubseteq^B b' \quad\Longrightarrow\quad d(a',b')$$
- *that is bi-represented by* $\uparrow_d : A \to B, \downarrow_d : B \to A$, i.e.,
$$\uparrow_d a \sqsubseteq^B b \quad\Leftrightarrow\quad d(a,b) \quad\Leftrightarrow\quad a \sqsubseteq^A \downarrow_d b,$$
- *such that the equality* $\downarrow_d \circ \uparrow_d = \mathrm{id}_A$ *holds.*

Note that here equiprecision of the retraction becomes an equality because of antisymmetry. Under these conditions, $\uparrow_d\colon A \hookrightarrow B$ is injective, $\downarrow_d\colon B \twoheadrightarrow A$ is surjective and both preserve bottom elements, explaining that we call $d : A \lhd B$ an embedding-projection pair. The definition of ep-pairs is based on a relation rather than just its pair of representing functions to highlight the connection between ep-pairs and parametricity [New et al. 2020]. Assuming function extensionality, being an ep-pair is a property of the underlying relation: there is at most one pair $(\uparrow_d, \downarrow_d)$ representing the underlying relation of d. An ep-pair straightforwardly induces the following relations that will be used in later proofs.

Lemma 27 (Properties of ep-pairs). *Let $d : A \lhd B$ be an ep-pair between posets.*

(1) If $a : A$ then $d\,(a, \uparrow_d a)$ and $a \sqsubseteq^A \downarrow_d \uparrow_d a$.
(2) If $b : B$ then $d\,(\downarrow_d b, b)$ and $\uparrow_d \downarrow_d b \sqsubseteq^B b$.

Posetal families. By monotonicity, a family $B : A \to \square^{\leq}$ over a poset A gives rise not only to a poset $B\,a$ for each $a \in A$, but also to ep-pairs $B_{a,a'} : B\,a \lhd B\,a'$ for each $a \sqsubseteq^A a'$. These ep-pairs need to satisfy functoriality conditions:
$$B_{a,a} = \sqsubseteq^{B\,a} \qquad\text{and}\qquad B_{a,a''} = B_{a',a''} \circ B_{a,a'} \quad\text{whenever}\quad a \sqsubseteq^A a' \sqsubseteq^A a''.$$
In particular, this ensures that heterogeneous transitivity is well defined:
$$B_{a,a'}(b,b') \wedge B_{a',a''}(b',b'') \Rightarrow B_{a,a''}(b,b'').$$

Dependent products. Given a poset A and a posetal family B over A, we can form the poset $\Pi^{\mathrm{mon}}\ A\,B$ of *monotone* dependent functions from A to B, equipped with the pointwise order. Its inhabitants are dependent functions $f : \Pi(a : A).B\,a$ such that $a \sqsubseteq^A a' \Rightarrow B_{a,a'}\ (f\,a, f\,a')$. Moreover, given ep-pairs $d_A : A \lhd A'$ and $d_B : B \lhd B'$, we can build an induced ep-pair $d_\Pi : \Pi^{\mathrm{mon}}\ A\,B \lhd \Pi^{\mathrm{mon}}\ A'\,B'$ with underlying relation
$$d_\Pi(f, f') := d_A(a,a') \Rightarrow d_B(f\,a, f'\,a'),$$
$$\uparrow_{d_\Pi} f := \uparrow_{d_B} \circ f \circ \downarrow_{d_A} \qquad\text{and}\qquad \downarrow_{d_\Pi} f := \downarrow_{d_B} \circ f \circ \uparrow_{d_A}.$$
The general case where B and B' actually depend on A, A' is obtained with similar formulas, but a larger amount of data is required to handle the dependency: we refer to the accompanying Agda development for details.

Inductive types. Generalizing the case of natural numbers, the order on an arbitrary extended inductive type $\ddot{I}$ uses the following scheme:

(1) $\bot_{\ddot{I}}$ is the least element
(2) $\top_{\ddot{I}} \sqsubseteq^{\ddot{I}} \top_{\ddot{I}}$
(3) $c\ \mathbf{t} \sqsubseteq^{\ddot{I}} \top_{\ddot{I}}$ whenever $t_i \sqsubseteq^{X_i} \top_{X_i}$ for all i
(4) each constructor c is monotone with respect to the order on its arguments

The precondition on subterms in the third case is unnecessary in simple cases and is kept to be uniform with definition of order on the monotone unknown type in the following section.

Similarly to dependent product, an ep-pair $\mathbf{X} \lhd \mathbf{X}'$ between the parameters of an extended inductive type $\ddot{I}$ induces an ep-pair $\ddot{I}\,\mathbf{X} \lhd \ddot{I}\,\mathbf{X}'$. For instance, ep-pairs $d_A : A \lhd A'$ and $d_B : B \lhd B'$ induce an ep-pair $d_{\ddot{\Sigma}} : \ddot{\Sigma}\,A\,B \lhd \ddot{\Sigma}\,A'\,B'$ defined by $d_{\ddot{\Sigma}}((a,b),(a',b')) := d_A(a,a') \wedge d_B(b,b')$.

6.3 Microcosm: the Monotone Unknown Type $\ddot{?}$

The interpretation $\ddot{?}_i$ of the unknown type in the monotone model should morally group together approximations of every type at the same universe level. Working in (bi)pointed orders, $\ddot{?}_i$ can be realized as a coalesced sum [Abramsky and Jung 1995, section 3.2.3] of the family $\mathrm{germ}_i\ h$ indexed by head constructors $h \in \mathrm{Head}_i$. A concrete presentation of $\ddot{?}_i$ is obtained as the quotient of $\ddot{\Sigma}\ \mathrm{Head}_i\ \mathrm{germ}_i$ identifying $\bot_{\ddot{\Sigma}\ \mathrm{Head}_i\ \mathrm{germ}_i}$ with any pair $(h; \mathtt{err}_{\widehat{\mathrm{germ}}_i\ h})$. The equivalence classes of $(h;x)$ is noted as $[h;x]$, $\bot_{\ddot{\Sigma}\ \mathrm{Head}_i\ \mathrm{germ}_i}$ as $\bot_{\ddot{?}_i}$ and $\top_{\ddot{\Sigma}\ \mathrm{Head}_i\ \mathrm{germ}_i}$ as $\top_{\ddot{?}_i}$. The obtained type $\ddot{?}_i$ is then equipped with a precision relation defined by the rules:

$$\bot_{\ddot{?}_i} \sqsubseteq^{\widehat{?}_i} z \qquad \top_{\ddot{?}_i} \sqsubseteq^{\widehat{?}_i} \top_{\ddot{?}_i} \qquad \frac{x \sqsubseteq^{\widehat{\mathrm{germ}}_i\ h} x'}{[h;x] \sqsubseteq^{\widehat{?}_i} [h;x']} \qquad [h;x] \sqsubseteq^{\widehat{?}_i} \top_{\ddot{?}_i} \tag{1}$$

These rules ensure that the exceptions $\bot_{\ddot{?}_i}$ and $\top_{\ddot{?}_i}$ are respectively the smallest and biggest elements of $\ddot{?}_i$. Non-exceptional elements are comparable only if they have the same head constructor h and if so are compared according to the interpretation of that head constructor as an ordered type $\mathrm{germ}_i\ h$. Because of the quotient, it is not immediate that this presentation of $\sqsubseteq^{\ddot{?}_i}$ is independent of the choice of representatives in equivalence classes and that it forms a proof-irrelevant relation. In the formal development, we define the relation by quotient-induction on each argument, thus verifying that it respects the quotient, and also show that it is irrelevant. This relies crucially on equality being decidable on head constructors when comparing $[h;x]$ and $[h';x']$.

In order to globally satisfy $\mathcal{G}$, $\ddot{?}_i$ should admit an ep-pair $d_h : \mathrm{germ}_i\ h \lhd \ddot{?}_i$ whenever we have a head constructor $h \in \mathrm{Head}_i$ such that $\widehat{\mathrm{germ}}_i\ h \sqsubseteq \widehat{?}_i$ (we return to that point in the next section §6.4). Embedding an element $x \in \mathrm{germ}_i\ h$ by $\uparrow_{d_h}\ x = [h;x]$ and projecting out of $\mathrm{germ}_i\ h$ by the following equations form a reasonable candidate:

$$\downarrow_{d_h}\ [h';x] = \begin{cases} x & \text{if } h = h' \\ \mathtt{err}_{\mathrm{germ}_i\ h} & \text{otherwise} \end{cases} \qquad \downarrow_{d_h}\ \top_{\ddot{?}_i} = ?_{\mathrm{germ}_i\ h} \qquad \downarrow_{d_h}\ \bot_{\ddot{?}_i} = \mathtt{err}_{\mathrm{germ}_i\ h}$$

Note that we rely again on Head having decidable equality to compute the $\downarrow_{d_h}$. Moreover $\uparrow_{d_h} \dashv \downarrow_{d_h}$ should be adjoints; in particular, the following precision relation needs to hold:

$$\mathtt{err}_{\mathrm{germ}_i\ h} \sqsubseteq^{\mathrm{germ}_i\ h} \downarrow_{d_h} \bot_{\ddot{?}_i} \qquad \Longleftrightarrow \qquad [h; \mathtt{err}_{\mathrm{germ}_i\ h}] = \uparrow_{d_h}\ \mathtt{err}_{\mathrm{germ}_i\ h} \sqsubseteq^{\ddot{?}_i} \bot_{\ddot{?}_i}$$

Since $\sqsubseteq^{\ddot{?}_i}$ should be antisymmetric, this is possible only if $[h; \mathtt{err}_{\mathrm{germ}_i\ h}]$ and $\bot_{\ddot{?}_i}$ are identified in $\ddot{?}_i$, explaining why we have to quotient in the first place.

6.4 Realization of the Monotone Universe Hierarchy

Following the discrete model, the monotone universe hierarchy is also implemented through an inductive-recursive datatype of codes $\square_i$ together with a decoding function $\mathrm{El} : \square_i \to \square$, both presented in Fig. 17. The precision relation $\sqsubseteq\ :\ \square_i\ \to\ \square_j\ \to \square$ presented below is an order (Theorem 28) on this universe hierarchy. The "diagonal" inference rules, providing evidence for relating type constructors from CIC, coincide with those of binary parametricity [Bernardy et al. 2012]. Outside the diagonal, $\widehat{\mathtt{err}}$ is placed at the bottom. More interestingly, the derivation of a precision proof $A \sqsubseteq \widehat{?}$ provides a unique decomposition of A through iterated germs directed by the relevant head constructors. For instance, in the gradual systems $\mathrm{CastCIC}^{\mathcal{G}}$ and $\mathrm{CastCIC}^{\uparrow}$

Monotone universes $\square_i$ and decoding function El $: \square_i \to \square^{\leq}$ (cases distinct from Fig. 12)

$$\frac{A \in \square_i \qquad B \in \Pi^{\mathrm{mon}}(a : \mathrm{El}\, A).\, \square_j}{\widehat{\Pi}\, A\, B \in \square_{s_\Pi(i,j)}} \qquad \mathrm{El}\,(\widehat{\Pi}\, A\, B) = \Pi^{\mathrm{mon}}(a : \mathrm{El}\, A).\, \mathrm{El}(B\, a) \qquad \mathrm{El}\,\widehat{?}_i = \ddot{?}_i$$

Precision order $\sqsubseteq$ on the universes (where $i \leq j$)

$$\widehat{\mathtt{err}}\text{-}\sqsubseteq\ \frac{A : \square_j}{\widehat{\mathtt{err}}_i \sqsubseteq A} \qquad \widehat{\mathbb{N}}\text{-}\sqsubseteq\ \frac{}{\widehat{\mathbb{N}} \sqsubseteq \widehat{\mathbb{N}}} \qquad \widehat{\square}\text{-}\sqsubseteq\ \frac{}{\widehat{\square}_i \sqsubseteq \widehat{\square}_i} \qquad \widehat{?}\text{-}\sqsubseteq\ \frac{}{\widehat{?}_i \sqsubseteq \widehat{?}_j}$$

$$\widehat{\Pi}\text{-}\sqsubseteq\ \frac{A \sqsubseteq A' \qquad a : \mathrm{El}\, A, a' : \mathrm{El}\, A', a_\epsilon : a \ {}_{A}{\sqsubseteq}_{A'}\ a' \vdash B\, a \sqsubseteq B'\, a'}{\widehat{\Pi}\, A\, B \sqsubseteq \widehat{\Pi}\, A'\, B'} \qquad \text{Head-}\sqsubseteq\ \frac{h = \mathrm{head}\, A \in \mathrm{Head}_i \qquad A \sqsubseteq \widehat{\mathrm{germ}}_j\, h}{A \sqsubseteq \widehat{?}_j}$$

Precision on terms ${}_{A}{\sqsubseteq}_{B} := \mathrm{El}_\varepsilon(A \sqsubseteq B)$

$$\mathrm{El}_\varepsilon(\widehat{\mathtt{err}}\text{-}\sqsubseteq)\ \frac{a : \mathrm{El}\, A}{()\ {}_{\widehat{\mathtt{err}}}{\sqsubseteq}_{A}\ a} \qquad \mathrm{El}_\varepsilon(\widehat{\mathbb{N}}\text{-}\sqsubseteq), \mathrm{El}_\varepsilon(\widehat{\square}\text{-}\sqsubseteq)\ \frac{A = \widehat{\mathbb{N}}, \widehat{\square}_i \qquad x \sqsubseteq^{\mathrm{El}\, A} y}{x\ {}_{A}{\sqsubseteq}_{A}\ y}$$

$$\mathrm{El}_\varepsilon(\widehat{?}\text{-}\sqsubseteq)\ \frac{z : \ddot{?}_j}{\mathtt{err}_{\widehat{?}_i}\ {}_{\widehat{?}_i}{\sqsubseteq}_{\widehat{?}_j}\ z} \qquad \frac{z : \ddot{?}_i}{z\ {}_{\widehat{?}_i}{\sqsubseteq}_{\widehat{?}_j}\ ?_{\widehat{?}_j}} \qquad \frac{x\ {}_{\widehat{\mathrm{germ}}_i\, h}{\sqsubseteq}_{\widehat{\mathrm{germ}}_j\, h}\ x'}{[h; x]\ {}_{\widehat{?}_i}{\sqsubseteq}_{\widehat{?}_j}\ [h; x']}$$

$$\mathrm{El}_\varepsilon(\widehat{\Pi}\text{-}\sqsubseteq)\ \frac{a : \mathrm{El}\, A, a' : \mathrm{El}\, A', a_\epsilon : a\ {}_{A}{\sqsubseteq}_{A'}\ a' \vdash f\ a\ {}_{B\, a}{\sqsubseteq}_{B'\, a'}\ f'\ a'}{f\ {}_{\widehat{\Pi}\, A\, B}{\sqsubseteq}_{\widehat{\Pi}\, A'\, B'}\ f'} \qquad \mathrm{El}_\varepsilon(\text{Head-}\sqsubseteq)\ \frac{a\ {}_{A}{\sqsubseteq}_{\widehat{\mathrm{germ}}_j\, (\mathrm{head}\, A)}\ x}{a\ {}_{A}{\sqsubseteq}_{\widehat{?}_j}\ [\mathrm{head}\, A; x]} \qquad \frac{a : \mathrm{El}\, A}{a\ {}_{A}{\sqsubseteq}_{\widehat{?}_j}\ ?_{\widehat{?}_j}}$$

Fig. 17. Monotone universe of codes and precision

where the equation $c_\Pi(s_\Pi(i,j)) = \max(i,j)$ holds for any universe levels i, j, the derivation of $(\widehat{\mathbb{N}} \to \widehat{\mathbb{N}}) \to \widehat{\mathbb{N}} \sqsubseteq \widehat{?}$ canonically decomposes as:

$$(\widehat{\mathbb{N}} \to \widehat{\mathbb{N}}) \to \widehat{\mathbb{N}} \quad \sqsubseteq \quad (\widehat{?} \to \widehat{?}) \to \widehat{\mathbb{N}} \quad \sqsubseteq \quad \widehat{?} \to \widehat{?} \quad \sqsubseteq \quad \widehat{?}$$

This unique decomposition is at the heart of the reduction of the cast operator given in Fig. 5, and it can be described informally as taking the path of maximal length between two related types.[20] Such a derivation of precision $A \sqsubseteq B$ gives rise through decoding to ep-pairs $\mathrm{El}_\varepsilon\ (A \sqsubseteq B) : \mathrm{El}\, A \lhd \mathrm{El}\, B$, with underlying relation noted ${}_{A}{\sqsubseteq}_{B} : \mathrm{El}\, A \to \mathrm{El}\, B \to \square$. This decoding function El_ε is described on generators of $\sqsubseteq$ at the bottom of Fig. 17. $\mathrm{El}_\varepsilon(\widehat{\mathtt{err}}\text{-}\sqsubseteq)$ states that the unique value $()$ of $\mathrm{El}\,\widehat{\mathtt{err}} = \mathtt{unit}$ is smaller than any other value. The diagonal cases $\mathrm{El}_\varepsilon(\widehat{\mathbb{N}}\text{-}\sqsubseteq)$ and $\mathrm{El}_\varepsilon(\widehat{\square}\text{-}\sqsubseteq)$ reuse the order specified on the carrier. The ep-pair $\mathrm{El}_\varepsilon(\widehat{?}\text{-}\sqsubseteq)$ between two unknown types $\ddot{?}_i \lhd \ddot{?}_j$ at potentially distinct universe levels $i \leq j$ stipulate that $\mathtt{err}_{\widehat{?}_i}$ and $?_{\widehat{?}_j}$ are respectively smaller and greater than any other value, and that the comparison between two injected terms with same head is induced by their second component. Note that these rules are redundant since $\ddot{?}$ is obtained through a quotient. Functions f, f' are related by $\mathrm{El}_\varepsilon(\widehat{\Pi}\text{-}\sqsubseteq)$ when they map related elements $a\ {}_{A}{\sqsubseteq}_{A'}\ a'$ to related elements $f\ a\ {}_{B\, a}{\sqsubseteq}_{B'\, a'}\ f'\ a'$. Finally, $\mathrm{El}_\varepsilon(\text{Head-}\sqsubseteq)$ embeds a type A into $\ddot{?}$ through its head.

[20] This decomposition is already present in [New and Ahmed 2018] and to be contrasted with the AGT approach [Garcia et al. 2016], which tends to pair a value with the most precise witness of its type, *i.e.*, canonical path of minimal length.

It is interesting to observe what happens in CastCIC$^{\mathcal{N}}$, where $c_{\Pi}(s_{\Pi}(i,j)) \neq \max(i,j)$, for instance on the previous example:

$$\widehat{\mathbb{N}}\widehat{\to}\widehat{\mathbb{N}} \quad \not\sqsubseteq \quad \widehat{\mathtt{err}} = \widehat{\mathrm{germ}}_0\,\Pi \quad \sqsubseteq \quad \widehat{?}$$

So $\widehat{\mathbb{N}}\widehat{\to}\widehat{\mathbb{N}}$ is not lower than $\widehat{?}$ in that setting.

One crucial point of the monotone model is the mutual definition of codes $\square_i$ together with the precision relation, particularly salient on codes for Π-types: in $\widehat{\Pi}\,A\,B$, $B : \mathrm{El}\,A \to \square_i$ is a monotone function with respect to the order on $\mathrm{El}\,A$ and the precision on $\square_i$. This intertwining happens because the order is required to be reflexive, a fact observed previously by Atkey et al. [2014] in the similar setting of reflexive graphs. Indeed, a dependent function $f : \Pi(a : \mathrm{El}\,A).\,\mathrm{El}\,(B\,a)$ is related to itself $f\ {}_{\widehat{\Pi}\,A\,B}{\sqsubseteq}_{\widehat{\Pi}\,A\,B}\ f$ if and only if f is *monotone*.

Theorem 28 (Properties of the universe hierarchy).

(1) $\sqsubseteq$ *is reflexive, transitive, antisymmetric and irrelevant so that* $(\square_i, \sqsubseteq)$ *is a poset.*
(2) $\square_i$ *has a bottom element* $\widehat{\mathtt{err}}_i$ *and a top element* $\widehat{?}_i$*; in particular,* $A \sqsubseteq \widehat{?}_i$ *for any* $A : \square_i$*.*
(3) $\mathrm{El} : \square_i \to \square$ *is a family of posets over* $\square_i$ *with underlying relation* ${}_{A}{\sqsubseteq}_{B}$ *whenever* $A \sqsubseteq B$*.*
(4) $\square_i$ *and* $\mathrm{El}\ A$ *for any* $A : \square_i$ *verify UIP*[21]*: the equality on these types is irrelevant.*

Proof sketch. All these properties are proved mutually, first by strong induction on the universe levels, then by induction on the codes of the universe or the derivation of precision. Here, we only sketch the proof of point (1) and refer to the Agda development (*cf.* `UnivPartial.agda`) for detailed formal proofs.

For reflexivity, all cases are immediate but for $\widehat{\Pi}\,A\,B$: the induction hypothesis provides $A \sqsubseteq A$ and by point (3) $\mathrm{El}_{\varepsilon}(A \sqsubseteq A) = \sqsubseteq^{A}$ so we can apply the monotonicity of B.

For anti-symmetry, assuming $A \sqsubseteq B$ and $B \sqsubseteq A$, we prove by induction on the derivation of $A \sqsubseteq B$ and case analysis on the other derivation that $A \equiv B$. Note that we never need to consider the rule Head-$\sqsubseteq$. The case Π-$\sqsubseteq$ holds by induction hypothesis and because the relation ${}_{A}{\sqsubseteq}_{A}$ is reflexive. All the other cases follow from antisymmetry of the order on universe levels.

For transitivity, assuming $AB : A \sqsubseteq B$ and $BC : B \sqsubseteq C$, we prove by induction on the (lexicographic) pair (AB, BC) that $A \sqsubseteq C$:

Case $AB = \widehat{?}$-$\sqsubseteq$**,** necessarily $BC = \widehat{?}$-$\sqsubseteq$, we conclude by $\widehat{?}$-$\sqsubseteq$.

Case $AB = \mathrm{Head}$-$\sqsubseteq$**,** necessarily $BC = \widehat{?}$-$\sqsubseteq, \widehat{?}_j \sqsubseteq \widehat{?}_{j'}$, we can thus apply the inductive hypothesis to $A \sqsubseteq \widehat{\mathrm{germ}}_j\,(\mathrm{head}\,A)$ and $\widehat{\mathrm{germ}}_j\,(\mathrm{head}\,A) \sqsubseteq \widehat{\mathrm{germ}}_j\,(\mathrm{head}\,A)$ in order to conclude with Head-$\sqsubseteq$.

Case $AB = \widehat{\mathtt{err}}$-$\sqsubseteq$**,** we conclude immediately by $\widehat{\mathtt{err}}$-$\sqsubseteq$.

Case $AB = \widehat{\mathbb{N}}$-$\sqsubseteq, BC = \widehat{\mathbb{N}}$-$\sqsubseteq$ we conclude with $\widehat{\mathbb{N}}$-$\sqsubseteq$.

Case $AB = \widehat{\square}$-$\sqsubseteq, BC = \widehat{\square}$-$\sqsubseteq$ immediate by $\widehat{\square}$-$\sqsubseteq$.

Case $AB = \widehat{\Pi}$-$\sqsubseteq, BC = \widehat{\Pi}$-$\sqsubseteq$ by hypothesis we have

$$A = \widehat{\Pi}\,A^{\mathrm{d}}\,A^{\mathrm{c}} \qquad B = \widehat{\Pi}\,B^{\mathrm{d}}\,B^{\mathrm{c}} \qquad C = \widehat{\Pi}\,C^{\mathrm{d}}\,C^{\mathrm{c}} \qquad A^{\mathrm{d}} \sqsubseteq B^{\mathrm{d}} \qquad B^{\mathrm{d}} \sqsubseteq C^{\mathrm{d}}$$

$$AB^{\mathrm{c}} : \forall a\,b, a\ {}_{A^{\mathrm{d}}}{\sqsubseteq}_{B^{\mathrm{d}}}\ b \to A^{\mathrm{c}}\,a \sqsubseteq B^{\mathrm{c}}\,b \qquad BC^{\mathrm{c}} : \forall b\,c, b\ {}_{B^{\mathrm{d}}}{\sqsubseteq}_{C^{\mathrm{d}}}\ c \to B^{\mathrm{c}}\,b \sqsubseteq C^{\mathrm{c}}\,c$$

By induction hypothesis applied to $A^{\mathrm{d}} \sqsubseteq B^{\mathrm{d}}$ and $B^{\mathrm{d}} \sqsubseteq C^{\mathrm{d}}$, the domains of the dependent product are related $A^{\mathrm{d}} \sqsubseteq C^{\mathrm{d}}$. For the codomains, we need to show that for any $a : A^{\mathrm{d}}, c : C^{\mathrm{d}}$ such that $a\ {}_{A^{\mathrm{d}}}{\sqsubseteq}_{C^{\mathrm{d}}}\ c$ we have $A^{\mathrm{c}}\,a \sqsubseteq C^{\mathrm{c}}\,c$. By induction hypothesis, it is enough to prove that $A^{\mathrm{c}}\,a \sqsubseteq B^{\mathrm{c}}\,(\uparrow_{A^{\mathrm{d}} \sqsubseteq B^{\mathrm{d}}}\,a)$ and $B^{\mathrm{c}}\,(\uparrow_{A^{\mathrm{d}} \sqsubseteq B^{\mathrm{d}}}\,a) \sqsubseteq C^{\mathrm{c}}\,c$. The former follows from AB^{c} applied

[21] Uniqueness of Identity Proofs; in HoTT parlance, $\square_i$ and $\mathrm{El}\,A$ are hSets.

to $a \ {}_{A^{\mathrm{d}}}{\sqsubseteq}_{B^{\mathrm{d}}} \uparrow_{A^{\mathrm{d}} \sqsubseteq B^{\mathrm{d}}} a \Leftrightarrow a \sqsubseteq^{A^{\mathrm{d}}} \downarrow \uparrow a \Leftrightarrow a \sqsubseteq^{A^{\mathrm{d}}} a$ which holds by reflexivity, and the latter follows from BC^{c} applied to $\uparrow_{A^{\mathrm{d}} \sqsubseteq B^{\mathrm{d}}} a \ {}_{B^{\mathrm{d}}}{\sqsubseteq}_{C^{\mathrm{d}}} c \Leftrightarrow a \ {}_{A^{\mathrm{d}}}{\sqsubseteq}_{C^{\mathrm{d}}} c$.

Otherwise, we are left with the cases where $AB = \widehat{\mathbb{N}}\text{-}\sqsubseteq$, $\widehat{\Pi}\text{-}\sqsubseteq$ or $\widehat{\square}\text{-}\sqsubseteq$ and $BC = \mathrm{Head}\text{-}\sqsubseteq$, we apply the inductive hypothesis to AB and $B \sqsubseteq \widehat{\mathrm{germ}}_j\ (\mathrm{head}\, B)$ in order to conclude with $\mathrm{Head}\text{-}\sqsubseteq$.

Finally, we show proof-irrelevance, *i.e.,* that for any A, B there is at most one derivation of $A \sqsubseteq B$. Since the conclusions of the rules do not overlap, we only have to prove that the premises of each rules are uniquely determined by the conclusion. This is immediate for $\widehat{\Pi}\text{-}\sqsubseteq$. For $\mathrm{Head}\text{-}\sqsubseteq$, $h = \mathrm{head}\, A \in \mathrm{Head}_i$ with $i = \mathrm{pred}\ j$ are uniquely determined by the conclusion so it holds too. □

6.5 Monotone Model of CastCIC$^{\uparrow}$

The monotone translation $\{-\}$ presented in Fig. 18 brings together the monotone interpretation of inductive types (e.g. $\ddot{\mathbb{N}}$), dependent products, the unknown type $\ddot{?}$ as well as the universe hierarchy. Following the approach of New and Ahmed [2018], casts are derived out of the canonical decomposition through the unknown type using the property (2) from Theorem 28:

$$\{\langle B \Leftarrow A \rangle\, t\} \quad := \quad \downarrow_{\mathrm{El}_\varepsilon \{B\} \sqsubseteq \widehat{?}} \uparrow_{\mathrm{El}_\varepsilon \{A\} \sqsubseteq \widehat{?}} \{t\}$$

Note that this definition formally depends on a chosen universe level j for $\widehat{?} : \square_j$, but the resulting operation is independent of this choice thanks to the section-retraction properties of ep-pairs. The difficult part of the model, the monotonicity of `cast`, thus holds by design. However, the translation of some terms do not reduce as in CastCIC: `cast` can get stuck on type variables eagerly, *e.g.,* on a Down-Err step.[22] These reduction rules still hold propositionally though so that we have at least a model in an extensional variant of the target theory (*i.e.,* in which two terms are *definitionally* equal whenever they are *propositionally* so).

Lemma 29. *If* $\Gamma \vdash_{\mathtt{cast}} t \rightsquigarrow u$ *then there exists a* $\mathrm{CIC}^{\mathrm{IR}}_{\mathrm{QIT}}$ *term* e *such that* $\{\Gamma\} \vdash_{\mathrm{IR}} e : \{t\} = \{u\}$.

We can further enhance this result using the fact that we assume functional extensionality in our target and can prove that the translation of all our types satisfy UIP. Under these assumptions, the conservativity results of Hofmann [1995] and Winterhalter et al. [2019] apply, so we can recover a translation targeting $\mathrm{CIC}^{\mathrm{IR}}_{\mathrm{QIT}}$.

Theorem 30 (Monotone model). *The translation* $\{-\}$ *of Fig. 18 extends to a model of* CastCIC *into* CIC *extended with induction-recursion and functional extensionality: if* $\Gamma \vdash_{\mathtt{cast}} t : A$ *then* $\{\!\!\{\Gamma\}\!\!\} \vdash_{\mathrm{IR}} \{t\} : \{\!\!\{A\}\!\!\}$.

It is unlikely that the principle that we demand in the target calculus $\mathrm{CIC}^{\mathrm{IR}}_{\mathrm{QIT}}$ are optimal. We conjecture that a variation of the translation described here could be developed in CIC extended only with induction-induction to describe the intensional content of the codes $\square$ in the universe, and strict propositions [Gilbert et al. 2019] following the construction of the setoid models of type theory [Altenkirch 1999; Altenkirch et al. 2021, 2019].

6.6 Back to Graduality

The precision order equipping each types of the monotone model can be reflected back to CastCIC, giving rise to the *propositional precision* judgment:

$$\Gamma \vdash_{\mathtt{cast}} t \ {}_{T}{\sqsubseteq}_{U}\ u \qquad := \qquad \exists e, \quad \{\!\!\{\Gamma\}\!\!\}_\varepsilon \vdash_{\mathrm{IR}} e : \{t\}\ {}_{\{T\}}{\sqsubseteq}_{\{U\}}\ \{u\}. \tag{2}$$

By the properties of the monotone model (Theorem 28), there is at most one witness up to propositional equality in the target that this judgment holds. This precision relation bears a similar relationship to the structural precision $\sqsubseteq_\alpha$ as propositional equality with definitional equality in

[22] An analysis of the correspondence between the discrete and monotone models can be found in Appendix C.

Monotone translation of contexts

$$
\begin{array}{lcl@{\qquad}lcl}
\{\!\!\{ \cdot \}\!\!\} & := & \cdot & \{\!\!\{ \Gamma, x : A \}\!\!\} & := & \{\!\!\{ \Gamma \}\!\!\}, x : \{\!\!\{ A \}\!\!\} \\
\{\!\!\{ \cdot \}\!\!\}_\varepsilon & := & \cdot & \{\!\!\{ \Gamma, x : A \}\!\!\}_\varepsilon & := & \{\!\!\{ \Gamma \}\!\!\}_\varepsilon, x_0 : \{\!\!\{ A \}\!\!\}_0, x_1 : \{\!\!\{ A \}\!\!\}_1, x_\varepsilon : \{\!\!\{ A \}\!\!\}_\varepsilon\, x_0\, x_1
\end{array}
$$

Monotone translation on terms and types

$$
\begin{array}{lcl@{\qquad}lcl}
\{\!\!\{ A \}\!\!\} & := & \mathrm{El}\,\{A\} : \square^{\leq} & \{\!\!\{ A \}\!\!\}_\varepsilon & := & \mathrm{El}_\varepsilon\,\{A\}_\varepsilon : \{\!\!\{ A \}\!\!\} \lhd \{\!\!\{ A \}\!\!\} \\
\\
\{x\} & := & x & \{x\}_\varepsilon & := & x_\varepsilon \\
\{\square_i\} & := & \widehat{\square}_i & \{\square_i\}_\varepsilon & := & \widehat{\square}\text{-}\sqsubseteq_i \\
\\
\{\Pi\, x : A.B\} & := & \widehat{\Pi}\,\{A\}\,\{\lambda\, x : A.B\}_\varepsilon & \{\Pi\, x : A.B\}_\varepsilon & := & \widehat{\Pi}\text{-}\sqsubseteq\,\{A\}_\varepsilon\,\{\lambda\, x : A.B\}_\varepsilon \\
\{t\ u\} & := & \{t\}\ \{u\} & \{t\ u\}_\varepsilon & := & \{t\}_\varepsilon\,\{u\}_0\,\{u\}_1\,\{u\}_\varepsilon \\
\{\lambda\, x : A.t\} & := & \lambda\, x : \{\!\!\{ A \}\!\!\}.\{t\} & \{\lambda\, x : A.\ t\}_\varepsilon & := & \lambda(x_0\, x_1 : \{\!\!\{ A \}\!\!\})(x_\varepsilon : \{\!\!\{ A \}\!\!\}_\varepsilon\, x_0\, x_1).\ \{t\}_\varepsilon \\
\{\mathbb{N}\} & := & \widehat{\mathbb{N}} & \{\mathbb{N}\}_\varepsilon & := & \widehat{\mathbb{N}}\text{-}\sqsubseteq \\
\{?_A\} & := & ?_{\{A\}} & \{?_A\}_\varepsilon & := & \mathtt{refl}\ \{\!\!\{ A \}\!\!\}\ ?_{\{A\}} \\
\{\mathtt{err}_A\} & := & \mathtt{err}_{\{A\}} & \{\mathtt{err}_A\}_\varepsilon & := & \mathtt{refl}\ \{\!\!\{ A \}\!\!\}\ \mathtt{err}_{\{A\}} \\
\{\langle B \Leftarrow A \rangle\, t\} & := & \downarrow_{\mathrm{El}_\varepsilon \{B\} \sqsubseteq \widehat{?}} \uparrow_{\mathrm{El}_\varepsilon \{A\} \sqsubseteq \widehat{?}} \{t\} & \{\langle B \Leftarrow A \rangle\, t\}_\varepsilon & := & \downarrow_{\mathrm{El}_\varepsilon \{B\} \sqsubseteq \widehat{?}}\text{-}\mathtt{mon} \uparrow_{\mathrm{El}_\varepsilon \{A\} \sqsubseteq \widehat{?}}\text{-}\mathtt{mon}\ \{t\}_\varepsilon
\end{array}
$$

$\{-\}_\alpha$ and $\{\!\!\{ - \}\!\!\}_\alpha$ where $\alpha \in \{0, 1\}$ stand for the variable-renaming counterparts of $\{-\}$ and $\{\!\!\{ - \}\!\!\}$.

Fig. 18. Translation of the monotone model

CIC. On the one hand, propositional precision can be used to prove precision statements inside the target type theory, for instance we can show by a straightforward case analysis on $b : \mathbb{B}$ that $b : \mathbb{B} \vdash_{\mathsf{cast}} \mathtt{if}\ b\ \mathtt{then}\ A\ \mathtt{else}\ A\ {}_{\square}{\sqsubseteq}_{\square}\ A$, a judgment that does not hold for syntactic precision. In particular, propositional precision is compatible with propositional equality, and a fortiori it is invariant by conversion in CastCIC: if $t \equiv t'$, $u \equiv u'$ and $\Gamma \vdash_{\mathsf{cast}} t\ {}_T{\sqsubseteq}_U\ u$ then $\Gamma \vdash_{\mathsf{cast}} t'\ {}_T{\sqsubseteq}_U\ u'$. On the other hand, propositional precision is not decidable, thus not suited for typechecking, where structural precision has to be used instead.

Lemma 31 (Compatibility of structural and propositional precision).

1. *If* $\vdash_{\mathsf{cast}} t : T$, $\vdash_{\mathsf{cast}} u : U$ *and* $\vdash t \sqsubseteq_\alpha u$ *then* $\vdash_{\mathsf{cast}} t\ {}_T{\sqsubseteq}_U\ u$.
2. *Conversely, if the target of the translation* $\mathsf{CIC}^{\mathsf{IR}}_{\mathsf{QIT}}$ *is logically consistent and* $\vdash_{\mathsf{cast}} v_1\ {}_{\mathbb{B}}{\sqsubseteq}_{\mathbb{B}}\ v_2$ *for normal forms* v_1, v_2, *then* $\vdash v_1 \sqsubseteq_\alpha v_2$.

Proof. For the first statement, we strengthen the inductive hypothesis, proving by induction on the derivation of structural precision the stronger statement:

If $\Gamma \vdash t \sqsubseteq_\alpha u$, $\Gamma_1 \vdash_{\mathsf{cast}} t : T$ and $\Gamma_2 \vdash_{\mathsf{cast}} u : U$ then there exists a term e such that

$$\{\!\!\{ \Gamma \}\!\!\} \vdash_{\mathrm{IR}} e : \{t\}\ {}_{\{T\}}{\sqsubseteq}_{\{U\}}\ \{u\}.$$

The cases for variables (Diag-Var) and universes (Diag-Univ) hold by reflexivity. The cases involving ? (Unk, Unk-Univ) and $\mathtt{err}$ (Err, Err-Lambda) amount to $\{?\}$ and $\{\mathtt{err}\}$ being respectively interpreted as top and bottom elements at each type. For Cast-R, we have $u = \langle B' \Leftarrow A' \rangle\, t'$, $B' = U$, and by induction hypothesis $\{\!\!\{ \Gamma \}\!\!\} \vdash e : \{t\}\ {}_{\{T\}}{\sqsubseteq}_{\{A'\}}\ \{t'\}$ and $\{\!\!\{ \Gamma \}\!\!\} \vdash \{T\} \sqsubseteq \{B'\}$. Let j be a universe level such that $\{A'\} \sqsubseteq \widehat{?}_j$, $\{B'\} \sqsubseteq \widehat{?}_j$. By (heterogeneous) transitivity of precision applied to e and a witness of $\{\!\!\{ \Gamma \}\!\!\} \vdash \{t'\}\ {}_{\{A'\}}{\sqsubseteq}_{\widehat{?}_j} \uparrow_{\{A'\} \sqsubseteq \widehat{?}^j} \{t'\}$ (Lemma 27), we obtain a proof e' of $\{\!\!\{ \Gamma \}\!\!\} \vdash e' : \{t\}\ {}_{\{T\}}{\sqsubseteq}_{\{B'\}} \uparrow_{\{A'\} \sqsubseteq \widehat{?}^j} \{t'\}$ and finally, using the adjunction property, a proof e'' of

$$\{\!\!\{ \Gamma \}\!\!\} \vdash e : \{t\}\ {}_{\{T\}}{\sqsubseteq}_{\{B'\}} \downarrow_{\{B'\} \sqsubseteq \widehat{?}^j} \uparrow_{\{A'\} \sqsubseteq \widehat{?}^j} \{t'\} \equiv \{\langle B' \Leftarrow A' \rangle\, t'\} \equiv \{u\}.$$

The case CAST-L proceeds in an entirely symmetric fashion since we only use the adjunction laws. All the other cases, being congruence rules with respect to some term constructor, are consequences of the monotonicity of said term constructor with a direct application of the inductive hypothesis and inversion of the typing judgments.

For the second statement, by progress (Theorem 8), both v_1 and v_2 are canonical booleans, so we can proceed by case analysis on the canonical forms v_1 and v_2 that are either $\mathtt{true}$, $\mathtt{false}$, $\mathtt{err}_{\mathbb{B}}$ or $?_{\mathbb{B}}$, ruling out the impossible cases by inversion of the premise $\vdash_{\mathsf{cast}} v_1 \;{}_{\mathbb{B}}{\sqsubseteq}_{\mathbb{B}}\; v_2$ and logical consistency of $\vdash_{\mathrm{IR}}$. Out of the 16 cases, we obtain that only the following 9 cases are possible:

$$\begin{array}{lll}
\vdash_{\mathsf{cast}} \mathtt{err}_{\mathbb{B}} \;{}_{\mathbb{B}}{\sqsubseteq}_{\mathbb{B}}\, \mathtt{err}_{\mathbb{B}} & \vdash_{\mathsf{cast}} \mathtt{err}_{\mathbb{B}} \;{}_{\mathbb{B}}{\sqsubseteq}_{\mathbb{B}}\, \mathtt{true} & \vdash_{\mathsf{cast}} \mathtt{err}_{\mathbb{B}} \;{}_{\mathbb{B}}{\sqsubseteq}_{\mathbb{B}}\, \mathtt{false} \\
\vdash_{\mathsf{cast}} \mathtt{err}_{\mathbb{B}} \;{}_{\mathbb{B}}{\sqsubseteq}_{\mathbb{B}}\, ?_{\mathbb{B}} & \vdash_{\mathsf{cast}} \mathtt{true} \;{}_{\mathbb{B}}{\sqsubseteq}_{\mathbb{B}}\, \mathtt{true} & \vdash_{\mathsf{cast}} \mathtt{true} \;{}_{\mathbb{B}}{\sqsubseteq}_{\mathbb{B}}\, ?_{\mathbb{B}} \\
\vdash_{\mathsf{cast}} \mathtt{false} \;{}_{\mathbb{B}}{\sqsubseteq}_{\mathbb{B}}\, \mathtt{false} & \vdash_{\mathsf{cast}} \mathtt{false} \;{}_{\mathbb{B}}{\sqsubseteq}_{\mathbb{B}}\, ?_{\mathbb{B}} & \vdash_{\mathsf{cast}} ?_{\mathbb{B}} \;{}_{\mathbb{B}}{\sqsubseteq}_{\mathbb{B}}\, ?_{\mathbb{B}}
\end{array}$$

For each case, a corresponding rule exists for the structural precision, proving that $\vdash v_1 \sqsubseteq_{\alpha} v_2$. □

With a similar method, we show that $\mathsf{CastCIC}^{\uparrow}$ satisfies graduality, which is the key missing point of §5 and the *raison d'etre* of the monotone model.

THEOREM 32 (Graduality for $\mathsf{CastCIC}^{\uparrow}$). *For $\Gamma \vdash_{\mathsf{cast}} t : T$, $\Gamma \vdash_{\mathsf{cast}} t' : T$ and $\Gamma \vdash_{\mathsf{cast}} u : U$, we have*

- *(DGG) If $\Gamma \vdash_{\mathsf{cast}} t \;{}_{T}{\sqsubseteq}_{T}\; t'$ then $t \sqsubseteq^{obs} t'$;*
- *(Ep-pairs) If $\Gamma \vdash_{\mathsf{cast}} T \;{}_{\square}{\sqsubseteq}_{\square}\; U$ then*

$$\Gamma \vdash_{\mathsf{cast}} \langle U \Leftarrow T\rangle\, t \;{}_{U}{\sqsubseteq}_{U}\; u \Leftrightarrow \Gamma \vdash_{\mathsf{cast}} t \;{}_{T}{\sqsubseteq}_{U}\; u \Leftrightarrow \Gamma \vdash_{\mathsf{cast}} t \;{}_{T}{\sqsubseteq}_{T}\; \langle T \Leftarrow U\rangle\, u,$$

Furthermore, $\Gamma \vdash_{\mathsf{cast}} \langle T \Leftarrow U\rangle \langle U \Leftarrow T\rangle\, t \sqsupseteq\sqsubseteq t$.

Proof.

- (DGG) Let $C[-] : (\Gamma \vdash T) \Rightarrow (\vdash \mathbb{B})$ be an observation context, by monotonicity of the translation $\vdash_{\mathsf{cast}} C[t] \;{}_{\mathbb{B}}{\sqsubseteq}_{\mathbb{B}}\; C[t']$. By progress, subject reduction (Theorem 8) and strong normalization (Theorem 9) of $\mathsf{CastCIC}^{\uparrow}$, there exists canonical forms $\vdash_{\mathsf{cast}} v, v' : \mathbb{B}$ such that $C[t] \equiv v$ and $C[t'] \equiv v'$. Since propositional precision is stable by conversion in CastCIC, $\vdash_{\mathsf{cast}} v' \sqsupseteq\sqsubseteq v$. Finally, we conclude that $v \equiv v'$ by a case analysis on the boolean normal forms v and v', that are either $\mathtt{true}$, $\mathtt{false}$, $\mathtt{err}_{\mathbb{B}}$ or $?_{\mathbb{B}}$: if v and v' are distinct normal forms then $\vdash_{\mathsf{cast}} v' \sqsupseteq\sqsubseteq v$ is a closed proof of an empty type, contradicting the consistency of the target.
- (Ep-pairs) The fact that propositional precision induces an adjunction is a direct reformulation of the fact that the relation ${}_{\{T\}}{\sqsubseteq}_{\{U\}}$ underlies an ep-pair (Theorem 28.(3)), using the fact that there is at most one upcast and downcast between two types. Similarly, the equi-precision statement is an application of the first point to the proofs

$$\begin{cases} \{\!|\Gamma|\!\} \vdash_{\mathrm{IR}} \_ : \{t\} \;{}_{\{T\}}{\sqsubseteq}_{\{T\}}\; \{\langle U \Leftarrow T\rangle \langle T \Leftarrow U\rangle\, t\} \\ \{\!|\Gamma|\!\} \vdash_{\mathrm{IR}} \_ : \{\langle T \Leftarrow U\rangle \langle U \Leftarrow T\rangle\, t\} \;{}_{\{T\}}{\sqsubseteq}_{\{T\}}\; \{t\} \end{cases}$$

  which holds because $\downarrow_{\{T\}\sqsubseteq\{U\}} \circ \uparrow_{\{T\}\sqsubseteq\{U\}} = \mathrm{id}$ in the monotone model.

□

We conjecture that the target $\mathsf{CIC}^{\mathsf{IR}}_{\mathsf{QIT}}$ mentioned in the above theorem and propositions is consistent relative to a strong enough metatheory,[23] that is the assumed inductive-recursive definition for the universe does not endanger consistency. As can be seen from the proof, this hypothesis allows to move from a contradiction internal to $\mathsf{CIC}^{\mathsf{IR}}_{\mathsf{QIT}}$ to a contradiction in the ambient metatheory.

[23] For instance, $\mathcal{ZFC}$ + the existence of Mahlo cardinals [Dybjer and Setzer 2003; Forsberg 2013; Setzer 2000].

6.7 Graduality of $\mathsf{CastCIC}^{\mathcal{G}}$

To prove graduality of $\mathsf{CastCIC}^{\mathcal{G}}$, we need to provide a model accounting for both monotony and non-termination. The monotone model presented in the previous sections, which gives us graduality for $\mathsf{CastCIC}^{\uparrow}$ and can be related to the pointed model of New and Licata [2020, Section 6.1], only accounts for terminating functions. In order to capture also non-termination, we can adapt the Scott model of New and Licata [2020, Section 6.2] based on pointed ω-cpo to our setting. We now explain the construction of the main type formers, overloading the notations from the previous sections.

Types are interpreted as bipointed ω-cpos, that is as orders $(A, \sqsubseteq)$ equipped with a smallest element $\mathtt{err}_A \in A$, a largest element $?_A \in A$ and an operation $\sup_i\ a_i$ computing the suprema of countable ascending chains, *i.e.*, sequences $(a_i)_{i\in\omega} \in A^\omega$ indexed by the ordinal $\omega = \{0 < 1 < \ldots\}$ such that $a_i \sqsubseteq^A a_j$ whenever $i < j$. A monotone function $f : A \to B$ between ω-cpos is called ω-*continuous* if for any ascending chain $(a_k)_{k\in\omega}$, $\sup_k f\ a_k = f(\sup_k a_k)$; we write $d : A \lhd^\omega B$ for an ep-pair between ω-cpos where $\uparrow_d$ preserves suprema (the left adjoint $\downarrow_d$ automatically preserves suprema).

A type-theoretical construction of the free ω-cpo on a set is described in [Bidlingmaier et al. 2019; Chapman et al. 2019] using quotient-inductive-inductive types (QIIT) [Altenkirch et al. 2018; Kaposi et al. 2019]. We can adapt this technique to provide an interpretation for inductive types, and in particular natural numbers, throwing in freely a new constructor sup denoting the suprema of any chain of elements and quotienting by the appropriate (in)equations: the suprema of a chain is greater than any of its parts $a_i \sqsubseteq \sup\ a_i$ and an element b that is greater than a chain $\forall i, a_i \sqsubseteq b$ is greater than its suprema $\sup a_i \sqsubseteq b$. Functions $f : A \to B$ between types are interpreted as continuous monotone maps, and the type $A \to^\omega B$ of continuous monotone functions is an ω-cpo with suprema computed pointwise. As in §6.3, the construction of the ω-cpo corresponding to the unknown type is intertwined with the universe hierarchy. Assuming by induction that we have ω-cpos $\ddot{\square}_0, \ldots, \ddot{\square}_i$ for universes at level lower than i, we follow the seminal work of Scott [1976] on domains and we take the unknown type $\ddot{?}_{i+1}$ to be a solution to the recursive equation:

$$\ddot{?}_{i+1} \quad \cong \quad \ddot{\mathbb{N}} + (\ddot{?}_{i+1} \to^\omega \ddot{?}_{i+1}) + \ddot{\square}_0 + \ldots + \ddot{\square}_i$$

The key techniques to build a solution to this equation in the setting of ω-cpos are detailed in [Smyth and Plotkin 1977; Wand 1979]. In a nutshell, this construction amounts to iterate the assignment $F(X) := \ddot{\mathbb{N}} + (X \to^\omega X) + \ddot{\square}_0 + \ldots + \ddot{\square}_i$ starting from the initial bipointed ω-cpo $\ddot{\mathbb{0}}$—the free bipointed ω-cpo on an empty type, consisting just of $\bot_{\ddot{\mathbb{0}}} \sqsubseteq \top_{\ddot{\mathbb{0}}}$—and to take the colimit of the induced sequence:

$$\ddot{\mathbb{0}} \lhd^\omega F(\ddot{\mathbb{0}}) \lhd^\omega \ldots\ F^k(\ddot{\mathbb{0}})\ \ldots\ \lhd^\omega \operatorname{colim}_k F^k(\ddot{\mathbb{0}}) =: \ddot{?}_{i+1} \tag{3}$$

For this construction to succeed, F should extend to an ω-continuous functor on the category of ω-cpos and ω-continuous ep-pairs that moreover preserves countable sequential colimits as above so that the following hold:

$$\ddot{?}_{i+1} = \operatorname{colim}_k F^k(\ddot{\mathbb{0}}) \cong \operatorname{colim}_k F^{k+1}(\ddot{\mathbb{0}}) \cong F(\operatorname{colim}_k F^k(\ddot{\mathbb{0}})) = F(\ddot{?}_{i+1}).$$

The construction of $\ddot{?}_{i+1}$ as a fixpoint for F should be contrasted with the construction from §6.3 where we essentially describe explicitly a construction of $\ddot{?}_{i+1} := F(\ddot{?}_i)$ in the setting of bipointed posets. The existence of countable sequential colimits in the category of ω-cpos and ep-pairs, as employed in Eq. (3), is an interesting fact proved in [Wand 1979, Theorem 3.1], which we also use to equip the next universe of codes $\ddot{\square}_{i+1}$ with an ω-cpo structure. In brief, we adapt the inductive description of the universe of codes $\ddot{\square}_i$ given in Fig. 17 with an additional code $\sup\ (A_k)_{k\in\omega} : \ddot{\square}_i$ for suprema of chains of codes $A_k : \ddot{\square}_i$, and decode it with the function El satisfying $\mathrm{El}\,(\sup\,(A_k)_{k\in\omega}) \cong \mathrm{colim}_k\,(\mathrm{El}\,A_k)$. However, the isomorphism above cannot be used as a definition because the definition of El has to respect the quotiented nature of $\square_i$. In particular, when the chain is the constant chain $(\widehat{\mathbb{N}})_{k\in\omega}$, $\sup\,(\widehat{\mathbb{N}})_{k\in\omega} = \widehat{\mathbb{N}}$ and thus $\mathrm{El}\,(\sup\,(\widehat{\mathbb{N}})_{k\in\omega})$ must also be equal to $\ddot{\mathbb{N}}$, which is different from $\mathrm{colim}_k\,\ddot{\mathbb{N}}$. Technically, we define $\mathrm{El}\,A$ as its isomorphic image onto $\ddot{?}_i$, recovering a canonical choice for the inhabitants of $\mathrm{El}\,(\sup\,(A_k)_{k\in\omega})$.

Note that in contrast with the construction from §6.3 that depended on germ_i and hence El_i, the present construction of $\ddot{?}_i$ does not depend on the construction of $\square_i$ and El_i, cutting the non-wellfounded loop observed in §6.1.

The components that we describe assemble as a model of $\mathsf{CastCIC}^{\mathcal{G}}$ into $\mathsf{CIC}^{\mathsf{IR}}_{\mathsf{QIT}}$. In order to be able to prove DGG for $\mathsf{CastCIC}^{\mathcal{G}}$, we first need to characterize the semantic interpretation of diverging terms of type $\mathbb{B}$ in the model.

Lemma 33. *If $\Gamma \vdash_{\mathsf{cast}} t \triangleright \mathbb{B}$ and t has no weak head normal form, then $\{t\} = \mathtt{err}_{\{\mathbb{B}\}}$.*

Proof. The proof of this lemma is based on the definition of a logical relation which is shown to relate t to its translation $\{t\}$ in the model (a.k.a. the fundamental lemma). The precise definition of the logical and proof of the fundamental lemma is given in Appendix D. □

Relativizing the notion of precision $\Gamma \vdash_{\mathsf{cast}} t \;{}_T{\sqsubseteq}_S\; u$ of the monotone model to use the order induced by this ω-cpo model instead of the monotone model, we can replay the steps of Theorem 32 and derive graduality for $\mathsf{CastCIC}^{\mathcal{G}}$.

Theorem 34 (Graduality for $\mathsf{CastCIC}^{\mathcal{G}}$). *For $\Gamma \vdash_{\mathsf{cast}} t : T$, $\Gamma \vdash_{\mathsf{cast}} t' : T$ and $\Gamma \vdash_{\mathsf{cast}} u : U$, we have*

- *(DGG) If $\Gamma \vdash_{\mathsf{cast}} t \;{}_T{\sqsubseteq}_T\; t'$ then $t \sqsubseteq^{obs} t'$;*
- *(Ep-pairs) If $\Gamma \vdash_{\mathsf{cast}} T \;{}_\square{\sqsubseteq}_\square\; U$ then*

$$\Gamma \vdash_{\mathsf{cast}} \langle U \Leftarrow T\rangle\, t \;{}_U{\sqsubseteq}_U\; u \Leftrightarrow \Gamma \vdash_{\mathsf{cast}} t \;{}_T{\sqsubseteq}_U\; u \Leftrightarrow \Gamma \vdash_{\mathsf{cast}} t \;{}_T{\sqsubseteq}_T\; \langle T \Leftarrow U\rangle\, u,$$

Furthermore, $\Gamma \vdash_{\mathsf{cast}} \langle T \Leftarrow U\rangle\, \langle U \Leftarrow T\rangle\, t \sqsupseteq\sqsubseteq t$.

Proof.

- (DGG) Similarly to the proof of Theorem 32, we consider a context $C[-] : (\Gamma \vdash T) \Rightarrow (\vdash \mathbb{B})$. We know by monotonicity of the translation that $\vdash_{\mathsf{cast}} C[t] \;{}_{\mathbb{B}}{\sqsubseteq}_{\mathbb{B}}\; C[t']$. We need to distinguish whether the evaluation of $C[t]$ and $C[t']$ terminates or not. If $C[t]$ diverges, we are done. If $C[t]$ terminates and $C[t']$ diverges, by progress, $C[t]$ reduces to a value v and by Lemma 33, $\{C[t']\} = \mathtt{err}_{\{\mathbb{B}\}}$. This means that $\{C[t]\} = \mathtt{err}_{\{\mathbb{B}\}}$ because $\mathtt{err}_{\{\mathbb{B}\}}$ is the smallest element of $\ddot{\mathbb{B}}$. Since $\{-\}$ is stable by conversion, $\{v\} = \{C[t]\} = \mathtt{err}_{\{\mathbb{B}\}}$, and so $v = \mathtt{err}_{\mathbb{B}}$ by case analysis of the possible values for v. If both terminates, then the reasoning is the same as in the proof of Theorem 32.
- (Ep-pairs) As for Theorem 32, this fact derives directly from the interpretation of the precision order as ep-pairs in the ω-cpo model.

□

7 GRADUAL INDEXED INDUCTIVE TYPES

We now explore how indexed inductive types, as used in the introduction (Example 1), can be handled in GCIC. Recall the definition of vec:

```
Inductive vec (A : □) : ℕ → □ :=
| nil  : vec A 0
| cons : A → forall n : ℕ, vec A n → vec A (S n).
```

and recall the difference between *parameters* (here, A), which are common to all constructors, and *indices* (here, n), which can differ between constructors. Also recall from §4 that our formal development does not consider indexed inductive types, only parametrized ones.

This section first explains two alternatives to indexed inductive types that can directly be expressed in GCIC (§ 7.1). We then describe how these alternatives actually behave in the gradual setting (§ 7.2 and 7.3). Finally, we present an extension of CastCIC to directly support indexed inductive types, focusing on the specific case of vectors (§ 7.4), showing that it combines the advantages of the other approaches. § 7.5 summarizes our findings.

7.1 Alternatives to indexed inductive types

Indexed inductive types make it possible to define structures that are intrinsically characterized by some property, which holds by construction, as opposed to extrinsically establishing such properties after the fact. There are two well-known alternatives to indexed inductive types for capturing properties intrinsically: type-level fixpoints, and "forded" inductive types.

Type-level fixpoint. The vector can be defined as a recursive function on the index, at the type level. For instance, the following formulation represents sized lists as nested pairs:

$\texttt{Fixpoint}\ \texttt{vec}_\mu\ (\mathsf{A} : \square)\ (\mathsf{n} : \mathbb{N}) : \square := \texttt{match}\ \mathsf{n}\ \texttt{with}\ 0 \Rightarrow \texttt{unit} \mid \mathsf{S}\ \mathsf{n} \Rightarrow \mathsf{A} * \texttt{vec}_\mu\ \mathsf{A}\ \mathsf{n}\ \texttt{end}.$

Type-level fixpoints can be used as soon as the indices are *concretely forceable* [Brady et al. 2004]. Intuitively, concretely forceable indices are those that can be matched upon (like n in this example definition). See Gilbert et al. [2019] for a description of a general translation.

Forded inductive type. Instead of using an indexed inductive type, one can use a parametrized inductive type, with *explicit* equalities as arguments to constructors.[24] For instance, vectors can be defined in this style as follows:

$\texttt{Inductive}\ \texttt{vec}_f\ (\mathsf{A} : \square)\ (\mathsf{n} : \mathbb{N}) : \square :=$
$\mid \texttt{nil}_f : \texttt{eq}_\mathbb{N}\ 0\ \mathsf{n} \to \texttt{vec}_f\ \mathsf{A}\ \mathsf{n}$
$\mid \texttt{cons}_f : \mathsf{A} \to \texttt{forall}\ \mathsf{m} : \mathbb{N},\ \texttt{eq}_\mathbb{N}\ (\mathsf{S}\ \mathsf{m})\ \mathsf{n} \to \texttt{vec}_f\ \mathsf{A}\ \mathsf{m} \to \texttt{vec}_f\ \mathsf{A}\ \mathsf{n}.$

Note that this definition uses $\texttt{eq}_\mathbb{N}$, the type of decidable equality proofs over natural numbers, for expressing the constraints on n instead of propositional equalities (*e.g.*, 0=n), because propositional equality is not available in GCIC (§ 8.3).

In CIC, these two alternative presentations of an indexed inductive type can be shown internally to be equivalent. But each of these presentations has advantages and drawbacks depending on the considered system and scenarios of use, so practitioners have different preferences in that respect. More important to us here, these presentations are *not* equivalent in GCIC.

7.2 Type-level fixpoints

Constructors. The definition of $\texttt{vec}_\mu$ above can directly be written in GCIC, as it uses only inductive types with parameters (here the unit and product types and natural numbers). The vector constructors can be defined as:

$\texttt{Definition}\ \texttt{nil}_\mu\ (\mathsf{A}{:}\square) : \texttt{vec}_\mu\ \mathsf{A}\ 0 := \texttt{tt}.$

$\texttt{Definition}\ \texttt{cons}_\mu\ (\mathsf{A}{:}\square)\ (\mathsf{a}{:}\mathsf{A})\ (\mathsf{n}{:}\mathbb{N})\ (\mathsf{v}{:}\texttt{vec}_\mu\ \mathsf{A}\ \mathsf{n}) : \texttt{vec}_\mu\ \mathsf{A}\ (\mathsf{S}\ \mathsf{n}) := (\mathsf{a}\ ,\ \mathsf{v}).$

whose definitions typecheck because $\texttt{vec}_\mu$ computes on its indices.

Behavior. Let us now look at the type computed at $?_\mathbb{N}$. Because $?_\mathbb{N}$ is an exceptional term, the fixpoint has to return unknown in the universe: $\texttt{vec}_\mu\ \mathsf{A}\ ?_\mathbb{N} \rightsquigarrow^* ?_\square$. This means that the mechanism for casting a vector into a vector with the unknown index is directly inherited from the generic mechanism for casting to the unknown type. Therefore, we get for free the following computation rules, because they involve embedding-projection pairs:

[24] This technique has reportedly been coined "fording" by McBride [1999, §3.5]. Fording is in allusion to the Henry Ford quote *"Any customer can have a car painted any color that he wants, so long as it is black."*

$\texttt{nil}_\mu\ \mathsf{A} :: \texttt{vec}_\mu\ \mathsf{A}\ ?_{\mathbb{N}} :: \texttt{vec}_\mu\ \mathsf{A}\ 0 \rightsquigarrow^* \texttt{nil}_\mu\ \mathsf{A}$

$\texttt{nil}_\mu\ \mathsf{A} :: \texttt{vec}_\mu\ \mathsf{A}\ ?_{\mathbb{N}} :: \texttt{vec}_\mu\ \mathsf{A}\ 1 \rightsquigarrow^* \texttt{err}$

Similarly, the eliminator $\texttt{vec}_\mu\texttt{\_rect}$ can be defined by first matching on the index, and then on the vector and satisfies the computation rule of vectors when the index is non-exceptional. The only drawback of this encoding is that the behavior of the eliminator is not satisfactory when the index is unknown. Consider for instance the following term from Example 1, which unfortunately reduces to $?_{\mathbb{N}}$:

$\texttt{head}_\mu\ ?_{\mathbb{N}}\ (\texttt{filter}_\mu\ \mathbb{N}\ 4\ \texttt{even}\ [\ 0\ ;\ 1\ ;\ 2\ ;\ 3\ ]) \rightsquigarrow^* ?_{\mathbb{N}}$

This behavior occurs because the eliminator starts by matching on the index, which is unknown, and thus has to return the unknown itself.

7.3 Fording with decidable equalities

Constructors. With the definition of the forded inductive type $\texttt{vec}_f$, the $\texttt{nil}_f$ constructor can legitimately be used to inhabit $\texttt{vec}_f\ \mathsf{A}\ ?_{\mathbb{N}}$, provided we have an inhabitant (possibly ?) of $\mathsf{eq}_{\mathbb{N}}\ 0\ \mathsf{n}$.

Note that we can provide the same vector interface as that of the indexed inductive type by defining the following constructor wrappers, using the term $\texttt{refl n}$ of reflexivity on $\mathsf{eq}_{\mathbb{N}}$:

```
Definition nil' (A:□) : vec_f A 0 := nil_f A (refl 0).

Definition cons' A a n (v: vec_f A n) : vec_f A (S n) := cons_f A a n (refl n) v.
```

and define the corresponding eliminator `vec_rect'` accordingly.

Behavior. The computational content of the eliminator on $\texttt{vec}_f\ \mathsf{A}\ ?_{\mathbb{N}}$ is more precise than with $\texttt{vec}_\mu$: the eliminator never matches on the proof of equality to produce a term, but only to guarantee that a branch is not accessible. Concretely, this means that we observe the expected reduction:

$\texttt{nil}_f\ \mathsf{A}\ \mathsf{e} :: \texttt{vec}_f\ \mathsf{A}\ ?_{\mathbb{N}} :: \texttt{vec}_f\ \mathsf{A}\ 0 \rightsquigarrow^* \texttt{nil}_f\ \mathsf{A}\ \mathsf{e}$

Again, the fact that upcasting to $\texttt{vec}_f\ \mathsf{A}\ ?_{\mathbb{N}}$ and then downcasting back is the identity relies on the CastCIC mechanism on the unknown for the universe, but this time only for the type representing the decidable equality. Likewise, the example of `filter` (Example 1) computes as expected:

$\texttt{head}_f\ ?_{\mathbb{N}}\ (\texttt{filter}_f\ \mathbb{N}\ 4\ \texttt{even}\ [\ 0\ ;\ 1\ ;\ 2\ ;\ 3\ ]) \rightsquigarrow^* 0$

On the other hand, an invalid assertion does not produce an error, but a term with an error in place of the equality proof:

$\texttt{nil}_f\ \mathsf{A}\ \mathsf{e} :: \texttt{vec}_f\ \mathsf{A}\ ?_{\mathbb{N}} :: \texttt{vec}_f\ \mathsf{A}\ 1 \rightsquigarrow^* \texttt{nil}_f\ \mathsf{A}\ \texttt{err}$

where `err` is at type $\mathsf{eq}_{\mathbb{N}}\ 1\ 0$. Consequently, we have $\texttt{head}_f\ ?_{\mathbb{N}}\ (\texttt{filter}_f\ \mathbb{N}\ 2\ \texttt{even}\ [\ 1\ ;\ 3\ ]) \rightsquigarrow^* \texttt{err}$, because the branch of $\texttt{head}_f$ that deals with the `nil` case matches on the (erroneous) equality proof. Invalid assertions are therefore very lazily observed, if at all, which is not satisfactory.

Finally, there is a drawback of using decidable equalities, which only manifests when working with the original vector interface (`nil'`/`cons'`/`vec_rect'`). In that case, the eliminator does not enjoy the expected computational rule on the constructor `cons'`. Because the eliminator is defined by induction on natural numbers, therefore it only reduces when the index is a concrete natural number, not a variable.

7.4 Direct support for indexed inductive types: the case of vectors

Extending GCIC/CastCIC with direct support for indexed inductive types can provide a fully satisfactory solution, in contrast to the two previously-exposed encodings that both have serious shortcomings. The idea is to reason about indices directly in the reduction of casts. Here, we expose this approach for the specific case of length-indexed vectors and leave a generalization to future

$$\frac{}{\text{canonical}\,\texttt{nil}_?\,A} \qquad \frac{}{\text{canonical}(\texttt{cons}_?\,A\;a\;n\;v)}$$

V-CONS-? : $\langle \texttt{vec}\;B\;?_{\mathbb{N}} \Leftarrow \texttt{vec}\;A\;(S\;n)\rangle\;(\texttt{cons}\;A\;a\;k\;v) \rightsquigarrow \texttt{cons}_?\;B\;(\langle B \Leftarrow A\rangle\;a)\;n\;(\langle \texttt{vec}\;B\;n \Leftarrow \texttt{vec}\;A\;k\rangle\;v)$

V-CONS : $\langle \texttt{vec}\;B\;(S\;m) \Leftarrow \texttt{vec}\;A\;(S\;n)\rangle\;(\texttt{cons}\;A\;a\;k\;v) \rightsquigarrow \texttt{cons}\;B\;(\langle B \Leftarrow A\rangle\;a)\;m\;(\langle \texttt{vec}\;B\;m \Leftarrow \texttt{vec}\;A\;k\rangle\;v)$

V-CONS-NIL : $\langle \texttt{vec}\;B\;0 \Leftarrow \texttt{vec}\;A\;(S\;n)\rangle\;(\texttt{cons}\;A\;a\;k\;v) \rightsquigarrow \texttt{err}_{\texttt{vec}\;B\;0}$

V-RECT-NILU : $\texttt{vec\_rect}\;P\;P_{nil}\;P_{cons}\;\texttt{nil}_?\;A \rightsquigarrow \langle P\;?_{\mathbb{N}} \Leftarrow P\;0\rangle\;(\texttt{vec\_rect}\;P\;P_{nil}\;P_{cons}\;\texttt{nil}\;A)$

V-RECT-CONSU : $\texttt{vec\_rect}\;P\;P_{nil}\;P_{cons}\;(\texttt{cons}_?\;A\;a\;n\;v) \rightsquigarrow$
$\langle P\;?_{\mathbb{N}} \Leftarrow P\;(S\;n)\rangle\;(\texttt{vec\_rect}\;P\;P_{nil}\;P_{cons}\;(\texttt{cons}\;A\;a\;n\;v))$

Fig. 19. New canonical forms and reduction rules for vectors in CastCIC (excerpt).

work. Appendix E describes the extension for vectors in full details; here, we only present selected rules (Fig. 19) and illustrate how reduction works.

Constructors. We add two new canonical forms, corresponding to the casts of `nil` and `cons` to $\texttt{vec}\;A\;?_{\mathbb{N}}$: namely, $\texttt{nil}_?\;A$ and $\texttt{cons}_?\;A\;a\;n\;v$ (Fig. 19). Note that we cannot simply declare casts such as $\langle \texttt{vec}\;A\;?_{\mathbb{N}} \Leftarrow \texttt{vec}\;A\;n\rangle\;t$ to be canonical, because they involve non-linear occurrences of types (here, A).

Reduction rules. We add reduction rules to conduct casts between vectors in canonical forms. Fig. 19 presents these rules when the argument of the cast is a `cons`. Rule V-CONS-? propagates the cast on the arguments, but using the newly-introduced $\texttt{cons}_?$, effectively converting precise information to less precise information. Rule V-CONS applies when both source and target indices are successors, and propagates the cast of the arguments, just like the standard rule for casting a constructor. As expected, Rule V-CONS-NIL raises an error when the indices do not match.

For the eliminator, there are two new computation rules, one for each new constructor: V-RECT-NILU and V-RECT-CONSU. They both apply the eliminator to the underlying non-exceptional constructor, and then cast the result back to $P\;?_{\mathbb{N}}$. Intuitively, these rules transfer the cast on vectors to a cast on the returned type of the predicate.

Behavior. Given these rules, we can actually realize the behavior described in Example 1. For instance, we have both

$\texttt{nil A} :: \texttt{vec A}\;?_{\mathbb{N}} :: \texttt{vec A}\;0 \rightsquigarrow^* \texttt{nil A}$

$\texttt{nil A} :: \texttt{vec A}\;?_{\mathbb{N}} :: \texttt{vec A}\;1 \rightsquigarrow^* \texttt{err}_{\texttt{A}}$

and coming back to Example 1, in all three GCIC variants the term:

$$\texttt{head}\;?_{\mathbb{N}}\;(\texttt{filter}\;\mathbb{N}\;4\;\texttt{even}\;[\;0\;;\;1\;;\;2\;;\;3\;])$$

typechecks and reduces to 0. Additionally, as expected:

$$\texttt{head}\;?_{\mathbb{N}}\;(\texttt{filter}\;\mathbb{N}\;2\;\texttt{even}\;[1\;;\;3])$$

typechecks and fails at runtime. And similarly for Example 4.

Note that to be able to define the action of casts on vectors, we have crucially used the fact that it is possible to discriminate between 0, `S n` and $?_{\mathbb{N}}$ in the reduction rule.

7.5 Summary

To summarize, the different approaches to define structures with intrinsic properties in GCIC compare as follows:

- The type-level fixpoint coincides with the indexed inductive presentation on non-exceptional terms, but is extremely imprecise in presence of unknown indices.
- The forded inductive is more accurate when dealing with unknown indices, but is arguably too permissive with invalid index assertions.
- The direct support of the indexed inductive type with additional constructors and reduction rules yields a satisfactory solution. We conjecture that this presentation can be generalized to support arbitrary indexed inductive types as long as they have concretely forceable indices; we leave such a general construction for future work.

Recall that fording is only an option in GCIC when the indices pertain to a type with decidable equality; properly handling general propositional equality in a gradual type theory is an open question (§8.3). The constraint of indices being concretely forceable (for type-level fixpoints, direct support) are intuitively understandable and expected: gradual typing requires synthesizing dynamic checks, therefore these checks need to be somehow computable.

8 LIMITATIONS AND PERSPECTIVES

Up to now, we have left aside three important aspects of CIC, namely, impredicativity, η-equality and propositional equality. This section explains the challenges induced by each feature, and possibly, venues to explore.

8.1 Impredicativity

In this work, we do not deal with the impredicative sort `Prop`, for multiple reasons. The models used in §6 to justify termination and graduality crucially rely on the predicativity of the universe hierarchy for the inductive-recursive definition of codes to be well-founded. Moreover, the results of Palmgren [1998, Theorem 6.1] show that it is not possible to endow an impredicative universe with an inductive-recursive structure in a consistent and strongly-normalizing theory, hinting that it may be difficult to devise an inductively-defined cast function between types that belong to an impredicative universe. Additionally, it seems difficult to avoid the divergence of Ω with an impredicative sort, as no universe levels can be used to prevent a self-application from being well-typed.

8.2 η-equality

In most presentations of CIC, and in particular its Coq implementation, conversion satisfies an additional rule, called η-equality, which corresponds to an extensional property for functions:

$$\Gamma \vdash f \equiv \lambda\, x : A.f\ x \quad \text{when} \quad \Gamma \vdash f : \Pi x : A.B.$$

The difficulty of integrating η-equality in the setting of GCIC is that the conversion we consider in CastCIC is entirely induced by a notion of reduction: two terms are convertible exactly when they have a common reduct up to α-equivalence. It is well-known that η-equality cannot be easily modeled using a rewrite rule, as both η-expansion and η-reduction have significant drawbacks [Goguen 2005], and so we would have to consider another approach to the one we took if we were to integrate η-equality. The most prominent alternative way is to define conversion as an alternation of reduction steps (for instance using a weak-head reduction strategy) not containing η and comparison of terms up to congruence and η-equality.

This approach has been recently formalized by Abel et al. [2018] in a fully-typed setting. That is, types participate crucially in the conversion relation: they are maintained during conversion, so that for instance comparison of terms at a Π-type systematically η-expands them before recursively calling conversion at the domain types. Defining a gradual variant of such a typed conversion might be quite interesting, but would require a significant amount of work.

On the contrary, a precise, formalized, account is still missing for η-equality for an untyped conversion as used in practice in the Coq proof assistant and in GCIC. The MetaCoq project, which aims at such a formalized account, leaves the treatment of η-equality to future work [Sozeau et al. 2020]. While we envision no specific issues to the adaptation to this approach to gradual typing once a clear and precise solution for CIC itself has been reached, solving the issue in a satisfactory way for CIC is obviously out of scope for this article. Thus, while it should in principle be possible to add η-equality to GCIC, either via typed or untyped conversion, we leave this for future work.

8.3 Propositional equality

In CIC, propositional equality `eq A x y`, corresponds to the Martin-Löf identity type [Martin-Löf 1975], with a single constructor `refl` for reflexivity, and the elimination principle known as `J`:

```
Inductive eq (A : □) (x : A) : A → □ := refl : eq A x x.

J : forall (A : □) (P : A → □) (x : A) (t : P x) (y : A) (e : eq A x y), P y
```

together with the conversion rule:

$$\texttt{J A P x t x (refl A x)} \equiv \texttt{t}$$

For the sake of exposing the problem, suppose that we can define this identity type in GCIC, while still satisfying canonicity, conservativity with respect to CIC and graduality. This means that for an equality `t = u` involving closed terms `t` and `u` of CIC, there should only be three possible canonical forms: `refl A t` whenever `t` and `u` are convertible terms (of type `A`), as well as `err` and ?.

Just under these assumptions, we can show that there exist two functions that are pointwise equal in CIC, and hence equal by extensionality, but are no longer equivalent in GCIC/CastCIC. Consider the two functions $\texttt{id}_{\mathbb{N}}$ and `add0` below:

$$\texttt{id}_{\mathbb{N}} := \lambda\ \texttt{n} : \mathbb{N} \Rightarrow \texttt{n} \qquad\qquad \texttt{add0} := \lambda\ \texttt{n} : \mathbb{N} \Rightarrow \texttt{n} + 0$$

In CIC, these functions are not convertible, but they are observationally equivalent. However, they would not be observationally equivalent in GCIC. To see why, consider the following term:

$$\texttt{test} := \lambda\ \texttt{f} \Rightarrow \texttt{J}\ (\mathbb{N} \to \mathbb{N})\ (\lambda\ \_ \Rightarrow \mathbb{B})\ \texttt{id}_{\mathbb{N}}\ \texttt{true}\ \texttt{f}\ (\texttt{refl}\ \texttt{id}_{\mathbb{N}} :: ?_{\square} :: \texttt{id}_{\mathbb{N}} = \texttt{f})$$

We have $\texttt{test id}_{\mathbb{N}} \leadsto^* \texttt{true}$ because, by $\mathcal{G}$, $\texttt{refl id}_{\mathbb{N}} :: ?_{\square} :: \texttt{id}_{\mathbb{N}} = \texttt{id}_{\mathbb{N}} \leadsto^* \texttt{refl id}_{\mathbb{N}}$. However, because `add0` is *not* convertible to $\texttt{id}_{\mathbb{N}}$, $\texttt{refl id}_{\mathbb{N}} :: \texttt{id}_{\mathbb{N}} = \texttt{add0}$ cannot possibly reduce to `refl`, and thus would need to reduce either to `err` or ?; and so does `test add0`.

This means that a model for such a gradual type theory would need to be intensional, conversely to the extensional models usually used to justify type theories. Studying such a model as well as exploring alternatives approaches to propositional equality in a gradual type theory are interesting venues for future work.

9 RELATED WORK

Bidirectional typing and unification. Our framework uses a bidirectional version of the type system of CIC. Although this presentation is folklore among type theory specialists [McBride 2019], the type system of CIC is rarely presented in this way on paper and has been studied in details only recently [Lennon-Bertrand 2021]. However, the bidirectional approach becomes necessary when dealing with unification and elaboration of implicit arguments. Bidirectional elaboration is a common feature of proof assistant implementations, for instance [Asperti et al. 2012], as it clearly delineates what information is available to the elaboration system in the different typing modes. In a context with missing information due to implicit arguments, those implementations face the undecidable higher order unification [Dowek 2001]. In this error-less context, the solution must be a form of under-approximation, using complex heuristics [Ziliani and Sozeau 2017]. Deciding consistency is very close to unification, as observed by Castagna et al. [2019], but our notion of consistency over-approximates unification, making sure that unifiable terms are always consistent, relying on errors to catch invalid over-approximations at runtime.

Dependent types with effects. As explained in this paper, introducing the unknown type of gradual typing also require, in a dependently-typed setting, to introduce unknown terms at any type. This means that a gradual dependent type theory naturally endorses an effectful mechanism which is similar to having exceptions. This connects GCIC to the literature on dependent types and effects. Several programming languages mix dependent types with effectful computation, either giving up on metatheoretical properties, such as Dependent Haskell [Eisenberg 2016], or by restricting the dependent fragment to pure expressions [Swamy et al. 2016; Xi and Pfenning 1998]. In the context of dependent type theories, Pédrot and Tabareau [2017, 2018] have leveraged the monadic approach to type theory, at the price of a weaker form of dependent large elimination for inductive types. The only way to recover full elimination is to accept a weaker form of logical consistency, as crystallized by the fire triangle between observable effects, substitution and logical consistency [Pédrot and Tabareau 2020].

Ordered and directed type theories. The monotone model of CastCIC interpret types as posets in order to give meaning to the notion of precision. Interpretations of dependent type theories in ordered structures goes back to various works on domain theoretic and realizability interpretations of (partial) Martin-Löf Type Theory [Ehrhard 1988; Palmgren and Stoltenberg-Hansen 1990]. More recently, Licata and Harper [2011] and North [2019] extend type theory with directed structures corresponding to a categorical interpretation of types, a higher version of the monotone model we consider.

Hybrid typing. [Ou et al. 2004] present a programming language with separate dependently- and simply-typed fragments, using arbitrary runtime checks at the boundary. Knowles and Flanagan [2010] support runtime checking of refinements. In a similar manner, [Tanter and Tabareau 2015] introduce casts for subset types with decidable properties in Coq. They use an axiom to denote failure, which breaks weak canonicity. Dependent interoperability [Dagand et al. 2018; Osera et al. 2012] supports the combination of dependent and non-dependent typing through deep conversions. All these approaches are more intended as programming languages than as type theories, and none support the notion of (im)precision that is at the heart of gradual typing.

Dependent contracts. [Greenberg et al. 2010] relates hybrid typing to dependent contracts, which are dynamically-checked assertions that can relate the result of a function application to its argument [Findler and Felleisen 2002]. The semantics of dependent contracts are subtle because contracts include arbitrary code, and in particular one must be careful not to violate the precondition on the argument in the definition of the postcondition contract [Blume and McAllester 2006]. Also, blame assignment when the result and/or argument are themselves higher-order is subtle. Different variants of dependent contracts have been studied in the literature, which differ in terms of the violations they report and the way they assign blame [Dimoulas et al. 2011; Greenberg et al. 2010]. An in-depth exploration of blame assignment for gradual dependent type theories such as GCIC is an important perspective for future work.

Gradual typing. The blame calculus of Wadler and Findler [2009] considers subset types on base types, where the refinement is an arbitrary term, as in hybrid type checking [Knowles and Flanagan 2010]. It however lacks the dependent function types found in other works. Lehmann and Tanter [2017] exploit the Abstracting Gradual Typing (AGT) methodology [Garcia et al. 2016] to design a language with imprecise formulas and implication. They support dependent function types, but gradual refinements are only on base types refined with decidable logical predicates. Eremondi et al. [2019] also use AGT to develop approximate normalization and GDTL. While being a clear initial inspiration for this work, the technique of approximate normalization cannot yield a computationally-relevant gradual type theory (nor was its intent, as clearly stated by the authors). We hope that the results in our work can prove useful in the design and formalization of such gradual dependently-typed programming languages. Eremondi et al. [2019] study the dynamic gradual guarantee, but not its reformulation as graduality [New and Ahmed 2018], which as we explain is strictly stronger in the full dependent setting. Finally, while AGT provided valuable intuitions for this work, graduality as embedding-projection pairs was the key technical driver in the design of CastCIC.

10 CONCLUSION

We have unveiled a fundamental tension in the design of gradual dependent type theories between conservativity with respect to a dependent type theory such as CIC, normalization, and graduality. We explore several resolutions of this Fire Triangle of Graduality, yielding three different gradual counterparts of CIC, each compromising with one edge of the Triangle. We develop the metatheory of all three variants of GCIC thanks to a common formalization, parametrized by two knobs controlling universe constraints on dependent product types in typing and reduction.

This work opens a number of perspectives for future work, in addition to addressing the limitations discussed in §8. The delicate interplay between universe levels and computational behavior of casts begs for a more flexible approach to the normalizing $\mathrm{GCIC}^{\mathcal{N}}$, for instance using gradual universes. The approach based on multiple universe hierarchies to support logically consistent reasoning about exceptional programs [Pédrot et al. 2019] could be adapted to our setting in order to provide a seamless integration inside a single theory of gradual features together with standard CIC without compromising normalization. This could also open the door to supporting consistent reasoning about gradual programs in the context of GCIC. On the more practical side, there is still a lot of challenges ahead in order to implement a gradual incarnation of GCIC in Coq or Agda, possibly parametrized in order to support the three variants presented in this work.

REFERENCES

Andreas Abel, Joakim Öhman, and Andrea Vezzosi. 2018. Decidability of Conversion for Type Theory in Type Theory. *Proceedings of the ACM on Programming Languages* 2, POPL, Article 23 (Jan. 2018), 29 pages. https://doi.org/10.1145/3158111

Samson Abramsky and Achim Jung. 1995. *Domain Theory*. Oxford University Press, Inc., USA, 1–168.

Thorsten Altenkirch. 1999. Extensional Equality in Intensional Type Theory. In *Proceedings of the 14th Symposium on Logic in Computer Science (LICS 2002)*. IEEE Computer Society Press, Trento, Italy, 412–420. https://doi.org/10.1109/LICS.1999.782636

Thorsten Altenkirch, Simon Boulier, Ambrus Kaposi, Christian Sattler, and Filippo Sestini. 2021. Constructing a universe for the setoid model *(Lecture Notes in Computer Science, Vol. 12650)*, Stefan Kiefer and Christine Tasson (Eds.). Springer, 1–21. https://doi.org/10.1007/978-3-030-71995-1_1

Thorsten Altenkirch, Simon Boulier, Ambrus Kaposi, and Nicolas Tabareau. 2019. Setoid Type Theory - A Syntactic Translation *(Lecture Notes in Computer Science, Vol. 11825)*, Graham Hutton (Ed.). Springer, 155–196. https://doi.org/10.1007/978-3-030-33636-3_7

Thorsten Altenkirch, Paolo Capriotti, Gabe Dijkstra, Nicolai Kraus, and Fredrik Nordvall Forsberg. 2018. Quotient Inductive-Inductive Types *(Lecture Notes in Computer Science, Vol. 10803)*, Christel Baier and Ugo Dal Lago (Eds.). Springer, 293–310. https://doi.org/10.1007/978-3-319-89366-2_16

Andrea Asperti, Wilmer Ricciotti, Claudio Sacerdoti Coen, and Enrico Tassi. 2012. A Bi-Directional Refinement Algorithm for the Calculus of (Co)Inductive Constructions. Volume 8, Issue 1 (2012). https://doi.org/10.2168/LMCS-8(1:18)2012

Robert Atkey, Neil Ghani, and Patricia Johann. 2014. A relationally parametric model of dependent type theory. In *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA,* *January 20-21, 2014*, Suresh Jagannathan and Peter Sewell (Eds.). ACM, 503–516. https://doi.org/10.1145/2535838.2535852

Felipe Bañados Schwerter, Alison M. Clark, Khurram A. Jafery, and Ronald Garcia. 2020. Abstracting Gradual Typing Moving Forward: Precise and Space-Efficient. arXiv:2010.14094 [cs.PL]

Felipe Bañados Schwerter, Ronald Garcia, and Éric Tanter. 2016. Gradual Type-and-Effect Systems. *Journal of Functional Programming* 26 (Sept. 2016), 19:1–19:69.

Henk Barendregt. 1991. Introduction to Generalized Type Systems. *Journal of Functional Programming* 1, 2 (April 1991), 125–154.

Henk P. Barendregt. 1984. *The Lambda Calculus: Its Syntax and Semantics.* North-Holland.

Jean-Philippe Bernardy, Patrik Jansson, and Ross Paterson. 2012. Proofs for free: Parametricity for dependent types. *Journal of Functional Programming* 22, 2 (March 2012), 107–152.

Martin E. Bidlingmaier, Florian Faissole, and Bas Spitters. 2019. Synthetic topology in Homotopy Type Theory for probabilistic programming. (2019). arXiv:1912.07339 http://arxiv.org/abs/1912.07339

Gavin Bierman, Erik Meijer, and Mads Torgersen. 2010. Adding Dynamic Types to C#. In *Proceedings of the 24th European Conference on Object-oriented Programming (ECOOP 2010) (Lecture Notes in Computer Science, 6183)*, Theo D'Hondt (Ed.). Springer-Verlag, Maribor, Slovenia, 76–100.

M. Blume and D. McAllester. 2006. Sound and complete models of contracts. *Journal of Functional Programming* 16, 4-5 (2006), 375–414.

Rastislav Bodík and Rupak Majumdar (Eds.). 2016. *Proceedings of the 43rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2016).* ACM Press, St Petersburg, FL, USA.

Simon Boulier, Pierre-Marie Pédrot, and Nicolas Tabareau. 2017. The next 700 syntactical models of type theory. In *Proceedings of the 6th ACM SIGPLAN Conference on Certified Programs and Proofs, CPP 2017, Paris, France, January 16-17, 2017.* 182–194. https://doi.org/10.1145/3018610.3018620

Edwin Brady. 2013. Idris, a General Purpose Dependently Typed Programming Language: Design and Implementation. *Journal of Functional Programming* 23, 5 (Sept. 2013), 552–593.

Edwin Brady, Conor McBride, and James McKinna. 2004. Inductive Families Need Not Store Their Indices. In *Types for Proofs and Programs (TYPES 2004) (Lecture Notes in Computer Science, Vol. 3085).* Springer-Verlag, 115–129.

Giuseppe Castagna (Ed.). 2009. *Proceedings of the 18th European Symposium on Programming Languages and Systems (ESOP 2009).* Lecture Notes in Computer Science, Vol. 5502. Springer-Verlag, York, UK.

Giuseppe Castagna, Victor Lanvin, Tommaso Petrucciani, and Jeremy G. Siek. 2019. Gradual typing: a new perspective. See[POPL 2019 2019], 16:1–16:32.

James Chapman, Tarmo Uustalu, and Niccolò Veltri. 2019. Quotienting the delay monad by weak bisimilarity. *Math. Struct. Comput. Sci.* 29, 1 (2019), 67–92. https://doi.org/10.1017/S0960129517000184

Matteo Cimini and Jeremy Siek. 2016. The gradualizer: a methodology and algorithm for generating gradual type systems, See [Bodík and Majumdar 2016], 443–455.

Thierry Coquand and Gérard Huet. 1988. The Calculus of Constructions. *Information and Computation* 76, 2-3 (Feb. 1988), 95–120.

Pierre-Évariste Dagand, Nicolas Tabareau, and Éric Tanter. 2018. Foundations of Dependent Interoperability. *Journal of Functional Programming* 28 (2018), 9:1–9:44.

Christos Dimoulas, Robert Bruce Findler, Cormac Flanagan, and Matthias Felleisen. 2011. Correct blame for contracts: no more scapegoating. In *Proceedings of the 38th annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2011).* ACM Press, Austin, Texas, USA, 215–226.

Gilles Dowek. 2001. Chapter 16 - Higher-Order Unification and Matching. In *Handbook of Automated Reasoning*, Alan Robinson and Andrei Voronkov (Eds.). North-Holland, 1009–1062. https://doi.org/10.1016/B978-044450813-3/50018-7

Peter Dybjer and Anton Setzer. 2003. Induction-recursion and initial algebras. *Ann. Pure Appl. Log.* 124, 1-3 (2003), 1–47. https://doi.org/10.1016/S0168-0072(02)00096-9

Thomas Ehrhard. 1988. A Categorical Semantics of Constructions. In *Proceedings of the Third Annual Symposium on Logic in Computer Science (LICS '88), Edinburgh, Scotland, UK, July 5-8, 1988.* IEEE Computer Society, 264–273. https://doi.org/10.1109/LICS.1988.5125

Richard A. Eisenberg. 2016. Dependent Types in Haskell: Theory and Practice. arXiv:1610.07978 [cs.PL]

Joseph Eremondi, Éric Tanter, and Ronald Garcia. 2019. Approximate Normalization for Gradual Dependent Types. See[ICFP 2019 2019], 88:1–88:30.

Luminous Fennell and Peter Thiemann. 2013. Gradual Security Typing with References. In *Proceedings of the 26th Computer Security Foundations Symposium (CSF).* 224–239.

Robert Bruce Findler and Matthias Felleisen. 2002. Contracts for Higher-Order Functions. In *Proceedings of the 7th ACM SIGPLAN Conference on Functional Programming (ICFP 2002).* ACM Press, Pittsburgh, PA, USA, 48–59.

Fredrik Nordvall Forsberg. 2013. *Inductive-inductive definitions.* Ph.D. Dissertation. Swansea University, UK. http://ethos.bl.uk/OrderDetails.do?uin=uk.bl.ethos.752308

Ronald Garcia, Alison M. Clark, and Éric Tanter. 2016. Abstracting Gradual Typing, See [Bodík and Majumdar 2016], 429–442. See erratum: https://www.cs.ubc.ca/ rxg/agt-erratum.pdf.

Ronald Garcia and Éric Tanter. 2020. Gradual Typing as if Types Mattered. In *Informal Proceedings of the ACM SIGPLAN Workshop on Gradual Typing (WGT20)*.

Neil Ghani, Lorenzo Malatesta, and Fredrik Nordvall Forsberg. 2015. Positive Inductive-Recursive Definitions. *Log. Methods Comput. Sci.* 11, 1 (2015). https://doi.org/10.2168/LMCS-11(1:13)2015

Gaëtan Gilbert, Jesper Cockx, Matthieu Sozeau, and Nicolas Tabareau. 2019. Definitional proof-irrelevance without K. See[POPL 2019 2019], 3:1–3:28. https://doi.org/10.1145/3290316

Eduardo Giménez. 1998. Structural Recursive Definitions in Type Theory. In *ICALP*. 397–408.

Healfdene Goguen. 2005. A Syntactic Approach to Eta Equality in Type Theory. *SIGPLAN Not.* 40, 1 (Jan. 2005), 75–84. https://doi.org/10.1145/1047659.1040312

Michael Greenberg, Benjamin C. Pierce, and Stephanie Weirich. 2010. Contracts Made Manifest, See [POPL 2010 2010], 353–364.

Robert Harper and Robert Pollack. 1991. Type checking with universes. *Theoretical Computer Science* 89, 1 (1991). https://doi.org/10.1016/0304-3975(90)90108-T

David Herman, Aaron Tomb, and Cormac Flanagan. 2010. Space-efficient gradual typing. *Higher-Order and Sympolic Computation* 23, 2 (June 2010), 167–189.

Martin Hofmann. 1995. Conservativity of Equality Reflection over Intensional Type Theory. In *Types for Proofs and Programs, International Workshop TYPES'95, Torino, Italy, June 5-8, 1995, Selected Papers*. 153–164. https://doi.org/10.1007/3-540-61780-9_68

ICFP 2019 2019.

Yuu Igarashi, Taro Sekiyama, and Atsushi Igarashi. 2017. On Polymorphic Gradual Typing. *Proceedings of the ACM on Programming Languages* 1, ICFP (Sept. 2017), 40:1–40:29.

Ambrus Kaposi, András Kovács, and Thorsten Altenkirch. 2019. Constructing quotient inductive-inductive types. See[POPL 2019 2019], 2:1–2:24. https://doi.org/10.1145/3290315

Kenneth Knowles and Cormac Flanagan. 2010. Hybrid type checking. *ACM Transactions on Programming Languages and Systems* 32, 2 (Jan. 2010), Article n.6.

Nico Lehmann and Éric Tanter. 2017. Gradual Refinement Types. In *Proceedings of the 44th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2017)*. ACM Press, Paris, France, 775–788.

Meven Lennon-Bertrand. 2021. Complete Bidirectional Typing for the Calculus of Inductive Constructions. In *12th International Conference on Interactive Theorem Proving (ITP 2021) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 193)*, Liron Cohen and Cezary Kaliszyk (Eds.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik. https://doi.org/10.4230/LIPIcs.ITP.2021.24

Meven Lennon-Bertrand, Kenji Maillard, Éric Tanter, and Nicolas Tabareau. 2020. https://github.com/pleiad/GradualizingCIC

Paul Blain Levy. 2004. *Call-By-Push-Value: A Functional/Imperative Synthesis*. Semantics Structures in Computation, Vol. 2. Springer.

Daniel R. Licata and Robert Harper. 2011. 2-Dimensional Directed Type Theory. In *Twenty-seventh Conference on the Mathematical Foundations of Programming Semantics, MFPS 2011, Pittsburgh, PA, USA, May 25-28, 2011 (Electronic Notes in Theoretical Computer Science, Vol. 276)*, Michael W. Mislove and Joël Ouaknine (Eds.). Elsevier, 263–289. https://doi.org/10.1016/j.entcs.2011.09.026

Saunders MacLane and Ieke Moerdijk. 1992. *Sheaves in Geometry and Logic: A First Introduction to Topos Theory*. Springer-Verlag.

Assia Mahboubi and Enrico Tassi. 2008. *Mathematical Components*.

Per Martin-Löf. 1975. An intuitionistic theory of types: predicative part. In *Logic Colloquium '73, Proceedings of the Logic Colloquium*, H.E. Rose and J.C. Shepherdson (Eds.). Studies in Logic and the Foundations of Mathematics, Vol. 80. North-Holland, 73–118.

Per Martin-Löf. 1984. *Intuitionistic type theory*. Studies in proof theory, Vol. 1. Bibliopolis.

Per Martin-Löf. 1996. On the Meanings of the Logical Constants and the Justifications of the Logical Laws. *Nordic Journal of Philosophical Logic* 1, 1 (1996), 11–60.

Conor McBride. 1999. *Dependently Typed Functional Programs and their Proofs*. Ph.D. Dissertation. University of Edinburgh.

Conor McBride. 2010. Outrageous but meaningful coincidences: dependent type-safe syntax and evaluation, Bruno C. d. S. Oliveira and Marcin Zalewski (Eds.). ACM, 1–12. https://doi.org/10.1145/1863495.1863497

Conor McBride. 2018. Basics of Bidirectionalism. https://pigworker.wordpress.com/2018/08/06/basics-of-bidirectionalism/

Conor McBride. 2019. Check the Box!. In *25th International Conference on Types for Proofs and Programs*. Invited presentation.

Max S. New and Amal Ahmed. 2018. Graduality from Embedding-Projection Pairs. , 73:1–73:30 pages.

Max S. New, Dustin Jamner, and Amal Ahmed. 2020. Graduality and Parametricity: Together Again for the First Time. See[POPL 2020 2020], 46:1–46:32.

Max S. New and Daniel R. Licata. 2020. Call-by-name Gradual Type Theory. *Logical Methods in Computer Science* Volume 16, Issue 1 (Jan. 2020). https://doi.org/10.23638/LMCS-16(1:7)2020

Max S. New, Daniel R. Licata, and Amal Ahmed. 2019. Gradual Type Theory. See[POPL 2019 2019], 15:1–15:31.

Phuc C. Nguyen, Thomas Gilray, and Sam Tobin-Hochstadt. 2019. Size-change termination as a contract: dynamically and statically enforcing termination for higher-order programs. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2019)*. ACM Press, Phoenix, AZ, USA, 845–859.

Ulf Norell. 2009. Dependently Typed Programming in Agda. In *Advanced Functional Programming (AFP 2008) (Lecture Notes in Computer Science, Vol. 5832)*. Springer-Verlag, 230–266.

Paige Randall North. 2019. Towards a Directed Homotopy Type Theory. In *Proceedings of the Thirty-Fifth Conference on the Mathematical Foundations of Programming Semantics, MFPS 2019, London, UK, June 4-7, 2019 (Electronic Notes in Theoretical Computer Science, Vol. 347)*, Barbara König (Ed.). Elsevier, 223–239. https://doi.org/10.1016/j.entcs.2019.09.012

Peter-Michael Osera, Vilhelm Sjöberg, and Steve Zdancewic. 2012. Dependent Interoperability. In *Proceedings of the 6th workshop on Programming Languages Meets Program Verification (PLPV 2012)*. ACM Press, 3–14.

Xinming Ou, Gang Tan, Yitzhak Mandelbaum, and David Walker. 2004. Dynamic Typing with Dependent Types. In *Proceedings of the IFIP International Conference on Theoretical Computer Science*. 437–450.

Erik Palmgren. 1998. On universes in type theory. In *Twenty Five Years of Constructive Type Theory.*, G. Sambin and J. Smith (Eds.). Oxford University Press, 191–204.

Erik Palmgren and Viggo Stoltenberg-Hansen. 1990. Domain Interpretations of Martin-Löf's Partial Type Theory. *Ann. Pure Appl. Log.* 48, 2 (1990), 135–196. https://doi.org/10.1016/0168-0072(90)90044-3

Christine Paulin-Mohring. 2015. Introduction to the Calculus of Inductive Constructions. In *All About Proofs, Proofs for All*, Bruno Woltzenlogel Paleo and David Delahaye (Eds.). College Publications.

Pierre-Marie Pédrot and Nicolas Tabareau. 2017. An effectful way to eliminate addiction to dependence. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*. IEEE Computer Society, 1–12. https://doi.org/10.1109/LICS.2017.8005113

Pierre-Marie Pédrot and Nicolas Tabareau. 2018. Failure is Not an Option - An Exceptional Type Theory. In *Proceedings of the 27th European Symposium on Programming Languages and Systems (ESOP 2018) (Lecture Notes in Computer Science, Vol. 10801)*, Amal Ahmed (Ed.). Springer-Verlag, Thessaloniki, Greece, 245–271.

Pierre-Marie Pédrot and Nicolas Tabareau. 2020. The fire triangle: how to mix substitution, dependent elimination, and effects. See[POPL 2020 2020], 58:1–58:28.

Pierre-Marie Pédrot, Nicolas Tabareau, Hans Fehrmann, and Éric Tanter. 2019. A Reasonably Exceptional Type Theory. See[ICFP 2019 2019], 108:1–108:29.

POPL 2010 2010. *Proceedings of the 37th annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2010)*. ACM Press, Madrid, Spain.

POPL 2019 2019. . Vol. 3. ACM Press.

POPL 2020 2020. . Vol. 4. ACM Press.

Dana Scott. 1976. Data Types as Lattices. *SIAM J. Comput.* 5, 3 (1976), 522–587.

Anton Setzer. 2000. Extending Martin-Löf Type Theory by one Mahlo-universe. *Arch. Math. Log.* 39, 3 (2000), 155–181. https://doi.org/10.1007/s001530050140

Michael Shulman. 2011. An interval type implies function extensionality. Blog article. https://homotopytypetheory.org/2011/04/04/an-interval-type-implies-function-extensionality/

Jeremy Siek, Ronald Garcia, and Walid Taha. 2009. Exploring the Design Space of Higher-Order Casts, See [Castagna 2009], 17–31.

Jeremy Siek and Walid Taha. 2006. Gradual Typing for Functional Languages. In *Proceedings of the Scheme and Functional Programming Workshop*. 81–92.

Jeremy Siek and Walid Taha. 2007. Gradual Typing for Objects. In *Proceedings of the 21st European Conference on Object-oriented Programming (ECOOP 2007) (Lecture Notes in Computer Science, 4609)*, Erik Ernst (Ed.). Springer-Verlag, Berlin, Germany, 2–27.

Jeremy Siek and Philip Wadler. 2010. Threesomes, with and without blame, See [POPL 2010 2010], 365–376.

Jeremy G. Siek, Michael M. Vitousek, Matteo Cimini, and John Tang Boyland. 2015. Refined Criteria for Gradual Typing. In *1st Summit on Advances in Programming Languages (SNAPL 2015) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 32)*. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Asilomar, California, USA, 274–293.

Michael B. Smyth and Gordon D. Plotkin. 1977. The Category-Theoretic Solution of Recursive Domain Equations (Extended Abstract). IEEE Computer Society, 13–17. https://doi.org/10.1109/SFCS.1977.30

Matthieu Sozeau, Simon Boulier, Yannick Forster, Nicolas Tabareau, and Théo Winterhalter. 2020. Coq Coq correct! verification of type checking and erasure for Coq, in Coq. *Proc. ACM Program. Lang.* 4, POPL (2020), 8:1–8:28. https://doi.org/10.1145/3371076

Nikhil Swamy, Catalin Hritcu, Chantal Keller, Aseem Rastogi, Antoine Delignat-Lavaud, Simon Forest, Karthikeyan Bhargavan, Cédric Fournet, Pierre-Yves Strub, Markulf Kohlweiss, Jean Karim Zinzindohoue, and Santiago Zanella Béguelin. 2016. Dependent types and multi-effects in F*, See [Bodík and Majumdar 2016], 256–270.

M. Takahashi. 1995. Parallel Reductions in λ-Calculus. *Information and Computation* 118, 1 (1995), 120 – 127. https://doi.org/10.1006/inco.1995.1057

Éric Tanter and Nicolas Tabareau. 2015. Gradual Certified Programming in Coq. In *Proceedings of the 11th ACM Dynamic Languages Symposium (DLS 2015)*. ACM Press, Pittsburgh, PA, USA, 26–40.

The Coq Development Team. 2020. *The Coq proof assistant reference manual*. https://coq.inria.fr/refman/ Version 8.12.

Peter Thiemann and Luminous Fennell. 2014. Gradual Typing for Annotated Type Systems. In *Proceedings of the 23rd European Symposium on Programming Languages and Systems (ESOP 2014) (Lecture Notes in Computer Science, Vol. 8410)*, Zhong Shao (Ed.). Springer-Verlag, Grenoble, France, 47–66.

Sam Tobin-Hochstadt and Matthias Felleisen. 2008. The Design and Implementation of Typed Scheme. In *Proceedings of the 35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2008)*. ACM Press, San Francisco, CA, USA, 395–406.

Matías Toro, Ronald Garcia, and Éric Tanter. 2018. Type-Driven Gradual Security with References. *ACM Transactions on Programming Languages and Systems* 40, 4 (Nov. 2018), 16:1–16:55.

Matías Toro and Éric Tanter. 2020. Abstracting Gradual References. *Science of Computer Programming* 197 (Oct. 2020), 1–65.

Philip Wadler and Robert Bruce Findler. 2009. Well-Typed Programs Can't Be Blamed, See [Castagna 2009], 1–16.

Mitchell Wand. 1979. Fixed-Point Constructions in Order-Enriched Categories. *Theor. Comput. Sci.* 8 (1979), 13–30. https://doi.org/10.1016/0304-3975(79)90053-7

Théo Winterhalter, Matthieu Sozeau, and Nicolas Tabareau. 2019. Eliminating reflection from type theory. In *Proceedings of the 8th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2019, Cascais, Portugal, January 14-15, 2019*, Assia Mahboubi and Magnus O. Myreen (Eds.). ACM, 91–103. https://doi.org/10.1145/3293880.3294095

Andrew K. Wright and Matthias Felleisen. 1994. A syntactic approach to type soundness. *Journal of Information and Computation* 115, 1 (Nov. 1994), 38–94.

Hongwei Xi and Frank Pfenning. 1998. Eliminating array bound checking through dependent types. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '98)*. ACM Press, 249–257.

Beta Ziliani and Matthieu Sozeau. 2017. A comprehensible guide to a new unifier for CIC including universe polymorphism and overloading. 27 (2017). https://doi.org/10.1017/S0956796817000028

A INDEX OF NOTATIONS

Description	Symbol	Ref	Remark
		Section 4	
Universe	$\square_i$	Page 19	At level i
Inductive type	$I@\{i\}(\mathbf{a})$	Page 19	At level i with parameters $\mathbf{a}$
Inductive constructor	$c_k^I@\{i\}(\mathbf{a}, \mathbf{b})$	Page 19	k-th constructor of I at level i with parameters $\mathbf{a}$ and arguments $\mathbf{b}$
Inductive destructor	$\mathtt{ind}_I(s, z.P, f.\mathbf{y}.\mathbf{b})$	Page 19	corresponds to fix + match in Coq
Substitution	$t[u/x]$	Page 19	extended to parallel substitution
Types of parameters	$\mathbf{Params}(I, i)$	Page 19	of inductive I at level i
Types of arguments	$\mathbf{Args}(I, i, c_k)$	Page 19	of constructor k of inductive I at level i
Substitution in parameters	$\mathbf{Params}(I, i)[\mathbf{a}]$	Page 19	
Substitution in arguments	$\mathbf{Args}(I, i, c_k)[\mathbf{a}, \mathbf{b}]$	Page 19	
Context checking	$\vdash \Gamma$	Fig. 1	
Type inference	$\Gamma \vdash t \triangleright T$	Fig. 1	
Type checking	$\Gamma \vdash t \triangleleft T$	Fig. 1	
Constrained inference	$\Gamma \vdash t \blacktriangleright_{\bullet}$	Fig. 1	$\bullet$ is either Π, I or $\square$
One-step reduction	$\rightsquigarrow$	Fig. 1	*full*, *i.e.*, with all congruences
Reduction	$\rightsquigarrow^*$	Fig. 1	reflexive, transitive closure of $\rightsquigarrow$
Conversion	$\equiv$	Fig. 1	
		Section 5	
Unknown type	$?_T$	Page 22	in CastCIC
Error	$\mathtt{err}_T$	Page 22	
Cast	$\langle T' \Leftarrow T \rangle\, t$	Page 22	
Level of product type	$s_\Pi(i, j)$	Fig. 2	
Level of product germ	$c_\Pi(i)$	Fig. 2	
Type heads	Head	Fig. 4	
Head of a type	head (T)	Fig. 4	
Germ	$\mathrm{germ}_i\ h$	Fig. 4	Least precise type with head h at level i
Parallel reduction	$\Rrightarrow$	Lemma 7	
Canonical term	canonical t	Fig. 7	inductive caracterization
Neutral term	neutral t	Fig. 7	inductive caracterization
α-consistency	$t \sim_\alpha t'$	Fig. 8	
Consistent conversion	$t \sim t'$	Definition 5	Also called *consistency*
Unknown type	$?@\{i\}$	Page 28	in GCIC, at level i
Elaboration (inference)	$\Gamma \vdash t \rightsquigarrow t' \triangleright T$	Fig. 9	
Elaboration (checking)	$\Gamma \vdash t \triangleleft T \rightsquigarrow t'$	Fig. 9	
Elaboration (constrained)	$\Gamma \vdash t \rightsquigarrow t' \blacktriangleright_{\bullet} T$	Fig. 9	
Structural precision	$\mathbb{\Gamma} \vdash t \sqsubseteq_\alpha t'$	Fig. 10	extended to contexts pointwise
Definitional precision	$\mathbb{\Gamma} \vdash t \sqsubseteq_{\rightsquigarrow} t'$	Fig. 10	extended to contexts pointwise
Typing in CIC/CastCIC	$\vdash_{\mathrm{CIC}}$ / $\vdash_{\mathrm{cast}}$	§5.5	to differentiate between systems
Equiprecision	$\mathbb{\Gamma} \vdash t \sqsupseteq\sqsubseteq_\alpha t'$	Definition 7	
Erasure	$\varepsilon(t)$	Definition 8	
Syntactic precision	$t \sqsubseteq_\alpha^{\mathrm{G}} t'$	Fig. 11	

Description	Symbol	Ref	Remark
	Section 6		
CIC + Induction-Recursion	$\mathsf{CIC}^{\mathsf{IR}}$	§6.1	Target for the discrete model
Judgements for $\mathsf{CIC}^{\mathsf{IR}}$	$\vdash_{\mathrm{IR}}$		
$\mathsf{CIC}^{\mathsf{IR}}$ + quotients	$\mathsf{CIC}^{\mathsf{IR}}_{\mathsf{QIT}}$		Target for the monotone models
Universe of codes	$\mathbb{\square}$	Fig. 12	
Bipointed poset on inductive I	$\ddot{I}$	§6.1	
Top element in $\ddot{I}$	$\top_{\ddot{I}}$	§6.1	
Bottom element in $\ddot{I}$	$\bot_{\ddot{I}}$	§6.1	
Bipointed poset on $\mathbb{N}$	$\ddot{\mathbb{N}}$	§6.1	
Bipointed poset on Σ	$\ddot{\Sigma}$	§6.1	
Code for nat	$\widehat{\mathbb{N}}$	§6.1	
Code for dependent product	$\widehat{\Pi}$	§6.1	
Code for universes	$\widehat{\mathbb{\square}}_i$	§6.1	
Code for unknown types	$\widehat{?}_i$	§6.1	
Code for error type	$\widehat{\mathtt{err}}$	§6.1	
Decoding function to types	El	Figs. 12 and 17	$\mathrm{El} : \mathbb{\square} \to \square^{\le}$
Type heads	Head_i	Fig. 4	
Head of a type	$\mathrm{head}\,(T)$	Fig. 4	
Germ as a code	$\widehat{\mathrm{germ}}_i\ h$	§6.1	
Germ	$\mathrm{germ}_i\ h$	Fig. 4	Least precise type with head $h \in \mathrm{Head}_i$ at level i
Cast in discrete model	$\mathtt{cast}$	Fig. 14	
Discrete translation of types	$[\![\cdot]\!]$	Fig. 15	
Discrete translation of terms	$[\cdot]$	Fig. 15	
Order on type A	$\sqsubseteq^A$	§6.2	
Type of posets	$\square^{\le}$	§6.2	
Monotone dependent product	$\Pi^{\mathrm{mon}}\ A\,B$	§6.2	
Ep-pairs	$A \lhd B$	Definition 9	
Upcast	$\uparrow_d$	Definition 9	Embedding part of an ep-pair d
Downcast	$\downarrow_d$	Definition 9	Projection part of an ep-pair d
Monotone unknown type	$\ddot{?}_i$	§6.3	
Quotiented pairs in $\ddot{?}_i$	$[h; x]$	§6.3	
Top element in $\ddot{?}_i$	$\top_{\ddot{?}_i}$	§6.3	
Bottom element in $\ddot{?}_i$	$\bot_{\ddot{?}_i}$	§6.3	
Decoding function to ep-pairs	El_ε	Fig. 17	$\mathrm{El}_\varepsilon(A {\sqsubseteq} B) : \mathrm{El}\,A \lhd \mathrm{El}\,B$
Precision on terms	${}_A{\sqsubseteq}_B$	Fig. 17	
Monotone translation of types	$\{\!\vert \cdot \vert\!\}$	Fig. 18	
Monotone translation of terms	$\{\cdot\}$	Fig. 18	
Propositional precision	$\Gamma \vdash_{\mathsf{cast}} t\ {}_T{\sqsubseteq}_U\ u$	Eq. (2)	
ω-continuous maps	$A \to^\omega B$	§6.7	A, B ω-cpos
ω-continuous ep-pair	$A \lhd^\omega B$	§6.7	

B COMPLEMENTS ON ELABORATION AND CastCIC

This section gives an extended account of §5. The structure is the same, and we refer to the main section when things are already spelled out there.

B.1 CastCIC

We state and prove a handful of standard, technical properties of CastCIC, that are useful in the next sections. They should not be very surprising, the main specific point here is their formulation in the bidirectional setting.

Property 1 (Weakening). *If* $\Gamma \vdash t \triangleright T$ *then* $\Gamma, \Delta \vdash t \triangleright T$*, and similarly for the other typing judgments.*

Proof. We show by (mutual) induction on the typing derivation the more general statement that if $\Gamma, \Gamma' \vdash t \triangleright T$ then $\Gamma, \Delta, \Gamma' \vdash t \triangleright T$. It is true for the base cases (including the variable), and we can check that all rules preserve it. □

Property 2 (Substitution). *If* $\Gamma, x : A, \Delta \vdash t \triangleright T$ *and* $\Gamma \vdash u \triangleleft A$ *then* $\Gamma, \Delta[u/x] \vdash t[u/x] \triangleright S$ *with* $S \equiv T[u/x]$.

Proof. Again, the proof is by mutual induction on the derivation. In the checking judgment, we use the transitivity of conversion to conclude. In the constrained inference, we need injectivity of type constructors, which is a consequence of confluence. □

Property 3 (Validity). *If* $\Gamma \vdash t \triangleright T$ *and* $\vdash \Gamma$*, then* $\Gamma \vdash T \blacktriangleright_{\square} \square_i$ *for some* i.

Proof. Once again, this is a routine induction on the inference derivation, using subject reduction to handle the reductions in the constrained inference rules, to ensure that the reduced type is still well-formed. The hypothesis of context well-formedness is needed for the base case of a variable, to get that the type obtained from the context is indeed well-typed. □

B.2 Precision and Reduction

Structural lemmas. Let us start our lemmas by counterparts to the weakening and substitution lemmas for precision.

Lemma 35 (Weakening of precision). *If* $\mathbb{\Gamma} \vdash t \sqsubseteq_\alpha t'$*, then* $\mathbb{\Gamma}, \mathbb{\Delta} \vdash t \sqsubseteq_\alpha t'$ *for any* $\mathbb{\Delta}$.

Proof. This is by induction on the precision derivation, using weakening of CastCIC to handle the uses of typing. □

Lemma 36 (Substitution and precision). *If* $\mathbb{\Gamma}, x : S \mid S', \mathbb{\Delta} \vdash t \sqsubseteq_\alpha t'$, $\mathbb{\Gamma} \vdash u \sqsubseteq_\alpha u'$, $\mathbb{\Gamma}_1 \vdash u \triangleleft S$ *and* $\mathbb{\Gamma}_2 \vdash u' \triangleleft S'$ *then* $\mathbb{\Gamma}, \mathbb{\Delta}[u \mid u'/x] \vdash t[u/x] \sqsubseteq_\alpha t'[u'/x]$.

Proof. The substitution property follows from weakening, again by induction on the precision derivation. Weakening is used in the variable case where x is replaced by u and u', and the substitution property of CastCIC appears to handle the uses of typing. □

Catch-up lemmas. With these structural lemmas at hand, let us turn to the proofs of the catch-up lemmas.

Proof of Lemma 15. We want to prove the following: under the hypothesis that $\mathbb{\Gamma}_1 \sqsubseteq_\alpha \mathbb{\Gamma}_2$, if $\mathbb{\Gamma} \vdash \square_i \sqsubseteq_{\rightsquigarrow} T'$ and $\mathbb{\Gamma}_2 \vdash T' \blacktriangleright_{\square} \square_j$, then either $T' \rightsquigarrow^* ?_{\square_j}$ with $i + 1 \leq j$, or $T' \rightsquigarrow^* \square_i$.

The proof is by induction on the precision derivation, mutually with the same property where $\sqsubseteq_{\rightsquigarrow}$ is replaced by $\sqsubseteq_\alpha$.

Let us start with the proof for $\sqsubseteq_\alpha$. Using the precision derivation, we can decompose T' into $\langle S_n \Leftarrow U_{n-1} \rangle \ldots \langle S_2 \Leftarrow U_1 \rangle\, T''$, where the casts come from Cast-R rules, and T'' is either $\square_i$ (rule Diag-Univ) or $?_S$ for some S (rule Unk), and we have $\mathbb{\Gamma} \vdash \square_{i+1} \sqsubseteq_{\rightsquigarrow} S_k$, $\mathbb{\Gamma} \vdash \square_{i+1} \sqsubseteq_{\rightsquigarrow} T_k$ and $\mathbb{\Gamma} \vdash \square_{i+1} \sqsubseteq_{\rightsquigarrow} S$. By induction hypothesis, all of S_k, T_k and S reduce either to $\square_{i+1}$ or some $?_{\square_l}$ with

$i + 1 \le l$. Moreover, because T' type-checks against $\square_j$, we must have $S_n \equiv \square_j$. This implies that S_n cannot reduce to $?_{\square_l}$ by confluence, and thus it must reduce to $\square_{i+1}$.

Using that $i + 1 \le l$ and rules Down-Unk, Univ-Univ and Up-Down giving respectively

$$\langle X \Leftarrow ?_{\square_l} \rangle \, ?_{?_{\square_l}} \rightsquigarrow ?_X$$
$$\langle \square_{i+1} \Leftarrow \square_{i+1} \rangle \, t \rightsquigarrow t$$
$$\langle X \Leftarrow ?_{\square_l} \rangle \, \langle ?_{\square_l} \Leftarrow \square_{i+1} \rangle \, t \rightsquigarrow \langle X \Leftarrow \square_{i+1} \rangle \, t$$

we can reduce away all casts. We thus get $T' \rightsquigarrow^* \square_i$ or $T' \rightsquigarrow^* ?_{\square_{i+1}}$, as expected.

For $\sqsubseteq_{\rightsquigarrow}$, if $\mathbb{\Gamma} \vdash \square_i \sqsubseteq_{\rightsquigarrow} T'$ then by decomposing the precision derivation there is an S' such that $T' \rightsquigarrow^* S'$, $\mathbb{\Gamma} \vdash \square_i \sqsubseteq_\alpha S'$, and by subject reduction $\Gamma_1 \vdash S' \blacktriangleright_\square \square_j$. By induction hypothesis, either $S' \rightsquigarrow^* \square_i$ or $S' \rightsquigarrow^* ?_{\square_{i+1}}$, and composing both reductions we get the desired result. □

Proof of Lemma 16. The proof of those catch-up lemmas is very similar to the previous one for structural precision, but this time without the need for induction—we use Lemma 15 instead. We show the one for product types, the others are identical.

First, let us show the property for $\sqsubseteq_\alpha$. Decompose T' into $\langle S_n \Leftarrow U_{n-1} \rangle \dots \langle S_2 \Leftarrow U_1 \rangle \, T''$, where T'' is not a cast, but either some $?_S$ or a product type structurally less precise than $\Pi \, x : A.B$. Now by Lemma 15, U_k, T_k and possibly S all reduce to $\square$ or $?_\square$. Using the same reduction rules as before, all casts can be reduced away, leaving us with either $?_\square$ or a product type structurally less precise than $\Pi \, x : A.B$, as stated.

□

Proof of Lemma 17. The proof still follows the same idea: decompose the less precise term as a series of casts, and show that all those casts can be reduced, using Lemma 16 for product types. However it is somewhat more complex, because the reduction of a cast between product types does a substitution, which we need to handle using the previous substitution lemma for precision.

Let us now detail the reasoning. First, decompose s' into $\langle S_n \Leftarrow U_{n-1} \rangle \dots \langle S_2 \Leftarrow U_1 \rangle \, u'$, where u' is either $\lambda \, x : A''.t''$ or $?_S$ for some S. All of the S_k, U_k and possibly S are definitionally less precise than $\Pi \, x : A.B$. By definition of $\sqsubseteq_{\rightsquigarrow}$ they all reduce to a term structurally less precise than a reduct of $\Pi \, x : A.B$, which must be a product type, and thus by Lemma 16 they all reduce to either some $?_{\square_j}$ or some product type. Moreover, given the typing hypothesis and confluence S_n can only be in the second case. By rule Down-Unk, we get

$$\langle X \Leftarrow ?_\square \rangle \, ?_{?_\square} \rightsquigarrow ?_X$$

so if S is $?_\square$ we can reduce the innermost casts until it is (knowing that we will encounter one because S_n is a product type), then use rule Prod-Unk on u' if it applies, so that without loss of generality we can suppose that u' is an abstraction.

Now we show that all casts reduce, and that this reduction preserves precision, starting with the innermost one. There are three possibilities for that innermost cast.

If it is $\langle ?_{\square_j} \Leftarrow \mathrm{germ}_j \, \Pi \rangle \, u'$, then by typing this cannot be the outermost cast, and thus rule Up-Down applies to get

$$\langle X \Leftarrow ?_{\square_j} \rangle \, \langle ?_{\square_j} \Leftarrow \mathrm{germ}_j \, \Pi \rangle \, u' \rightsquigarrow \langle X \Leftarrow \mathrm{germ}_j \, \Pi \rangle \, u'$$

In the second case, the cast is some $\langle \Pi \, x : A_2.B_2 \Leftarrow \Pi \, x : A_1.B_1 \rangle \, \lambda \, x : A''.t''$, and rule Prod-Prod applies to give

$$\begin{aligned} &\langle \Pi \, x : A_2.B_2 \Leftarrow \Pi \, x : A_1.B_1 \rangle \, \lambda \, x : A''.t'' \rightsquigarrow \\ &\qquad \lambda \, x : A''.\langle B_2 \Leftarrow B_1[\langle A_1 \Leftarrow A_2 \rangle \, x/x] \rangle \, t''[\langle A'' \Leftarrow A_2 \rangle \, x/x] \end{aligned}$$

Moreover, using the precision hypothesis of Cast-R, we know that $\mathbb{\Gamma} \vdash \Pi \, x : A.B \sqsubseteq_{\rightsquigarrow} \Pi \, x : A_1.B_2$ and $\mathbb{\Gamma} \vdash \Pi \, x : A.B \sqsubseteq_{\rightsquigarrow} \Pi \, x : A_2.B_2$. From the first one, using substitution and rule Cast-R, we get that $\Gamma, x : A \mid A_2 \vdash B \sqsubseteq_{\rightsquigarrow} B_1[\langle A_1 \Leftarrow A_2\rangle\, x/x]$. The second gives in particular that $\Gamma \vdash A \sqsubseteq_{\rightsquigarrow} A_2$. Finally, inverting the proof of $\Gamma \vdash \lambda\, x : A.t \sqsubseteq_\alpha \lambda\, x : A''.t''$ we also have $\Gamma \vdash A \sqsubseteq_\alpha A''$ and $\Gamma, x : A \mid A'' \vdash t \sqsubseteq_\alpha t''$. From this, again by substitution, we can derive $\Gamma, x : A \mid A'' \vdash t \sqsubseteq_\alpha t''[\langle A'' \Leftarrow A_2\rangle\, x/x]$. Combining all of those, we can construct a derivation of

$$\Gamma \vdash \lambda\, x : A.t \sqsubseteq_\alpha \lambda\, x : A_2.\langle B_2 \Leftarrow B_1[\langle A_1 \Leftarrow A_2\rangle\, x/x]\rangle\, t'[\langle A'' \Leftarrow A_2\rangle\, x/x]$$

by a use of Diag-Abs followed by one of Cast-R.

The last case corresponds to $\langle ?_{\square_j} \Leftarrow \Pi\, x : A''.B''\rangle\, u'$ when $\Pi\, x : A''.B''$ is not $\mathrm{germ}_j\ h$, in which case the reduction that applies is Prod-Germ, giving

$$\langle ?_{\square_j} \Leftarrow \Pi\, x : A''.B''\rangle\, u' \rightsquigarrow \langle ?_{\square_j} \Leftarrow ?_{\square_{c_\Pi(j)}} \to ?_{\square_{c_\Pi(j)}}\rangle\, \langle ?_{\square_{c_\Pi(j)}} \to ?_{\square_{c_\Pi(j)}} \Leftarrow \Pi\, x : A''.B''\rangle\, u'$$

For this reduct to be less precise that $\lambda\, x : A.t$, we need that all types involved in the casts are definitionally precise than $\Pi\, x : A.B$, as we already have that $\Gamma \vdash \lambda\, x : A.t \sqsubseteq_\alpha u'$. For $?_{\square_j}$ and $\Pi\, x : A''.B''$ it is direct, as they were obtained using Lemma 16 with a reduct of $\Pi\, x : A.B$. Thus only the germ remains, for which it suffices to show that both A and B are less precise than $?_{\square_{c_\Pi(j)}}$. Because $\Pi\, x : A.B$ is typable and less precise than $?_{\square_j}$, we know that $\Gamma_1 \vdash A \blacktriangleright_\square \square_k$ and $\Gamma_1, x : A \vdash B \blacktriangleright_\square \square_l$ with $s_\Pi(k,l) \le j$, thus $k \le c_\Pi(j)$ and $l \le c_\Pi(j)$. Therefore $\Gamma \vdash A \sqsubseteq_\alpha ?_{\square_{c_\Pi(j)}}$ using rule Unk-Univ, and similarly for B.

Note that this last reduction is the point where the system under consideration plays a role: in $\mathrm{CastCIC}^{\mathcal{N}}$, the reasoning does not hold. However, when considering only terms without $?$, this case never happens, and thus the rest of the proof still applies.

Thus, all casts must reduce, and each of those reductions preserves precision, so we end up with a term $\lambda\, x : A'.t'$ such that $\Gamma \vdash \lambda\, x : A.t \sqsubseteq_\alpha \lambda\, x : A'.t'$, as expected. □

Proof of Lemma 18. We start by the proof of the second property. We have as hypothesis that $\Gamma \vdash ?_{I(\mathbf{a})} \sqsubseteq_\alpha s'$, $\Gamma_1 \vdash ?_{I(\mathbf{a})} \triangleright I(\mathbf{a})$ and $\Gamma_2 \vdash s' \blacktriangleright_I I(\mathbf{a}')$, and wish to prove that $s' \rightsquigarrow^* ?_{I(\mathbf{a}')}$ with $\Gamma \vdash I(\mathbf{a}) \sqsubseteq_\alpha I(\mathbf{a}')$.

As previously, decompose s' as $\langle S_n \Leftarrow U_{n-1}\rangle \ldots \langle S_2 \Leftarrow U_1\rangle\, ?_{I(\mathbf{a}'')}$, where all U_k, S_k and $I(\mathbf{a}'')$ are definitionally less precise than $I(\mathbf{a})$, and thus reduce to either $?_{\square_l}$ for some l, or $I(\mathbf{c})$ for some $\mathbf{c}$, and S_n can only be the second by typing. Using the three rules Ind-Unk, Up-Down and Ind-Germ, we respectively get

$$\langle I(\mathbf{c}') \Leftarrow I(\mathbf{c})\rangle\, ?_{I(\mathbf{c}'')} \rightsquigarrow ?_{I(\mathbf{c}')}$$
$$\langle X \Leftarrow ?_{\square_j}\rangle\, \langle ?_{\square_j} \Leftarrow \mathrm{germ}_j\ I\rangle\, u' \rightsquigarrow \langle X \Leftarrow \mathrm{germ}_j\ I\rangle\, u'$$
$$\langle ?_{\square_j} \Leftarrow I(\mathbf{c})\rangle\, u' \rightsquigarrow \langle ?_{\square_j} \Leftarrow \mathrm{germ}_j\ I\rangle\, \langle \mathrm{germ}_j\ I \Leftarrow I(\mathbf{c})\rangle\, u'$$

we can reduce all casts: Up-Down (possibly using §5.1 first) removes all casts through $?_\square$; we can then use Ind-Unk to propagate $?_{I(\mathbf{a}'')}$ all the way through the casts, ending up with $?_{S_n}$ which is the term we sought.

For the first property, again decompose s' as $\langle S_n \Leftarrow U_{n-1}\rangle \ldots \langle S_2 \Leftarrow U_1\rangle\, u'$ where u' does not start with a cast. If u' is some $?_{I(\mathbf{a}'')}$, we can re-use the proof above and are finished. Otherwise u' must be of the form $c(\mathbf{a}'', \mathbf{b}'')$. Again we reduce the casts starting with the innermost, using rules Up-Down and Ind-Germ to remove the occurrences of $?_\square$. The last case to handle is $\langle I(\mathbf{c}') \Leftarrow I(\mathbf{c})\rangle\, c(\mathbf{a}_3, \mathbf{b}_3)$. Then rule Ind-Ind applies, and it preserves precision by repeated uses of the substitution property, and giving a term with c as a head constructor. Thus, we get the desired term with c as a head constructor and arguments less precise than $\mathbf{a}$ and $\mathbf{b}$, respectively. □

Simulation.

Proof of Theorem 20. Both are shown by mutual induction on the precision derivation. We use a stronger induction principle that the one given by the induction rules. Indeed, we need extra induction hypothesis on the inferred type for a term. Proving this stronger principle is done by making the proof of Property 3 slightly more general: instead of proving that an inferred type is always well-formed, we prove that any property consequence of typing is true of all inferred types. Let us now detail the most important cases of the inductive proof.

Definitional precision. We start with the easier second point. The proof is summarized by the following diagram:

$$
\begin{tikzcd}
 & t \arrow[dl] \arrow[d] & \sqsubseteq_{\rightsquigarrow} & t' \arrow[d] \\
s \arrow[dr, dashed] & u \arrow[d, dashed] & \sqsubseteq_{\alpha} & u' \arrow[d, dashed] \\
 & v & \sqsubseteq_{\alpha} & v'
\end{tikzcd}
$$

By definition of $\sqsubseteq_{\rightsquigarrow}$, there exists u and u', reducts respectively of t and t', and such that $\Gamma \vdash u \sqsubseteq_{\alpha} u'$. By confluence, there exists some v that is a reduct of both u and s. By subject reduction, u and u' are both well-typed, and thus by induction hypothesis there exists v' such that $u' \rightsquigarrow^* v'$ and $\Gamma \vdash v \sqsubseteq_{\alpha} v'$. But then v is a reduct of s and v' is a reduct of t', and so $\Gamma \vdash s \sqsubseteq_{\rightsquigarrow} t'$.

This implies in particular that if $\Gamma \vdash t \triangleright T$, $\Gamma \vdash T \sqsubseteq_{\rightsquigarrow} T'$, $t \rightsquigarrow^* s$ and $\Gamma_1 \vdash s \triangleright S$, then $\Gamma \vdash S \sqsubseteq_{\rightsquigarrow} T'$. Indeed $\Gamma_1 \vdash s \triangleleft T$ by subject reduction, thus S and T are convertible, and have a common reduct U by confluence. The property just stated then gives $\Gamma \vdash U \sqsubseteq_{\rightsquigarrow} T'$, hence $\Gamma \vdash S \sqsubseteq_{\rightsquigarrow} T'$.

Syntactic precision—Non-diagonal precision rules. Let us now turn to $\sqsubseteq_{\alpha}$. It is enough to show that one step of reduction can be simulated, by induction on the path $t \rightsquigarrow^* s$.

First, we get rid of most cases where the last rule used for $\Gamma \vdash t \sqsubseteq_{\alpha} t'$ is not a diagonal rule. For Unk we must handle the side-condition involving the type of t. However, by the previous property, the inferred type of s is also definitionally less precise than T'. Thus the reduction in t can be simulated by zero reduction steps. The reasoning for rules Err and Err-Lambda is similar. As for rule Diag-Univ, subject reduction is enough to get what we seek, without even resorting to the previous property. Rule Cast-R is treated in the same way as Unk, as the typing side-conditions are similar. Thus the only non-diagonal rule left for $\sqsubseteq_{\alpha}$ is Cast-L.

Syntactic precision—Non-top-level reduction. Next, we can get rid of reductions that do not happen at top level. Indeed, if the last rule used was Cast-L, and the reduction happens in one of the types of the cast, the same reasoning as for Cast-R applies. If it happens in the term, we can use the induction hypothesis on this term to conclude. Also, if the last rule used was a diagonal rule, then the reduction in t can be simulated by a similar congruence rules in t'.

So we are left with the simulation of a reduction that happens at the top-level in t, and where the last precision rule used is either Cast-L or a diagonal one, and this is the real core of the proof.

Syntactic precision—non-diagonal cast. Let us first turn to the case where the last precision rule is Cast-L, and that cast reduces. More precisely, t is some $\langle T \Leftarrow S \rangle\, u$, with $\Gamma \vdash u \sqsubseteq_{\alpha} t'$. There are four possibilities for the reduction.

- The cast fails. When it does, whatever the rule, it always reduces to $\mathtt{err}_T$. But then we know that $\Gamma_2 \vdash t' \triangleright T'$ and $\Gamma \vdash T \sqsubseteq_{\rightsquigarrow} T'$. Thus $\Gamma \vdash \mathtt{err}_T \sqsubseteq_{\alpha} t'$ by rule Err, and the reduction is simulated by zero reductions.
- The cast disappears (Univ-Univ) or expands into two casts without changing u (Ind-Germ, Prod-Germ). In those cases the reduct of t is still smaller than t'. In the case of cast

expansion, we must use Cast-L twice, and thus prove that the type of t' is less precise than the introduced germ. But by the Cast-L rule that was used to prove $\mathbb{\Gamma} \vdash t \sqsubseteq_\alpha t'$, we know that t' infers a type T' which is definitionally less precise than some $?_{\square_i}$. Thus, T' reduces to some S' such that $\mathbb{\Gamma} \vdash ?_{\square_i} \sqsubseteq_\alpha S'$, and this implies that also $\mathbb{\Gamma} \vdash \mathrm{germ}_i\ h \sqsubseteq_\alpha S'$, *i.e.*, what we sought.

- Both T and S are either product types or inductive types, and u starts with an abstraction or an inductive constructor. In that case, by Lemmas 17 and 18, t' reduces to a term u' with the same head constructor as u or some $?_{I(\mathbf{a})}$. In the first case, by the substitution property of precision we have $\mathbb{\Gamma} \vdash s \sqsubseteq_\alpha u'$. In the second, we can use Unk to conclude.
- The reduction rule is Up-Down, that is t is $\langle T \Leftarrow ?_{\square_i}\rangle \langle ?_{\square_i} \Leftarrow \mathrm{germ}_i\ h\rangle\, u$ which reduces to $\langle T \Leftarrow \mathrm{germ}_i\ h\rangle\, u$. If rule Cast-L was used twice in a row then we directly have $\mathbb{\Gamma} \vdash u \sqsubseteq_\alpha t'$ and so $\mathbb{\Gamma} \vdash \langle X \Leftarrow \mathrm{germ}_i\ h\rangle\, u \sqsubseteq_\alpha t'$. Otherwise, rule Diag-Cast was used, t' is some $\langle T' \Leftarrow S'\rangle\, u'$ and we have $\mathbb{\Gamma} \vdash u \sqsubseteq_\alpha u'$ and $\mathbb{\Gamma}_1 \vdash \mathrm{germ}_i\ h \sqsubseteq_{\rightsquigarrow} S'$. Moreover, Cast-L also gives $\mathbb{\Gamma}_1 \vdash X \sqsubseteq_{\rightsquigarrow} B'$, since $\mathbb{\Gamma}_2 \vdash \langle B' \Leftarrow A'\rangle\, u' \triangleright B'$. Thus $\mathbb{\Gamma} \vdash \langle X \Leftarrow \mathrm{germ}_i\ h\rangle\, u \sqsubseteq_\alpha \langle B' \Leftarrow A'\rangle\, u'$ by a use of Diag-Cast.

Syntactic precision—β redex. Next we consider the case where t is a β redex $(\lambda\, x : A.t_1)\ t_2$. Because the last applied precision rule is diagonal, t' must also decompose as $t_1''\ t_2'$. If t_1 is some $\mathtt{err}_T$, then the reduct is $\mathtt{err}_T$ and must be still smaller that t'. Otherwise, Lemma 17 applies, thus t_1'' reduces to some $\lambda\, x : A'.t_1'$ that is syntactically less precise than $\lambda\, x : A.t_1$. Then the β reduction of t can be simulated with a β reduction in t', and using the substitution property we conclude that the redexes are still related by precision.

Syntactic precision—ι redex. If t is a ι redex $\mathtt{ind}_{c(\mathbf{a},\mathbf{b})}(I, z.P, \mathbf{f}.\mathbf{y}.\mathbf{t})$, the reasoning is similar. Because the last precision rule is diagonal, t' must also be a fixpoint. We thus can use Lemma 18 to ensure that its scrutinee reduces either to $c(\mathbf{a}', \mathbf{b}')$ or $?_{I(\mathbf{a}')}$. In the first case, a ι reduction of t' and the substitution property is enough to conclude. In the second case, t' reduces to a term $s' := ?_{P'[?_{I(\mathbf{a}')}/z]}$, and we must show this term to be less precise than s, which is $t_k[\lambda\, x : I(\mathbf{a}).\ \mathtt{ind}_I(x, z.P, \mathbf{f}.\mathbf{y}.\mathbf{t})/z][\mathbf{b}/\mathbf{y}]$. Let S be the type inferred for s, by rule Unk, it is enough to show $\mathbb{\Gamma} \vdash S \sqsubseteq_{\rightsquigarrow} P'[?_{I(\mathbf{a}')}/z]$. By subject reduction, S and $P[c_k(\mathbf{a}, \mathbf{b})/z]$ (the type of t) are convertible, thus they have a common reduct U. Now we also have by substitution that $\mathbb{\Gamma} \vdash P[c_k(\mathbf{a}, \mathbf{b})/z] \sqsubseteq_\alpha P'[?_{I(\mathbf{a}')}/z]$. Because $P[c_k(\mathbf{a}, \mathbf{b})/z]$ is the inferred type for t, the induction hypothesis applies to it, and thus there is some U' such that $P'[?_{I(\mathbf{a}')}z/] \rightsquigarrow^* U'$ and also $\mathbb{\Gamma} \vdash U \sqsubseteq_\alpha U'$.

Syntactic precision—$\mathtt{err}$ and $?$ reductions. For reductions Prod-Err, *i.e.,* when $\mathtt{err}_{\Pi\, x:A.B} \rightsquigarrow \lambda\, x : A.\, \mathtt{err}_B$, we can replace the use of Err by a use of Err-Lambda. For reduction Ind-Err, *i.e.,* when t is $\mathtt{ind}_I(\mathtt{err}_{I(\mathbf{a})}, z.P, \mathbf{f}.\mathbf{y}.\mathbf{t})$ we distinguish three cases depending on t'. If t' is $?_{T'}$ (the precision rule between t and t' was Unk) or $\langle T' \Leftarrow S'\rangle\, t'$, then $\mathbb{\Gamma} \vdash P[\mathtt{err}_{I(\mathbf{a})}/z] \sqsubseteq_{\rightsquigarrow} T'$, and thus $\mathbb{\Gamma} \vdash \mathtt{err}_{P[\mathtt{err}_{I(\mathbf{a})}/z]} \sqsubseteq_\alpha t'$ by using Err. Otherwise, the last rule was Diag-Fix, and again we can conclude using Err and the substitution property of $\sqsubseteq_\alpha$.

Conversely, let us consider the reduction rules for $?$. If t is $?_{\Pi\, x:A.B}$ and reduces to $\lambda\, x : A.?_B$, then t' must be $?_T$, possibly surrounded by casts. If there are casts, they can all be reduced away until we are left with $?_{T'}$ for some T' such that $\mathbb{\Gamma} \vdash \Pi\, x : A.B \sqsubseteq_{\rightsquigarrow} T$. By Lemma 16, $T \rightsquigarrow^* ?_{?_\square}$ or $T \rightsquigarrow^* T_{\Pi\, x:A'.B'}$. In the first case, $?_{?_\square}$ is still less precise than $\lambda\, x : A.B$, and in the second case, $?_{\Pi\, x:A'.B'}$ can reduce to $\lambda\, x : A'.?_{B'}$, which is less precise than s'. If t is $\mathtt{ind}_I(?_{I(\mathbf{a})}, P, \mathbf{b})$, reducing to $?_{P[?_{I((a))}/z]}$, we use the second part of Lemma 18 to conclude that also t' reduces to some $\mathtt{ind}_I(?_{I(\mathbf{a}')}, P', \mathbf{b}')$ that is less precise than t. From this, $t' \rightsquigarrow ?_{P'[?_{I((a'))}/z]}$, which is less precise than s.

Syntactic precision—diagonal cast reduction. This only leaves us with the reduction of a cast when the precision rule is Diag-Cast: we have some $\langle T \Leftarrow S\rangle\, u$ and $\langle T' \Leftarrow S'\rangle\, u'$ that are pointwise related by precision, such that $\langle T \Leftarrow S\rangle\, t \rightsquigarrow^* s$ by a head reduction, and we must show that $\langle T \Leftarrow S\rangle\, u$ simulates that reduction.

First, if the reduction for $\langle T \Leftarrow S\rangle\, t$ is any reduction to an error, then the reduct is $\mathtt{err}_T$, and since $\mathbb{\Gamma}_2 \vdash \langle T' \Leftarrow S'\rangle\, u' \triangleright T'$ and $\mathbb{\Gamma} \vdash T \sqsubseteq_{\rightsquigarrow} T'$ we can use rule Err to conclude.

Next, consider Prod-Prod. We are in the situation where t is $\langle \Pi\, x : A_2.B_2 \Leftarrow \Pi\, x : A_1.B_1\rangle\, \lambda\, x : A.v$. If v is $\mathtt{err}_{B_1}$ then the reduct is more precise than any term. Otherwise, by Lemma 16, S' reduces either to $?_\square$ or to a product type. In the first case, u' must reduce to $?_{?_\square}$ by Lemma 17, since it is less precise than $\lambda\, x : A.v$ and by typing it cannot start with a λ. In that case, $\langle T' \Leftarrow S'\rangle\, u' \rightsquigarrow ?_{T'}$, and since $\mathbb{\Gamma} \vdash \Pi\, x : A_2.B_2 \sqsubseteq_{\rightsquigarrow} T'$, we have that $\mathbb{\Gamma} \vdash s \sqsubseteq_\alpha ?_{T'}$. Otherwise S' reduces to some $\Pi\, x : A_1'.B_1'$. By Lemma 17, t' reduces either to some $?$ or to an abstraction. In the first case, the previous reasoning still applies. Otherwise, t' reduces to some $\lambda\, x : A'.v'$. Again, by Lemma 16, T' reduces either to a product type or to $?$. In the first case t' can simply do the same cast reduction as t, and the substitution property of precision enables us to conclude. Thus, the only case left is that where t' is $\langle ?_{\square_i} \Leftarrow \Pi\, x : A_1'.B_1'\rangle\, \lambda\, x : A'.v'$. If $\Pi\, x : A_1'.B_1'$ is $\mathrm{germ}_i\ \Pi$, then all of A, A_1, A_2, B_1 and B_2 are more precise than $?_{\square_{c_\Pi(i)}}$, and this is enough to conclude that s is less precise than $\langle \mathrm{germ}_i\ \Pi \Leftarrow ?_{\square_i}\rangle\, \lambda\, x : ?_{\square_{c_\Pi(i)}}.u'$, using the substitution property of precision to relate u' with the substituted u, and the Diag-Abs, Cast-L and Cast-R rules. The last case is when $\Pi\, x : A_1'.B_1'$ is not a germ. Then the reduction of t' first does a cast expansion through $\mathrm{germ}_i\ \Pi$, followed by a reduction of the cast between $\Pi\, x : A_1'.B_1'$ and $\mathrm{germ}_i\ \Pi$. The reasoning of the two previous cases can be used again to conclude. The proof is similar for rule Ind-Ind.

Next, let us consider Prod-Germ, that is when t is $\langle ?_{\square_i} \Leftarrow \Pi\, x : A_1.B_1\rangle\, f$. We have that $T' \rightsquigarrow ?_{\square_j}$ by Lemma 16 with $i \leq j$, and thus $\mathbb{\Gamma} \vdash \mathrm{germ}_i\ \Pi \sqsubseteq_{\rightsquigarrow} T'$. Thus, using Diag-Cast for the innermost cast in s, and Cast-L for the outermost one, we conclude $\mathbb{\Gamma} \vdash s \sqsubseteq_\alpha \langle T' \Leftarrow S'\rangle\, u'$. Again, the reasoning is similar for Ind-Germ.

As for Univ-Univ, t is $\langle \square_i \Leftarrow \square_i\rangle\, A$, and we can replace rule Diag-Cast by rule Cast-R. Indeed $\mathbb{\Gamma}_1 \vdash A \triangleleft \square_i$ by typing, thus $\mathbb{\Gamma}_1 \vdash A \triangleright T$ for some T such that $T \rightsquigarrow \square_i$. Therefore, since $\mathbb{\Gamma} \vdash \square_i \sqsubseteq_{\rightsquigarrow} T'$, we have $\mathbb{\Gamma} \vdash T \sqsubseteq_{\rightsquigarrow} T'$ and similarly $\mathbb{\Gamma} \vdash T \sqsubseteq_{\rightsquigarrow} S'$. Thus, rule Cast-R gives $\mathbb{\Gamma} \vdash A \sqsubseteq_\alpha t'$.

The last case left is the one of Up-Down, where t is $\langle X \Leftarrow ?_{\square_i}\rangle\, \langle ?_{\square_i} \Leftarrow \mathrm{germ}_i\ h\rangle\, v$. We distinguish on the rule used to prove $\mathbb{\Gamma} \vdash \langle ?_{\square_i} \Leftarrow \mathrm{germ}_i\ h\rangle\, v \sqsubseteq_\alpha u'$. If it is Cast-L, then we simply have $\mathbb{\Gamma} \vdash \langle X \Leftarrow \mathrm{germ}_i\ h\rangle\, t \sqsubseteq_\alpha \langle T' \Leftarrow S'\rangle\, u'$ using rule Diag-Cast, as $\mathbb{\Gamma} \vdash \mathrm{germ}_i\ h \sqsubseteq_{\rightsquigarrow} S'$ since $\mathbb{\Gamma} \vdash ?_{\square_i} \sqsubseteq_{\rightsquigarrow} S'$. Otherwise the rule is Diag-Cast, t' reduces to $\langle T' \Leftarrow ?_{\square_j}\rangle\, \langle ?_{\square_j} \Leftarrow U'\rangle\, u'$, using Lemma 16 to reduce types less precise than $?_{\square_i}$ to some $?_{\square_j}$ with $i \leq j$. We can use Diag-Cast on the outermost cast, and Cast-R on the innermost to prove that this term is less precise than s, as $\mathbb{\Gamma} \vdash \mathrm{germ}_i\ h \sqsubseteq_{\rightsquigarrow} ?_{\square_j}$ since $i \leq j$. □

B.3 Properties of GCIC

Conservativity is an equivalence, so to prove it we break it down into two implications. We now state and prove those in an open context and for the three different judgments.

Theorem 37 (GCIC is weaker than CIC—Open context). *Let $\tilde{t}$ be a static term and Γ an erasable context. Then*

- *if $\varepsilon(\Gamma) \vdash_{\mathrm{CIC}} t \triangleright T$ then $\Gamma \vdash \tilde{t} \rightsquigarrow t \triangleright T'$ for some erasable t and T' such that $\varepsilon(t) = \tilde{t}$ and $\varepsilon(T') = T$;*
- *if T' is an erasable term of CastCIC, and $\varepsilon(\Gamma) \vdash_{\mathrm{CIC}} \tilde{t} \triangleleft \varepsilon(T')$ then $\Gamma \vdash \tilde{t} \triangleleft T' \rightsquigarrow t$ for some erasable t such that $\varepsilon(t) = \tilde{t}$;*
- *if $\varepsilon(\Gamma) \vdash_{\mathrm{CIC}} \tilde{t} \blacktriangleright_{\mathrm{h}} T$ then $\Gamma \vdash \tilde{t} \rightsquigarrow t \blacktriangleright_{\mathrm{h}} T'$ for some erasable t and T' such that $\varepsilon(t) = \tilde{t}$ and $\varepsilon(T') = T$.*

Proof. Once again, the proof is by mutual induction, on the typing derivation of $\tilde{t}$ in CIC.

All inference rules are direct: one needs to combine the induction hypothesis together, using the substitution property of precision and the fact that erasure commutes with substitution to handle the cases of substitution in the inferred types.

Let us consider the case of Prod-Inf next. We are given Γ erasable, and suppose $\varepsilon(\Gamma) \vdash_{\mathrm{CIC}} \tilde{t} \triangleright T$ and $T \rightsquigarrow^* \Pi x : A.B$. By induction hypothesis there exists t and T' erasable such that $\Gamma \vdash t \rightsquigarrow \tilde{t} \triangleright T'$ and $\varepsilon(t) = \tilde{t}$, $\varepsilon(T') = T$. Because T' is erasable, it is less precise than T. By Corollary 21, it must reduce to either $?_{\square}$ or a product type. The first case is impossible because T' does not contain any $?$ as it is erasable. Thus there are some A' and B' such that $T' \rightsquigarrow^* \Pi x : A'.B'$ and $\Gamma \vdash \Pi x : A.B \sqsubseteq_\alpha \Pi x : A'.B'$. Since also $\Gamma \vdash T' \sqsubseteq_\alpha T$, by the same reasoning there are also some A'' and B'' such that $T \rightsquigarrow^* \Pi x : A''.B''$ and $\Gamma \vdash \Pi x : A'.B' \sqsubseteq_\alpha \Pi x : A''.B''$. Now because T is static, so are $\Pi x : A.B$ and $\Pi x : A''.B''$, and because of the comparisons with $\Pi x : A'.B'$ we must have $\varepsilon(\Gamma) \vdash \Pi x : A.B \sqsubseteq_\alpha \Pi x : A''.B''$. Since both are static, this means they must be α-equal, since no non-diagonal rule can be used on static terms. Hence, $\Pi x : A.B = \Pi x : A''.B'' = \varepsilon(\Pi x : A'.B')$, implying that $\Pi x : A'.B'$ is erasable. Thus, $\Gamma \vdash \tilde{t} \rightsquigarrow t \blacktriangleright_{\Pi} \Pi x : A'.B'$, both t and $\Pi x : A'.B'$ are erasable, and moreover $\varepsilon(t) = \tilde{t}$ and $\varepsilon(\Pi x : A'.B') = \Pi x : A.B$, which is what had to be proven.

The other constrained inference rules being very similar, let us turn to Check. We are given Γ and T' erasable, and suppose that $\varepsilon(\Gamma) \vdash_{\mathrm{CIC}} \tilde{t} \triangleright S$ such that $S \equiv \varepsilon(T')$. By induction hypothesis, $\Gamma' \vdash \tilde{t} \rightsquigarrow t \triangleright S'$ with t and S' erasable, $\varepsilon(t) = \tilde{t}$ and $\varepsilon(S') = S$. But convertibility implies consistency, so $S \sim \varepsilon(T')$. By monotonicity of consistency, this implies $S' \sim T'$. Thus $\Gamma \vdash \tilde{t} \triangleleft T' \rightsquigarrow \langle T' \Leftarrow S' \rangle\, t$. We have $\varepsilon(\langle T' \Leftarrow S' \rangle\, t) = \varepsilon(t) = \tilde{t}$, so we are left with showing that $\Gamma \vdash \langle T' \Leftarrow S' \rangle\, t \sqsupseteq\sqsubseteq_\alpha \tilde{t}$. Using rules Cast-L and Cast-R, and knowing already that $\Gamma \vdash S' \sqsupseteq\sqsubseteq_\alpha S$, it remains to show that $\Gamma \vdash T' \sqsubseteq_{\rightsquigarrow} S$ and $\Gamma \vdash S \sqsubseteq_{\rightsquigarrow} T'$. As S and $\varepsilon(T')$ are convertible, let U be a common reduct. Using Theorem 20, $T' \rightsquigarrow^* U'$ with $\Gamma \vdash U \sqsubseteq_\alpha U'$. Simulating that reduction again, we get $\varepsilon(T') \rightsquigarrow^* U''$ with $\Gamma \vdash U'' \sqsubseteq_\alpha U'$. As before, this implies $U = U'' = \varepsilon(U')$. Thus, using the reduct U' of T' that is equiprecise with U, we can conclude $\Gamma \vdash S \sqsubseteq_{\rightsquigarrow} T'$ and $\Gamma \vdash T' \sqsubseteq_{\rightsquigarrow} S$. □

Theorem 38 (CIC is weaker than GCIC—Open context). *Let $\tilde{t}$ be a static term and Γ an erasable context of* CastCIC. *Then*

- *if $\Gamma \vdash \tilde{t} \rightsquigarrow t \triangleright T$, then t and T are erasable, $\varepsilon(t') = \tilde{t}$ and $\varepsilon(\Gamma) \vdash \tilde{t} \triangleright \varepsilon(T')$;*
- *if T' is an erasable term of* CastCIC *such that $\Gamma' \vdash \tilde{t} \triangleleft T' \rightsquigarrow t'$, then t' is erasable, $\varepsilon(t') = \tilde{t}$ and $\varepsilon(\Gamma) \vdash \tilde{t} \triangleleft \varepsilon(T')$;*
- *if $\Gamma' \vdash \tilde{t} \rightsquigarrow t' \blacktriangleright_{h} T'$, then t' and T' are erasable, $\varepsilon(t') = \tilde{t}$ and $\varepsilon(\Gamma) \vdash \tilde{t} \blacktriangleright_{h} \varepsilon(T')$.*

Proof. The proof is similar to the previous one. Again, the tricky part is to handle reduction steps, and we use equiprecision in the same way to conclude in those. □

As a direct corollary of those propositions in an empty context, we get conservativity Theorem 23.

Elaboration graduality. Now for the elaboration graduality: again, we state it in an open context for all three typing judgments.

Theorem 39 (Elaboration graduality—Open context). *Let $\mathbb{\Gamma}$ be a context such that $\mathbb{\Gamma}_1 \sqsubseteq_\alpha \mathbb{\Gamma}_2$, and $\tilde{t}$ and $\tilde{t}'$ be two* GCIC *terms such that $\tilde{t} \sqsubseteq^{G}_{\alpha} \tilde{t}'$. Then*

- *if $\mathbb{\Gamma}_1 \vdash \tilde{t} \rightsquigarrow t \triangleright T$ is universe adequate, then there exists t' and T' such that $\mathbb{\Gamma}_2 \vdash \tilde{t}' \rightsquigarrow t' \triangleright T'$, $\mathbb{\Gamma} \vdash t \sqsubseteq_\alpha t'$ and $\mathbb{\Gamma} \vdash T \sqsubseteq_\alpha T'$;*
- *If $\mathbb{\Gamma}_1 \vdash t \triangleleft T \rightsquigarrow t'$ is universe adequate, then for all T' such that $\mathbb{\Gamma} \vdash T \sqsubseteq_\alpha T'$ there exists t' such that $\mathbb{\Gamma}_2 \vdash \tilde{t}' \triangleleft T' \rightsquigarrow t'$ and $\mathbb{\Gamma} \vdash t \sqsubseteq_\alpha t'$;*
- *If $\mathbb{\Gamma}_1 \vdash t \rightsquigarrow t' \blacktriangleright_{h} T$ is universe adequate, then there exists t' and T' such that $\mathbb{\Gamma}_2 \vdash \tilde{t}' \rightsquigarrow t' \blacktriangleright_{h} T'$, $\mathbb{\Gamma} \vdash t \sqsubseteq_\alpha t'$ and $\mathbb{\Gamma} \vdash T \sqsubseteq_\alpha T'$.*

Proof. Once again, we use our favorite tool: induction on the typing derivation of $\tilde{t}$.

Inference—Non-diagonal precision. For inference, we have to make a distinction on the rule used to prove *tildet* $\sqsubseteq_\alpha^{\mathrm{G}} \tilde{t}'$: we have to handle specifically the non-diagonal one, where $\tilde{t}'$ is some $?$. We start with this, and treat the ones where the rule is diagonal (*i.e.*, when $\tilde{t}$ and $\tilde{t}'$ have the same head) next.

We have $\mathbb{\Gamma}_1 \vdash \tilde{t} \rightsquigarrow t' \triangleright T'$ and $\mathbb{\Gamma}_2 \vdash ?_{@\{i\}} \rightsquigarrow ?_{?_{\square_i}} \triangleright ?_{\square_i}$. Correctness of elaboration gives $\mathbb{\Gamma}_1 \vdash t' \triangleright T'$, and by validity $\Gamma_1 \vdash T' \triangleright \square_i$, universe adequacy ensuring us that this i is the same as the one in $\tilde{t}'$. Thus we have $\mathbb{\Gamma} \vdash T' \sqsubseteq_\alpha ?_{\square_i}$ by rule Unk, and in turn $\mathbb{\Gamma} \vdash t' \sqsubseteq_\alpha ?_{?_{\square_i}}$ by a second use of the same rule, giving us the required conclusions.

Inference—Variable. Rule Var gives us $(x : T) \in \mathbb{\Gamma}_1$. Because $\vdash \mathbb{\Gamma}_1 \sqsubseteq_\alpha \mathbb{\Gamma}_2$, there exists some T' such that $(x : T') \in \mathbb{\Gamma}_2$, and $\mathbb{\Gamma} \vdash T \sqsubseteq_\alpha T'$ using weakening. Thus, $\mathbb{\Gamma}_2 \vdash x \rightsquigarrow x \triangleright T'$, and of course $\mathbb{\Gamma} \vdash x \sqsubseteq_\alpha x$.

Inference—Product. Premises of rule Prod give $\mathbb{\Gamma}_1 \vdash \tilde{A} \rightsquigarrow A \blacktriangleright_\square \square_i$ and $\mathbb{\Gamma}_1, x : A \vdash \tilde{B} \rightsquigarrow B \blacktriangleright_\square \square_j$, and the diagonal precision one gives $\tilde{A} \sqsubseteq_\alpha^{\mathrm{G}} \tilde{A}'$ and $\tilde{B} \sqsubseteq_\alpha^{\mathrm{G}} \tilde{B}'$. Applying the induction hypothesis, we get some A' such that $\mathbb{\Gamma}_2 \vdash \tilde{A}' \rightsquigarrow A' \blacktriangleright_\square \square_i$ and $\mathbb{\Gamma} \vdash A \sqsubseteq_\alpha A'$. The inferred type for $\tilde{A}'$ must be $\square_i$ as it is some $\square_j$ because of the constrained elaboration, and it is less precise than $\square_i$ by the induction hypothesis. From this, we also deduce that $\mathbb{\Gamma}_1, x : A \sqsubseteq_\alpha \mathbb{\Gamma}_2, x : A'$. Hence the induction hypothesis can be applied to $\tilde{B}$, giving $\mathbb{\Gamma}_2 \vdash \tilde{B}' \rightsquigarrow B' \blacktriangleright_\square \square_j$. Combining this with the elaboration for $\tilde{A}'$, we obtain $\mathbb{\Gamma}_2 \vdash \Pi x : \tilde{A}'.\tilde{B}' \rightsquigarrow \Pi x : A'.B' \triangleright \square_{s_\Pi(i,j)}$. Moreover, $\mathbb{\Gamma} \vdash \Pi x : A.B \sqsubseteq_\alpha \Pi x : A'.B'$ by combining the precision hypothesis on A and B, and also $\mathbb{\Gamma} \vdash \square_{s_\Pi(i,j)} \sqsubseteq_\alpha \square_{s_\Pi(i,j)}$.

Inference—Application. From rule App, we have $\mathbb{\Gamma}_1 \vdash \tilde{t} \rightsquigarrow t \blacktriangleright_\Pi \Pi x : A.B$ and $\mathbb{\Gamma}_1 \vdash \tilde{u} \triangleleft A \rightsquigarrow u$, and the diagonal precision gives $\tilde{t} \sqsubseteq_\alpha^{\mathrm{G}} \tilde{t}'$ and $\tilde{u} \sqsubseteq_\alpha^{\mathrm{G}} \tilde{u}'$. By induction, we have $\mathbb{\Gamma}_1 \vdash \tilde{t}' \rightsquigarrow t' \blacktriangleright_\Pi \Pi x : A'.B'$ for some t', A' and B' such that $\mathbb{\Gamma} \vdash t \sqsubseteq_\alpha t'$, $\mathbb{\Gamma} \vdash A \sqsubseteq_\alpha A'$ and $\mathbb{\Gamma}, x : A \mid A' \vdash B \sqsubseteq_\alpha B'$. Using the induction hypothesis again with that precision property on A and A' gives $\mathbb{\Gamma}_2 \vdash \tilde{u}' \triangleleft A' \rightsquigarrow u'$ with $\mathbb{\Gamma} \vdash u \sqsubseteq_\alpha u'$. Therefore combining those we get $\mathbb{\Gamma}_2 \vdash \tilde{t}'\ \tilde{u}' \rightsquigarrow t'\ u' \triangleright B'[u'/x]$, $\mathbb{\Gamma} \vdash t\ u \sqsubseteq_\alpha t'\ u'$ and, by substitution property of precision, $\mathbb{\Gamma} \vdash B[u/x] \sqsubseteq_\alpha B'[u'/x]$.

Inference—Other diagonal cases. All other cases are similar to those: combining the induction hypothesis directly leads to the desired result, handling the binders in a similar way to that of products when needed.

Checking. For Check, we have that $\mathbb{\Gamma}_1 \vdash \tilde{t} \rightsquigarrow t \triangleright S$, with $S \sim T$. By induction hypothesis, $\mathbb{\Gamma}_2 \vdash \tilde{t}' \rightsquigarrow \tilde{t}' \triangleright S'$ with $\mathbb{\Gamma} \vdash t \sqsubseteq_\alpha t'$ and $\mathbb{\Gamma} \vdash S \sqsubseteq_\alpha S'$. But we also have as an hypothesis that $\mathbb{\Gamma} \vdash T \sqsubseteq_\alpha T'$. By monotonicity of consistency, we conclude that $S' \sim T'$, and thus $\mathbb{\Gamma}_2 \vdash \tilde{t}' \triangleleft T' \rightsquigarrow \langle T' \Leftarrow S' \rangle\, t'$. A use of Diag-Cast then ensures that $\mathbb{\Gamma} \vdash \langle T \Leftarrow S \rangle\, t \sqsubseteq_\alpha \langle T' \Leftarrow S' \rangle\, t'$, as desired.

Constrained inference—Inf-Prod rule. We are in the situation where $\mathbb{\Gamma}_1 \vdash \tilde{t} \rightsquigarrow t \triangleright S$ and $S \leadsto^* \Pi x : A.B$. By induction hypothesis, $\mathbb{\Gamma}_2 \vdash \tilde{t}' \rightsquigarrow t' \triangleright S'$ with $\mathbb{\Gamma} \vdash S \sqsubseteq_\alpha S'$. Using Corollary 21, we get that $S' \leadsto^* \Pi x : A'.B'$ such that $\mathbb{\Gamma} \vdash \Pi x : A.B \sqsubseteq_\alpha \Pi x : A'.B'$, or $S' \leadsto^* ?_{\square_i}$. In the first case, by rule Inf-Prod we get $\mathbb{\Gamma}_2 \vdash \tilde{t}' \rightsquigarrow t' \blacktriangleright_\Pi \Pi x : A'.B'$ together with the precision inequalities for t' and $\Pi x : A'.B'$. In the second case, we can use rule Inf-Prod? instead, and get $\mathbb{\Gamma}_2 \vdash \tilde{t}' \rightsquigarrow \langle \mathrm{germ}_i\ \Pi \Leftarrow S' \rangle\, t' \blacktriangleright_\Pi \mathrm{germ}_i\ \Pi$, and $c_\Pi(i)$ is larger than the universe levels of both A' and B'. A use of Cast-R, together with the fact that $\mathbb{\Gamma} \vdash A \sqsubseteq_\alpha ?_{\square_{c_\Pi(i)}}$ by Unk-Univ and similarly for B, gives that $\mathbb{\Gamma} \vdash t' \sqsubseteq_\alpha \langle \mathrm{germ}_i\ \Pi \Leftarrow S' \rangle\, t'$, and the precision between types has been established already.

Constrained inference—Inf-Prod?. This time, $\mathbb{\Gamma}_1 \vdash \tilde{t} \rightsquigarrow t \triangleright S$, but $S \leadsto^* ?_{\square_i}$. By induction hypothesis, $\mathbb{\Gamma}_2 \vdash \tilde{t}' \rightsquigarrow t' \triangleright S'$ with $\mathbb{\Gamma} \vdash S \sqsubseteq_\alpha S'$. By Corollary 21, we get that $S' \leadsto^* ?_{\square_i}$. Thus $\mathbb{\Gamma}_2 \vdash \tilde{t}' \rightsquigarrow \langle \mathrm{germ}_i\ \Pi \Leftarrow S' \rangle\, t' \blacktriangleright_\Pi \mathrm{germ}_i\ \Pi$. A use of Diag-Cast is enough to conclude.

Constrained inference—Other rules. All other cases are similar to the previous ones, albeit with a simpler handling of universe levels (since c_Π does not appear).

□

C CONNECTING THE DISCRETE AND MONOTONE MODELS

Comparing the discrete and the monotone translations, we can see that they coincide on ground types such as $\mathbb{N}$. On functions over ground types, for instance $\mathbb{N}\to\mathbb{N}$, the monotone interpretation is more conservative: any monotone function $f : \{\!| \mathbb{N}\to\mathbb{N} |\!\}$ induces a function $\tilde{f} : [\![\mathbb{N}\to\mathbb{N}]\!]$ by forgetting the monotonicity, but not all functions from $[\![\mathbb{N}\to\mathbb{N}]\!]$ are monotone[25].

Extending the sketched correspondence at higher types, we obtain a (binary) logical relation $\wr - \int$ between terms of the discrete and monotone translations described in Fig. 20, that forgets the monotonicity information on ground types. More precisely we define for each types A in the source a relation $\wr A \int : [\![A]\!] \to \{\!| A |\!\} \to \square$ and for each term $t : A$ a witness $\wr t \int : \wr A \int\ [t]\ \{t\}$.

The logical relation employs a an inductively defined relation $\square_{\mathrm{rel},i}$ between $\square_i^{\mathrm{dis}} := [\![\square_i]\!]$ and $\square_i^{\mathrm{mon}} := \{\!| \square_i |\!\}$ whose constructors are relational codes relating codes of discrete and monotone types. These relational codes are then decoded to relations between the corresponding decoded types thanks to $\mathrm{El}_{\mathrm{rel}}$. The main difficult case in establishing the logical relation lie in relating the `cast`s, since that's the main point of divergence of the two models.

Lemma 40 (Basis lemma).

(1) There exists a term $\mathtt{cast}_{rel} : \wr \Pi(A\,B : \square).A \to B \int\ [\mathtt{cast}]\ \{\mathtt{cast}\}$.
(2) More generally, if $\Gamma \vdash_{\mathtt{cast}} t : A$ *then* $\wr \Gamma \int \vdash_{\mathrm{IR}} \wr t \int : \wr A \int\ [t]\ \{t\}$.

In particular CastCIC terms of ground types behave similarly in both models.

Proof. Expanding the type of $\mathtt{cast}_{\mathrm{rel}}$, we need to provide a term

$$c_{\mathrm{rel}} = \mathtt{cast}_{\mathrm{rel}}\ A\,A'\,A_{\mathrm{rel}}\,B\,B'\,B_{\mathrm{rel}}\,a\,a'\,a_{\mathrm{rel}} : \mathrm{El}_{\mathrm{rel}}\,B_{\mathrm{rel}}\ ([\mathtt{cast}]\,A\,B\,a)\ (\{\mathtt{cast}\}\ A'\,B'\,a')$$

where

$$\begin{array}{lll} A : [\![\square_i]\!], & A' : \{\!| \square_i |\!\}, & A_{\mathrm{rel}} : \square_{\mathrm{rel}}\,A\,A', \\ B : [\![\square_i]\!], & B' : \{\!| \square_i |\!\}, & B_{\mathrm{rel}} : \square_{\mathrm{rel}}\,B\,B', \\ a : \mathrm{El}\,A, & a' : \mathrm{El}\,A', & a_{\mathrm{rel}} : \mathrm{El}_{\mathrm{rel}}\,A_{\mathrm{rel}}\ a\ a' \end{array}$$

We proceed by induction on A_{rel}, B_{rel}, following the defining cases for $[\mathtt{cast}]$ (see Fig. 14).

Case $A_{\mathrm{rel}} = \widehat{\Pi}_{\mathrm{rel}}\,A^{\mathbf{d}}_{\mathrm{rel}}\,A^{\mathbf{c}}_{\mathrm{rel}}$ and $B_{\mathrm{rel}} = \widehat{\Pi}_{\mathrm{rel}}\,B^{\mathbf{d}}_{\mathrm{rel}}\,B^{\mathbf{c}}_{\mathrm{rel}}$: we pose $A' = \widehat{\Pi}\,A'^{\mathrm{d}}\,A'^{\mathrm{c}}$ and $B' = \widehat{\Pi}\,B'^{\mathrm{d}}\,B'^{\mathrm{c}}$

$$\begin{aligned} \{\mathtt{cast}\}\ A'\ B'\ f' &= \downarrow^{\widehat{?}}_{B'}\ (\uparrow^{\widehat{?}}_{A'}\ f') && \text{(by definition of } \{\mathtt{cast}\}) \\ &= \downarrow^{\widehat{?}\to\widehat{?}}_{B'} \circ \downarrow^{\widehat{?}}_{\widehat{?}\to\widehat{?}} \circ \uparrow^{\widehat{?}}_{\widehat{?}\to\widehat{?}} \circ \uparrow^{\widehat{?}\to\widehat{?}}_{A'}\ (f) && \text{(by decomposition of } \widehat{\Pi} \sqsubseteq \widehat{?}) \\ &= \downarrow^{\widehat{?}\to\widehat{?}}_{B'} \circ \uparrow^{\widehat{?}\to\widehat{?}}_{A'}\ (f) && \text{(by section-retraction identity)} \\ &= \lambda(b' : \mathrm{El}\,A'^{\mathrm{d}}).\ \mathrm{let}\ a' = \downarrow^{\widehat{?}}_{B'^{\mathrm{d}}} \circ \uparrow^{\widehat{?}}_{A'^{\mathrm{d}}}\ (b)\ \mathrm{in} && \text{(by def. of ep-pair on } \Pi) \\ &\qquad \downarrow^{\widehat{?}}_{B'^{\mathrm{c}}\,b'} \circ \uparrow^{\widehat{?}}_{A'^{\mathrm{c}}\,a'}\ (f\,a') \\ &= \lambda(b' : \mathrm{El}\,A'^{\mathrm{d}}).\ \mathrm{let}\ a' = \{\mathtt{cast}\}\ B'^{\mathrm{d}}\ A'^{\mathrm{d}}\ b'\ \mathrm{in} && \text{(by definition of } \{\mathtt{cast}\}) \\ &\qquad \{\mathtt{cast}\}\ (A'^{\mathrm{c}}\,a')\ (B'^{\mathrm{c}}\,b')\ (f\,a') \end{aligned}$$

[25] For instance the function swapping $\mathtt{err}_{\tilde{\mathbb{N}}}$ and $?_{\tilde{\mathbb{N}}}$ is not monotone.

Translation of contexts

$$\wr \cdot \wr \quad := \quad \cdot \qquad \wr \Gamma, x : A \wr \quad := \quad \wr \Gamma \wr, x_{\text{dis}} : [\![A]\!], x_{\text{mon}} : \{\!|A|\!\}, x_{\text{rel}} : \wr A \wr\, x_{\text{dis}}\, x_{\text{mon}}$$

Logical relation on terms and types

$$\begin{array}{lcl}
\wr A \wr & := & \mathrm{El}_{\text{rel}}\ \wr A \wr \\
\wr x \wr & := & x_{\text{rel}} \\
\wr \square_i \wr & := & \widehat{\square}_{\text{rel},i} \\
\wr t\ u \wr & := & \wr t \wr\ [u]\ \{u\}\ \wr u \wr \\
\wr \lambda x : A.t \wr & := & \lambda (x_{\text{dis}} : [\![A]\!])(x_{\text{mon}} : \{\!|A|\!\})(x_{\text{rel}} : \wr A \wr\, x_{\text{dis}}\, x_{\text{mon}}).\ \wr t \wr \\
\wr \Pi x : A.B \wr & := & \widehat{\Pi}_{\text{rel}}\ \wr A \wr\ (\lambda (x_{\text{dis}} : [\![A]\!])(x_{\text{mon}} : \{\!|A|\!\})(x_{\text{rel}} : \wr A \wr\, x_{\text{dis}}\, x_{\text{mon}}). \wr B \wr) \\
\wr \mathbb{N} \wr & := & \widehat{\mathbb{N}}_{\text{rel}} \\
\wr ?_A \wr & := & ?_{\wr A \wr} : \wr A \wr\, ?_{[\![A]\!]}\ ?_{\{A\}} \\
\wr \mathtt{err}_A \wr & := & \mathtt{err}_{\wr A \wr} : \wr A \wr\ \mathtt{err}_{[\![A]\!]}\ \mathtt{err}_{\{A\}} \\
\wr \mathtt{cast} \wr & := & \mathtt{cast}_{\text{rel}}
\end{array}$$

Inductive-recursive relational universe $\square_{\text{rel}} : \square^{\text{dis}} \to \square^{\text{mon}} \to \square$

$$\frac{A_{\text{rel}} \in \square_{\text{rel},i}\ A\ A' \qquad B \in \Pi(a : A)(a' : A').\mathrm{El}_{\text{rel}}\ A_{\text{rel}}\ a\ a' \to \square_{\text{rel},j}\ (B\ a)\ (B'\ a')}{\widehat{\Pi}_{\text{rel}}\ A_{\text{rel}}\ B_{\text{rel}} \in \square_{\text{rel},s_\Pi(i,j)}\ (\widehat{\Pi}\ A\ B)\ (\widehat{\Pi}\ A'\ B')}$$

$$\frac{j < i}{\widehat{\square}_{\text{rel},j} \in \square_{\text{rel},i}\ \widehat{\square}_j\ \widehat{\square}_j} \qquad \widehat{\mathbb{N}}_{\text{rel}} \in \square_{\text{rel},i}\ \widehat{\mathbb{N}}\ \widehat{\mathbb{N}} \qquad \widehat{?}_{\text{rel}} \in \square_{\text{rel},i}\ \widehat{?}\ \widehat{?} \qquad \maltese_{\text{rel}} \in \square_{\text{rel},i}\ \maltese\ \maltese$$

Decoding function $\mathrm{El}_{\text{rel}} : \square_{\text{rel}}\ A\ A' \to \mathrm{El}\ A \to \mathrm{El}\ A' \to \square$

$$\begin{array}{c}
\mathrm{El}_{\text{rel}}\ \widehat{\square}_{\text{rel},j}\ A\ A' := \square_{\text{rel},j}\ A\ A' \\
\mathrm{El}_{\text{rel}}\ \widehat{\mathbb{N}}_{\text{rel}}\ n\ m := n = m \\
\mathrm{El}_{\text{rel}}\ \maltese_{\text{rel}}\ ()\ () := \mathtt{unit} \\
\mathrm{El}_{\text{rel}}\ \widehat{?}_{\text{rel}}\ (c; x)\ y := \mathrm{El}_{\text{rel}}\ (\mathrm{germ}_{\text{rel}}\ c)\ x\ (\mathtt{downcast}_{\widehat{?},\mathrm{germ}\ c}\ y) \\
\mathrm{El}_{\text{rel}}\ (\widehat{\Pi}_{\text{rel}}\ A_{\text{rel}}\ B_{\text{rel}})\ f\ f' := \Pi(a : \mathrm{El}\ A)(a' : \mathrm{El}\ A')(a_{\text{rel}} : \mathrm{El}_{\text{rel}}\ A_{\text{rel}}\ a\ a'). \\
\qquad\qquad \mathrm{El}_{\text{rel}}\ (B_{\text{rel}}\ a\ a'\ a_{\text{rel}})\ (f\ a)\ (f'\ a')
\end{array}$$

Fig. 20. Logical relation between the discrete and monotone models

For any $b : \mathrm{El}\, B^{\mathrm{d}}$ and $b' : \mathrm{El}\, B'^{\mathrm{d}}$, $b_{\text{rel}} : \mathrm{El}_{\text{rel}}\, B^{\mathrm{d}}_{\text{rel}}\, b\, b'$, we have by inductive hypothesis

$$a_{\text{rel}} := \wr \mathtt{cast} \wr\ B^{\mathrm{d}}_{\text{rel}}\ A^{\mathrm{d}}_{\text{rel}}\ b_{\text{rel}} : \mathrm{El}_{\text{rel}}\ A_{\text{rel}}\ ([\mathtt{cast}]\ B^{\mathrm{d}}\ A^{\mathrm{d}}\ b)\ (\{\mathtt{cast}\}\ B'^{\mathrm{d}}\ A'^{\mathrm{d}}\ b')$$

so that, posing $a = [\mathtt{cast}]\ B^{\mathrm{d}}\ A^{\mathrm{d}}\ b$ and $a' = \{\mathtt{cast}\}\ B'^{\mathrm{d}}\ A'^{\mathrm{d}}\ b'$,

$$f_{\text{rel}}\ a\ a'\ a_{\text{rel}} : \mathrm{El}_{\text{rel}}\ (A^{\mathrm{c}}_{\text{rel}}\ a\ a'\ a_{\text{rel}})\ (f\ a)\ (f'\ a')$$

and by another application of the inductive hypothesis

$\wr\mathtt{cast}\int\,(B^{\mathrm{c}}_{\mathrm{rel}}\,b\,b'\,b_{\mathrm{rel}})\,(A^{\mathrm{c}}_{\mathrm{rel}}\,a\,a'\,a_{\mathrm{rel}})\,(f_{\mathrm{rel}}\,a\,a'\,a_{\mathrm{rel}}) : \wr B^{\mathrm{c}}_{\mathrm{rel}}\,b\,b'\,b_{\mathrm{rel}}\int\,([\mathtt{cast}]\,A\,B\,f\,a)\,(\{\mathtt{cast}\}\,A'\,B'\,f'\,a')$

Packing these together, we obtain a term

$$\wr\mathtt{cast}\int\ A_{\mathrm{rel}}\,B_{\mathrm{rel}}\,f_{\mathrm{rel}} : \mathrm{El}_{\mathrm{rel}}\,(\widehat{\Pi}\,B^{\mathrm{d}}_{\mathrm{rel}}\,B^{\mathrm{c}}_{\mathrm{rel}})\,([\mathtt{cast}]\,A\,B\,f)\,(\{\mathtt{cast}\}\,A'\,B'\,f').$$

Case $A_{\mathrm{rel}} = \widehat{\Pi}_{\mathrm{rel}}\,A^{\mathrm{d}}_{\mathrm{rel}}\,A^{\mathrm{c}}_{\mathrm{rel}}$ and $B_{\mathrm{rel}} = \widehat{?}_{\mathrm{rel}}$: By definition of the logical relation at $\widehat{?}_{\mathrm{rel}}$, we need to build a witness of type

$$\mathrm{El}_{\mathrm{rel}}\,(\widehat{?}^{c_\Pi(i)} \to \widehat{?}^{c_\Pi(i)})\,([\mathtt{cast}]\,A\,(\widehat{?}\to\widehat{?})\,f)\,(\downarrow^{\widehat{?}}_{\widehat{?}\to\widehat{?}}\,(\{\mathtt{cast}\}\,A'\,\widehat{?}\,f'))$$

We compute that

$$\downarrow^{\widehat{?}}_{\widehat{?}\to\widehat{?}}\,(\{\mathtt{cast}\}\,A'\,\widehat{?}\,f') = \downarrow^{\widehat{?}}_{\widehat{?}\to\widehat{?}} \circ \downarrow^{\widehat{?}}_{\widehat{?}} \circ \uparrow^{\widehat{?}}_{A'}\,f' = \downarrow^{\widehat{?}}_{\widehat{?}\to\widehat{?}} \circ \uparrow^{\widehat{?}}_{A'}\,f' = \{\mathtt{cast}\}\,A'\,(\widehat{?}\to\widehat{?})\,f'$$

So the result holds by induction hypothesis.

Other cases with $A_{\mathrm{rel}} = \widehat{\Pi}_{\mathrm{rel}}\,A^{\mathrm{d}}_{\mathrm{rel}}\,A^{\mathrm{c}}_{\mathrm{rel}}$: It is enough to show that $\{\mathtt{cast}\}\,A'\,B'\,f' = \maltese_{B'}$ when $B' = \maltese$ (trivial) or head $B' \neq \mathtt{pi}$. The latter case holds because $\downarrow^{\widehat{?}}_{\mathrm{germ}\ c}\uparrow^{\widehat{?}}_{\mathrm{germ}\ c'}\,x = \maltese_{\mathrm{El}_{\mathrm{Head}}\,c}$ whenever $c \neq c'$ and downcasts preserve $\maltese$.

Case $A_{\mathrm{rel}} = \widehat{?}_{\mathrm{rel}}$, $B_{\mathrm{rel}} = \widehat{\Pi}_{\mathrm{rel}}\,B^{\mathrm{d}}_{\mathrm{rel}}\,B^{\mathrm{c}}_{\mathrm{rel}}$ and $a = (\mathtt{pi}; f)$: By hypothesis, $a_{\mathrm{rel}} : \mathrm{El}_{\mathrm{rel}}\,(\widehat{?}\to\widehat{?})\,f\,(\downarrow^{\widehat{?}}_{\widehat{?}\to\widehat{?}}\,a')$ and $\{\mathtt{cast}\}\,\widehat{?}\,B'\,a' = \{\mathtt{cast}\}\,(\widehat{?}\to\widehat{?})\,B'\,(\downarrow^{\widehat{?}}_{\widehat{?}\to\widehat{?}}\,a')$ so by induction hypothesis

$$\wr\mathtt{cast}\int\ (\widehat{?}_{\mathrm{rel}} \to_{\mathrm{rel}} \widehat{?}_{\mathrm{rel}})\,B_{\mathrm{rel}}\,f\,(\downarrow^{\widehat{?}}_{\widehat{?}\to\widehat{?}}\,a')\,a_{\mathrm{rel}} : \mathrm{El}_{\mathrm{rel}}\,B_{\mathrm{rel}}\,([\mathtt{cast}]\,\widehat{?}\,B\,(\mathtt{pi}; f))\,(\{\mathtt{cast}\}\,\widehat{?}\,B'\,a')$$

The others cases with $A_{\mathrm{rel}} = \widehat{?}_{\mathrm{rel}}$ proceed in a similarly fashion. All cases with $A_{\mathrm{rel}} = \maltese_{\mathrm{rel}}$ are immediate since $\maltese^{\mathrm{dis}}$ and $\maltese^{\mathrm{mon}}$ are related at any related types. Finally, the cases with $A_{\mathrm{rel}} = \widehat{\mathbb{N}}_{\mathrm{rel}}$ follow the same pattern as for $\widehat{\Pi}_{\mathrm{rel}}$. □

D DIVERGING TERMS DENOTE AS ERRORS IN ω-CPOS

In this section we define a logical relation between $\mathsf{CastCIC}^{\mathcal{G}}$ and $\mathsf{CIC}^{\mathsf{IR}}_{\mathsf{QIT}}$ and prove a fundamental lemma, obtaining Lemma 33 as a corollary. The logical relation is presented in Figs. 21 to 23 and relates types A in $\mathsf{CastCIC}^{\mathcal{G}}$ with sub-ω-cpos of $\ddot{?}$, following the description of El in that model. A type A related to an ω-cpo A' by the logical relation, noted $A \sim A'$, induces a relation between terms of type A and elements of A'. We use variables with ε subscript to name proof witnesses of relatedness between two objects, for instance $A_\varepsilon : A \sim A'$, and bold variables such as $\mathbb{\Gamma}, \mathbb{\Delta}$ for the corresponding double contexts consisting of variable bindings $a \sim a' : A_\varepsilon$. The projections $\mathbb{\Gamma}_1$ and $\mathbb{\Gamma}_2$ are then respectively contexts in $\mathsf{CastCIC}^{\mathcal{G}}$ and $\mathsf{CIC}^{\mathsf{IR}}_{\mathsf{QIT}}$.

The logical relation uses weak head reduction to characterize divergence. We note $t \twoheadrightarrow_{\mathrm{wh}} u$ when a CastCIC term t reduces to a weak head normal form, that is a term u such that canonical u hold (see Fig. 7), using only weak head reduction steps. We note $t \not\twoheadrightarrow_{\mathrm{wh}}$ when weak head reduction paths from t never reach a weak head normal form, that is t is unsolvable.

We first state a lemma making explicit how divergence is accounted for by the logical relation.

Lemma 41 (Diverging terms relate to errors).

(1) If $\mathbb{\Gamma}_1 \vdash t : T$, $T_\varepsilon : \mathbb{\Gamma} \Vdash T \sim T'$ and $t \not\twoheadrightarrow_{\mathrm{wh}}$ then $\mathbb{\Gamma} \Vdash t \sim \mathtt{err}_{T'} : T_\varepsilon$.

(2) Conversely, if $\Gamma \vdash t : T$, $T_\varepsilon : \mathbb{\Gamma} \Vdash T \sim T'$, $\mathbb{\Gamma} \Vdash t \sim t' : T_\varepsilon$ and $t \not\twoheadrightarrow_{\mathrm{wh}}$ then $t' = \mathtt{err}_{T'}$.

Proof. In the two parts of the lemma, we proceed by induction on T_ε. For the first part, the cases $T_\varepsilon = \square_\varepsilon, \mathbb{N}_\varepsilon, \mathbb{B}_\varepsilon$ and $?_\varepsilon$ are immediate because in each case a rule apply for diverging terms. If $T_\varepsilon = \mathtt{err}_\varepsilon$, then $\mathbb{\Gamma} \Vdash t \sim () : T_\varepsilon$ which is enough because $() = \mathtt{err}_{\mathtt{unit}} = \mathtt{err}_{\mathrm{El}\,\widehat{\mathtt{err}}} = \mathtt{err}_{\mathrm{El}\,T'}$. Finally, if $T_\varepsilon = \Pi_\varepsilon\, A_\varepsilon\, B_\varepsilon$, then for any $\rho : \mathbb{\Delta} \subset \mathbb{\Gamma}$ and $a_\varepsilon : \mathbb{\Delta} \Vdash a \sim a' : A_\varepsilon\, \rho$ we have that $\mathbb{\Gamma}_1 \vdash t[\rho_1]\ a : B[\rho_1, a]$, $B_\varepsilon\, \rho\, a_\varepsilon : \mathbb{\Delta} \Vdash B[\rho_1, a] \sim B'[\rho_2, a']$ and $t\ a \not\twoheadrightarrow_{\mathrm{wh}}$, so by induction hypothesis $\mathbb{\Delta} \Vdash t[\rho_1]\ a \sim \mathtt{err}_{B'[\rho_2,a']} : B_\varepsilon\, \rho\, a_\varepsilon$, hence $\mathbb{\Gamma} \Vdash t \sim \mathtt{err}_{\widehat{\Pi}\,A'\,B'} : T_\varepsilon$.

We now turn to the second part of the lemma. When $T_\varepsilon = \square_\varepsilon, \mathbb{N}_\varepsilon, \mathbb{B}_\varepsilon, ?_\varepsilon$ and $\mathtt{err}_\varepsilon$, there is exactly one rule that apply to relate to a term without weak head normal form t so that necessarily $t' = \mathtt{err}_{T'}$. When $T_\varepsilon = \Pi_\varepsilon\, A_\varepsilon\, B_\varepsilon$, any $\rho : \mathbb{\Delta} \subset \mathbb{\Gamma}$ and $a_\varepsilon : \mathbb{\Delta} \Vdash a \sim a' : A_\varepsilon\, \rho$ we have that $B_\varepsilon\, \rho\, a_\varepsilon : \mathbb{\Delta} \Vdash B[\rho_1, a] \sim B'[\rho_2, a']$, $\mathbb{\Delta} \Vdash t[\rho_1]\ a \sim t'[\rho_2]\ a' : B_\varepsilon\, \rho\, a_\varepsilon$ and $t\ a \not\twoheadrightarrow_{\mathrm{wh}}$, so by induction hypothesis $t'\ a' = \mathtt{err}_{B'[\rho_2,a']}$. Taking ρ to be the weakening $\mathbb{\Gamma}, a \sim a' : A_\varepsilon \subset \mathbb{\Gamma}$, we have by function extensionality that $t' = \lambda(a' : A').\, \mathtt{err}_{B'} = \mathtt{err}_{\widehat{\Pi}\,A'\,B'}$. ◻

Lemma 42 (Fundamental lemma).

- *If $\Gamma \vdash$ then there exists $\mathbb{\Gamma}$ such that $\mathbb{\Gamma} \Vdash$, $\mathbb{\Gamma}_1 = \Gamma$ and $\mathbb{\Gamma}_2 = \{\!|\, \Gamma \,|\!\}$;*
- *If $\Gamma \vdash T \triangleright \square_i$ there exists a derivation $T_\varepsilon : \mathbb{\Gamma} \Vdash T \sim \{\!|\, T \,|\!\}$*
- *If $\Gamma \vdash t \triangleright T$ then there exists a derivation $t_\varepsilon : \mathbb{\Gamma} \Vdash t \sim \{t\} : T_\varepsilon$*
- *If $\Gamma \vdash T \triangleright \square_i$, $\Gamma \vdash T' \triangleright \square_i$ and $T \equiv T'$ then $\{\!|\, T \,|\!\} = \{\!|\, T' \,|\!\}$ and $T_\varepsilon = T'_\varepsilon : \mathbb{\Gamma} \Vdash T \sim \{\!|\, T \,|\!\}$.*

Proof. Since the translation $\{\!|\, - \,|\!\}$ underly a model of $\mathrm{CastCIC}^{\mathcal{N}}$, it sends convertible types T, T' in the source to provably equal types in the target $\{\!|\, T \,|\!\} = \{\!|\, T' \,|\!\}$, proving the last claim.

The three other claims are proved by mutual induction on the input derivation, assuming an undirected variant of the rules in Figs. 1 and 3, which is possible by [Lennon-Bertrand 2021]. Concretely, this modification means that we assume additional well-formedness premises in the derivations, *e.g.*, for contexts and types, and do not show that input well-formedness is preserved. Moreover the induction hypothesis needs to be strenghened to quantify over an arbitrary context $\mathbb{\Delta}$ with a substition $\sigma : \mathbb{\Delta} \to \mathbb{\Gamma}$ whose components are related according to the logical relation.

For contexts, if the derivation ends with a rule Empty, it is enough take $\mathbb{\Gamma} = \cdot$. If it ends with Concat, then by induction hypothesis there exists $\mathbb{\Gamma}$ and A_ε such that $\mathbb{\Gamma}_1 = \Gamma$, $\mathbb{\Gamma}_2 = \{\!|\, \Gamma \,|\!\}$, $\mathbb{\Gamma} \Vdash$ and $A_\varepsilon : \mathbb{\Gamma} \Vdash A \sim \{\!|\, A \,|\!\}$, so taking $\mathbb{\Gamma}, a \sim a' : A_\varepsilon$ suffices.

For Univ by induction hypothesis $\mathbb{\Gamma} \Vdash$ with $\mathbb{\Gamma}_1 = \Gamma$. Moreover, $\Gamma \vdash \square_i \triangleright \square_{i+1}$ and $\{\square_i\} = \widehat{\square}_i$ so $\mathbb{\Gamma} \Vdash \square_i \sim \{\square_i\} : \square_\varepsilon\,(i+1)$ and $\mathbb{\Gamma} \Vdash \square_i \sim \{\square_i\}$. The rules Ind (for $\mathbb{N}, \mathbb{B}$) and Cons (for $0, \mathtt{suc}, \mathtt{true}, \mathtt{false}$), introducing types and terms that are already in weak head normal form follow the same pattern as Univ. In the case of the rules Prod and Abs, the context needs to be extended and we need to take advantage of the full induction hypothesis strenghened under arbitrary reducible substitutions.

Dually, the rule App is immediate by induction hypothesis and the definition the logical relation at function types. A bit more work is needed for the rule Fix for $\mathtt{ind}_{\mathbb{B}}$ and $\mathtt{ind}_{\mathbb{N}}$, doing a case analysis on the proof of relatedness of their main argument. If the main argument diverges, then the applied eliminator diverges too so it is related to $\mathtt{err}$ which is its translation because eliminators send errors at an inductive type to errors at the adequate type in ω-cpos. Otherwise the main argument weak head reduces to a normal form and we can conclude by induction hypothesis and closure by anti-reduction.

For the variable case, rule Var, we can show by induction on the proof of relatedness of its type that it is related to its η-expansion at Π types an to itself at any other type using the rules for neutrals. We conclude by extensionality of the ω-cpo model.

Conversion rules Check, Prod-Inf, Ind-Inf and Univ-Inf satisfy the fundamental lemma because convertible types induce the same relation on their term.

For Err, we have by induction hypothesis that $T_\varepsilon : \mathbb{\Gamma} \Vdash T \sim \{T\} : \square_\varepsilon$. By case analysis, T_ε is necessarily one of $\mathtt{err}_\varepsilon, \Pi_\varepsilon, ?_\varepsilon, \mathbb{B}_\varepsilon, \mathbb{N}_\varepsilon$ or $\square_\varepsilon$. If $T_\varepsilon = \mathtt{err}_\varepsilon\ T$ then $\mathbb{\Gamma} \Vdash \mathtt{err}_T \sim () : \mathtt{err}_\varepsilon\ T$ since $\mathbb{\Gamma}_1 \vdash \mathtt{err}_T \triangleright T$, and we can conclude using extensionality of $\mathtt{unit} = \{\!| T |\!\}$, that is $\{\mathtt{err}_T\} = ()$. If $T_\varepsilon = \Pi_\varepsilon\ A_\varepsilon\ B_\varepsilon$, then for any $\rho : \mathbb{\Delta} \subset \mathbb{\Gamma}$ and $a_\varepsilon : \mathbb{\Delta} \Vdash a \sim a' : A_\varepsilon\ \rho$ we have that $T[\rho_1] \twoheadrightarrow_{\mathrm{wh}} \Pi(a : A)B$ and $\mathtt{err}_T\ \ a \twoheadrightarrow_{\mathrm{wh}} \mathtt{err}_{B[a]}$, so we conclude this case by induction hypothesis $\mathbb{\Delta} \Vdash \mathtt{err}_{B[a]} \sim \{\mathtt{err}_{B[a]}\} : B_\varepsilon\ a_\varepsilon$, closure by anti-reduction and the fact that $\{\mathtt{err}_{B[a]}\} = \mathtt{err}_{\{B[a]\}} = \mathtt{err}_{\{B\}\,[a']}$.

In all the other cases T weak head reduces to a type in weak head normal form $?_\square, \square_i, \mathbb{N}$ or $\mathbb{B}$, and a corresponding rule is present in the logical relation to conclude directly. A similar proof apply for the rule Unk.

Finally, for the rule Cast with conclusion $\Gamma \vdash \langle B \Leftarrow A\rangle\, t \triangleright B$, we have by induction hypothesis we have that $\mathbb{\Gamma} \Vdash, A_\varepsilon : \mathbb{\Gamma} \Vdash A \sim \{\!| A |\!\}, B_\varepsilon : \mathbb{\Gamma} \Vdash B \sim \{\!| B |\!\}$ and $\mathbb{\Gamma} \Vdash t \sim \{t\} : A_\varepsilon$. By analysing all possible weak head reduction paths from $\langle B \Leftarrow A\rangle\, t$, either:

(a) $\langle B \Leftarrow A\rangle\, t \twoheadrightarrow_{\mathrm{wh}} u$ such that $\mathbb{\Gamma} \Vdash u \sim \{u\} : B_\varepsilon$ using inversions on $A_\varepsilon, B_\varepsilon$ and t_ε, or
(b) one of A, B or t never reduces to a weak head normal form.

In case **(a)**, we conclude that $\mathbb{\Gamma} \Vdash \langle B \Leftarrow A\rangle\, t \sim \{\langle B \Leftarrow A\rangle\, t\} : B_\varepsilon$ by closure under anti-reduction and using the fact that $\{\langle B \Leftarrow A\rangle\, t\} = \{u\}$ (because $\{-\}$ maps convertible terms to equal terms in the model). In case **(b)**, we have that $\mathbb{\Gamma} \Vdash \langle B \Leftarrow A\rangle\, t \sim \mathtt{err}_{\{B\}} : B_\varepsilon$ by the first part of Lemma 41 and the second part of that lemma ensures that one of $\{A\}$, $\{B\}$ or $\{t\}$ is an error at the adequate type so that $\{\langle B \Leftarrow A\rangle\, t\} = \downarrow^{\widehat{?}}_{\{B\}} \uparrow^{\widehat{?}}_{\{A\}}\ \{t\} = \mathtt{err}_{\{B\}}$. □

Corollary 43. *If* $\Gamma \vdash t \triangleright T$ *and* $t \not\twoheadrightarrow_{\mathrm{wh}}$ *then* $\{t\} = \mathtt{err}_{\{T\}}$.

Proof. By the fundamental lemma, $\mathbb{\Gamma} \Vdash t \sim \{t\} : T_\varepsilon$ with $T_\varepsilon : \mathbb{\Gamma} \Vdash T \sim \{T\} : \square_\varepsilon$ and by the second part of Lemma 41, $\{t\} = \mathtt{err}_{\{T\}}$. □

E A DIRECT PRESENTATION OF VECTORS

Vectors have two new normal forms, corresponding to cast of `nil` and `cons` to $\mathtt{vec}\ A\ ?_{\mathbb{N}}$. The difference with the treatment of the universe is that the corresponding term, for instance $\langle \mathtt{vec}\ A\ ?_{\mathbb{N}} \Leftarrow \mathtt{vec}\ A\ n\rangle\, t$ for the case of `nil`, can not be considered as canonical form because they involve a non-linear occurrence of A. To remedy to this issue, we add two new canonical forms ($\mathtt{nil}_?\ A$ and $\mathtt{cons}_?\ A\ a\ n\ v$) to vectors with introduction typing rules defined in Appendix E.

Regarding cast on vectors, it does not only compute in the argument of the cast as it is the case for inductive types without indices, but it also computes on the indices. That is, a cast on vectors is neutral when either one of the indices is neutral or the argument is neutral (see Appendix E). Other kind of neutral can be derived from the one of inductive types without indices and are omitted here.

Similarly, we do not detail the other typing rules for vectors as they are similar to the one for inductive types wihtout indices, and focus on explaining the new reduction rules, presented also in Appendix E.

The two first reduction rules v-rect-nil and v-rect-cons are standard reduction rules in CIC for the recursor `vect_rect` on vectors. The rules v-rect-err and v-rect-unk are the standard rules dealing with exceptions. Additionally, there are two computation rules for the eliminator on the two new constructors v-rect-nilu and v-rect-consu which basically consist in the underlying non-exceptional constructor to the eliminator and cast the result back to $P\ ?_{\mathbb{N}}$. This rule somehow transfers the cast on vectors to a cast on the returned type of the predicate.

Finally, there are rules to conduct casts between vectors in canonical forms. The last three rules (V-unk, V-err and V-to-err) are simply propogation of errors. Then, there remains 12 rules, 3 by constructors of vectors. We just explain the one on cons. Rule V-cons applies when both indices of the form S of somthing and progates the cast of the arguments, as does the standard rule for casting a constructor. Rule V-cons-nil detects that the indices do not match and raise an error. Finally, Rule V-cons-? propagates the cast on the arguments, but this time applied to $\mathtt{cons}_?$, thus converting precise information to a less precise information.

$\boxed{\Gamma \Vdash}$ logical relation between CastCIC$^{\mathcal{N}}$ contexts and ω-cpos.

$$\cdot \Vdash \qquad \frac{\Gamma \Vdash \qquad A_\varepsilon : \Gamma \Vdash A \sim A'}{\Gamma, a \sim a' : A_\varepsilon \Vdash} \qquad \begin{array}{l} \cdot_1 = \cdot \\ (\Gamma, a \sim a' : A_\varepsilon)_1 = \Gamma_1, a : A \end{array}$$

$\boxed{\Gamma \Vdash A \sim B}$ logical relation between CastCIC$^{\mathcal{G}}$ types A and ω-cpos $B \subset \ddot{?}$.

$$\frac{\Gamma \Vdash T \sim U : \square_\varepsilon}{\Gamma \Vdash T \sim \mathrm{El}\, U}$$

$\boxed{\Gamma \Vdash t \sim u : \mathbb{B}_\varepsilon}$ logical relation between CastCIC$^{\mathcal{G}}$ terms of type $\mathbb{B}$ and elements of $\ddot{\mathbb{B}}$.

$$\frac{\Gamma_1 \vdash t \triangleright \mathbb{B} \qquad t \twoheadrightarrow_{\mathrm{wh}} \mathtt{true} \qquad \Gamma \Vdash}{\Gamma \Vdash t \sim \mathtt{true} : \mathbb{B}_\varepsilon} \qquad \frac{\Gamma_1 \vdash t \triangleright \mathbb{B} \qquad t \twoheadrightarrow_{\mathrm{wh}} \mathtt{false} \qquad \Gamma \Vdash}{\Gamma \Vdash t \sim \mathtt{false} : \mathbb{B}_\varepsilon}$$

$$\frac{\Gamma_1 \vdash t \triangleright \mathbb{B} \qquad t \twoheadrightarrow_{\mathrm{wh}} ?_{\mathbb{B}} \qquad \Gamma \Vdash}{\Gamma \Vdash t \sim \top_{\ddot{\mathbb{B}}} : \mathbb{B}_\varepsilon} \qquad \frac{\Gamma_1 \vdash t \triangleright \mathbb{B} \qquad t \twoheadrightarrow_{\mathrm{wh}} \mathtt{err}_{\mathbb{B}} \lor t \not\twoheadrightarrow_{\mathrm{wh}} \qquad \Gamma \Vdash}{\Gamma \Vdash t \sim \bot_{\ddot{\mathbb{B}}} : \mathbb{B}_\varepsilon}$$

$$\frac{\Gamma_1 \vdash t \triangleright \mathbb{B} \qquad t \twoheadrightarrow_{\mathrm{wh}} t' \qquad \Gamma \Vdash t' \sim_{\mathrm{ne}} u : \mathbb{B}_\varepsilon}{\Gamma \Vdash t \sim u : \mathbb{B}_\varepsilon}$$

$\boxed{\Gamma \Vdash t \sim u : \mathbb{N}_\varepsilon}$ logical relation between CastCIC$^{\mathcal{G}}$ terms of type $\mathbb{N}$ and elements of $\ddot{\mathbb{N}}$.

$$\frac{\Gamma_1 \vdash t \triangleright \mathbb{N} \qquad t \twoheadrightarrow_{\mathrm{wh}} 0 \qquad \Gamma \Vdash}{\Gamma \Vdash t \sim 0 : \mathbb{N}_\varepsilon} \qquad \frac{\Gamma_1 \vdash t \triangleright \mathbb{N} \qquad t \twoheadrightarrow_{\mathrm{wh}} \mathtt{suc}\, t' \qquad \Gamma \Vdash t' \sim u' : \mathbb{N}_\varepsilon}{\Gamma \Vdash t \sim \mathtt{suc}\, u' : \mathbb{N}_\varepsilon}$$

$$\frac{\Gamma_1 \vdash t \triangleright \mathbb{N} \qquad t \twoheadrightarrow_{\mathrm{wh}} ?_{\mathbb{N}} \qquad \Gamma \Vdash}{\Gamma \Vdash t \sim \top_{\ddot{\mathbb{N}}} : \mathbb{N}_\varepsilon} \qquad \frac{\Gamma_1 \vdash t \triangleright \mathbb{N} \qquad t \twoheadrightarrow_{\mathrm{wh}} \mathtt{err}_{\mathbb{N}} \lor t \not\twoheadrightarrow_{\mathrm{wh}} \qquad \Gamma \Vdash}{\Gamma \Vdash t \sim \bot_{\ddot{\mathbb{N}}} : \mathbb{N}_\varepsilon}$$

$$\frac{\Gamma_1 \vdash t \triangleright \mathbb{N} \qquad t \twoheadrightarrow_{\mathrm{wh}} t' \qquad \Gamma \Vdash t' \sim_{\mathrm{ne}} u : \mathbb{N}_\varepsilon}{\Gamma \Vdash t \sim u : \mathbb{N}_\varepsilon}$$

$\boxed{\Gamma \Vdash t \sim u : ?_\varepsilon\, i}$ logical relation between CastCIC$^{\mathcal{G}}$ terms of type $?_{\square_i}$ and elements of $\ddot{?}_i$.

$$\frac{\Gamma_1 \vdash t \triangleright ?_{\square} \qquad t \twoheadrightarrow_{\mathrm{wh}} ?_{?_{\square_i}} \qquad \Gamma \Vdash}{\Gamma \Vdash t \sim \top_{\ddot{?}_i} : ?_\varepsilon\, i} \qquad \frac{\Gamma_1 \vdash t \triangleright ?_{\square} \qquad t \twoheadrightarrow_{\mathrm{wh}} \mathtt{err}_{?_{\square_i}} \lor t \not\twoheadrightarrow_{\mathrm{wh}} \qquad \Gamma \Vdash}{\Gamma \Vdash t \sim \bot_{\ddot{?}_i} : ?_\varepsilon\, i}$$

$$\frac{\Gamma_1 \vdash t \triangleright ?_{\square} \qquad t \twoheadrightarrow_{\mathrm{wh}} \langle ? \Leftarrow A \rangle\, t' \qquad \Gamma \Vdash t' \sim u' : A_\varepsilon}{\Gamma \Vdash t \sim \uparrow^{\ddot{?}}_{A'} u' : ?_\varepsilon\, i} \qquad \frac{\Gamma_1 \vdash t \triangleright ?_{\square} \qquad t \twoheadrightarrow_{\mathrm{wh}} t' \qquad \Gamma \Vdash t' \sim_{\mathrm{ne}} u : ?_\varepsilon\, i}{\Gamma \Vdash t \sim u : ?_\varepsilon\, i}$$

Fig. 21. Logical relation between CastCIC$^{\mathcal{G}}$ and ω-cpos

$\boxed{\Gamma \Vdash t \sim u : \mathtt{err}_\varepsilon\ A}$ logical relation between CastCIC$^{\mathcal{G}}$ terms of type A and elements of $\mathtt{unit}$.

$$\frac{\Gamma_1 \vdash t \triangleright A}{\Gamma \Vdash t \sim () : \mathtt{err}_\varepsilon\ A}$$

$\boxed{\Gamma \Vdash t \sim u : \square_\varepsilon\ i}$ logical relation between CastCIC$^{\mathcal{G}}$ terms of type $\square_i$ and elements of $\widehat{\square}_i$.

$$\mathbb{N}_\varepsilon\ \frac{\Gamma_1 \vdash T \triangleright \square_i \qquad T \twoheadrightarrow_{\text{wh}} \mathbb{N} \qquad \Gamma \Vdash}{\Gamma \Vdash T \sim \widehat{\mathbb{N}} : \square_\varepsilon\ i} \qquad ?_\varepsilon\ \frac{\Gamma_1 \vdash T \triangleright \square_i \qquad T \twoheadrightarrow_{\text{wh}} ?_{\square_i} \qquad \Gamma \Vdash}{\Gamma \Vdash T \sim \widehat{?}_i : \square_\varepsilon\ i}$$

$$\mathtt{err}_\varepsilon\ \frac{\Gamma_1 \vdash T \triangleright \square_i \qquad T \twoheadrightarrow_{\text{wh}} \mathtt{err}_{\square_i} \vee T \not\twoheadrightarrow_{\text{wh}} \qquad \Gamma \Vdash}{\Gamma \Vdash T \sim \widehat{\mathtt{err}} : \square_\varepsilon\ i} \qquad \square_\varepsilon\ \frac{\Gamma_1 \vdash T \triangleright \square_i \qquad T \twoheadrightarrow_{\text{wh}} \square_j \qquad \Gamma \Vdash}{\Gamma \Vdash T \sim \widehat{\square}_j : \square_\varepsilon\ i}$$

$$\Pi_\varepsilon\ \frac{\begin{array}{c}\Gamma_1 \vdash T \triangleright \square_i \qquad T \twoheadrightarrow_{\text{wh}} \Pi(a : A)\, B \qquad A_\varepsilon : \forall \rho : \Delta \subseteq \Gamma, \Delta \Vdash A[\rho_1] \sim A'[\rho_2] : \square_\varepsilon\ i \\ \forall \rho : \Delta \subseteq \Gamma, \Delta \Vdash a \sim a' : A_\varepsilon\ \rho \implies \Gamma \Vdash B[a] \sim B'[a'] : \square_\varepsilon\ i\end{array}}{\Gamma \Vdash T \sim \widehat{\Pi}\, A'\, (\lambda(a' : A').B') : \square_\varepsilon\ i}$$

$$\mathtt{ne}_\varepsilon\ \frac{\Gamma_1 \vdash t \triangleright \square_i \qquad t \twoheadrightarrow_{\text{wh}} t' \qquad \Gamma \Vdash t' \sim_{\text{ne}} u : \square_\varepsilon\ i}{\Gamma \Vdash t \sim u : \square_\varepsilon\ i}$$

$\boxed{\Gamma \Vdash t \sim u : \Pi_\varepsilon\ A_\varepsilon\, B_\varepsilon}$ logical relation between CastCIC$^{\mathcal{G}}$ terms of Π type.

$$\frac{\forall (\rho : \Delta \subseteq \Gamma)\ (a_\varepsilon : \Delta \Vdash a \sim a' : A_\varepsilon\ \rho),\ \Delta \Vdash t[\rho_1]\ a \sim u[\rho_2]\ a' : B_\varepsilon\ \rho\ a_\varepsilon}{\Gamma \Vdash t \sim u : \Pi_\varepsilon\ A_\varepsilon\, B_\varepsilon}$$

where

- $\Gamma_1 \vdash t \triangleright \Pi\,(a : A)\, B$,
- $\Gamma_2 \vdash_{\text{IR}} u : \Pi^\omega\,(a' : A')\, B'$,
- $A_\varepsilon : \forall \rho : \Delta \subseteq \Gamma, \Delta \Vdash A[\rho_1] \sim A'[\rho_2] : \square_\varepsilon\ i$
- $B_\varepsilon : \forall \rho : \Delta \subseteq \Gamma, \Delta \Vdash a \sim a' : A_\varepsilon\ \rho \implies \Delta \Vdash B[a] \sim B'[a'] : \square_\varepsilon\ i$

Fig. 22. Logical relation between CastCIC$^{\mathcal{G}}$ and ω-cpos

$\boxed{\Gamma \Vdash t \sim u : \mathsf{ne}_\varepsilon\, A_\varepsilon}$ logical relation on CastCIC$^{\mathcal{G}}$ terms of a neutral type A ($A_\varepsilon : \Gamma \Vdash A \sim A'$).

$$\frac{\Gamma \Vdash t \sim_{\mathrm{ne}} t' : \mathsf{ne}_\varepsilon\, A_\varepsilon}{\Gamma \Vdash t \sim () : \mathsf{ne}_\varepsilon\, A_\varepsilon}$$

$\boxed{\Gamma \Vdash t \sim_{\mathrm{ne}} u : A_\varepsilon}$ logical relation on neutral CastCIC$^{\mathcal{G}}$ terms (excerpt).

$$\frac{\Gamma \Vdash \quad a \sim a' : A_\varepsilon \in \Gamma}{\Gamma \Vdash a \sim_{\mathrm{ne}} a' : A_\varepsilon} \qquad \frac{\Gamma \Vdash f \sim_{\mathrm{ne}} f' : \Pi_\varepsilon\, A_\varepsilon\, B_\varepsilon \qquad t_\varepsilon : \Gamma \Vdash t \sim t' : A_\varepsilon}{\Gamma \Vdash f\ t \sim_{\mathrm{ne}} g\ u : B_\varepsilon\ t_\varepsilon}$$

$$\frac{\begin{array}{cc} b_\varepsilon : \Gamma \Vdash b \sim_{\mathrm{ne}} b' : \mathbb{B}_\varepsilon & P_\varepsilon : \Gamma, z \sim z' : \mathbb{B}_\varepsilon \Vdash P \sim P' \\ \Gamma \Vdash t_{\mathtt{true}} \sim t'_{\mathtt{true}} : P_\varepsilon[\mathtt{true} \sim \mathtt{true}] & \Gamma \Vdash t_{\mathtt{false}} \sim t'_{\mathtt{false}} : P_\varepsilon[\mathtt{false} \sim \mathtt{false}] \end{array}}{\Gamma \Vdash \mathtt{ind}_{\mathbb{B}}(b, z.P, (t_{\mathtt{true}}, t_{\mathtt{false}})) \sim_{\mathrm{ne}} \mathtt{ind}_{\mathbb{B}}(b', z'.P', (t'_{\mathtt{true}}, t'_{\mathtt{false}})) : P_\varepsilon[b_\varepsilon]}$$

$$\frac{A_\varepsilon : \Gamma \Vdash A \sim_{\mathrm{ne}} A' : \square_\varepsilon}{\Gamma \Vdash ?_A \sim_{\mathrm{ne}} ?_{A'} : A_\varepsilon} \qquad \frac{A_\varepsilon : \Gamma \Vdash A \sim_{\mathrm{ne}} A' : \square_\varepsilon}{\Gamma \Vdash \mathtt{err}_A \sim_{\mathrm{ne}} \mathtt{err}_{A'} : A_\varepsilon}$$

$$\frac{A_\varepsilon : \Gamma \Vdash A \sim_{\mathrm{ne}} A' : \square_\varepsilon \qquad B_\varepsilon : \Gamma \Vdash B \sim B' : \square_\varepsilon \qquad \Gamma \Vdash t \sim t' : A_\varepsilon}{\Gamma \Vdash \langle B \Leftarrow A \rangle\, t \sim_{\mathrm{ne}} \downarrow^{\widehat{?}}_{B'} \uparrow^{\widehat{?}}_{A'}\, t' : B_\varepsilon}$$

Fig. 23. Logical relation between CastCIC$^{\mathcal{G}}$ and ω-cpos

$$\frac{}{\text{canonical}\,\mathtt{nil}\ A} \qquad \frac{}{\text{canonical}(\mathtt{cons}\ A\ a\ n\ v)} \qquad \frac{}{\text{canonical}\,\mathtt{nil}_?\ A} \qquad \frac{}{\text{canonical}(\mathtt{cons}_?\ A\ a\ n\ v)}$$

$$\frac{\text{neutral}\,v \quad \vee \quad \text{neutral}\,n \quad \vee \quad \text{neutral}\,m}{\text{neutral}(\langle \mathtt{vec}\ B\ m \Leftarrow \mathtt{vec}\ A\ n\rangle\ v)} \qquad \frac{\text{neutral}\,v}{\text{neutral}(\mathtt{vec\_rect}\ P\ P_{nil}\ P_{cons}\ v)}$$

$$\frac{\Gamma \vdash A \triangleleft \square_i}{\Gamma \vdash \mathtt{nil}_{?@\{i\}}\ A \triangleright \mathtt{vec}\ A\ ?_{\mathbb{N}}}\ \textsc{nilU} \qquad \frac{\Gamma \vdash A \triangleleft \square_i \qquad \Gamma \vdash a \triangleleft A \qquad \Gamma \vdash n \triangleleft \mathbb{N} \qquad \Gamma \vdash v \triangleleft \mathtt{vec}\ A\ n}{\Gamma \vdash \mathtt{cons}_{?@\{i\}}\ A\ a\ n\ v \triangleright \mathtt{vec}\ A\ ?_{\mathbb{N}}}\ \textsc{consU}$$

$\textsc{v-rect-nil}$: $\mathtt{vec\_rect}\ P\ P_{nil}\ P_{cons}\ \mathtt{nil}\ A \rightsquigarrow P_{nil}$

$\textsc{v-rect-cons}$: $\mathtt{vec\_rect}\ P\ P_{nil}\ P_{cons}\ (\mathtt{cons}\ A\ a\ n\ v) \rightsquigarrow P_{cons}\ a\ n\ (\mathtt{vec\_rect}\ P\ P_{nil}\ P_{cons}\ v)$

$\textsc{v-rect-err}$: $\mathtt{vec\_rect}\ P\ P_{nil}\ P_{cons}\ \mathtt{err}_{\mathtt{vec}\ A\ n} \rightsquigarrow \mathtt{err}_{P\ \mathtt{err}_{\mathtt{vec}\ A\ n}}$

$\textsc{v-rect-unk}$: $\mathtt{vec\_rect}\ P\ P_{nil}\ P_{cons}\ ?_{\mathtt{vec}\ A\ n} \rightsquigarrow ?_{P\ ?_{\mathtt{vec}\ A\ n}}$

$\textsc{v-rect-nilU}$: $\mathtt{vec\_rect}\ P\ P_{nil}\ P_{cons}\ \mathtt{nil}_?\ A \rightsquigarrow \langle P\ ?_{\mathbb{N}} \Leftarrow P\ 0\rangle\ (\mathtt{vec\_rect}\ P\ P_{nil}\ P_{cons}\ \mathtt{nil}\ A)$

$\textsc{v-rect-consU}$: $\mathtt{vec\_rect}\ P\ P_{nil}\ P_{cons}\ (\mathtt{cons}_?\ A\ a\ n\ v) \rightsquigarrow$

$$\langle P\ ?_{\mathbb{N}} \Leftarrow P\ (S\ n)\rangle\ (\mathtt{vec\_rect}\ P\ P_{nil}\ P_{cons}\ (\mathtt{cons}\ A\ a\ n\ v))$$

$\textsc{V-nil}$: $\langle \mathtt{vec}\ B\ 0 \Leftarrow \mathtt{vec}\ A\ 0\rangle\ \mathtt{nil}\ A \rightsquigarrow \mathtt{nil}\ B$

$\textsc{V-nil-cons}$: $\langle \mathtt{vec}\ B\ (S\ n) \Leftarrow \mathtt{vec}\ A\ 0\rangle\ \mathtt{nil}\ A \rightsquigarrow \mathtt{err}_{\mathtt{vec}\ B\ (S\ n)}$

$\textsc{V-nil-?}$: $\langle \mathtt{vec}\ B\ ?_{\mathbb{N}} \Leftarrow \mathtt{vec}\ A\ 0\rangle\ \mathtt{nil}\ A \rightsquigarrow \mathtt{nil}_?\ B$

$\textsc{V-cons}$: $\langle \mathtt{vec}\ B\ (S\ m) \Leftarrow \mathtt{vec}\ A\ (S\ n)\rangle\ (\mathtt{cons}\ A\ a\ k\ v) \rightsquigarrow \mathtt{cons}\ B\ (\langle B \Leftarrow A\rangle\ a)\ m\ (\langle \mathtt{vec}\ B\ m \Leftarrow \mathtt{vec}\ A\ k\rangle\ v)$

$\textsc{V-cons-nil}$: $\langle \mathtt{vec}\ B\ 0 \Leftarrow \mathtt{vec}\ A\ (S\ n)\rangle\ (\mathtt{cons}\ A\ a\ k\ v) \rightsquigarrow \mathtt{err}_{\mathtt{vec}\ B\ 0}$

$\textsc{V-cons-?}$: $\langle \mathtt{vec}\ B\ ?_{\mathbb{N}} \Leftarrow \mathtt{vec}\ A\ (S\ n)\rangle\ (\mathtt{cons}\ A\ a\ k\ v) \rightsquigarrow \mathtt{cons}_?\ B\ (\langle B \Leftarrow A\rangle\ a)\ n\ (\langle \mathtt{vec}\ B\ n \Leftarrow \mathtt{vec}\ A\ k\rangle\ v)$

$\textsc{V-nilU}$: $\langle \mathtt{vec}\ B\ ?_{\mathbb{N}} \Leftarrow \mathtt{vec}\ A\ ?_{\mathbb{N}}\rangle\ \mathtt{nil}_?\ A \rightsquigarrow \mathtt{nil}_?\ B$

$\textsc{V-nilU-nil}$: $\langle \mathtt{vec}\ B\ 0 \Leftarrow \mathtt{vec}\ A\ ?_{\mathbb{N}}\rangle\ \mathtt{nil}_?\ A \rightsquigarrow \mathtt{nil}\ B$

$\textsc{V-nilU-cons}$: $\langle \mathtt{vec}\ B\ (S\ n) \Leftarrow \mathtt{vec}\ A\ ?_{\mathbb{N}}\rangle\ \mathtt{nil}_?\ A \rightsquigarrow \mathtt{err}_{\mathtt{vec}\ B\ (S\ n)}$

$\textsc{V-consU}$: $\langle \mathtt{vec}\ B\ ?_{\mathbb{N}} \Leftarrow \mathtt{vec}\ A\ ?_{\mathbb{N}}\rangle\ (\mathtt{cons}_?\ A\ a\ k\ v) \rightsquigarrow \mathtt{cons}_?\ B\ (\langle B \Leftarrow A\rangle\ a)\ k\ (\langle \mathtt{vec}\ B\ k \Leftarrow \mathtt{vec}\ A\ k\rangle\ v)$

$\textsc{V-consU-nil}$: $\langle \mathtt{vec}\ B\ 0 \Leftarrow \mathtt{vec}\ A\ ?_{\mathbb{N}}\rangle\ (\mathtt{cons}_?\ A\ a\ k\ v) \rightsquigarrow \mathtt{err}_{\mathtt{vec}\ B\ 0}$

$\textsc{V-consU-cons}$: $\langle \mathtt{vec}\ B\ (S\ n) \Leftarrow \mathtt{vec}\ A\ ?_{\mathbb{N}}\rangle\ (\mathtt{cons}_?\ A\ a\ k\ v) \rightsquigarrow$

$$\mathtt{cons}\ B\ (\langle B \Leftarrow A\rangle\ a)\ n\ (\langle \mathtt{vec}\ B\ n \Leftarrow \mathtt{vec}\ A\ k\rangle\ v)$$

$\textsc{V-unk}$: $\langle \mathtt{vec}\ B\ m \Leftarrow \mathtt{vec}\ A\ n\rangle\ ?_{\mathtt{vec}\ A\ n} \rightsquigarrow ?_{\mathtt{vec}\ B\ m}$ $\qquad m, n \in \{0, S\ m, ?_{\mathbb{N}}, \mathtt{err}_{\mathbb{N}}\}$

$\textsc{V-err}$: $\langle \mathtt{vec}\ B\ m \Leftarrow \mathtt{vec}\ A\ n\rangle\ \mathtt{err}_{\mathtt{vec}\ A\ n} \rightsquigarrow \mathtt{err}_{\mathtt{vec}\ B\ m}$ $\qquad m, n \in \{0, S\ m, ?_{\mathbb{N}}, \mathtt{err}_{\mathbb{N}}\}$

$\textsc{V-to-err}$: $\langle \mathtt{vec}\ B\ \mathtt{err}_{\mathbb{N}} \Leftarrow \mathtt{vec}\ A\ n\rangle\ v \rightsquigarrow \mathtt{err}_{\mathtt{vec}\ B\ \mathtt{err}_{\mathbb{N}}}$ $\qquad n \in \{0, S\ m, ?_{\mathbb{N}}, \mathtt{err}_{\mathbb{N}}\}$ and $v \neq ?_{\mathtt{vec}\ A\ n}$

Fig. 24. Canonical forms and reduction rule for vectors.