Gradualizing the Calculus of Inductive Constructions

Meven Bertrand, Kenji Maillard, Nicolas Tabareau, Éric Tanter

▶ **To cite this version:**

Meven Bertrand, Kenji Maillard, Nicolas Tabareau, Éric Tanter. Gradualizing the Calculus of Inductive Constructions. 2020. hal-02896776v2

HAL Id: hal-02896776

https://hal.science/hal-02896776v2

Preprint submitted on 20 Nov 2020 (v2), last revised 17 Nov 2021 (v5)

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

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

Acknowledging the ordeal of a fully formal development in a proof assistant such as Coq, 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 that encompasses all three variations, and develop their metatheory. We first present a bidirectional elaboration of Gradual CIC to a dependently-typed cast calculus, which elucidates the interrelation between typing, conversion, and the gradual guarantees. We use a syntactic model into CIC to inform the design of a safe, confluent reduction, and establish, when applicable, normalization. We also study the stronger notion of graduality as 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.

ACM Reference Format:
Meven Lennon-Bertrand, Kenji Maillard, Nicolas Tabareau, and Éric Tanter. 202Y. Gradualizing the Calculus of Inductive Constructions . *ACM Trans. Program. Lang. Syst.* V, N, Article 1 (January 202Y), 64 pages.

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 subtle challenges for gradualization. In the most elaborate effort to date, Eremondi et al. [2019]

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.

. 0164-0925/202Y/1-ART1
https://doi.org/

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 `vect A n` as defined in Coq:[1]

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

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

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:

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

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

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

typechecks, and is successfully convertible to 0, while:

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

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:

$$\texttt{def foo(n)(m) \{ if (n > m) then m + 1 else m > 0 \}}$$

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:

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

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:

$\texttt{foo : ?}$

$\texttt{foo : } \mathbb{N} \to \mathbb{N} \to \texttt{?}$

$\texttt{foo : } \mathbb{N} \to \mathbb{N} \to \texttt{if ? then } \mathbb{N} \texttt{ else ?}$

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

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

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

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 : vect A n), vect 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: vect 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 : vect 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: vect 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? One might wonder whether adapting the ideas of gradual typing to a dependent type theory does not make more sense for programs than it does for proofs. This observation is however misguided: 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 novel, 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 we could not readily find in the literature (§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 into CIC extended with induction-reduction [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 the normalizing variants, we provide a model of CastCIC that captures the notion of monotonicity with respect to precision. Finally, we discuss an extension of Scott's model based on ω-complete partial orders [Scott 1976] to prove graduality for the variant with divergence.
- We explain the challenging issue of equality in gradual type theories, and propose an approach to handling equality in GCIC through elaboration (§7).

We finally discuss related work (§8) and conclude (§9). 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).

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

[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.

`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 syntactical 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, $0 :: ? :: \mathbb{B}$ 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. The reason is that the argument type of δ, the unknown type $?$, is *consistent* with the type of δ itself, $? \to ?$, 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 $\mathtt{ax}_A$, 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 $\mathtt{ax}_A$ 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 → 𝔹), vect A n → vect A` $\mathtt{ax}_{\mathbb{N}}$, in order to avoid the complications related to specifying the size of the vector produced by `filter`. The problem is that the term:

```
head ℕ ax_ℕ (filter ℕ 4 even [ 0 ; 1 ; 2 ; 3 ])
```

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

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.

[4] We write $a :: A$ for a type ascription, which in some systems is syntactic sugar for $(\lambda x : A.x)\ a$ [Siek and Taha 2006], and is primitive in others [Garcia et al. 2016].

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 $\mathtt{ax}_{\mathbb{B}}$, but an infinite number of terms stuck on eliminations of ax, such as $\mathtt{match}\ \mathtt{ax}_{\mathtt{A}}\ \mathtt{with}\ \ldots$ or $\mathtt{ax}_{\mathbb{N}\to\mathbb{B}}\ 1$.

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 $\mathtt{err}_{\mathtt{A}}$ that can inhabit any type A. But instead of being treated as a computational black box like $\mathtt{ax}_{\mathtt{A}}$, $\mathtt{err}_{\mathtt{A}}$ 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:

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

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, $\mathtt{err}_{\mathtt{A}\to\mathtt{B}}$ and $\lambda\ \_ : \mathtt{A} \Rightarrow \mathtt{err}_{\mathtt{B}}$ are convertible, and the latter is considered to be in normal form. So $\mathtt{err}_{\mathtt{A}}$ 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 $\mathtt{err}_{\mathtt{A}}$ 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.,* $\mathtt{err}_{\mathbb{B}}$). 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 $\mathtt{err}_{\mathtt{False}}$, 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 $\mathtt{err}_{\mathtt{A}}$ 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 $\mathtt{vect}\ \mathtt{A}\ \mathtt{err}_{\mathbb{N}}$ is still not convertible to $\mathtt{vect}\ \mathtt{A}\ (\mathtt{S}\ \mathtt{err}_{\mathbb{N}})$. The reason is that $\mathtt{err}_{\mathbb{N}}$ behaves like an extra constructor to $\mathbb{N}$, so $\mathtt{S}\ \mathtt{err}_{\mathbb{N}}$ 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 possible set of types that it stands for [Garcia et al. 2016]. This allows to naturally understand the meaning of partially-specified types, for instance $\mathbb{B} \to ?$ 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 $\mathbb{B} \to ?$, 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. This property is hereafter called **Conservativity** (C), and parametrized with the considered static system. For instance, we write that GTLC satisfies $C_{/\mathrm{STLC}}$. 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 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.

[5] Not to be confused with logical consistency!

Note that, despite its many issues, CIC+ax (§2.2) satisfies $C_{/\mathsf{CIC}}$: 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 $\mathtt{A}$ and $\mathtt{B}$ are consistent if and only if there exists $\mathtt{T}$ such that $\mathtt{T} \sqsubseteq \mathtt{A}$ and $\mathtt{T} \sqsubseteq \mathtt{B}$. The unknown type $\mathtt{?}$ is the most imprecise type of all, *i.e.,* $\mathtt{T} \sqsubseteq \mathtt{?}$ for any $\mathtt{T}$. 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) states that if $\mathtt{t} \sqsubseteq \mathtt{u}$ and $\mathtt{t}$ is well-typed at type $\mathtt{T}$, then $\mathtt{u}$ is also well-typed at some type $\mathtt{U} \sqsupseteq \mathtt{T}$. The *dynamic gradual guarantee* (DGG) is the key result that bridges the syntactic notion of precision to reduction: if $\mathtt{t} \sqsubseteq \mathtt{u}$ and $\mathtt{t}$ reduces to some value $\mathtt{v}$, then $\mathtt{u}$ reduces to some value $\mathtt{v'} \sqsupseteq \mathtt{v}$; and if $\mathtt{t}$ diverges, then so does $\mathtt{u}$. This property entails that $\mathtt{t} \sqsubseteq \mathtt{u}$ means that $\mathtt{t}$ may error more than $\mathtt{u}$, but otherwise they should behave the same.

Note the clear separation between the two forms of precision that follows from the strict type/term distinction: the unknown type $\mathtt{?}$ is part of the type precision preorder, but not part of the term precision preorder; dually, the error term $\mathtt{err}$ is part of the term precision preorder, but not of the type precision preorder.

Graduality. New and Ahmed [2018] give a semantic account of precision that directly captures the property expressed as the DGG by Siek et al. [2015]. Considering precision as a generalization of parametricity [Reynolds 1983], they *define* precision as relating terms that only differ in their error behavior, with the more precise term able to fail more. To do so, they consider the following notion of observational error-approximation.

Definition 1 (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 context* $C : (\Gamma \vdash A) \Rightarrow (\vdash \mathbb{B})$ *closing over all free variables either*

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

Based on this notion, New and Ahmed [2018] can express the DGG in a more semantic fashion by saying that term precision implies observational error-approximation:

$$\text{If } \mathtt{t} \sqsubseteq \mathtt{u} \text{ then } \mathtt{t} \preccurlyeq^{obs} \mathtt{u}.$$

A key insight of their work is that this property, although desirable, is not enough to characterize the good dynamic behavior of precision. Indeed, their notion of graduality mandates 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. Technically, the adjunction part states that if we have $\mathtt{A} \sqsubseteq \mathtt{B}$, a term $\mathtt{a}$ of type $\mathtt{A}$, and a term $\mathtt{b}$ of type $\mathtt{B}$, then $\mathtt{a} \sqsubseteq \mathtt{b} :: \mathtt{A} \Leftrightarrow \mathtt{a} :: \mathtt{B} \sqsubseteq \mathtt{b}$. The retraction part further states that $\mathtt{t}$ is not only more precise than $\mathtt{t} :: \mathtt{B} :: \mathtt{A}$ (which is given by the unit of the adjunction) but is *equi-precise* to it,

noted $\mathtt{t} \sqsupseteq\sqsubseteq \mathtt{t} :: \mathtt{B} :: \mathtt{A}$. Because 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.

New and Ahmed [2018] introduce the term **Graduality** ($\mathcal{G}$) for these properties, which elegantly generalize parametricity [Reynolds 1983]: the parametric relation $\mathtt{R}$ between two types $\mathtt{A} \sqsubseteq \mathtt{B}$ being described by

$$\mathtt{a} \sqsubseteq \mathtt{b} :: \mathtt{A} \Leftrightarrow \mathtt{R\ a\ b} \Leftrightarrow \mathtt{a} :: \mathtt{B} \sqsubseteq \mathtt{b}.$$

Graduality is also based on important underlying structural properties of precision on terms, namely that term precision is stable by reduction (if $\mathtt{t} \sqsubseteq \mathtt{t'}$ and $\mathtt{t}$ reduces to $\mathtt{v}$ and $\mathtt{t'}$ to $\mathtt{v'}$, then $\mathtt{v} \sqsubseteq \mathtt{v'}$), and that the term and type constructors of the language are monotone (*e.g.*, if $\mathtt{t} \sqsubseteq \mathtt{t'}$ and $\mathtt{u} \sqsubseteq \mathtt{u'}$ then $\mathtt{t\ u} \sqsubseteq \mathtt{t'\ u'}$). These technical conditions, natural in a categorical setting [New et al. 2019], coincide with the programmer-level interpretation of precision and the DGG.

Finally, because of its reliance on observational equivalence, $\mathcal{G}$ is tied to safety ($\mathcal{S}$), in that it implies that valid gains of precision cannot get stuck.

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, $\mathtt{err}$ is at the term level. Likewise, *type precision*, with ? as greatest element, is separate from *term precision*, with $\mathtt{err}$ 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 $?_{\mathtt{A}}$, indexed by their type $\mathtt{A}$. 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 $\mathtt{err}$ 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, $\mathtt{err}_{\mathtt{A}}$ is the most precise term of type $\mathtt{A}$.

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 $?_{\mathtt{A}}$ inhabiting any type $\mathtt{A}$, we have one new canonical form for each type $\mathtt{A}$. In particular, $?_{\mathbb{B}}$ cannot possibly reduce to either $\mathtt{true}$ or $\mathtt{false}$ or $\mathtt{err}_{\mathbb{B}}$, because doing so would collapse the precision order. Therefore, $?_{\mathtt{A}}$ should propagate computationally, like $\mathtt{err}_{\mathtt{A}}$ (§2.3).

The difference between errors and unknown terms is rather on their static interpretation. In essence, the unknown term $?_{\mathtt{A}}$ is a dual form of exceptions: it propagates, but is optimistically comparable, *i.e.,* consistent with, any other term of type $\mathtt{A}$. Conversely, $\mathtt{err}_{\mathtt{A}}$ should not be consistent with any term of type $\mathtt{A}$. 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:

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

now typechecks: $\mathtt{vect\ A}\ ?_{\mathbb{N}}$ can be deemed consistent with $\mathtt{vect\ A\ (S}\ ?_{\mathbb{N}})$, because $\mathtt{S}\ ?_{\mathbb{N}}$ 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 `t:T` and `T≡U`, then `t: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' ≡ 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.[6]

Precision and the Gradual Guarantees. The static gradual guarantee (SGG) captures the intuition that "sprinkling" `?` over a term maintains its typeability. As such, the notion of precision used to formulate the SGG is inherently syntactic, over as-yet-untyped terms: typeability is the *consequence* of the SGG theorem [Siek et al. 2015]. In contrast, graduality ($\mathcal{G}$) operates over well-typed terms, so a semantic notion of precision can be type-directed, as formulated by New and Ahmed [2018]. In the simply-typed setting, these subtleties have no fundamental consequences: the syntactic notion of precision and the semantic notion coincide.

With dependent types, however, using a syntactic notion of precision is only a possibility to capture the SGG (and DGG), but cannot be used to capture $\mathcal{G}$. This is because in the simply typed setting, the syntactic notion of precision is shown to induce $\mathcal{G}$ using the stability of precision with respect to reduction. However, a syntactic notion of precision cannot be stable by conversion in type theory: conversion can operate on open terms, yielding neutral terms such as $1 :: \mathtt{X} :: \mathbb{N}$ where `X` is a type variable. Such a term cannot reduce further, while less precise variants such as $1 :: ? :: \mathbb{N}$ would reduce to 1. Depending on the upcoming substitution for `X`, $1 :: \mathtt{X} :: \mathbb{N}$ can either raise an error or reduce to 1. Those stuck open terms cannot be handled using syntactic definitions. Rather, $\mathcal{G}$ has to be established relative to a semantic notion of precision. Unfortunately, such a semantic notion presumes typeability, and therefore cannot be used to state the SGG.

DGG vs Graduality. In the simply-typed setting, graduality ($\mathcal{G}$) can be seen as an equivalent, semantic formulation of (the key property underlying) the DGG. We observe that, in a dependently-typed setting, $\mathcal{G}$ is in fact a much stronger and useful property, due to the embedding-projection pairs requirement.

To see why, consider a system in which any term of type `A` that is not fully-precise immediately reduces to $?_{\mathtt{A}}$. This system would satisfy $\mathcal{C}$, $\mathcal{S}$, $\mathcal{N}$, and ...the DGG. Recall that the DGG only

[6] 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].

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.

In the rest of this article, we use graduality $\mathcal{G}$ as the property that refers to *both* the DGG and ep-pairs properties.

Observational refinement. Let us now 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, but 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 gradual theory admits divergence, then a *diverging term* is more precise than the unknown term, which does not diverge, thereby breaking the left-to-right implication of equi-divergence. Second, 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 setting, the motto of graduality ought to be adjusted: more precise programs produce more type error *or diverge more*. This leads to the following definition of *observational refinement*.

Definition 2 (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, $C[u] \rightsquigarrow^* \mathtt{err}_\mathbb{B}$ or diverges implies $C[t] \rightsquigarrow^* \mathtt{err}_\mathbb{B}$ or diverges.*

Note that, in a gradual dependent theory that admits divergence, equi-refinement does not imply observational equivalence, because errors and divergence are collapsed. If the gradual dependent theory is strongly normalizing, then both notions 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 $\mathtt{A} \sqsubseteq \mathtt{B}$, any term $\mathtt{t}$ of type $\mathtt{A}$ is equi-precise to $\mathtt{t} :: \mathtt{B} :: \mathtt{A}$. We additionally require that for any context $\mathtt{C}$, if $\mathtt{C}[\mathtt{t}]$ reduces at least k steps, then $\mathtt{C}[\mathtt{t} :: \mathtt{B} :: \mathtt{A}]$ also reduces at least k steps. Intuitively, this means that the reduction of $\mathtt{C}[\mathtt{t} :: \mathtt{B} :: \mathtt{A}]$, 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 $C_{/\mathsf{STLC}}$ 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 $C_{/\mathsf{STLC}}$), 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 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). *Suppose a gradual dependent type theory that satisfies properties $C_{/\mathsf{CIC}}$ and $\mathcal{G}$. Then $\mathcal{N}$ cannot hold.*

Proof. 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 $C_{/\mathsf{CIC}}$ implies $C_{/\mathsf{STLC}}$. Additionally, $\mathcal{G}$ can be specialized to the simply-typed fragment of the theory, by setting $? = ?_{\square_0}$ to be the unknown type. Therefore, we can apply Theorem 5 and we get a well-typed term that diverges, which is a violation of $\mathcal{N}$. □

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 $C_{/\mathsf{CIC}}$), 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 ($C_{/\mathsf{CIC}}$).

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 during typing: 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 $\mathsf{CIC}^{\uparrow}$ (read "CIC-shift"). $\mathsf{CIC}^{\uparrow}$ is a subset of CIC, with a stricter constraint on universe levels. In particular $\mathsf{CIC}^{\uparrow}$ loses the closure of universes under dependent product that CIC enjoys. As a consequence, some well-typed CIC terms are not well-typed in $\mathsf{CIC}^{\uparrow}$.[7]
- The second parameter is the reduction counterpart of the first parameter. Note that it is only meaningful to allow this reduction parameter to be loose (*i.e.,* using maximum) if the typing parameter is also loose. Letting the typing parameter be strict (*i.e.,* using successor) while the reduction parameter is loose breaks subject reduction (and hence $\mathcal{S}$).

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

	$\mathcal{S}$	$\mathcal{N}$	$C_{/X}$	$\mathcal{G}$	SGG	DGG
GCIC$^{\mathcal{G}}$	✓(Th. 8)	✗	CIC (Th. 21)	✓(Th. 30)	✓(Th. 22)	✓(Th. 23)
GCIC$^{\uparrow}$	✓(idem)	✓(Th. 9 & 24)	CIC$^{\uparrow}$ (idem)	✓(Th. 29)	✓(idem)	✓(Th. 23)
GCIC$^{\mathcal{N}}$	✓(idem)	✓(idem)	CIC (idem)	✗	✗	✗

Table 1. GCIC variants and their properties
$\mathcal{S}$: safety — $\mathcal{N}$: normalization — $C_{/X}$: conservativity wrt theory X — $\mathcal{G}$: graduality (DGG + ep-pairs) — SGG: static gradual guarantee — DGG: dynamic gradual guarantee

Based on these parameters, this work develops the following three variants of GCIC, whose properties are summarized in Table 1:

(1) GCIC$^{\mathcal{G}}$**: a theory that satisfies both** $C_{/\text{CIC}}$ **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.

(2) GCIC$^{\uparrow}$**: a theory that satisfies both** $\mathcal{N}$ **and** $\mathcal{G}$**, and supports** C **with respect to** 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 $C_{/\text{CIC}^{\uparrow}}$.

(3) GCIC$^{\mathcal{N}}$**: a theory that satisfies both** $C_{/\text{CIC}}$ **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. Still, GCIC$^{\mathcal{N}}$ satisfies a partial version of graduality: $\mathcal{G}$ holds on all terms that live in CIC$^{\uparrow}$, seen as a subsystem of CIC. This is arguably a strength of GCIC$^{\mathcal{N}}$ over Approximate Normalization [Eremondi et al. 2019], which breaks the ep-pairs requirement of $\mathcal{G}$ on all terms, even first-order, simply-typed ones.

Table 1 also includes pointers to the respective theorems. Note that because GCIC is one common framework with two parameters, we are able to establish several properties for all variants at once.

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 → 𝔹), vect A n → vect A ?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 GCIC$^{\mathcal{G}}$, it reduces forever, as it would in the untyped lambda calculus. In that sense, GCIC$^{\mathcal{G}}$ can embed the untyped lambda calculus just as GTLC [Siek et al. 2015]. In 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 GCIC$^{\mathcal{G}}$ and in GCIC$^{\mathcal{N}}$ is given in full details in §5.3. In GCIC$^{\uparrow}$, Ω fails in the same way as in 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 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 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 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 GCIC$^{\mathcal{N}}$, even of the dynamic gradual guarantee. Finally, in 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:

- 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.
- 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 outset of the CIC$^{\uparrow}$ subset. In 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.
- 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 and Agda support different ways to customize their semantics (such as allowing Type-in-Type, or switching off termination checking)—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.[8]

In order to satisfy conservativity with respect to CIC ($C_{/\mathsf{CIC}}$), 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 $\langle \mathbb{B} \Leftarrow ?_\Box \rangle\ \langle ?_\Box \Leftarrow \mathbb{N} \rangle\ 0$ 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 $C_{/\mathsf{CIC}}$ 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 $\Box$, 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, $?_{\Box_i}$ behaves as a dependent sum $\Sigma\ \mathtt{A}{:}\Box_i.\ \mathtt{A}$. 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 CastCIC$^{\mathcal{N}}$ and CastCIC$^{\uparrow}$. Together with safety ($\mathcal{S}$), this gives us weak logical consistency for CastCIC$^{\mathcal{N}}$ and CastCIC$^{\uparrow}$.

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

3.4 Varieties of Precision and Graduality

As explained earlier (§2.5), we need two different notions of precision to deal with SGG and DGG (or rather $\mathcal{G}$). 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 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 non-semantic precision, we prove *elaboration graduality* (Theorem 22), that is the equivalent of SGG in our setting: if a term t of GCIC elaborates to a term t' of CastCIC, then a term u less syntactically precise than t in GCIC elaborates to a term u' less structurally precise than t' in CastCIC.

However, we cannot expect to prove $\mathcal{G}$ for CastCIC (in its variants $\text{CastCIC}^{\mathcal{G}}$ and $\text{CastCIC}^{\uparrow}$) with respect to structural precision (§2.5) directly. This is because, contrarily to GTLC, more precise terms can sometimes be more stuck, because of type variables. For instance, $\langle \mathbb{N} \Leftarrow X \rangle\ \langle X \Leftarrow \mathbb{B} \rangle\ 0$ is neutral due to the type variable X, while $\langle \mathbb{N} \Leftarrow ?_{\square} \rangle\ \langle ?_{\square} \Leftarrow \mathbb{B} \rangle\ 0$ reduces to $\mathtt{err}_{\mathbb{B}}$ even though it is less precise. Semantically, we are able to say that the more precise term will error whatever is picked for X, but the syntactic notion does not capture this. To prove $\mathcal{G}$ inductively, one however needs to reason about such open terms as soon as one goes under a binder, and terms like the one above cannot be handled 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 (semantic) *propositional precision* and establish that it gives rise to embedding-projection pairs [New and Ahmed 2018]. As propositional precision subsumes structural precision, it allows us to establish $\mathcal{G}$ for $\text{CastCIC}^{\uparrow}$ (Theorem 29) as a corollary on closed terms. The monotone model only works for a normalizing gradual type theory, thus we then establish $\mathcal{G}$ for $\text{CastCIC}^{\mathcal{G}}$ 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 is folklore among type theory specialists [McBride 2019], but has never been spelled out in details — to our knowledge. 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 $C_{/\text{CIC}}$ in a gradual setting where conversion is replaced by consistency, which is not transitive.

Syntax. Our syntax for CIC terms, featuring a predicative universe hierarchy $\square_i$, is the following:[9]

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

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 syntactical 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 strictly positive

[9] In this work, we do not deal with the impredicative sort Prop, for multiple reasons. First, the models of §6 rely on the predicativity of the universe hierarchy, see that section for details. More fundamentally, it seems inherently impossible to avoid the normalization problem of Ω with an impredicative sort; and in the non-terminating setting, Prop can be interpreted just as Type, following Palmgren [1998].

ones to preserve normalization, following [Giménez 1998]. At this point we consider only inductive types without indices: § 7 explains how to recover usual indexed inductive types with just one equality type. Inductive types are formally annotated with a universe level $@i$, controlling the level of its parameters: for instance $\mathsf{List}_{@i}(A)$ expects A to be a type in $\square_i$. This level is omitted when inessential. An inductive type at level i with parameters $\mathbf{a} : \mathrm{Params}(I, i)$ is noted $I_{@i}(\mathbf{a})$. Similarly $c^I_{k@i}(\mathbf{a}, \mathbf{b})$ denotes the k-th constructor of the inductive I, taking parameters $\mathbf{a} : \mathrm{Params}(I, i)$ and arguments $\mathbf{b} : \mathrm{Args}(I, i, c_k)$.

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 $C_{/\mathsf{CIC}}$. 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 $C_{/\mathsf{CIC}}$.

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, for instance that it should be a universe, a Π-type, or an instance of an inductive I. These 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

$\boxed{\vdash \Gamma}$

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

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

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

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

$$\frac{\Gamma \vdash a_k \triangleleft X_k[\mathbf{a}/\mathbf{x}]}{\Gamma \vdash I@i(\mathbf{a}) \triangleright \square_i}\textsc{Ind} \qquad \frac{\Gamma \vdash a_k \triangleleft X_k[\mathbf{a}/\mathbf{x}] \qquad \Gamma \vdash b_l \triangleleft Y_l[\mathbf{a}/\mathbf{x}][\mathbf{b}/\mathbf{y}]}{\Gamma \vdash c^I@i(\mathbf{a},\mathbf{b}) \triangleright I@i(\mathbf{a})}\textsc{Cons}$$

with $\mathrm{Params}(I, i) = \mathbf{X}$ and $\mathrm{Args}(I, i, c) = \mathbf{Y}$

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

with $\mathrm{Args}(I, i, c_k) = \mathbf{Y_k}$

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

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

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

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

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

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

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

$\boxed{t \equiv u}$

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

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

Fig. 1. CIC: Bidirectional typing

$$
\begin{array}{lll}
s_\Pi(i,j) := \max(i,j) & c_\Pi(i) := i & (\text{GCIC}^{\mathcal{G}}\text{-CastCIC}^{\mathcal{G}}) \\
s_\Pi(i,j) := \max(i,j) & c_\Pi(i) := i-1 & (\text{GCIC}^{\mathcal{N}}\text{-CastCIC}^{\mathcal{N}}) \\
s_\Pi(i,j) := \max(i,j)+1 & c_\Pi(i) := i-1 & (\text{GCIC}^{\uparrow}\text{-CastCIC}^{\uparrow})
\end{array}
$$

Fig. 2. Universe parameters

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 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.

Finally, we observe that, despite being folklore [McBride 2019], the equivalence of this bidirectional formulation with standard CIC relies on the transitivity of conversion and this has never been spelled out in details in the literature. In any case, this does not matter in our work, as in the gradual setting, this conjecture does not hold. This is precisely the point of using a bidirectional formulation in a gradual setting where consistency is not a transitive relation.

5 **FROM** GCIC **TO** CastCIC

In this section we present the elaboration of the source gradual system GCIC into 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 $\text{CIC}^{\uparrow}$, as well as the gradual guarantees (§ 5.5).

5.1 CastCIC

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

$$t ::= \cdots \mid ?_t \mid \mathtt{err}_t \mid \langle t \Leftarrow t \rangle\ t. \qquad \text{(Syntax of CastCIC)}$$

The unknown term and dynamic failure 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, using the failure $\mathtt{err}_T$ 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

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

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

$$\frac{\Gamma \vdash T \blacktriangleright_{\square} \square_i}{\Gamma \vdash \star_T \triangleright T}\textsc{Exc} \qquad \frac{\Gamma \vdash A \blacktriangleright_{\square} \square_i \qquad \Gamma \vdash B \blacktriangleright_{\square} \square_j \qquad \Gamma \vdash t \triangleleft A}{\Gamma \vdash \langle B \Leftarrow A \rangle\ t \triangleright B}\textsc{Cast}$$

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

$$\mathbb{H} := \square_i \mid \Pi \mid I$$

$$\mathrm{head}\,(\Pi\, A\, B) := \Pi \qquad \mathrm{head}\,(\square_i) := \square_i \qquad \mathrm{head}\,(I\,\mathbf{a}) := I \qquad \text{undefined otherwise}$$

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

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

Fig. 4. Head constructor and germ

updated Prod rule in Fig. 3). The second function c_Π controls the universe levels in the reduction of casts between $? \to ?$ and $?$ (see Fig. 5).

Typing. Fig. 3 gives the typing rules for the three new primitives of CastCIC. Apart from the modified Prod rule, 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 rule for $?_T$ and $\mathtt{err}_T$ both say that it infers T when T is a type. Hereafter, we use when possible the notation $\star_T$ to mean either $?_T$ or $\mathtt{err}_T$. Note, that in CastCIC, no consistency premise appears when typing a cast: consistency only plays a role in the GCIC layer, but disappears after the elaboration. Instead, we rely on the usual conversion, defined as in CIC as the existence of α-equal reducts for the reduction described hereafter.

Reduction. The typing rules of the new primitives are rather crude; the interesting part is really their reduction behavior. The reduction rules of CastCIC are given in Fig. 5 (congruence rules omitted). Reduction relies on two auxiliary functions that mediate between types and their head constructor $h \in \mathbb{H}$ (Fig. 4). The first is the partial function head that returns the head constructor of a term, if it exists. The second is the germ[10] $\mathrm{Germ}_i\, h$, which constructs the least precise type (when it exists) with head h at level i.

[10]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.

Reduction rules for cast:

$$\textsc{Prod-Prod}: \langle \Pi(x:A_2).B_2 \Leftarrow \Pi(x:A_1).B_1\rangle\ (\lambda\, x:A.t) \rightsquigarrow \lambda\, y:A_2.\langle B_2 \Leftarrow B_1[a_1/x]\rangle\ (t[a/x])$$

$$\text{with } a := \langle A \Leftarrow A_2\rangle\ y \text{ and } a_1 := \langle A_1 \Leftarrow A_2\rangle\ y$$

$$\textsc{Prod-Germ}: \left\langle ?_{\square_i} \Leftarrow \Pi(x:A^{\mathrm{d}}).A^{\mathrm{c}}\right\rangle f \rightsquigarrow \left\langle ?_{\square_i} \Leftarrow \mathrm{Germ}_i\,\Pi\right\rangle \left(\left\langle \mathrm{Germ}_i\,\Pi \Leftarrow \Pi(x:A^{\mathrm{d}}).A^{\mathrm{c}}\right\rangle f\right) \quad \text{when } \Pi(x:A^{\mathrm{d}}).A^{\mathrm{c}} \neq \mathrm{Germ}_i\,\Pi$$

$$\textsc{Ind-Ind}: \langle I(\mathbf{a}') \Leftarrow I(\mathbf{a})\rangle\ c(\mathrm{a}, b_1, \ldots, b_n) \rightsquigarrow c(\mathrm{a}', b_1', \ldots, b_n')$$

$$\text{with } b_k' := \langle Y_k[\mathbf{a}'/\mathbf{x}][\mathbf{b}'/\mathbf{y}] \Leftarrow Y_k[\mathbf{a}/\mathbf{x}][\mathbf{b}/\mathbf{y}]\rangle\ b_k$$

$$\textsc{Ind-Germ}: \left\langle ?_{\square_i} \Leftarrow I(\mathbf{a})\right\rangle\ t \rightsquigarrow \left\langle ?_{\square_i} \Leftarrow \mathrm{Germ}_i\,I\right\rangle\ (\langle \mathrm{Germ}_i\,I \Leftarrow I(\mathbf{a})\rangle\ t) \quad \text{when } I(\mathbf{a}) \neq \mathrm{Germ}_i\,I$$

$$\textsc{Ind-Err}: \langle I(\mathbf{a}') \Leftarrow I(\mathbf{a}'')\rangle\ \star_{I(\mathbf{a})} \rightsquigarrow \star_{I(\mathbf{a}')} \qquad \textsc{Univ-Univ}: \langle \square_i \Leftarrow \square_i\rangle\ A \rightsquigarrow A$$

$$\textsc{Up-Down}: \left\langle X \Leftarrow ?_{\square_i}\right\rangle \left\langle ?_{\square_i} \Leftarrow \mathrm{Germ}_i\,h\right\rangle\ t \rightsquigarrow \langle X \Leftarrow \mathrm{Germ}_i\,h\rangle\ t \quad \text{when } \mathrm{Germ}_i\,h \neq \mathtt{err}_{\square_i}$$

$$\textsc{Size-Err}: \left\langle ?_{\square_i} \Leftarrow \mathrm{Germ}_j\,h\right\rangle\ t \rightsquigarrow \mathtt{err}_{?_{\square_i}} \quad \text{when } j > i$$

$$\textsc{Head-Err}: \langle T' \Leftarrow T\rangle\ t \rightsquigarrow \mathtt{err}_{T'} \quad \text{when head}\, T \neq \text{head}\, T'$$

$$\textsc{Down-Err}: \langle X \Leftarrow ?_{\square}\rangle\ \star_{?_{\square}} \rightsquigarrow \star_X \qquad \textsc{Err-Dom}: \langle X \Leftarrow \mathtt{err}_{\square}\rangle\ t \rightsquigarrow \mathtt{err}_X$$

$$\textsc{Err-Codom}: \langle \mathtt{err}_{\square} \Leftarrow Z\rangle\ t \rightsquigarrow \mathtt{err}_{\mathtt{err}_{\square}} \quad \text{when head}\, Z \text{ is defined}$$

Reduction rules for ? and err:

$$\textsc{Prod-}\star: \star_{\Pi(x:A).B} \rightsquigarrow \lambda(x:A).\star_B \qquad \textsc{Ind-}\star: \mathtt{ind}_I(\star_{I(\mathbf{a})}, z.P, f.\mathbf{y}.\mathbf{t}) \rightsquigarrow \star_{P[\star_{I(\mathbf{a})}/z]}$$

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 in §6. Nevertheless, we now provide some intuition about their meaning. Let us start with rules Prod-⋆ and Ind-⋆. These rules simply specify the error-like behavior of both ? and err. Similarly, rules Err-Dom, Err-Codom specify that err behaves like an error also with respect to the cast—contrarily to ?, whose behavior is more specific.

Next are rules Prod-Prod, Ind-Ind and Univ-Univ, which correspond to success cases cast/dynamic type-checking, where the cast was used between types with the same head. In that case, casts are either completely erased when possible, or propagated. We restrict Prod-Prod to abstractions because of the absence of η-expansion in the system: allowing the cast between product types of an arbitrary function f to reduce to a λ-abstraction would perform a kind of η-expansion on f, that an uncast function cannot match in the absence of η. Since constructors and inductives are fully-applied, this rule cannot be blocked on a partially applied constructor on inductive. As for inductive types, the restriction to reduce only on constructors means that a cast between $\mathbb{N}$ and $\mathbb{N}$ does not simply reduce away, since it waits for its argument to be a constructor to reduce. We

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

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

follow this strategy to be consistent for all inductive types, since as soon as inductive types have type arguments, such as `list A`, casts need the recursive behavior of Ind-Ind.

Rule Head-Err specifies failure of dynamic checking when the considered types have different heads. Rule Size-Err corresponds to another kind of error, which does not happen in absence of a type hierarchy: we are trying to upcast a term to ?, but at a universe level that is too low, and hence also leads to a failure.

Rule Ind-Err propagates both ? and `err` between the same inductive type (applied to potentially different parameters). Similarly, Rule Down-Err propagates both ? and `err` from the unknown type to any type X.

Finally, there are specific rules pertaining to casts to and from ?. Rules Prod-Germ and Ind-Germ decompose the upcast of a generic type into ? as a succession of simple upcasts going from a germ to ?. Rule Up-Down erases casts through ?, and combined with the previous success and failure cases, completes the picture of dynamic type-checking.

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 show a crucial lemma, namely the confluence of the rewriting system induced by reduction.

Lemma 7 (Confluence of CastCIC). *If* $T\ (\leftsquigarrow \rightsquigarrow)^*\ U$ *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 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 will be proven using the standard progress and subject reduction properties. 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, we distinguish between canonical forms and neutral terms. Neutral terms are special kind of canonical forms that have the additional property that they can trigger a reduction (such as a λ in an application, or a constructor of an inductive type in an elimination). Intuitively, neutral terms correspond to (blocked) destructors, waiting for a substitution to happen, while other canonical forms correspond to constructors.

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 a 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

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

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

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

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

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

$$\frac{\text{neutral}\ T}{\text{neutral}\ \langle T \Leftarrow \square\rangle\ t} \qquad \frac{\text{neutral}\ T}{\text{neutral}\ \langle T \Leftarrow \Pi\, x : A.B\rangle\ t} \qquad \frac{\text{neutral}\ T}{\text{neutral}\ \langle T \Leftarrow I(\mathbf{a})\rangle\ t}$$

Fig. 7. Head neutral and canonical forms for CastCIC

terms reduce on Π-types (§2.3). Second, their 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. These canonical forms represent constructors for $?_{\square}$. Finally, the cast operation behaves as a destructor, but not on an inductive type as usually the case in CIC, but rather on the universe $\square$. 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. Then, when the source type reduces to a head constructor, the cast scrutinizes the target type, so the cast is neutral when the target type is neutral. Note that there is however a special case when casting from $?_{\square}$: in that case, the cast is neutral when its argument is neutral.

Equipped with the notion of canonical form, 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* $t \rightsquigarrow t'$ *with a weak-head reduction.*

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. If t is already a canonical form, we are done. Otherwise, its head term former must be a destructor (application, eliminator or cast). Let us consider the case of application. We have a well-typed term $u\ v$ and we know that u either takes a head reduction step or is a canonical form. In the first case, this reduction step is a head reduction for t, and we are done. Otherwise, u has a product type because t is well-typed. Since it is canonical, it can only be an abstraction or a neutral term (it cannot be an error or unknown term because it is on a Π-type). In the first case, the application β-reduces. In the second, t is a neutral term, and thus canonical. All other cases

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

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

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

Fig. 8. CastCIC: α-consistency

are similar: either a deeper reduction happens, t itself reduces because some canonical form was not neutral and creates a redex, or t is neutral.

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

Theorem 9 (Normalization of CastCIC$^{\mathcal{N}}$ and CastCIC$^{\uparrow}$ ($\mathcal{N}$)). *Every reduction path for a well-typed term in* CastCIC$^{\mathcal{N}}$ *or* 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, see Theorem 24. So strong normalization holds whenever 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 (§3).

Syntax. The syntax of GCIC 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 turn to conversion. Indeed, to account for the imprecision introduced by ?, elaboration employs *consistent conversion* to compare CastCIC terms instead of the usual conversion relation.

Definition 3 (Consistent conversion). *Two 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* $s \rightsquigarrow^* s'$, $t \rightsquigarrow^* t'$ *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 any term. 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 ($C_{/\mathrm{CIC}}$), 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. 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.

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. □

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 how only the subject of the judgment is a source term in GCIC, both inputs (that have already been elaborated) and outputs (that are to be elaborated) are target terms in CastCIC.[11] Let us comment a bit on the specific modifications and additions.

The most salient feature of elaboration is the cast insertions 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?, where the type $?_{\square_i}$ is replaced by the least precise type of the right level verifying the constraint, which is exactly what Germ 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 the compatibility of the partial inference judgments with the imprecision of ?. Indeed, without them a term of type $?_{\square_i}$ could not be used as a function, or as a scrutinee of a match.

Rule Ukn also deserves some explanation: $?@i$ is elaborated to $?_{?_{\square_i}}$, the least precise term 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 that reduces 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 to have ? infer ?, as we choose. However, 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 [Harper and Pollack 1991], alleviating the user from the need to give any annotations when they use ?—much in the same way a Coq user almost never specifies explicit universe levels.

Finally, in order to obtain uniqueness of elaboration—which is not a priori guaranteed because casts that depend on computation are inserted during elaboration—we fix a reduction strategy,

[11] Colors are used to help with readability, making the distinction between terms of GCIC and terms of CastCIC clearer, but are not essential.

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

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

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

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

$$\frac{\Gamma \vdash \tilde{a}_m \triangleleft X_m[\mathbf{a}/\mathbf{x}] \rightsquigarrow a_m}{\Gamma \vdash I@i(\tilde{\mathbf{a}}) \rightsquigarrow I@i(\mathbf{a}) \triangleright \square_i}\textsc{Ind} \qquad \frac{\Gamma \vdash \tilde{a}_m \triangleleft X_m[\mathbf{a}/\mathbf{x}] \rightsquigarrow a_m \qquad \Gamma \vdash \tilde{b}_n \triangleleft Y_n[\mathbf{a}/\mathbf{x}][\mathbf{b}/\mathbf{y}] \rightsquigarrow b_n}{\Gamma \vdash c_k@i(\tilde{\mathbf{a}}, \tilde{\mathbf{b}}) \rightsquigarrow c(\mathbf{a}, \mathbf{b}) \triangleright I(\mathbf{a})}\textsc{Cons}$$

with $\mathrm{Params}(I, i) = \mathbf{X}$ and $\mathrm{Args}(I, i, c) = \mathbf{Y}$

$$\frac{\Gamma \vdash \tilde{s} \rightsquigarrow s \blacktriangleright_I I(\mathbf{a}) \qquad \Gamma, z : I(\mathbf{a}) \vdash \tilde{P} \rightsquigarrow P \blacktriangleright_\square \square_i \qquad \Gamma, f : (\Pi\, z : I(\mathbf{a}), P), \mathbf{y} : \mathbf{Y_k}[\mathbf{a}/\mathbf{x}] \vdash \tilde{t}_k \triangleleft P[c_k(\mathbf{a}, \mathbf{y})/z] \rightsquigarrow t_k}{\Gamma \vdash \mathsf{ind}_I(\tilde{s}, z.\tilde{P}, f.\mathbf{y}.\tilde{\mathbf{t}}) \rightsquigarrow \mathsf{ind}_I(s, z.P, \mathbf{f}.\mathbf{y}.\mathbf{t}) \triangleright P[s/z]}\textsc{Fix}$$

with $\mathrm{Args}(I, i, c_k) = \mathbf{Y_k}$

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

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

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

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

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

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

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

typically weak-head reduction, for $\rightsquigarrow^*$ in constrained inference rules. This ensures that for instance the A and B in a derivation of $\Gamma \vdash \rightsquigarrow t \triangleright \Pi\, x : A.B$ are unique. This could be avoided, in which case we would obtain uniqueness of the inferred type only up to conversion, but would also make the rest of the technical development more complex.

Direct properties. With this strategy fixed, the reduction rules then immediately translate to an algorithm for elaboration. Coupled with the decidability of consistency (Prop. 10), this makes elaboration decidable in $\mathrm{GCIC}^{\mathcal{N}}$ and $\mathrm{GCIC}^{\uparrow}$, although the same algorithm might diverge in $\mathrm{GCIC}^{\mathcal{G}}$, only giving us semi-decidability of typing.

Theorem 11 (Decidability of elaboration). *Whenever $\rightsquigarrow^*$ is normalizing, the relations of inference, checking and partial inference of Fig. 9 are decidable. When not, they are only semi-decidable.*

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

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* $\Gamma \vdash_{\mathsf{cast}} t \triangleleft T$.

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 typing rules are very similar for both systems, 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. □

Theorem 13 (Uniqueness of elaboration). *Elaboration is 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 moste one* t *such that* $\Gamma \vdash \tilde{t} \triangleleft T \rightsquigarrow t$.

Proof. Like for Theorem 12, uniqueness of elaboration for type inference is defined and proven mutually with similar properties for all typing judgments.

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. Moreover, because of the fixed reduction strategy, the inferred type is unique. The reasoning is similar for the other constrained inference judgments. □

5.3 Back to Omega

Now that GCIC, with its elaboration phase, has been entirely presented, let us come back to the important example of Ω, and precise the behavior described in §3.1. Recall that Ω is the term $\delta\ \delta$, with $\delta := \lambda\, x : ?@i + 1.x\ x$. We leave out the casts present in §2 and 3 for clarity, knowing that they will be introduced by elaboration. We also shift the level of ? up by one, because $?@i + 1$, when elaborated as a type, becomes $?_{\square_i}$. In all three systems, Ω is elaborated (in inference mode) to

$$\Omega' := \delta'\ \left\langle ?_{c_\Pi(i)} \Leftarrow T \to ?_{c_\Pi(i)} \right\rangle\ \delta'$$

where we use $?_j$ instead of $?_{\square_j}$ to ease readability, T is the elaboration of $?_{@i+1}$ as a type, namely $\langle \square_i \Leftarrow ?_{i+1} \rangle\ ?_{?_{i+1}}$, and

$$\delta' := \lambda\, x : T.\, (\langle \mathrm{Germ}_i\, \Pi \Leftarrow T \rangle\ x)\ \left(\left\langle ?_{c_\Pi(i)} \Leftarrow T \right\rangle\ x\right)$$

The only difference at this point between the systems is the fact that this elaboration fails in GCIC$^\uparrow$ and GCIC$^\mathcal{N}$ if i is 0 because in that case $c_\Pi(0)$ is undefined, and thus the first use of x in δ fails to infer under the Π constraint, since its type reduces to $?_0$ (Rule Inf-Prod?).

However, upon reduction we can observe how this Ω' reduces seamlessly in GCIC$^\mathcal{G}$, while having $c_\Pi(i) < i$ makes it fail. Let us first look at Ω' in GCIC$^\mathcal{G}$. To ease readability further, we have compacted multiple successive casts to avoid repeating the same type.

$$\begin{aligned}
\Omega' &\rightsquigarrow^* (\lambda\, x : ?_i.\, (\langle ?_i \to ?_i \Leftarrow ?_i \rangle\ x)\ (\langle ?_i \Leftarrow ?_i \rangle\ x))\, \langle ?_i \Leftarrow ?_i \to ?_i \rangle\ \delta' \\
&\rightsquigarrow^* (\langle ?_i \to ?_i \Leftarrow ?_i \Leftarrow ?_i \to ?_i \rangle\ \delta')\ (\langle ?_i \Leftarrow ?_i \Leftarrow ?_i \to ?_i \rangle\ \delta') \\
&\rightsquigarrow^* (\langle ?_i \to ?_i \Leftarrow ?_i \to ?_i \rangle\ \delta')\ (\langle ?_i \Leftarrow ?_i \to ?_i \rangle\ \delta') \\
&\rightsquigarrow^* (\lambda\, x : ?.\, \langle ?_i \Leftarrow ?_i \rangle\ ((\langle ?_i \to ?_i \Leftarrow ?_i \rangle\ x)\ (\langle ?_i \Leftarrow ?_i \rangle\ x)))\ (\langle ?_i \Leftarrow ?_i \to ?_i \rangle\ \delta')
\end{aligned}$$

And there the reduction has almost looped, apart from the cast $\langle ?_i \Leftarrow ?_i \rangle$ in the first occurrence of δ', which will simply accumulate through reduction, but without hindering the non-normalizing behavior. On the contrary, in GCIC$^\uparrow$ and GCIC$^\mathcal{N}$, the reductions are the same, and go as follows:

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

The error is generated when downcasting from $?_i$ to $?_{i-1}$, which can be seen as a dynamic universe inconsistency.

5.4 Precision and 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 on computation 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, a weaker simulation property—implying DG—can still be shown; fortunately, this is enough to conclude on the graduality of elaboration. The purpose of this section is to establish this property.

As we will see in details in § 6, we can recover graduality for a *semantic* notion of precision defined using a model construction. However, this semantic notion of precision cannot distinguish between convertible terms. As such, it cannot inform us on the behavior of reduction, which is why we cannot rely on it to establish graduality of elaboration.

This section was partly inspired from the proof of DGG by Siek et al. [2015] while of course having 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.

Technically, we need to distinguish between two notions of precision: (i) syntactic precision on terms in GCIC that corresponds to the usual syntactic precision in gradual typing, (ii) structural precision on terms of CastCIC that corresponds to syntactical precision together with a proper account of cast operations. We first concentrate on the notion of precision in CastCIC.

In this section, we only state and discuss the various lemmas and theorems, and differ the reader to Appendix A.2 for the detailed proofs.

Structural precision. As emphasized already, the key property we want to establish is that precision is a simulation for reduction, *i.e.,* less precise terms reduce at least as well as more precise ones. This property guided the quite involved definition we are about to give for structural precision: it is rigid enough to give us the induction hypothesis needed to prove simulation, while being lax enough to be a consequence of a loss of precision before elaboration, which is needed to establish elaboration graduality.

Before diving into the definition, let us note that the definition of structural precision relies on typing, in order to handle casts that might appear or disappear in one term but not the other during reduction. Similarly to $\sim_\alpha$, precision can ignore some casts. But in order to control what kind of casts can be erased, we need to impose some restriction on the types involved in the cast, typically to ensure that these ignored casts would not have raised an error: *e.g.,* we want to prevent $0 \sqsubseteq_\alpha \langle \mathbb{B} \Leftarrow \mathbb{N} \rangle\ 0$. Technically, to allow the definition of structural precision to rely on typing, we need to record the two contexts of the compared terms, to know in which context they shall be typed. 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.

Structural precision, denoted $\mathbb{\Gamma} \vdash t \sqsubseteq_\alpha t'$, is defined in Fig. 10. Its definition uses the auxiliary notion of *definitional precision*, denoted $\mathbb{\Gamma} \vdash t \sqsubseteq_\rightsquigarrow t'$, which is the closure by reduction of structural precision. Namely, $t \sqsubseteq_\rightsquigarrow t'$ means that there exist s and s' such that $t \rightsquigarrow^* s$, $t' \rightsquigarrow^* s'$ 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, the notion of definitional precision is also used in the definition of structural precision, in order to permit some computation in the types,[12] and thus the two notions are mutually defined. We write $\Gamma \sqsubseteq_\alpha \Gamma'$ and $\Gamma \sqsubseteq_\rightsquigarrow \Gamma'$ for the pointwise extensions to contexts.

Let us now explain the rules defining structural precision. Diagonal rules are completely structural, apart from the Diag-Fix rule, where we add typing assumptions to 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, where "the right type" can itself use loss of precision (rule Ukn), and accommodate for a small bit of cumulativity (rule Ukn-Univ), needed because of the typing rule for Π-types that allows some flexibility on the levels of A and B within a fixed level for $\Pi\, x : A.B$. 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 type restriction similar to the unknown. Finally, casts on the right-hand side can be erased 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 erased 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 induces 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

[12] Recall that we are in a dependently typed setting and so the two types involved in a cast may need to be reduced before the cast itself can be reduced.

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

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

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

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

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

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

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

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

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

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

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

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

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

Fig. 10. Structural precision in CastCIC

head, or is some ?. We call these *catch-up* lemmas, as they enable the less precise term to catch-up on the more precise one whose head is already fixed. 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 14, then on other types in Lemma 15, and finally on terms, namely on λ-abstractions in Lemma 16 and inductive constructors in Lemma 17. Each time, the previously proven catch-up lemmas are used to

reduce types in casts appearing in the less precise term, apart from Lemma 14, where the induction hypothesis of the lemma being proven is used instead.

Lemma 14 (Universe catch-up). *Under the hypothesis that* $\Gamma_1 \sqsubseteq_\alpha \Gamma_2$, *if* $\Gamma \vdash \Box_i \sqsubseteq_\rightsquigarrow T'$ *and* $\Gamma_2 \vdash T' \blacktriangleright_\Box \Box_j$, *then either* $T' \rightsquigarrow^* ?_{\Box_j}$ *with* $i+1 \leq j$, *or* $T' \rightsquigarrow^* \Box_i$.

Lemma 15 (Types catchup). *Under the hypothesis that* $\Gamma_1 \sqsubseteq_\alpha \Gamma_2$, *we have the following:*

- *if* $\Gamma \vdash ?_{\Box_i} \sqsubseteq_\alpha T'$ *and* $\Gamma_2 \vdash T' \blacktriangleright_\Box \Box_j$, *then* $T' \rightsquigarrow^* ?_{\Box_j}$ *and* $i \leq j$;
- *if* $\Gamma \vdash \Pi x : A.B \sqsubseteq_\alpha T'$, $\Gamma_1 \vdash \Pi x : A.B \triangleright \Box_i$ *and* $\Gamma_2 \vdash T' \blacktriangleright_\Box \Box_j$ *then either* $T' \rightsquigarrow^* ?_{\Box_j}$ *and* $i \leq j$, *or* $T' \rightsquigarrow^* \Pi x : A'.B'$ *for some* A' *and* B' *such that* $\Gamma \vdash \Pi x : A.B \sqsubseteq_\alpha \Pi x : A'.B'$;
- *if* $\Gamma \vdash I(\mathbf{a}) \sqsubseteq_\alpha T'$, $\Gamma_1 \vdash I(\mathbf{a}) \triangleright \Box_i$ *and* $\Gamma_2 \vdash T' \blacktriangleright_\Box \Box_j$ *then either* $T' \rightsquigarrow^* ?_{\Box_j}$ *and* $i \leq j$, *or* $T' \rightsquigarrow^* I(\mathbf{a}')$ *for some* a' *such that* $\Gamma \vdash I(\mathbf{a}) \sqsubseteq_\alpha I(\mathbf{a}')$.

Lemma 16 (λ-abstraction catch-up). *If* $\Gamma \vdash \lambda x : A.t \sqsubseteq_\alpha s'$, *where* t *is not an error,* $\Gamma_1 \vdash \lambda x : A.t \triangleright \Pi x : A.B$ *and* $\Gamma_2 \vdash s' \blacktriangleright_\Pi \Pi x : A'.B'$, *then* $s' \rightsquigarrow^* \lambda x : A'.t'$ *with* $\Gamma \vdash \lambda x : A.t \sqsubseteq_\alpha \lambda x : A'.t'$.

The previous lemma is the point where the difference between the three variants of CastCIC manifests: it holds in full generality only in CastCIC$^{\mathcal{G}}$ and CastCIC$^{\uparrow}$, but only on terms not containing ? in CastCIC$^{\mathcal{N}}$. Indeed, the fact that $i, j \leq c_\Pi(s_\Pi(i,j))$ 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 those casts never happen. This is important with regard to conservativity, as elaboration produces terms with casts but without ?, and Lemma 16 ensures that for those precision is still a simulation, even in CastCIC$^{\mathcal{N}}$.

A typical example of those differences is the following term t_i

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

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 happened on a function, for instance when elaborating a term like $(\lambda x : \mathbb{N}.\mathsf{suc}(x)) :: ?\,0$. Now t_i always reduces to

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

and this is where the real difference kicks in: if $\mathrm{Germ}_i\,\Pi$ is $\mathtt{err}_{?_{\Box_i}}$ (*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

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

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 CastCIC$^{\mathcal{G}}$, we have $\vdash \lambda x : \mathbb{N}.\mathsf{suc}(x) \sqsubseteq_\alpha t_0$, since $\vdash \mathbb{N} \to \mathbb{N} \triangleright \Box_0$, but $c_\Pi(0) = 0$ so t_0 reduces safely and Lemma 16 holds. In CastCIC$^{\uparrow}$, t_0 errors but because $s_\Pi(0,0) = 1$ we have $\vdash \mathbb{N} \to \mathbb{N} \triangleright \Box_1$, and thus t_0 is not less precise than $\lambda x : \mathbb{N}x\mathsf{suc}(x)$ thanks to the typing restriction in Cast-R, so this error does not contradict Lemma 16. On the contrary, one has $\vdash \lambda x : \mathbb{N}.\mathsf{suc}(x) \sqsubseteq_\alpha t_1$, but since $0 \leq c_\Pi(1)$, t_1 reduces safely. In CastCIC$^{\mathcal{N}}$, however, $\vdash \lambda x : \mathbb{N}.\mathsf{suc}(x) \sqsubseteq_\alpha t_0$ because $s_\Pi(0,0) = 0$, but $c_\Pi(0) < 0$, so the term errors even if it is less precise than the identity – Lemma 16 does not hold in that case.

Lemma 17 (Constructors and inductive error catch-up). *If* $\Gamma \vdash c(\mathbf{a},\mathbf{b}) \sqsubseteq_\alpha s'$, $\Gamma_1 \vdash c(\mathbf{a},\mathbf{b}) \triangleright I(\mathbf{a})$ *and* $\Gamma_2 \vdash s' \blacktriangleright_I I(\mathbf{a}')$, *then either* $s' \rightsquigarrow^* ?_{I(\mathbf{a}')}$ *or* $s' \rightsquigarrow^* c(\mathbf{a}',\mathbf{b}')$ *with* $\Gamma \vdash c(\mathbf{a},\mathbf{b}) \sqsubseteq_\alpha c(\mathbf{a}',\mathbf{b}')$.

Similarly, if $\Gamma \vdash ?_{I(\mathbf{a})} \sqsubseteq_\alpha s'$, $\Gamma_1 \vdash ?_{I(\mathbf{a})} \triangleright I(\mathbf{a})$ *and* $\Gamma_2 \vdash s' \blacktriangleright_I I(\mathbf{a}')$, *then* $s' \rightsquigarrow^* ?_{I(\mathbf{a}')}$ *with* $\Gamma \vdash I(\mathbf{a}) \sqsubseteq_\rightsquigarrow I(\mathbf{a}')$.

Note that for Lemma 17, we need to deal with unknown terms specifically, which is not necessary for Lemma 16 because the unknown term in a Π-type reduces to a λ-abstraction.

Simulation. We finally come to the main property of this section, the advertised simulation. As discussed above, the proposition holds in CastCIC$^{\mathcal{G}}$, CastCIC$^{\uparrow}$ and for terms without **?** in CastCIC$^{\mathcal{N}}$. Remark that the simulation property needs to be stated mutually for structural and definitional precision, but it is really informative for structural precision as definitional precision is somehow a simulation by construction.

Theorem 18 (Simulation of reduction). *Let* $\Gamma_1 \sqsubseteq_{\rightsquigarrow} \Gamma_2$, $\Gamma_1 \vdash t \triangleright T$, $\Gamma_2 \vdash u \triangleright U$ *and* $t \rightsquigarrow^* t'$.

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

Proof sketch. The case of definitional precision follows by confluence of the reduction. For the case of structural precision, the hardest point is to simulate β and ι redexes, that is terms of the shape $\mathtt{ind}_I(c(\mathbf{a}), z.P, f.\mathbf{y}.\mathbf{t})$. This is where we use Lemmas 16 and 17, 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 s. Finally, the restriction to the gradual systems show up again when considering the reduction rules with germs are involved, where the synchronization between s_Π and c_Π is required to conclude. □

From this simulation property, we get as direct corollaries the properties we sought to handle reduction (Corollary 19) and consistency (Corollary 20) in elaboration. Again those corollaries are true in GCIC$^{\mathcal{G}}$, GCIC$^{\uparrow}$ and for terms in GCIC$^{\mathcal{N}}$ containing no **?**.

Corollary 19 (Reduction and types). *Let* Γ, T *and* T' *be such that* $\Gamma_1 \vdash T \blacktriangleright_\square \square_i$, $\Gamma_2 \vdash T' \blacktriangleright_\square \square_j$, $\Gamma \vdash T \sqsubseteq_\alpha T'$. *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* $\Gamma \vdash \Pi x : A.B \sqsubseteq_\alpha \Pi x : A'.B'$;
- *if* $T \rightsquigarrow^* I(\mathbf{a})$ *then either* $T' \rightsquigarrow^* ?_{\square_j}$ *with* $i \leq j$, *or* $T' \rightsquigarrow^* I(\mathbf{a}')$ *and* $\Gamma \vdash I(\mathbf{a}) \sqsubseteq_\alpha I(\mathbf{a}')$.

Proof. Simulate the reductions of T by using Theorem 18, then use Lemmas 14 and 15 to conclude. Note that head reductions are simulated using head reductions in Theorem 18, and the reductions of Lemmas 14 and 15 are also head reductions. Thus the corollary still holds when fixing weak-head reduction as a reduction strategy. □

Corollary 20 (Monotonicity of consistency). *If* $\Gamma \vdash T \sqsubseteq_\alpha T'$, $\Gamma \vdash S \sqsubseteq_\alpha S'$ *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 18, we can simulate these reductions to get some U' and V' such that $T' \rightsquigarrow^* U'$ and $S' \rightsquigarrow^* V'$, and also $\Gamma_1 \vdash U \sqsubseteq_\alpha U'$ and $\Gamma_1 \vdash V \sqsubseteq_\alpha V'$. 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 conservativity and elaboration graduality for 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 A.3.

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

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

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

Fig. 11. GCIC: Syntactic precision

Conservativity. Elaboration systematically inserts casts during checking, thus even static terms are not elaborated to themselves. Therefore we use a (partial) erasure function ε, that (partially) 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 4 (Equiprecision). *Two terms s and t are equiprecise in a context Γ, denoted $\Gamma \vdash s \sqsupseteq\sqsubseteq_{\alpha} t$ if both $\Gamma \vdash s \sqsubseteq_{\alpha} t$ and $\Gamma \vdash t \sqsubseteq_{\alpha} s$.*

Definition 5 (Erasure, erasability). *Erasure ε is a partial function from the syntax of* CastCIC *to the syntax of* CIC*, which is undefined on* ? *and* `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 Γ we say that a term t is erasable if $\varepsilon(t)$ is defined, well-typed, and equiprecise to t. 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, typability 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 21 (Conservativity). *Let t be a static term. If $\vdash_{\mathsf{CIC}} t \triangleright T$ for some type T, then there exists t' and T' such that $\vdash t \rightsquigarrow t' \triangleright T'$, and moreover $\varepsilon(t') = t$ and $\varepsilon(T') = T$. Conversely if $\vdash t \rightsquigarrow t' \triangleright T'$ for some t' and T', then $\vdash t \triangleright \varepsilon(T')$.*

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 18), 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 can prove that constraint reductions ($\blacktriangleright_{\Pi}$, $\blacktriangleright_{\square}$ and $\blacktriangleright_{I}$) 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* $\sqsubseteq^{\mathrm{G}}_{\alpha}$, defined in Fig. 11. It is generated by a single non trivial rule $\tilde{t} \sqsubseteq^{\mathrm{G}}_{\alpha} ?@i$, and congruence rules for all term formers.

Distinctively to the simply-typed setting, the presence of multiple types $?$, one for each universe level i, requires an additional hypothesis relating elaboration and precision. We say that two judgments $\tilde{t} \sqsubseteq^{\mathrm{G}}_{\alpha} ?@i$ and $\Gamma \vdash \tilde{t} \rightsquigarrow t \triangleright T$ are *universe adequate* if the universe level j given by the induced judgment $\Gamma \vdash T \blacktriangleright_{\square} \square_j$ satisfies $i = j$. More generally, $\tilde{t} \sqsubseteq^{\mathrm{G}}_{\alpha} \tilde{s}$ and $\vdash \tilde{t} \rightsquigarrow t \triangleright T$ are *universe adequate* if any subterm $\tilde{t}_0$ of $\tilde{t}$ inducing judgments $\tilde{t}_0 \sqsubseteq^{\mathrm{G}}_{\alpha} ?@i$ and $\Gamma_0 \vdash \tilde{t}_0 \rightsquigarrow t \triangleright T$ is universe adequate. Note that this extraneous technical assumption on universe levels is not needed if we use typical ambiguity, as described in §5.2, where the universe level is not given explicitly. Elaboration graduality holds in the two systems satisfying $\mathcal{G}$, *i.e.,* $\mathrm{GCIC}^{\mathcal{G}}$ and $\mathrm{GCIC}^{\uparrow}$.

Theorem 22 (Elaboration Graduality / SGG). *In* $\mathrm{GCIC}^{\mathcal{G}}$ *and* $\mathrm{GCIC}^{\uparrow}$, *if* $\tilde{t} \sqsubseteq^{\mathrm{G}}_{\alpha} \tilde{s}$ *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 hypothesis.

Here again the technical difficulties arise in the rules involving reduction. This is where Corollary 19 is useful, proving that the less structurally precise term obtained by induction in a constrained inference reduces to a less precise type, and thus that either the rule can still be used; alternatively one has to trade a Inf-Ukn, 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 was not. Similarly Corollary 20 proves that in the checking rule the less precise types are still consistent. □

DGG. Following Siek et al. [2015], using the fact that structural precision is a simulation (Theorem 18), we can prove the DGG for $\mathrm{CastCIC}^{\mathcal{G}}$ and $\mathrm{CastCIC}^{\uparrow}$.

Theorem 23 (DGG for $\mathrm{CastCIC}^{\mathcal{G}}$ and $\mathrm{CastCIC}^{\uparrow}$). *Let* $\Gamma \vdash t \triangleright A$, $\Gamma \vdash u \triangleright A$.
If $\Gamma \mid \Gamma \vdash t \sqsubseteq_{\alpha} u$ *then* $t \sqsubseteq^{obs} u$.

Proof. Let $C : (\Gamma \vdash A) \Rightarrow (\vdash \mathbb{B})$ closing over all free variables. By all 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 a value, an error, 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 a value v that is either a constructor of $\mathbb{B}$ or $?_{\mathbb{B}}$, then by the catch-up Lemma 17, $C[u]$ either reduces to the same constructor, or to $?_{\mathbb{B}}$. In particular, it cannot diverge or reduce to an error. □

As observed in §2.5, there is no hope to prove graduality ($\mathcal{G}$)—that is, that structural precision induces ep-pairs—directly in the syntactic approach that we have used so far. Therefore, we defer the proof of $\mathcal{G}$ for $\mathrm{CastCIC}^{\uparrow}$ to the next section, where the notion of propositional precision based on the monotone model is introduced to solve this issue. For $\mathrm{CastCIC}^{\mathcal{G}}$, the proof cannot be based on the monotone model as the cast operation is not well-founded (hence the presence of non-terminating terms). We thus turn to a Scott-style interpretation of $\mathrm{CastCIC}^{\mathcal{G}}$ using ω-cpos to derive graduality for the diverging variant (§6.7).

6 REALIZING CastCIC AND GRADUALITY

To prove normalization of $\mathrm{CastCIC}^{\mathcal{N}}$ and $\mathrm{CastCIC}^{\uparrow}$, 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 to reduce the normalization of CastCIC to the normalization of the host theory (§6.1).

Then, to prove graduality of $\mathrm{CastCIC}^{\uparrow}$, we build a more elaborate *monotone model* inducing a precision relation well-behaved with respect to conversion. Following generalities about the interpretation of CIC in poset in §6.2, we describe the construction of a monotone unknown type $?$ 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 $\mathrm{CastCIC}^{\uparrow}$ (§6.6). In both the discrete and monotone case, the parameters

$$\star_{\pi\, A\, B} := \lambda\, x : \mathrm{El}\, A.\star_{B\, x} : \Pi\, A\, B \qquad ?_{\mathbb{U}_j} := ?^j : \mathbb{U}_j \qquad ?_? := ?_{\tilde{\Sigma}\, \mathbb{H}\, \mathrm{Germ}} : \tilde{\Sigma}\, \mathbb{H}\, \mathrm{Germ} \qquad ?_{\mathbb{nat}} := ?_{\mathbb{N}} : \tilde{\mathbb{N}}$$

$$\star_{\maltese^j} := * : \top \qquad \mathtt{err}_{\mathbb{U}_j} := \maltese^j : \mathbb{U}_j \qquad \mathtt{err}_? := \maltese_{\tilde{\Sigma}\, \mathbb{H}\, \mathrm{Germ}} : \tilde{\Sigma}\, \mathbb{H}\, \mathrm{Germ} \qquad \mathtt{err}_{\mathbb{nat}} := \maltese_{\mathbb{N}} : \tilde{\mathbb{N}}$$

Fig. 12. Realization of exceptions ($\star$ stands for either ? or err)

$c_\Pi(-)$ 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, CastCIC$^{\mathcal{G}}$, we describe at the end of this section a model based on ω-complete partial orders, extending the well-known Scott's model [Scott 1976] to CastCIC$^{\mathcal{G}}$ (§6.7).

The models embed into a variant of CIC extended with induction-recursion [Dybjer and Setzer 2003] as well as function extensionality for the monotone model, whose judgments will be denoted with $\vdash_{\mathbb{IR}}$. We use Agda [Norell 2009] as a practical counterpart to typecheck the components of the models[13] and assume that the implementation satisfies standard metatheoretical properties,[14] namely subject reduction and strong normalization.

6.1 Discrete Model of CastCIC

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

- exceptions, following the approach of ExTT [Pédrot and Tabareau 2018], in order to interpret both ? and err; and
- case-analysis on types [Boulier et al. 2017] to define the cast operator.

Exceptions. Following the general pattern of ExTT, we interpret each inductive type I by an inductive type $\tilde{I}$ featuring all constructors of I and extended with two new constructors $?_{\tilde{I}}$ and $\maltese_{\tilde{I}}$, corresponding respectively to $?_I$ and $\mathtt{err}_I$ of CastCIC. Figure 16 on the left describes the leading example of natural numbers $\tilde{\mathbb{N}}$ with 4 constructors. In the rest of this section, we only illustrate inductive types on natural numbers. The definition of exceptions $?_A, \mathtt{err}_A$ at an arbitrary type A then follows by case analysis on a code for A in Fig. 12. On types defined inductively – $\mathbb{U}$, $\tilde{\Sigma}\, \mathbb{H}\, \mathrm{Germ}$, $\tilde{\mathbb{N}}$ – we use the newly added constructors. On functions, the exceptions are defined by re-raising the exception at the codomain in a pointwise fashion, whereas on the error type $\maltese$ they are forced to take the only value $* : \top$ of its interpretation as a type (see the description of El below).

[13] We detail the correspondence between the notions developed in the following sections and the formal development in Agda [Bertrand et al. 2020]. 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 can endow a poset structure on the translation of each type formers from CastCIC: natural numbers in `nat.agda`, booleans in `bool.agda`, dependent product in `pi.agda`. The definition of the monotone unknown type $\tilde{?}$ is more involved since we need to use a quotient (that we axiomatize together with a rewriting rule in `Unknown/Quotient.agda`) and is defined in the subdirectory `Unknown/`.

Finally, all these building blocks are put together when assembling the inductive-recursive hierarchies of universes (`UnivPartial.agda`, `DiscreteModelPartial.agda` and `DiscreteModelTotal.agda`).

[14] These properties are conjectured but are still open problems to our knowledge.

$$\frac{A \in \mathbb{U}_i \qquad B \in \mathrm{El}\, A \to \mathbb{U}_j}{\pi\, A\, B \in \mathbb{U}_{s_\Pi(i,j)}} \qquad \frac{j < i}{\mathfrak{u}_j \in \mathbb{U}_i} \qquad \mathfrak{nat} \in \mathbb{U}_i \qquad ? \in \mathbb{U}_i \qquad \maltese \in \mathbb{U}_i$$

$$\mathrm{El}\,(\pi\, A\, B) = \Pi(a : \mathrm{El}\, A).\, \mathrm{El}(B\, a) \qquad \mathrm{El}\,\mathfrak{u}_j = \mathbb{U}_j \qquad \mathrm{El}\,\mathfrak{nat} = \tilde{\mathbb{N}} \qquad \mathrm{El}\,? = \tilde{\Sigma}(h : \mathbb{H}).\, \mathrm{Germ}\, h \qquad \mathrm{El}\,\maltese = \top$$

Fig. 13. Inductive-recursive encoding of the discrete universe hierarchy

Universe and type-case. Case analysis on types is obtained through an explicit inductive description of the universes, translating $\square_i$ to a type of codes $\mathbb{U}_i$ described in Fig. 13. It contains dependent product (π), universes ($\mathfrak{u}$), inductives (*e.g.,* $\mathfrak{nat}$) as well as codes $?$ for the unknown type and $\maltese$ for the error type. Accompanying the inductive definition of $\mathbb{U}_i$, the recursively defined decoding function El provides a semantics for these codes. The error $\maltese$ is decoded to the type $\top$ containing a unique element $* \equiv ?_{\maltese} \equiv \mathtt{err}_{\maltese}$. The unknown $?$ is decoded to the extended dependent sum whose elements are either:

- one of the two freely added constructors $?_{\tilde{\Sigma}}, \maltese_{\tilde{\Sigma}}$ following the scheme of inductive types;
- or a dependent pair $(h; t)$ of a head constructor $h \in \mathbb{H}_i$ together with an element $t \in \mathrm{Germ}_i\, h$ where we stratify the head constructors $\mathbb{H}$ and germs (see Fig. 4) according to the universe level i, and just adapt the definition to the target type theory $\vdash_{\mathrm{IR}}$.

Crucially, the code for Π-types depends on the choice of parameter for $s_\Pi(i, j)$. For the choice of parameters corresponding to the system CastCIC$^{\mathcal{G}}$, the inductive-recursive definition of $\mathbb{U}_i$ is ill-founded: since $c_\Pi(s_\Pi(i, i)) = s_\Pi(i, i)$, we can inject $\mathrm{Germ}_{s_\Pi(i,i)}\Pi = \mathrm{El}_{s_\Pi(i,i)}? \to \mathrm{El}_{s_\Pi(i,i)}?$ into $\mathrm{El}_{s_\Pi(i,i)}?$ and project back in the other direction thanks to errors, exhibiting an embedding-retraction suitable to interpret the untyped λ-calculus and in particular Ω. In the Agda implementation, we deactivate the termination-checker on the definition of the universe, thus effectively working in a partial, inconsistent type theory.

In order to maintain normalization, the construction of the unknown type and the universe thus needs to be stratified, which is possible when $c_\Pi(s_\Pi(i, i)) < s_\Pi(i, i)$. This strict inequality occurs for both CastCIC$^{\mathcal{N}}$ and CastCIC$^{\uparrow}$. We proceed by strong induction on the universe level, and note that thanks to the level gap, the decoding $\mathrm{El}\,?^i$ 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 $\mathbb{U}_i$. We can then define the rest of the universe $\mathbb{U}_i$ and the decoding function El at level i without trouble.

Note that results of Palmgren [1998] indicate that we couldn't apply this inductive-recursive translation to an impredicative universe in a consistent and strongly-normalizing meta theory $\vdash_{\mathrm{IR}}$.

Cast. Equipped with exceptions and the capability to do induction on types, we define $\mathtt{cast} : \Pi(A : \mathbb{U}_i)(B : \mathbb{U}_j).A \to B$ in Fig. 14 by induction on the universe levels and case analysis on the codes of the types A, B. In the total setting, the definition of `cast` is well-founded: each recursive call happens either at a strictly smaller universe (the two cases for π) or on a strict subterm of the term being cast (case of inductive/$\mathfrak{nat}$ and $?$). Note that each defining equations of `cast` corresponds straightforwardly to the a reduction rule of Fig. 5.

Discrete translation. The translations $[-]$ and $[\![-]\!]$ from CastCIC to CIC+IR is defined by induction on the syntax of terms and types in Fig. 15. The following theorem shows that it is a syntactic model in the sense of [Boulier et al. 2017].

Theorem 24 (Discrete syntactic model). *The translation preserves conversion and typing derivations:*

(1) if $\Gamma \vdash_{\mathtt{cast}} t \rightsquigarrow u$ *then* $[\![\Gamma]\!] \vdash_{\mathrm{IR}} [t] \rightsquigarrow_{\mathrm{IR}}{}^{+} [u]$, *in particular* $[\![\Gamma]\!] \vdash_{\mathrm{IR}} [t] \equiv [u]$,

$$
\begin{array}{lcll}
\mathtt{cast}\ (\pi\, A^{\mathrm{d}}\, A^{\mathrm{c}})\ (\pi\, B^{\mathrm{d}}\, B^{\mathrm{c}})\ f & := & \lambda\, b : \mathrm{El}\, B^{\mathrm{d}}.\ \mathrm{let}\, a = \mathtt{cast}\ B^{\mathrm{d}}\ A^{\mathrm{d}}\ b\ \mathrm{in}\ \mathtt{cast}\ (A^{\mathrm{c}}\ a)\ (B^{\mathrm{c}}\ b)\ (f\ a) & \\
\mathtt{cast}\ (\pi\, A^{\mathrm{d}}\, A^{\mathrm{c}})\ ?^{i}\ f & := & (\Pi; \mathtt{cast}\ (\pi\, A^{\mathrm{d}}\, A^{\mathrm{c}})\ (\mathrm{Germ}_i\ \Pi)\ f) & \text{if } \mathrm{Germ}_i\ \Pi \neq \maltese \\
\mathtt{cast}\ (\pi\, A^{\mathrm{d}}\, A^{\mathrm{c}})\ X\ f & := & \maltese_X & \text{otherwise}
\end{array}
$$

$$
\begin{array}{lcllcll}
\mathtt{cast}\ \mathfrak{nat}\ \mathfrak{nat}\ n & := & x & \mathtt{cast}\ \mathfrak{u}_j\ \mathfrak{u}_j\ A & := & A & \\
\mathtt{cast}\ \mathfrak{nat}\ ?\ n & := & (\mathbb{N}; x) & \mathtt{cast}\ \mathfrak{u}_j\ ?^{i}\ A & := & (\Box_j; A) & \text{if } j < i \\
\mathtt{cast}\ \mathfrak{nat}\ X\ n & := & \maltese_X & \mathtt{cast}\ \mathfrak{u}_j\ X\ A & := & \maltese_X & \text{otherwise} \\
\mathtt{cast}\ \maltese\ Z\ * & := & \maltese_Z & \mathtt{cast}\ ?^{i}\ Z\ (c; x) & := & \mathtt{cast}\ (\mathrm{Germ}_i\ c)\ Z\ x & \\
 & & & \mathtt{cast}\ ?^{i}\ Z\ ?_? & := & ?_Z & \\
 & & & \mathtt{cast}\ ?^{i}\ Z\ \maltese_? & := & \maltese_Z & \text{otherwise}
\end{array}
$$

Fig. 14. Implementation of cast (discrete models)

$$
\begin{array}{lcllcl}
[\![\cdot]\!] & := & \cdot & [\![\Gamma, x : A]\!] & := & [\![\Gamma]\!], x : [\![A]\!] \\
[\![A]\!] & := & \mathrm{El}\ [A] & [\mathbb{N}] & := & \mathfrak{nat} \\
 & & & [0] & := & 0 \\
[x] & := & x & [\mathtt{suc}] & := & \mathtt{suc} \\
[\Box_i] & := & \mathfrak{u}_i & [\mathtt{ind}_{\mathbb{N}}]\ P\ h_0\ h_{\mathtt{suc}} & := & \mathtt{ind}_{\tilde{\mathbb{N}}}\ P\ h_0\ h_{\mathtt{suc}}\ ?_{P\, ?_{\tilde{\mathbb{N}}}}\ \mathtt{err}_{(P\, \mathtt{err}_{\tilde{\mathbb{N}}})} \\
[\Pi\, x : A.B] & := & \pi\ [A]\ (\lambda\, x : [\![A]\!].[B]) & [?_A] & := & ?_{[A]} \\
[t\ u] & := & [t]\ [u] & [\mathtt{err}_A] & := & \mathtt{err}_{[A]} \\
[\lambda\, x : A.t] & := & \lambda\, x : [\![A]\!].[t] & [\langle B \Leftarrow A \rangle\ t] & := & \mathtt{cast}\ [A]\ [B]\ [t]
\end{array}
$$

Fig. 15. Discrete translation from CastCIC to CIC +IR

(2) if $\Gamma \vdash_{\mathsf{cast}} t : A$ *then* $[\![\Gamma]\!] \vdash_{\mathrm{IR}} [t] : [\![A]\!]$.

Proof. For the first part, 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 a cast. The preservation for these hold directly by a careful inspection once we observe that the CastCIC stuck term $\langle ?_{\Box_i} \Leftarrow \mathrm{Germ}_i\ h \rangle\ t$ is in one-to-one correspondence with the one-step reduced form of its translation $(h; [t]) : \tilde{\Sigma}(h : \mathbb{H}_i).\, \mathrm{Germ}_i\ h$. The second part is proved by a direct induction on the typing derivation of $\Gamma \vdash_{\mathsf{cast}} t : A$, using that exceptions and casts are well-typed $\vdash_{\mathrm{IR}} ?, \mathtt{err} : \Pi(A : \mathbb{U}_i)\, \mathrm{El}\, A$, $\vdash_{\mathrm{IR}} \mathtt{cast} : \Pi(A : \mathbb{U}_i)(B : \mathbb{U}_i) \to \mathrm{El}\, A \to \mathrm{El}\, B$, and relying on assertion (1) to handle the conversion rule. □

As explained in Theorem 9, Theorem 24 implies in particular that CastCIC$^{\uparrow}$ and 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 cast are sound, *i.e.,* a graduality theorem. To overcome this limitation, we develop a monotone model where, by construction, each type A comes equipped with an order structure $\sqsubseteq^{A}$—a reflexive, transitive, antisymmetric and and proof-irrelevant relation—modeling precision between terms. 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 can not be defined for CastCIC$^{\mathcal{N}}$ because this type theory lakes graduality.

$$0, ?_{\tilde{\mathbb{N}}}, \maltese_{\tilde{\mathbb{N}}} \in \tilde{\mathbb{N}} \qquad \frac{n \in \tilde{\mathbb{N}}}{\mathsf{suc}\, n \in \tilde{\mathbb{N}}} \qquad \Big| \qquad 0 \sqsubseteq^{\tilde{\mathbb{N}}} 0 \qquad \maltese_{\tilde{\mathbb{N}}} \sqsubseteq^{\tilde{\mathbb{N}}} n \qquad 0 \sqsubseteq^{\tilde{\mathbb{N}}} ?_{\tilde{\mathbb{N}}} \qquad ?_{\tilde{\mathbb{N}}} \sqsubseteq^{\tilde{\mathbb{N}}} ?_{\tilde{\mathbb{N}}} \qquad \frac{n \sqsubseteq^{\tilde{\mathbb{N}}} m}{\mathsf{suc}\, n \sqsubseteq^{\tilde{\mathbb{N}}} \mathsf{suc}\, m} \qquad \frac{n \sqsubseteq^{\tilde{\mathbb{N}}} ?_{\tilde{\mathbb{N}}}}{\mathsf{suc}\, n \sqsubseteq^{\tilde{\mathbb{N}}} ?_{\tilde{\mathbb{N}}}}$$

Fig. 16. Order structure on extended natural numbers

As an illustration, the order on extended natural numbers (Fig. 16) makes $\maltese_{\tilde{\mathbb{N}}}$ the smallest element and $?_{\tilde{\mathbb{N}}}$ the biggest element. The "standard" natural numbers then lay in between failure and indeterminacy, but are never related to each other by precision, since $\sqsubseteq^{\tilde{\mathbb{N}}}$ must coincide with CIC's conversion on static closed natural numbers so that conservativity with respect to CIC is maintained.

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* $\uparrow : A \to B$ and a *downcast* $\downarrow : B \to A$ satisfying a handful of properties with gradual guarantees as a corollary.

Definition 6 (Embedding-projection pairs). *An ep-pair $d : A \lhd B$ between posets A, B consists of*

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

Note that equiprecision of the retraction becomes here an equality because of antisymmetry. Under these conditions, $\uparrow_d : A \hookrightarrow B$ is injective, $\downarrow_d : B \twoheadrightarrow A$ is surjective and both preserve bottom elements, explaining that we call $d : A \lhd B$ an embedding-projection pair. Note that to highlight the connection between ep-pairs and parametricity, we present a definition of ep-pairs which makes use of a relation. Assuming propositional and 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.

Posetal families. By monotonicity, a family $B : A \to \square$ 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 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 depends 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 the details.

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

(1) $\maltese_I$ is below any element,
(2) $?_I \sqsubseteq^I ?_I$,
(3) $c\ \mathbf{t} \sqsubseteq^I ?_I$ whenever $t_i \sqsubseteq^{X_i} ?_{X_i}$ for all i
(4) each constructor c is monotone with respect to the order on its arguments.

Similarly to dependent product, an ep-pair $\mathbf{X} \lhd \mathbf{X}'$ between the parameters of an inductive type I induces an ep-pair $I\,\mathbf{X} \lhd I\,\mathbf{X}'$.

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

In order to build the interpretation $\overline{?}$ of the unknown type in the monotone model, we equip the extended dependent sum $\tilde{\Sigma}(h : \mathbb{H}_i).\,\mathrm{Germ}_i\ h$ from the discrete model with the precision relation generated by the rules:

$$\maltese_? \sqsubseteq z \qquad ?_? \sqsubseteq ?_? \qquad \frac{x \sqsubseteq^{\mathrm{Germ}\ h} x'}{(h;x) \sqsubseteq (h;x')} \qquad \frac{x \sqsubseteq^{\mathrm{Germ}\ h} ?_{\mathrm{Germ}\ h}}{(h;x) \sqsubseteq ?_?} \tag{1}$$

These rules ensure that the errors $\maltese_?$ and $?_?$ are respectively the smallest and biggest elements of $\tilde{\Sigma}(h : \mathbb{H}_i).\,\mathrm{Germ}_i\ h$. Non-error 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 $\mathrm{Germ}\ h$.

In order to globally satisfy $\mathcal{G}$, $\overline{?}$ should admit an ep-pair $d_h : \mathrm{Germ}_i\ h \lhd ?_{\mathrm{u}_i}$ whenever we have a head constructor $h \in \mathbb{H}_i$ such that $\mathrm{Germ}_i\ h \sqsubseteq ?_{\mathrm{u}_i}$. Embedding an element $x \in \mathrm{Germ}\ h$ by $\uparrow_{d_h} x = (h;x)$ and projecting out of $\mathrm{Germ}\ h$ by the following equations form a reasonable candidate.

$$\downarrow_{d_h} (h',x) = \begin{cases} x & \text{if } h = h' \\ \maltese_{\mathrm{Germ}\ h} & \text{otherwise} \end{cases} \qquad \downarrow_{d_h} ? = ?_{\mathrm{Germ}\ h}, \qquad \downarrow_{d_h} \maltese = \maltese_{\mathrm{Germ}\ h}.$$

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

$$\maltese_{\mathrm{Germ}\ h} \sqsubseteq^{\mathrm{Germ}\ h} \downarrow_{d_h} \maltese \qquad \Longleftrightarrow \qquad (h, \maltese_{\mathrm{Germ}\ h}) = \uparrow_{d_h} \maltese_{\mathrm{Germ}\ h} \sqsubseteq^{\overline{?}} \maltese_{\overline{?}}$$

Since $\sqsubseteq^{\overline{?}}$ should be antisymmetric, this is possible only if $(h, \maltese_{\mathrm{Germ}\ h})$ and $\maltese_{\overline{?}}$ are identified in $\overline{?}$.

Finally, we *define* $\overline{?}$ to be the quotient of $\tilde{\Sigma}(h : \mathbb{H}).\,\mathrm{Germ}\ h$ identifying $\maltese_{\overline{?}}$ and $(h, \maltese_{\mathrm{Germ}\ h})$. The precision relation described above descends on the quotient as well as $\uparrow_{d_h}$ and $\downarrow_{d_h}$, effectively giving rise to the required ep-pair d_h.

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 $\mathbb{U}_i$ together with a decoding function $\mathrm{El} : \mathbb{U}_i \to \square$ presented in Fig. 17. The precision relation $\sqsubseteq : \mathbb{U}_i \to \mathbb{U}_j \to \square$ presented below is an order (Theorem 25) 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, $\maltese$ is placed at the bottom. More interestingly, the derivation of a precision proof $A \sqsubseteq ?_?$ provides a unique decomposition of A through iterated germs directed by the relevant head constructors. For instance, in the gradual systems where $c_\Pi(s_\Pi(i,j)) = \max(i,j)$, the derivation of $(\mathfrak{nat} \to \mathfrak{nat}) \to \mathfrak{nat} \sqsubseteq ?$ canonically decomposes as:

$$(\mathfrak{nat}\to\mathfrak{nat})\to\mathfrak{nat} \quad \sqsubseteq \quad (?\to?)\to\mathfrak{nat} \quad \sqsubseteq \quad ?\to? \quad \sqsubseteq \quad ?$$

Monotone universes $\mathbb{U}_i$ and decoding function $\mathrm{El} : \mathbb{U}_i \to \square$ (cases distinct from Fig. 13)

$$\frac{A \in \mathbb{U}_i \qquad B \in \Pi^{\mathrm{mon}}(a : \mathrm{El}\, A).\, \mathbb{U}_j}{\pi\, A\, B \in \mathbb{U}_{s_\Pi(i,j)}} \qquad \mathrm{El}\,(\pi\, A\, B) = \Pi^{\mathrm{mon}}(a : \mathrm{El}\, A).\, \mathrm{El}(B\, a) \qquad \mathrm{El}\,\mathsf{?} = \overline{?}$$

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

$$\frac{}{\mathfrak{nat}^i \sqsubseteq \mathfrak{nat}^j}\ \mathfrak{nat}\text{-}\sqsubseteq \qquad \frac{}{\mathsf{?}^i \sqsubseteq \mathsf{?}^j}\ \mathsf{?}\text{-}\sqsubseteq \qquad \frac{}{\mathfrak{u}^i_k \sqsubseteq \mathfrak{u}^j_k}\ \mathfrak{u}\text{-}\sqsubseteq \qquad \frac{}{\maltese^i \sqsubseteq A^j}\ \maltese\text{-}\sqsubseteq$$

$$\frac{A \sqsubseteq A' \qquad a : \mathrm{El}\, A, a' : \mathrm{El}\, A', a_\epsilon : a\ {}_A\sqsubseteq_{A'}\ a' \vdash B\, a \sqsubseteq B'\, a'}{\pi\, A\, B \sqsubseteq \pi\, A'\, B'}\ \pi\text{-}\sqsubseteq \qquad \frac{h = \mathrm{head}\, A^i \in \mathbb{H}_i \qquad A \sqsubseteq \mathrm{Germ}_j\, h}{A^i \sqsubseteq \mathsf{?}^j}\ \mathbb{H}\text{-}\sqsubseteq$$

Precision on terms ${}_A\sqsubseteq_B$ (presupposing $A \sqsubseteq B$)

$$*\ \maltese\sqsubseteq_A\ a \qquad \frac{x \sqsubseteq^A y}{x\ {}_A\sqsubseteq_A\ y} \qquad \frac{a\ {}_A\sqsubseteq_{\mathrm{Germ}(\mathrm{head}\, A)}\ x}{a\ {}_A\sqsubseteq_{\mathsf{?}}\ [\mathrm{head}\, A, x]} \qquad \frac{x \sqsubseteq^{\mathrm{Germ}_i\, h}\ x'}{[h, x]\ {}_{\mathsf{?}^i}\sqsubseteq_{\mathsf{?}^j}\ [h, x']}$$

$$\frac{a : \mathrm{El}\, A, a' : \mathrm{El}\, A', a_\epsilon : a\ {}_A\sqsubseteq_{A'}\ a' \vdash f\ a\ {}_{B\, a}\sqsubseteq_{B'\, a'}\ f'\ a'}{f\ {}_{\pi\, A\, B}\sqsubseteq_{\pi\, A'\, B'}\ f'} \qquad a\ {}_A\sqsubseteq_{\mathsf{?}}\ \mathsf{?}_{\mathsf{?}} \qquad \maltese_A\ {}_A\sqsubseteq_{\mathsf{?}}\ z$$

Fig. 17. Monotone universe of codes and precision

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[15]. Such a derivation of precision $A \sqsubseteq B$ gives rise through decoding to ep-pairs $\mathrm{El}_\varepsilon\,(A \sqsubseteq B) : A \lhd B$, with underlying relation noted ${}_A\sqsubseteq_B : \mathrm{El}\, A \to \mathrm{El}\, B \to \square$.

It is interesting to observe what happens when $c_\Pi(s_\Pi(i,j)) \neq \max(i,j)$, that is in the non-gradual setting of CastCIC$^{\mathcal{N}}$, on an example,:

$$\mathfrak{nat} \to \mathfrak{nat} \quad \not\sqsubseteq \quad \maltese = \mathrm{Germ}_0\, \Pi \quad \sqsubseteq \quad \mathsf{?}$$

So $\mathfrak{nat} \to \mathfrak{nat}$ is not lower than $\mathsf{?}$ in that setting.

One crucial point of the monotone model is the mutual definition of codes $\mathbb{U}_i$ together with the precision relation, particularly salient on codes for Π-types: in $\pi\, A\, B$, $B : \mathrm{El}\, A \to \mathbb{U}_i$ is a monotone function with respect to the order on $\mathrm{El}\, A$ and the precision on $\mathbb{U}_i$. 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 $f\ {}_{\pi\, A\, B}\sqsubseteq_{\pi\, A\, B}\ f$ if and only f is *monotone*.

Theorem 25 (Properties of the universe hierarchy).

(1) $\sqsubseteq$ is reflexive, transitive, antisymmetric and irrelevant so that $(\mathbb{U}_i, \sqsubseteq)$ is a poset.
(2) $\mathbb{U}_i$ has a bottom element $\maltese^i$ and a top element $\mathsf{?}^i$; in particular, $A \sqsubseteq \mathsf{?}^i$ for any $A : \mathbb{U}_i$.
(3) $\mathrm{El} : \mathbb{U}_i \to \square$ is a family of posets over $\mathbb{U}_i$ with underlying relation ${}_A\sqsubseteq_B$ whenever $A \sqsubseteq B$.
(4) $\mathbb{U}_i$ and $\mathrm{El}\, A$ for any $A : \mathbb{U}_i$ verify UIP[16]: the equality on these types is irrelevant.

[15] This decomposition is already present in [New and Ahmed 2018] and to be contrasted with the approaches based on AGT [Garcia et al. 2016] that tend to pair values with most static witness of their type, i.e. canonical path of minimal length.
[16] Uniqueness of Identity Proofs; in HoTT parlance, $\mathbb{U}_i$ and $\mathrm{El}\, A$ are hSets.

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. We only prove point (1) and refer to the Agda development for detailled formal proofs.

For reflexivity, all cases are immediate but for $\pi\, A\, B$: 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 $\mathbb{H}$-$\sqsubseteq$. 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 $AB = ?$-$\sqsubseteq$, necessarily $BC = ?$-$\sqsubseteq$, we conclude by $?$-$\sqsubseteq$.

Case $AB = \mathbb{H}$-$\sqsubseteq$, necessarily $BC = ?$-$\sqsubseteq, ?^j \sqsubseteq ?^{j'}$, we can thus apply the inductive hypothesis to $A \sqsubseteq \mathrm{Germ}_j\ \mathrm{head}\, A$ and $\mathrm{Germ}_j\ \mathrm{head}\, A \sqsubseteq \mathrm{Germ}_j\ \mathrm{head}\, A$ in order to conclude with $\mathbb{H}$-$\sqsubseteq$.

Case $AB = \maltese$-$\sqsubseteq$, we conclude immediately by $\maltese$-$\sqsubseteq$.

Case $AB = \mathfrak{nat}$-$\sqsubseteq, BC = \mathfrak{nat}$-$\sqsubseteq$ we conclude with $\mathfrak{nat}$-$\sqsubseteq$.

Case $AB = \mathfrak{u}$-$\sqsubseteq, BC = \mathfrak{u}$-$\sqsubseteq$ immediate by $\mathfrak{u}$-$\sqsubseteq$.

Case $AB = \pi$-$\sqsubseteq, BC = \pi$-$\sqsubseteq$ by hypothesis we have

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

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

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 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 $AB = \mathfrak{nat}$-$\sqsubseteq, \pi$-$\sqsubseteq$ or $\mathfrak{u}$-$\sqsubseteq$ and $BC = \mathbb{H}$-$\sqsubseteq$, we apply the inductive hypothesis to AB and $B \sqsubseteq \mathrm{Germ}_j\ \mathrm{head}\, B$ in order to conclude with $\mathbb{H}$-$\sqsubseteq$.

So $\sqsubseteq$ is a reflexive, transitive and antisymmetric relation, we are only left with proof-irrelevance, that for any A, B there is at most one derivation of $A \sqsubseteq B$. Since the conclusion 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 π-$\sqsubseteq$. For $\mathbb{H}$-$\sqsubseteq$, $c = \mathrm{head}\, A$ and $j' = \mathrm{pred}\, j$ 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 ($\mathbb{N}$), dependent products, the unknown type $\overline{?}$ 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 25:

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

Note that this definition formally depends on a chosen universe level j for $?^j$, 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, as a consequence the translation does not validate the reduction rules of CastCIC on the nose: `cast`

Monotone translation of contexts

$$\{\!|\cdot|\!\} := \cdot \qquad \{\!|\Gamma, x : A|\!\} := \{\!|\Gamma|\!\}, x : \{\!|A|\!\}$$
$$\{\!|\cdot|\!\}_\varepsilon := \cdot \qquad \{\!|\Gamma, x : A|\!\}_\varepsilon := \{\!|\Gamma|\!\}_\varepsilon, x_0 : \{\!|A|\!\}_0, x_1 : \{\!|A|\!\}_1, x_\varepsilon : \{\!|A|\!\}_\varepsilon\, x_0\, x_1$$

Monotone translation on terms and types

$$\begin{array}{lcllcl}
\{\!|A|\!\} & := & \mathrm{El}\,\{A\} : \mathrm{Poset} & \{\!|A|\!\}_\varepsilon & := & \mathrm{El}_\varepsilon\,\{A\}_\varepsilon : \{\!|A|\!\} \lhd \{\!|A|\!\} \\
\{x\} & := & x & \{x\}_\varepsilon & := & x_\varepsilon \\
\{\Box_i\} & := & \mathfrak{u}_i & \{\Box_i\}_\varepsilon & := & \mathfrak{u}\text{-}\sqsubseteq_i \\
\{\Pi\, x : A.B\} & := & \pi\,\{A\}\,\{\lambda\, x : A.B\}_\varepsilon & \{\Pi\, x : A.B\}_\varepsilon & := & \pi\text{-}\sqsubseteq\,\{A\}_\varepsilon\,\{\lambda\, x : A.B\}_\varepsilon \\
\{t\ u\} & := & \{t\}\,\{u\} & \{t\ u\}_\varepsilon & := & \{t\}_\varepsilon\,\{u\}_0\,\{u\}_1\,\{u\}_\varepsilon \\
\{\lambda\, x : A.t\} & := & \lambda\, x : \{\!|A|\!\}.\{t\} & \{\lambda\, x : A.\ t\}_\varepsilon & := & \lambda(x_0\, x_1 : \{\!|A|\!\})(x_\varepsilon : \{\!|A|\!\}_\varepsilon\, x_0\, x_1).\ \{t\}_\varepsilon \\
\{\mathbb{N}\} & := & \mathfrak{nat} & \{\mathbb{N}\}_\varepsilon & := & \mathfrak{nat}\text{-}\sqsubseteq \\
\{?_A\} & := & ?_{\{A\}} & \{?_A\}_\varepsilon & := & \mathtt{refl}\,\{\!|A|\!\}\,?_{\{A\}} \\
\{\mathtt{err}_A\} & := & \mathtt{err}_{\{A\}} & \{\mathtt{err}_A\}_\varepsilon & := & \mathtt{refl}\,\{\!|A|\!\}\,\mathtt{err}_{\{A\}} \\
\{\langle B \Leftarrow A\rangle\ t\} & := & \downarrow_{\mathrm{El}_\varepsilon\{B\}\sqsubseteq?}\uparrow_{\mathrm{El}_\varepsilon\{A\}\sqsubseteq?}\{t\} & \{\langle B \Leftarrow A\rangle\ t\}_\varepsilon & := & \downarrow_{\mathrm{El}_\varepsilon\{B\}\sqsubseteq?}\text{-}\mathsf{mon}\ \uparrow_{\mathrm{El}_\varepsilon\{A\}\sqsubseteq?}\text{-}\mathsf{mon}\ \{t\}_\varepsilon
\end{array}$$

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

Fig. 18. Translation of the monotone model

can get stuck on type variables eagerly.[17] These rules still hold propositionally though so that we have at least a model in an extensional variant ECIC[18] of CIC.

Lemma 26. *If* $\Gamma \vdash_{\mathsf{cast}} t \rightsquigarrow u$ *then there exists a* CIC *term* e *such that* $\{\Gamma\} \vdash e : \{t\} = \{u\}$.

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 CIC.

Theorem 27. *The translation* $\{-\}$ *of Fig. 18 extends to a model of* CastCIC *into* CIC *extended with induction-recursion and functional extensionality: if* $\Gamma \vdash_{\mathsf{cast}} t : A$ *then* $\{\!|\Gamma|\!\} \vdash_{\mathrm{IR}} \{t\} : \{\!|A|\!\}$.

It is unlikely that the hypothesis that we make on the target calculus 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 $\mathbb{U}$ in the universe, and strict propositions.

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

$$\Gamma \vdash_{\mathsf{cast}} t\ {}_T\sqsubseteq_S\ u \qquad := \qquad \exists e,\quad \{\!|\Gamma|\!\}_\varepsilon \vdash_{\mathrm{IR}} e : \{t\}\ {}_{\{T\}}\sqsubseteq_{\{S\}}\ \{u\}.$$

By the properties of the monotone model (Theorem 25), there is at most one witness up to propositional equality in the target that this judgment hold. This precision relation bears a similar relationship to the structural precision $\sqsubseteq_\alpha$ as propositional equality with definitional equality in CIC. On the one hand, the propositional precision allows to prove precision statement inside the target type theory, for instance we can show by a straightforward case analysis on $b : \mathbb{B}$ that $b : \mathbb{B} \vdash_{\mathsf{cast}} \mathtt{if}\ b\ \mathtt{then}\ A\ \mathtt{else}\ A\ {}_\Box\sqsubseteq_\Box\ A$, a judgment that does not hold for syntactic precision.

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

[18] ECIC enjoy *equality reflection*: two terms are *definitionally* equal whenever they are *propositionally* so.

In particular, propositional precision is stable by propositional equality, and a fortiori it is invariant by conversion in CastCIC: if $t \equiv t'$, $u \equiv u'$ and $\Gamma \vdash_{\mathsf{cast}} t \ {}_T\sqsubseteq_S \ u$ then $\Gamma \vdash_{\mathsf{cast}} t' \ {}_T\sqsubseteq_S \ u'$. On the other hand, the propositional precision relation is not decidable, thus not suited for typechecking where structural precision has to be used instead.

Lemma 28 (Compatibility of structural and propositional precision).

(1) If $\vdash_{\mathsf{cast}} t : T$, $\vdash_{\mathsf{cast}} u : S$ *and* $\vdash t \sqsubseteq_\alpha u$ *then* $\vdash_{\mathsf{cast}} t \ {}_T\sqsubseteq_S \ u$.

(2) Conversely, under the assumption that the meta-theory $\vdash_{\mathrm{IR}}$ *is logically consistent, if* $\vdash_{\mathsf{cast}} v_1 \ {}_{\mathbb{B}}\sqsubseteq_{\mathbb{B}} \ v_2$ *for normal forms* v_1, v_2, *then* $\vdash v_1 \sqsubseteq_\alpha v_2$.

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

$$\text{If } \llbracket \Gamma \vdash t \sqsubseteq_\alpha u, \llbracket \Gamma_1 \vdash_{\mathsf{cast}} t : T \text{ and } \llbracket \Gamma_2 \vdash_{\mathsf{cast}} u : U \text{ then there exists a term } e \text{ such that}$$
$$\{\!|\, \llbracket \Gamma \rrbracket \vdash_{\mathrm{IR}} e : \{t\} \ {}_{\{T\}}\sqsubseteq_{\{U\}} \ \{u\}.$$

The cases for variables (Diag-Var) and universes (Diag-Univ) hold by reflexivity. The cases involving $?$ (Ukn, Ukn-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 $\{\!|\, \llbracket \Gamma \rrbracket \vdash e : \{t\} \ {}_{\{T\}}\sqsubseteq_{\{A'\}} \ \{t'\}$ and $\{\!|\, \llbracket \Gamma \rrbracket \vdash \{T\} \sqsubseteq \{B'\}$. Let j be a universe level such that $\{A'\} \sqsubseteq ?^j$, $\{B'\} \sqsubseteq ?^j$. By (heterogeneous) transitivity of precision applied to e and the proof of $\{\!|\, G \rrbracket \vdash \_ \vdash \{t\} \ {}_{\{A'\}}\sqsubseteq_{?^j}\uparrow_{\{A'\}\sqsubseteq ?^j} \ \{t'\}$, we obtain a proof e' of $\{\!|\, \llbracket \Gamma \rrbracket \vdash e' : \{t\} \ {}_{\{T\}}\sqsubseteq_{\{B'\}}\uparrow_{\{A'\}\sqsubseteq ?^j} \ \{t'\}$ and by adjunction a proof e'' of

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

The case Cast-L proceed in an entirely symmetric fashion since we only used 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 boolean, 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 possible 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 $\vdash v_1 \sqsubseteq_\alpha v_2$. □

We conjecture that the target for GCIC$^\uparrow$ is consistent, that is the assumed inductive-recursive definition for the universe does not endanger consistency. A direct corollary of this lemma is that GCIC$^\uparrow$ satisfies computational graduality, which is the key missing point of §5 and the *raison d'etre* of the monotone model.

Theorem 29 (Graduality for GCIC$^\uparrow$). GCIC$^\uparrow$ *is gradual: for* $\Gamma \vdash_{\mathsf{cast}} t : T, \Gamma \vdash_{\mathsf{cast}} t' : T$ *and* $\Gamma \vdash_{\mathsf{cast}} u : U$

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*

$$\Gamma \vdash_{\mathsf{cast}} \langle S \Leftarrow T \rangle \ t \ {}_S\sqsubseteq_S \ u \Leftrightarrow \Gamma \vdash_{\mathsf{cast}} t \ {}_T\sqsubseteq_S \ u \Leftrightarrow \Gamma \vdash_{\mathsf{cast}} t \ {}_T\sqsubseteq_T \ \langle T \Leftarrow S \rangle \ u,$$

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

Proof. **DGG:** Let $C[-] : (\Gamma \vdash T) \Rightarrow (\vdash \mathbb{B})$ be an observation context, by monotonicity $\vdash_{\mathsf{cast}} C[t] \ {}_{\mathbb{B}}\sqsubseteq_{\mathbb{B}} \ C[t']$. Let $\vdash_{\mathsf{cast}} v, v' : \mathbb{B}$ be the normal forms of $C[t]$ and $C[t']$ (which exist by Theorem 9), since propositional precision is invariant by reduction $\vdash_{\mathsf{cast}} v \ {}_{\mathbb{B}}\sqsubseteq_{\mathbb{B}} \ v'$. Lemma 28.(2) ensures that $\vdash v \sqsubseteq_\alpha v'$ and we conclude using Theorem 23.

Ep-pairs: The fact that propositional precision induces an adjunction is a direct reformulation of the fact that the relation ${}_{\{T\}}\sqsubseteq_{\{S\}}$ underlies an ep-pair (Theorem 25.(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

$$\begin{cases} \{\!|\Gamma|\!\} \vdash_{\mathbb{R}} \_ : \{t\} \ {}_{\{T\}}\sqsubseteq_{\{T\}} \ \{\langle S \Leftarrow T\rangle \ \langle T \Leftarrow S\rangle \ t\} \\ \{\!|\Gamma|\!\} \vdash_{\mathbb{R}} \_ : \{\langle S \Leftarrow T\rangle \ \langle T \Leftarrow S\rangle \ t\} \ {}_{\{T\}}\sqsubseteq_{\{T\}} \ \{t\} \end{cases}$$

that hold because $\downarrow_{\{T\}\sqsubseteq\{S\}} \circ \uparrow_{\{T\}\sqsubseteq\{S\}} = \mathrm{id}$ in the monotone model.

□

In particular, combining Lemma 28.(1) with Theorem 29, we obtain the retract equation: that is for structurally related types $\vdash T \sqsubseteq_\alpha U$, a term $\vdash_{\mathsf{cast}} t : T$ is observationnaly equivalent to $\langle U \Leftarrow T\rangle \ \langle T \Leftarrow U\rangle \ t$.

6.7 Graduality of GCIC$^{\mathcal{G}}$

The monotone model presented in the previous sections can be related to the pointed model of New and Licata [2020, Section 6.1]. As noted there, such a simple model is limited to first order functions in the simply-typed setting. In our dependent setting featuring a universe hierarchy, we can mitigate this limitation, yielding a model for CIC$^{\uparrow}$.

However, this model does not allow us to account for the non-terminating variant GCIC$^{\mathcal{G}}$. In order to go beyond this limitation, we explain how to adapt the Scott model of New and Licata [2020, Section 6.2] based on pointed ω-cpo to our setting. Types are interpreted as pointed ω-cpos, that is as orders $(A, \sqsubseteq)$ equipped with a smallest element $\bot\!\!\!\bot_A \in A$ and an operation $\sup_i a_i$ computing the suprema of countable ascending chains $(a_i)_{i\in\omega} \in A^\omega$. Functions $f : A \to B$ between types are interpreted as monotone *ω-continuous* maps, that is, for any ascending chain $(a_i)_i$, $\sup_i f\, a_i = f(\sup_i a_i)$. In the same spirit, an ep-pair $d : A \lhd B$ should have its two representing functions preserve suprema of ascending chains.[19] Following the seminal work of Scott [1976], the unknown type $?_i$ can be constructed as a solution to the recursive equation:

$$?_i \quad \cong \quad \tilde{\mathbb{N}} + (?_i \to ?_i) + \mathbb{U}_0 + \ldots + \mathbb{U}_i$$

The key technical property that allows us to extend the model from the previous sections to ω-cpos is that the universe can be extended with such a structure. This structure relies on the folkflore lemma that the category of ω-cpos and ep-pairs admit countable sequential colimits that are furthermore preserved by the constructions on the universe. The same facts are also underlying the construction of $?_i$.

Adapting the notion of precision $\Gamma \vdash_{\mathsf{cast}} t \ {}_T\sqsubseteq_S \ u$ of the monotone model to use the order induced by this model, and by using a compatibility with structural precision together with Theorem 23, we can derive graduality for GCIC$^{\mathcal{G}}$.

Theorem 30 (Graduality for GCIC$^{\mathcal{G}}$). GCIC$^{\mathcal{G}}$ *is gradual for the precision induced by the model based on ω-cpos.*

[19] The left adjoint automatically preserves suprema, not the right one

7 DEALING WITH EQUALITY

Up to now, we have left aside a very important aspect, namely, how to deal with *equality* in a gradual dependent type theory.

7.1 Indexed Inductives and Equality

In dependent type theories with inductive types such as CIC, inductive types can be *indexed*, meaning that each constructor can produce values with different type indices. The canonical example is of course the length-indexed type of lists, `vect A n` (see definition in Example 1).

In a gradual dependent type theory, the monotonicity of constructors with respect to precision raises a non-trivial challenge: by monotonicity, we should have `vect A 0` $\sqsubseteq$ `vect A` $?_{\mathbb{N}}$, and by graduality ($\mathcal{G}$), the roundtrip `nil` :: `vect A` $?_{\mathbb{N}}$ :: `vect A 0` should be in equi-precision with `nil`. However, no constructor of `vect` can possibly inhabit `vect A` $?_{\mathbb{N}}$! Therefore, by safety ($\mathcal{S}$), the only inhabitant of `vect A` $?_{\mathbb{N}}$ is `?` (omitting its type): too much precision is lost in the embedding to recover `nil` in the projection.

A systematic way to expose a constructor yielding potentially unknown indices is to encode these indices with additional parameters and *explicit equalities* to capture the constraints on indices:[20]

```
Inductive vect_p (A : □) (n : ℕ) : □ :=
| nil_p : 0 = n → vect_p A n
| cons_p : A → forall m : ℕ, S m = n → vect_p A m → vect_p A n.
```

With this definition, the `nil`$_p$ constructor can legitimately be used to inhabit `vect`$_p$ `A` $?_{\mathbb{N}}$, provided we have an inhabitant (possibly `?`) of `0 = n`. Therefore, the challenge of supporting indexed inductive types gradually is reduced to that of one indexed family, equality.

In CIC, propositional equality `eq A x y`, noted `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 : x = y), P y
```

together with the *definitional* equality:

```
J A P x t x (refl A x) ≡ t.
```

By $\mathcal{G}$, whenever `x` $\sqsubseteq$ `y`, we have `x = x` $\sqsubseteq$ `x = y`, so going from `x = x` to `x = y` should not fail. This in turn means that there has to be a canonical inhabitant of `x = y` whenever `x` $\sqsubseteq$ `y`. If precision were internalized in CIC, as equality is, this would mean that `x` $\sqsubseteq$ `y` iff `x = y`, because by `J`, `x = y` would imply `x` $\sqsubseteq$ `y`. In other words, precision ought to supplant (eq) equality. The problem is that by $\mathcal{G}$, precision must have an extensional flavor, akin to parametricity. Internalizing parametricity [Bernardy et al. 2015], extensional equality (with univalence [Cohen et al. 2015] or with uniqueness of identity proof [Altenkirch et al. 2019]) or even mixing both [Cavallo and Harper 2019], is an active area of research that is very likely to take us quite far from CIC.

Another option is to treat precision as an external relation that is used metatheoretically and implemented via a decision procedure, just as conversion in CIC is external, and decided by reduction. The problem here is that extensionality is not decidable—likewise, in CIC, conversion does not satisfy extensionality, *i.e.*, `f n ≡ g n` for any closed term `n` does not imply that `f ≡ g`.

A gradual dependent type theory therefore needs to address this conundrum.

[20] This technique has reportedly be coined "fording" by Conor McBride [McBride 1999, §3.5], 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."*

7.2 Equality in GCIC

We propose a resolution to the treatment of equality that allows us to remain close to CIC, and is compatible with all four properties, in particular $\mathcal{S}$ and $\mathcal{G}$.

If we have a proof $\mathsf{e} : \mathsf{a} = \mathsf{b}$, then $\mathsf{e} :: \mathsf{a} = ?_{\mathsf{A}} :: \mathsf{a} = \mathsf{b}$ reduces in CastCIC to e, so we do have a proper embedding-projection. However, due to the conundrum exposed previously, in the case of an invalid gain of precision with respect to an equality, such as $\mathtt{refl}\,\mathsf{A}\,\mathsf{a} :: \mathsf{a} = ?_{\mathsf{A}} :: \mathsf{a} = \mathsf{b}$, with $\mathsf{a} \neq \mathsf{b}$, we are not able in general to *eagerly* detect the error, and so we obtain a fake inhabitant of $\mathsf{a} = \mathsf{b}$, in addition to the ones obtained with `err` and $?$. This is because detecting the error at this stage amounts to deciding propositional equality in CIC, which is not possible in general.

Technically, we (grossly) over-approximate equality/precision in GCIC with a universal inductive relation in CastCIC:

```
Inductive universal (A : □) (x y : A) : □ := all : universal A x y.
```

In particular, `refl A x` in GCIC is interpreted as `all A x x` in CastCIC. Importantly, we restrict the *use* of this degenerate relation through its elimination principle, by defining it as casting:

```
J' := λ A P x t y e ⇒ ⟨(P x)<= (P y)⟩ t.
```

`J` in GCIC is interpreted as `J'` in CastCIC. A drawback is that `J' A P x t x (all A x x)` is definitionally equal to `t` only in a closed context; otherwise in general, it is just propositionally equal. This is because the cast operator may be blocked on types containing variables.

Decidable equalities. Finally, we highlight that the decidable forms of equality, although equivalent to the identity type `eq`, have a better behavior in this setting thanks to their computational content. While this is obviously not a novel observation, the impact of decidable equality in the gradual setting is worth highlighting.

For instance, encoding `vect` as $\mathtt{vect}_p$ but using the decidable equality on $\mathbb{N}$ `eqdec`, we obtain the expected conversions:

$$\mathtt{nil}_p\ \mathsf{e} :: \mathtt{vect}_p\ \mathsf{A}\ ?_{\mathbb{N}} :: \mathtt{vect}_p\ \mathsf{A}\ 0 \equiv \mathtt{nil}_p\ \mathsf{e} \qquad \mathtt{nil}_p\ \mathsf{e} :: \mathtt{vect}_p\ \mathsf{A}\ ?_{\mathbb{N}} :: \mathtt{vect}_p\ \mathsf{A}\ 1 \equiv \mathtt{err}_{\mathsf{B}}$$

whereas using the identity type, we get the following, where the casts are stuck on the variable e:

$$\mathtt{nil}_p\ \mathsf{e} :: \mathtt{vect}_p\ \mathsf{A}\ ?_{\mathbb{N}} :: \mathtt{vect}_p\ \mathsf{A}\ 0 \equiv \mathtt{nil}_p\ (\mathsf{e} :: 0 = ?_{\mathbb{N}} :: 0 = 0)$$

$$\mathtt{nil}_p\ \mathsf{e} :: \mathtt{vect}_p\ \mathsf{A}\ ?_{\mathbb{N}} :: \mathtt{vect}_p\ \mathsf{A}\ 1 \equiv \mathtt{nil}_p\ (\mathsf{e} :: 0 = ?_{\mathbb{N}} :: 0 = 1)$$

Coming back to Example 3, this means that using the encoding of vectors with decidable equality, then in all three GCIC variants the term:

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

typechecks and reduces to 0. Additionally, as expected:

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

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

8 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. 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 explain in this paper, introducing the unknown type of gradual typing also requires–in 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]; 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 approaches. [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.

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 it 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.

9 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. 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 lead the way to support 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, possibly parametrized in order to support the different modes reflecting the three variants develop in this work.

REFERENCES

Thorsten Altenkirch, Simon Boulier, Ambrus Kaposi, and Nicolas Tabareau. 2019. Setoid type theory - a syntactic translation. In *MPC 2019 - 13th International Conference on Mathematics of Program Construction (LNCS, Vol. 11825)*. Springer, Porto, Portugal, 155–196. https://doi.org/10.1007/978-3-030-33636-3_7

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, Thierry Coquand, and Guilhem Moulin. 2015. A Presheaf Model of Parametric Type Theory. *Electronic Notes in Theoretical Computer Science* 319 (2015), 67–82.

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.

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

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.

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.

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.

Evan Cavallo and Robert Harper. 2019. Parametric Cubical Type Theory. arXiv:1901.00489.

Cyril Cohen, Thierry Coquand, Simon Huber, and Anders Mörtberg. 2015. Cubical Type Theory: a constructive interpretation of the univalence axiom. (May 2015), 262 pages.

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.

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.

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

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

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.

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.

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. 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. 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 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. *Proceedings of the ACM on Programming Languages* 4, POPL (Jan. 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 2019 2019.

John C. Reynolds. 1983. Types, abstraction, and parametric polymorphism. In *Information Processing 83*, R. E. A. Mason (Ed.). Elsevier, 513–523.

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

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

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.

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

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 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.

A.1 CastCIC

We state and prove a handful 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. It suffices to prove it for Δ of length 1. For this we show by (mutual) induction on the typing derivation the more general statement that if $\Gamma, \Delta \vdash t \triangleright T$ then $\Gamma, x : A, \Delta \vdash t \triangleright T$. 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 ensuring that the reduced type is still well-formed. The hypothesis of context well-formedness is needed for the base case of a variable, to ensured that the type drawn from the context is indeed well-typed. □

A.2 Precision and Reduction

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

Lemma 31 (Weakening of precision). *If* $\Gamma \vdash t \sqsubseteq_\alpha t'$*, then* $\Gamma, \Delta \vdash t \sqsubseteq_\alpha t'$ *for any* Δ.

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

Lemma 32 (Substitution and precision). *If* $\Gamma, x : S \mid S', \Delta \vdash t \sqsubseteq_\alpha t'$, $\Gamma \vdash u \sqsubseteq_\alpha u'$, $\Gamma_1 \vdash u \triangleleft S$ *and* $\Gamma_2 \vdash u' \triangleleft S'$ *then* $\Gamma, \Delta[u \mid u'/x] \vdash t[u/x] \sqsubseteq_\alpha t'[u'/x]$.

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 14. We want to prove the following: under the hypothesis that $\Gamma_1 \sqsubseteq_\alpha \Gamma_2$, if $\Gamma \vdash \square_i \sqsubseteq_\rightsquigarrow T'$ and $\Gamma_2 \vdash T' \blacktriangleright_\square \square_j$, 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 structural precision. 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 Ukn), and we have $\Gamma \vdash \square_{i+1} \sqsubseteq_\rightsquigarrow S_k$, $\Gamma \vdash \square_{i+1} \sqsubseteq_\rightsquigarrow T_k$ and $\Gamma \vdash \square_{i+1} \sqsubseteq_\rightsquigarrow S$. By induction hypothesis, all of S_k, T_k and S reduce either

to $\square_{i+1}$ or some $?_{\square_l}$ with $i+1 \leq l$. 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 $i+1 \leq l$ and the reduction rules

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

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

For the definitional precision, if $\Gamma \vdash \square_i \sqsubseteq_{\rightsquigarrow} T'$ then by decomposing the precision derivation there is an S' such that $T' \rightsquigarrow^* S'$, $\Gamma \vdash \square_i \sqsubseteq_\alpha S'$, 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 15. The proof of those catch-up lemmas is very similar to the previous one for structural precision, but without the need for induction this time – we use the lemma just proven instead. We show the one for product types.

First, let us show the property for $\sqsubseteq_\alpha$. Decompose T' into $\langle S_n \Leftarrow U_{n-1} \rangle \ldots \langle S_2 \Leftarrow U_1 \rangle\ T''$, where T'' is not a cast, but either some $?_S$ or a product type structurally less precise than $\Pi\, x : A.B$. Now by the previous lemma, 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 16. 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 the previous lemma for product types. The proof is somewhat more complex however, 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 proof. First, decompose s' into $\langle S_n \Leftarrow U_{n-1} \rangle \ldots \langle S_2 \Leftarrow U_1 \rangle\ u'$, where u' is either $\lambda\, x : A''.t''$ or $?_S$ for some S. Moreover, 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 15 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 the rule

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

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 the rule

$$?_{\Pi\, x:A''.B''} \rightsquigarrow \lambda\, x : A''.?_{B''}$$

Thus 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 we can use the rule

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

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 we can use the rule

$$\langle \Pi\, x : A_2.B_2 \Leftarrow \Pi\, x : A_1.B_1 \rangle\ \lambda\, x : A''.t'' \rightsquigarrow \lambda\, x : A''.\langle B_2 \Leftarrow B_1[\langle A_1 \Leftarrow A_2 \rangle\ x/x] \rangle\ t''[\langle A'' \Leftarrow A_2 \rangle\ x/x]$$

Moreover, using the precision hypothesis of Cast-R, we know that $\Gamma \vdash \Pi\, x : A.B \sqsubseteq_{\rightsquigarrow} \Pi\, x : A_1.B_2$ and $\Gamma \vdash \Pi\, x : A.B \sqsubseteq_{\rightsquigarrow} \Pi\, x : A_2.B_2$. From the first one, using substitution and the 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 $\left\langle ?_{\square_j} \Leftarrow \Pi\, x : A''.B''\right\rangle\ u'$ when $\Pi\, x : A''.B''$ is not $\mathrm{Germ}_j\ h$, in which case the reduction that applies is

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

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 15 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 Ukn-Univ, and similarly for B.

Note that this last reduction is the point where the system under consideration plays a role: in 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 17. We start by the proof of the second property. We have as hypothesis that $\Gamma \vdash ?_{I(\mathbf{a})} \sqsubseteq_\alpha s'$, $\Gamma_1 \vdash ?_{I(\mathbf{a})} \blacktriangleright I(\mathbf{a})$ 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 $\langle S_n \Leftarrow U_{n-1}\rangle \dots \langle S_2 \Leftarrow U_1\rangle\ ?_{I(\mathbf{a}'')}$, 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

$$\langle I(\mathbf{c}') \Leftarrow I(\mathbf{c})\rangle\ ?_{I(\mathbf{c}'')} \rightsquigarrow^* ?_{I(\mathbf{c}')}$$

$$\left\langle X \Leftarrow ?_{\square_j}\right\rangle \left\langle ?_{\square_j} \Leftarrow \mathrm{Germ}_j\ I\right\rangle\ u' \rightsquigarrow \left\langle X \Leftarrow \mathrm{Germ}_j\ I\right\rangle\ u'$$

$$\left\langle ?_{\square_j} \Leftarrow I(\mathbf{c})\right\rangle\ u' \rightsquigarrow \left\langle ?_{\square_j} \Leftarrow \mathrm{Germ}_j\ I\right\rangle \left\langle \mathrm{Germ}_j\ I \Leftarrow I(\mathbf{c})\right\rangle\ u'$$

we can reduce all casts: the second one (maybe using the last one first) removes all casts through $?_\square$, and then we can use the first one to propagate $?_{I(\mathbf{a}'')}$ all the way through the casts, ending up with a term $?_{S_n}$, which is the one we sought.

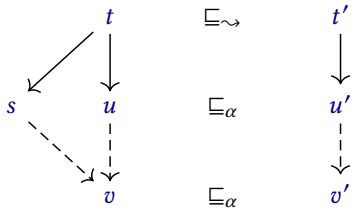
For the first property, again decompose s' as $\langle S_n \Leftarrow U_{n-1}\rangle \dots \langle S_2 \Leftarrow U_1\rangle\ u'$ where u' does not start with a cast. If u' is some $?_{I(\mathbf{a}'')}$, then we can use the proof above and we are finished. Otherwise u' must be of the form $c(\mathbf{a}'', \mathbf{b}'')$. Again we reduce the casts starting with the innermost, using the same two rules to remove the occurrences of $?_\square$. The last case to handle is $\langle I(\mathbf{c}') \Leftarrow I(\mathbf{c})\rangle\ c(\mathbf{c}'', \mathbf{d})$. The reduction that applies there preserves precision by repeated uses of the substitution property of precision, and gives us a term with c as a head constructor. Thus, we get the desired term with c as a head constructor, and argument that are related to $\mathbf{a}$ and $\mathbf{b}$. □

Simulation.

Proof of Theorem 18. 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.

Denotational precision

We start with the second point, which is easiest. The proof is summarized by the following diagram:



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, t and t' are all well typed, and thus by induction hypothesis, there exists some 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 $\Gamma \vdash s \sqsubseteq_{\rightsquigarrow} t'$.

As for inferred types, 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 $\Gamma \vdash S \sqsubseteq_{\rightsquigarrow} T'$.

Syntactical 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 consider the cases where the last rule used for $\Gamma \vdash t \sqsubseteq_\alpha t'$ is not a diagonal rule.

For Ukn 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 step of 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.

Finally, we are left with non-diagonal cast-rules. Rule Cast-R is treated in the same way as for Ukn, as the typing side-conditions are similar.

Thus, the only non-diagonal rule left for $\sqsubseteq_\alpha$ is Cast-L.

Syntactical precision – Congruence reduction rules

Next, we can get rid of the congruence rules of reduction. Indeed, if the last rule used was Cast-L, and the reduction happens in one of the types, of the cast, again 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. More generally, if the last rule used was a diagonal rule, then the congruence rule in t can be simulated by a similar congruence rules in t, since t and t' have the same head.

Syntactical precision – non-diagonal cast

Let us now turn to the case where the last precision rule is Cast-L, and that cast does a head reduction. 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 of that cast.

The first one is when the cast fails. When it does, whatever the rule, it always reduces to $\mathtt{err}_T$. But then we know that $\mathbb{\Gamma}_2 \vdash t' \triangleright T'$ and $\mathbb{\Gamma} \vdash T \sqsubseteq_\rightsquigarrow T'$. Thus $\mathbb{\Gamma} \vdash \mathtt{err}_T \sqsubseteq_\alpha t'$ using rule Err, and the reduction is simulated by zero reductions.

The second case is when the cast disappears (cast between universes) or expands into two casts without changing u (cast through a 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 $?_{\square_i}$, and the germ under consideration is $\mathrm{Germ}_i\ h$. Thus, T' reduces to some S' such that $\mathbb{\Gamma} \vdash ?_{\square_i} \sqsubseteq_\alpha S'$, and this implies that also $\mathbb{\Gamma} \vdash \mathrm{Germ}_i\ h \sqsubseteq_\alpha S'$.

The third case is when A and B are both product types or inductive types, and u starts with an abstraction or an inductive constructor. In that case, by Lemmas 16 and 17, 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 Ukn to conclude.

In the fourth case, t is $\langle X \Leftarrow ?_{\square_i} \rangle\ \langle ?_{\square_i} \Leftarrow \mathrm{Germ}_i\ h \rangle\ u$ reducing to $\langle X \Leftarrow \mathrm{Germ}_i\ h \rangle\ u$. If $\mathbb{\Gamma} \vdash u \sqsubseteq_\alpha t'$ (*i.e.,* rule Cast-L was used twice in a row), then we directly have $\mathbb{\Gamma} \vdash \langle X \Leftarrow \mathrm{Germ}_i\ h \rangle\ u \sqsubseteq_\alpha t'$. Otherwise, rule Diag-Cast was used, t' is some $\langle B' \Leftarrow A' \rangle\ u'$ and we have $\mathbb{\Gamma} \vdash u \sqsubseteq_\alpha u'$ and $\mathbb{\Gamma}_1 \vdash \mathrm{Germ}_i\ h \sqsubseteq_\rightsquigarrow A'$. 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.

Syntactical precision – β and ι redexes

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 16 applies, thus t_1'' reduces to some $\lambda\, x : A'.t_1'$ that is structurally less precise than $\lambda\, x : A.t_1$. Then the β reduction of t can be simulated with another β reduction in t', and using the substitution property we conclude that the redexes are still related by precision.

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. Then, we use Lemma 17 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 Ukn, 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 $\mathbb{\Gamma} \vdash U \sqsubseteq_\alpha U'$.

Syntactical precision – error and ? reductions

For reduction Prod-$\star$, *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 $\mathtt{ind}_I(\mathtt{err}_{I(\mathbf{a})}, z.P, \mathbf{f}.\mathbf{y}.\mathbf{t})$ we distinguish three cases depending on t'. If t' is $?_{T'}$ (the precision rule between t and t' was Ukn) 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 be reduced away, until we are left with $?_T$ with $\mathbb{\Gamma} \vdash \Pi\, x : A.B \sqsubseteq_\rightsquigarrow T$. By Lemma 15, $T \rightsquigarrow^* ?_{?_\square}$ 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 also 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 17 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, $t' \rightsquigarrow ?_{P'[?_{I((\mathbf{a}'))}/z]}$, which is less precise than s.

Syntactical 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 $\Gamma_2 \vdash \langle T \Leftarrow S\rangle\ u \triangleright t$ and $\Gamma \vdash T \sqsubseteq_\rightsquigarrow T'$ we can use rule Err to conclude. Now, let us consider all other reduction rules for casts from top to bottom.

First, 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 15, S' reduces either to $?_\square$ or to a product type. In the first case, u' must reduce to $?_{?_\square}$ by Lemma 16, 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 $\Gamma \vdash \Pi\, x : A_2.B_2 \sqsubseteq_\rightsquigarrow T'$, we have that $\Gamma \sqsubseteq_\alpha s \sqsubseteq_\alpha ?_{T'}$. Otherwise S' reduces to some $\Pi\, x : A'_1.B'_1$. By Lemma 16, 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 15, 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 $\left\langle ?_{\square_i} \Leftarrow \Pi\, x : A'_1.B'_1\right\rangle\ \lambda\, x : A'.v'$. 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 $\left\langle \mathrm{Germ}_i\ \Pi \Leftarrow ?_{\square_i}\right\rangle\ \lambda\, x :?_{\square_{c_\Pi(i)}} u'$, using the substitution property of precision to relate u' with the substituted u, and 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 the corresponding reduction of cast between the same inductive type on an inductive constructor.

Next, let us consider the case where t is $\left\langle ?_{\square_i} \Leftarrow \Pi\, x : A_1.B_1\right\rangle\ f$. We have that $T' \rightsquigarrow ?_{\square_j}$ by Lemma 15 with $i \leq j$, and thus $\Gamma \vdash \mathrm{Germ}_i\ \Pi \sqsubseteq_\rightsquigarrow T'$. Thus, using Diag-Cast for the innermost cast in s, and Cast-L for the outermost one, we conclude $\Gamma \vdash s \sqsubseteq_\alpha \langle T' \Leftarrow S'\rangle\ u'$. Again, the reasoning is similar for the corresponding rule for inductive types.

As for $\langle \square_i \Leftarrow \square_i\rangle\ A$, we can replace rule Diag-Cast by rule Cast-R: indeed $\Gamma_1 \vdash A \triangleleft \square_i$ by typing, thus $\Gamma_1 \vdash A \triangleright T$ for some T such that $T \rightsquigarrow \square_i$. Therefore, since $\Gamma \vdash \square_i \sqsubseteq_\rightsquigarrow T'$, we have $\Gamma \vdash T \sqsubseteq_\rightsquigarrow T'$ and similarly $\Gamma \vdash T \sqsubseteq_\rightsquigarrow S'$. Thus, rule Cast-R gives $\Gamma \vdash A \sqsubseteq_\alpha t'$.

The last case left is the one where t is $\left\langle X \Leftarrow ?_{\square_i}\right\rangle\ \left\langle ?_{\square_i} \Leftarrow \mathrm{Germ}_i\ h\right\rangle\ v$. We distinguish on the rule used to prove $\Gamma \vdash \left\langle ?_{\square_i} \Leftarrow \mathrm{Germ}_i\ h\right\rangle\ v \sqsubseteq_\alpha u'$. If it is Cast-L, then we simply have $\Gamma \vdash \langle X \Leftarrow \mathrm{Germ}_i\ h\rangle\ t \sqsubseteq_\alpha \langle T' \Leftarrow S'\rangle\ u'$ using rule Diag-Cast, as $\Gamma \vdash \mathrm{Germ}_i\ h \sqsubseteq_\rightsquigarrow S'$ since $\Gamma \vdash ?_{\square_i} \sqsubseteq_\rightsquigarrow S'$. Otherwise the rule is Diag-Cast, t' reduces to $\left\langle T' \Leftarrow ?_{\square_j}\right\rangle\ \left\langle ?_{\square_j} \Leftarrow U'\right\rangle\ u'$, using Lemma 15 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 $\Gamma \vdash \mathrm{Germ}_i\ h \sqsubseteq_\rightsquigarrow ?_{\square_j}$ since $i \leq j$. □

A.3 Properties of GCIC

First, let us prove the critical lemma about erasable terms: they have the same reduction behavior as their erasure.

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 33 (GCIC is weaker than CIC – Open context). *Let t be a static term and Γ an erasable context. Then*

- *if $\varepsilon(\Gamma) \vdash_{\mathsf{CIC}} t \triangleright T$ then $\Gamma \vdash t \rightsquigarrow t' \triangleright T'$ for some erasable t' and T' containing no $?$ and such that $\varepsilon(t') = t$ and $\varepsilon(T') = T$;*
- *if T' is an erasable term of* CastCIC *containing no $?$, and $\varepsilon(\Gamma) \vdash_{\mathsf{CIC}} t \triangleleft \varepsilon(T')$ then $\Gamma \vdash t \triangleleft T' \rightsquigarrow t'$ for some erasable t' containing no $?$ such that $\varepsilon(t') = t$;*
- *if $\varepsilon(\Gamma) \vdash_{\mathsf{CIC}} t \blacktriangleright_h T$ then $\Gamma \vdash t \rightsquigarrow t' \blacktriangleright_h T'$ for some erasable t' and T' containing no $?$ such that $\varepsilon(t') = t$ and $\varepsilon(T') = T$.*

Proof. Once again, the proof is by mutual induction, on the elaboration derivation of t.

The inference steps 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 Π-constrained inference next. We are given Γ erasable, and suppose that $\varepsilon(\Gamma) \vdash_{\mathsf{CIC}} t \triangleright T$ and $T \rightsquigarrow^* \Pi x : A.B$. By induction hypothesis there exists t' and T' erasable such that $\Gamma \vdash t \rightsquigarrow t' \triangleright T'$ and $\varepsilon(t') = t$, $\varepsilon(T') = T$. Because T' is erasable, it is less precise than T. By Corollary 19, 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 $\varepsilon(\Gamma) \vdash \Gamma\, \Pi x : A.B \sqsubseteq_\alpha \Pi x : A''.B''$. 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, $\Gamma \vdash t \rightsquigarrow t' \blacktriangleright_\Pi \Pi x : A'.B'$, both t' and $\Pi x : A'.B'$ are erasable, and moreover $\varepsilon(t') = t$ and $\varepsilon(\Pi x : A'.B') = \Pi x : A.B$, which is what had to be proved.

The other cases of constrained inference being very similar, let us turn to checking. We are given Γ and T' erasable, and suppose that $\varepsilon(\Gamma) \vdash_{\mathsf{CIC}} t \triangleright S$ such that $S \equiv \varepsilon(T')$. By induction hypothesis, $\Gamma' \vdash t \rightsquigarrow t' \triangleright S'$ with t' and S' erasable, $\varepsilon(t') = t$ and $\varepsilon(S') = S$. But convertibility implies consistency, so $S \sim \varepsilon(T')$. By monotonicity of consistency, this implies $S' \sim T'$. Thus $\Gamma \vdash t \triangleleft T' \rightsquigarrow \langle T' \Leftarrow S' \rangle\ t'$. We have $\varepsilon(\langle T' \Leftarrow S' \rangle\ t') = \varepsilon(t') = t$, so we are left to show that $\Gamma \vdash \langle T' \Leftarrow S' \rangle\ t' \sqsupseteq\sqsubseteq_\alpha t$. 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 18, $T' \rightsquigarrow^* U'$ 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 34 (CIC is weaker than GCIC – Open context). *Let t be a static term and Γ an erasable context of* CastCIC. *Then*

- *if $\Gamma' \vdash t \rightsquigarrow t' \triangleright T'$, then t' and T' are erasable and contain no $?$, $\varepsilon(t') = t$ and $\varepsilon(\Gamma') \vdash t \triangleright \varepsilon(T')$;*
- *if T' is an erasable term of* CastCIC *containing no $?$ such that $\Gamma' \vdash t \triangleleft T' \rightsquigarrow t'$, then t' is erasable, $\varepsilon(t') = t$ and $\varepsilon(\Gamma') \vdash t \triangleleft \varepsilon(T')$;*
- *if $\Gamma' \vdash t \rightsquigarrow t' \blacktriangleright_h T'$, then t' and T' are erasable and contain no $?$, $\varepsilon(t') = t$ and $\varepsilon(\Gamma') \vdash t \blacktriangleright_h \varepsilon(T')$.*

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, we get conservativity Theorem 21.

Elaboration graduality. Now for the elaboration graduality: again, we state it in an open context for all three typing judgments.

Theorem 35 (Elaboration graduality – Open context). *Let Γ be a context such that $\Gamma_1 \sqsubseteq_\alpha \Gamma_2$, and $\tilde{t}$ and $\tilde{t}'$ be two* GCIC *terms such that $\tilde{t} \sqsubseteq^{\mathrm{G}}_\alpha \tilde{t}'$. Then*

- *if $\Gamma_1 \vdash \tilde{t} \rightsquigarrow t \triangleright T$ and each subterm of $\tilde{t}$ that is against a $?^{@i}$ in $\tilde{t}'$ infers a type in $\square_i$, then there exists t' and T' such that $\Gamma_2 \vdash \tilde{t}' \rightsquigarrow t' \triangleright T'$, $\Gamma \vdash t \sqsubseteq_\alpha t'$ and $\Gamma \vdash T \sqsubseteq_\alpha T'$;*
- *If $\Gamma_1 \vdash t \triangleleft T \rightsquigarrow t'$ and each subterm of $\tilde{t}$ that is against a $?^{@i}$ in $\tilde{t}'$ infers a type in $\square_i$, then for all T' such that $\Gamma \vdash T \sqsubseteq_\alpha T'$ there exists t' such that $\Gamma_2 \vdash \tilde{t}' \triangleleft T' \rightsquigarrow t'$ and $\Gamma \vdash t \sqsubseteq_\alpha t'$;*
- *If $\Gamma_1 \vdash t \rightsquigarrow t' \blacktriangleright_h T$ and each subterm of $\tilde{t}$ that is against a $?^{@i}$ in $\tilde{t}'$ infers a type in $\square_i$, then there exists t' and T' such that $\Gamma_2 \vdash \tilde{t}' \rightsquigarrow t' \blacktriangleright_h T'$, $\Gamma \vdash t \sqsubseteq_\alpha t'$ and $\Gamma \vdash T \sqsubseteq_\alpha T'$.*

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 $\tilde{t} \sqsubseteq^{\mathrm{G}}_\alpha \tilde{t}'$: 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 $\Gamma_1 \vdash \tilde{t} \rightsquigarrow t' \triangleright T'$ and $\Gamma_2 \vdash ?^{@i} \rightsquigarrow ?^{?_{\square_i}} \triangleright ?_{\square_i}$. Correctness of elaboration gives $\Gamma_1 \vdash t' \triangleright T'$, and by validity $\Gamma_1 \vdash T' \triangleright \square_i$, the hypothesis on universe levels assuring us that this i is the same as the one in $\tilde{t}'$. Thus we have $\Gamma \vdash T' \sqsubseteq_\alpha ?_{\square_i}$ by rule Ukn, and in turn $\Gamma \vdash t' \sqsubseteq_\alpha ?^{?_{\square_i}}$ by a second use of the same rule, giving us the required conclusions.

Inference – Variable

The inference rule for a variable gives us $(x : T) \in \Gamma_1$. Because $\vdash \Gamma_1 \sqsubseteq_\alpha \Gamma_2$, there exists some T' such that $(x : T') \in \Gamma_2$, and $\Gamma \vdash T \sqsubseteq_\alpha T'$ using weakening. Thus, $\Gamma_2 \vdash x \rightsquigarrow x \triangleright T'$, and of course $\Gamma \vdash x \sqsubseteq_\alpha x$.

Inference – Product

The type inference rule for product gives $\Gamma_1 \vdash \tilde{A} \rightsquigarrow A \blacktriangleright_\square \square_i$ and $\Gamma_1, x : A \vdash \tilde{B} \rightsquigarrow B \blacktriangleright_\square \square_j$, and the diagonal precision one gives $\tilde{A} \sqsubseteq^{\mathrm{G}}_\alpha \tilde{A}'$ and $\tilde{B} \sqsubseteq^{\mathrm{G}}_\alpha \tilde{B}'$. Applying the induction hypothesis, we get some A' such that $\Gamma_2 \vdash \tilde{A}' \rightsquigarrow A' \blacktriangleright_\square \square_i$ and $\Gamma \vdash A \sqsubseteq_\alpha A'$. The inferred type for $\tilde{A}'$ must be $\square_i$ because 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 $GG_1, x : A \sqsubseteq_\alpha \Gamma_2, x : A'$. Hence the induction hypothesis can be applied to $\tilde{B}$, giving $\Gamma_2 \vdash \tilde{B}' \rightsquigarrow B' \blacktriangleright_\square \square_j$. Combining this with the elaboration for $\tilde{A}'$, we obtain $\Gamma_2 \vdash \Pi\, x : \tilde{A}'.\tilde{B}' \rightsquigarrow \Pi\, x : A'.B' \triangleright \square_{s_\Pi(i,j)}$. Moreover, $\Gamma \vdash \Pi\, x : A.B \sqsubseteq_\alpha \Pi\, x : A'.B'$ by combining the precision hypothesis on A and B, and also $\Gamma \vdash \square_{s_\Pi(i,j)} \sqsubseteq_\alpha \square_{s_\Pi(i,j)}$, relating the two types.

Inference – Application

From the type inference rule for application, we have $\Gamma_1 \vdash \tilde{t} \rightsquigarrow t \blacktriangleright_\Pi \Pi\, x : A.B$ and $\Gamma_1 \vdash \tilde{u} \triangleleft A \rightsquigarrow u$, and the diagonal precision gives $\tilde{t} \sqsubseteq^{\mathrm{G}}_\alpha \tilde{t}'$ and $\tilde{u} \sqsubseteq^{\mathrm{G}}_\alpha \tilde{u}'$. By induction, we have $\Gamma_1 \vdash \tilde{t}' \rightsquigarrow t' \blacktriangleright_\Pi \Pi\, x : A'.B'$ for some t', A' and B' such that $\Gamma \vdash t \sqsubseteq_\alpha t'$, $\Gamma \vdash A \sqsubseteq_\alpha A'$ and $\Gamma, x : A \mid A' \vdash B \sqsubseteq_\alpha B'$. Using the induction hypothesis again with that precision property on A and A' gives $\Gamma_2 \vdash \tilde{u}' \triangleleft A' \rightsquigarrow u'$ with $\Gamma \vdash u \sqsubseteq_\alpha u'$. Now it is just a matter to combine those to get $\Gamma_2 \vdash \tilde{t}'\ \tilde{u}' \rightsquigarrow t'\ u' \triangleright B'[u'/x]$, $\Gamma \vdash t\ u \sqsubseteq_\alpha t'\ u'$ and, by substitution property of precision, $\Gamma \vdash B[u/x] \sqsubseteq_\alpha B'[u'/x]$.

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 checking, we have the following $\Gamma_1 \vdash \tilde{t} \rightsquigarrow t \triangleright S$, with $S \sim T$. By induction hypothesis, $\Gamma_2 \vdash \tilde{t}' \rightsquigarrow \tilde{t}' \triangleright S'$ with $\Gamma \vdash t \sqsubseteq_\alpha t'$ and $\Gamma \vdash S \sqsubseteq_\alpha S'$. But we also have as an hypothesis that $\Gamma \vdash T \sqsubseteq_\alpha T'$. By the monotonicity of consistency, we conclude that $S' \sim T'$, and thus $\Gamma_2 \vdash \tilde{t}' \triangleleft T' \rightsquigarrow \langle T' \Leftarrow S' \rangle\ t'$. A use of Diag-Cast then ensures that $\Gamma \vdash \langle T \Leftarrow S \rangle\ t \sqsubseteq_\alpha \langle T' \Leftarrow S' \rangle\ t'$, as desired. The precision between types T and T' has already been established.

Constrained inference – Inf-Prod rule
We are in the situation where $\Gamma_1 \vdash \tilde{t} \rightsquigarrow t \triangleright S$ and $S \rightsquigarrow^* \Pi x : A.B$. By induction hypothesis, $\Gamma_2 \vdash \tilde{t}' \rightsquigarrow t' \triangleright S'$ with $\Gamma \vdash S \sqsubseteq_\alpha S'$. Using Corollary 19, we get that $S' \rightsquigarrow^* \Pi x : A'.B'$ such that $\Gamma \vdash \Pi x : A.B \sqsubseteq_\alpha \Pi x : A'.B'$, or $S' \rightsquigarrow^* ?_{\square_i}$. In the first case, by rule Inf-Prod we get $\Gamma_2 \vdash \tilde{t}' \rightsquigarrow t' \blacktriangleright_\Pi \Pi x : A'.B'$ 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 $\Gamma_2 \vdash \tilde{t}' \rightsquigarrow \langle \mathrm{Germ}_i\, \Pi \Leftarrow S' \rangle\ t' \blacktriangleright_\Pi \mathrm{Germ}_i\, \Pi$, 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 $\Gamma \vdash A \sqsubseteq_\alpha ?_{\square_{c_\Pi(i)}}$ by Ukn-Univ and similarly for B, gives that $\Gamma \vdash t' \sqsubseteq_\alpha \langle \mathrm{Germ}_i\, \Pi \Leftarrow S' \rangle\ t'$, and the precision between types has been established already.

Constrained inference – Inf-Prod?
This time, $\Gamma_1 \vdash \tilde{t} \rightsquigarrow t \triangleright S$, but $S \rightsquigarrow^* ?_{\square_i}$. By induction hypothesis, $\Gamma_2 \vdash \tilde{t}' \rightsquigarrow t' \triangleright S'$ with $\Gamma \vdash S \sqsubseteq_\alpha S'$. By Corollary 19, we get that $S' \rightsquigarrow^* ?_{\square_i}$. Thus $\Gamma_2 \vdash \tilde{t}' \rightsquigarrow \langle \mathrm{Germ}_i\, \Pi \Leftarrow S' \rangle\ t' \blacktriangleright_\Pi \mathrm{Germ}_i\, \Pi$. 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 we do not have to handle c_Π).

□

B 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[21].

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. 19, 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 $\mathbb{U}_{\mathrm{rel},i}$ between $\mathbb{U}_i^{\mathrm{dis}} := [\![\square_i]\!]$ and $\mathbb{U}_i^{\mathrm{mon}} := \{\!|\, \square_i \,|\!\}$ 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 `casts`, since that's the main point of divergence of the two models.

Lemma 36 (Basis lemma).

(1) There exists a term $\mathtt{cast}_{rel} : \wr \Pi(A\, B : \mathbb{U}).A \to B \int\ [\mathtt{cast}]\ \{\mathtt{cast}\}$.
(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

$$\begin{array}{lll} A : [\![\square_i]\!], & A' : \{\!|\, \square_i \,|\!\}, & A_{\mathrm{rel}} : \mathbb{U}_{\mathrm{rel}}\, A\, A', \\ B : [\![\square_i]\!], & B' : \{\!|\, \square_i \,|\!\}, & B_{\mathrm{rel}} : \mathbb{U}_{\mathrm{rel}}\, B\, B', \\ a : \mathrm{El}\, A, & a' : \mathrm{El}\, A', & a_{\mathrm{rel}} : \mathrm{El}_{\mathrm{rel}}\, A_{\mathrm{rel}}\, a\, a' \end{array}$$

We proceed by induction on $A_{\mathrm{rel}}, B_{\mathrm{rel}}$, following the defining cases for $[\mathtt{cast}]$ (see Fig. 14).

[21] For instance the function swapping $✠_{\tilde{\mathbb{N}}}$ and $?_{\tilde{\mathbb{N}}}$ is not monotone.

Case $A_{\mathbf{rel}} = \pi_{\mathbf{rel}}\, A^{\mathbf{d}}_{\mathbf{rel}}\, A^{\mathbf{c}}_{\mathbf{rel}}$ and $B_{\mathbf{rel}} = \pi_{\mathbf{rel}}\, B^{\mathbf{d}}_{\mathbf{rel}}\, B^{\mathbf{c}}_{\mathbf{rel}}$: we pose $A' = \pi\, A'^{\mathrm{d}}\, A'^{\mathrm{c}}$ and $B' = \pi\, B'^{\mathrm{d}}\, B'^{\mathrm{c}}$

$$\begin{aligned}
\{\mathtt{cast}\}\ A'\ B'\ f' &= \downarrow^{?}_{B'}\ (\uparrow^{?}_{A'}\ f') && \text{(by definition of } \{\mathtt{cast}\}\text{)}\\
&= \downarrow^{?\to ?}_{B'} \circ \downarrow^{?}_{?\to ?} \circ \uparrow^{?}_{?\to ?} \circ \uparrow^{?\to ?}_{A'}\ (f) && \text{(by decomposition of } \pi \sqsubseteq ?\text{)}\\
&= \downarrow^{?\to ?}_{B'} \circ \uparrow^{?\to ?}_{A'}\ (f) && \text{(by section-retraction identity)}\\
&= \lambda(b' : \mathrm{El}\, A'^{\mathrm{d}}).\ \mathrm{let}\ a' = \downarrow^{?}_{B'^{\mathrm{d}}} \circ \uparrow^{?}_{A'^{\mathrm{d}}}\ (b)\ \mathrm{in} && \text{(by def. of ep-pair on } \Pi\text{)}\\
&\qquad \downarrow^{?}_{B'^{\mathrm{c}}\, b'} \circ \uparrow^{?}_{A'^{\mathrm{c}}\, a'}\ (f\ a')\\
&= \lambda(b' : \mathrm{El}\, A'^{\mathrm{d}}).\ \mathrm{let}\ a' = \{\mathtt{cast}\}\ B'^{\mathrm{d}}\ A'^{\mathrm{d}}\ b'\ \mathrm{in} && \text{(by definition of } \{\mathtt{cast}\}\text{)}\\
&\qquad \{\mathtt{cast}\}\ (A'^{\mathrm{c}}\ a')\ (B'^{\mathrm{c}}\ b')\ (f\ a')
\end{aligned}$$

For any $b : \mathrm{El}\, B^{\mathrm{d}}$ and $b' : \mathrm{El}\, B'^{\mathrm{d}}$, $b_{\mathrm{rel}} : \mathrm{El}_{\mathrm{rel}}\, B^{\mathrm{d}}_{\mathrm{rel}}\, b\, b'$, we have by inductive hypothesis

$$a_{\mathrm{rel}} := \wr \mathtt{cast} \int\ B^{\mathrm{d}}_{\mathrm{rel}}\, A^{\mathrm{d}}_{\mathrm{rel}}\, b_{\mathrm{rel}} : \mathrm{El}_{\mathrm{rel}}\, A_{\mathrm{rel}}\ ([\mathtt{cast}]\, B^{\mathrm{d}}\, A^{\mathrm{d}}\, b)\ (\{\mathtt{cast}\}\, B'^{\mathrm{d}}\, A'^{\mathrm{d}}\, b')$$

so that, posing $a = [\mathtt{cast}]\, B^{\mathrm{d}}\, A^{\mathrm{d}}\, b$ and $a' = \{\mathtt{cast}\}\, B'^{\mathrm{d}}\, A'^{\mathrm{d}}\, b'$,

$$f_{\mathrm{rel}}\, a\, a'\, a_{\mathrm{rel}} : \mathrm{El}_{\mathrm{rel}}\ (A^{\mathrm{c}}_{\mathrm{rel}}\, a\, a'\, a_{\mathrm{rel}})\ (f\ a)\ (f'\ a')$$

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

$$\wr \mathtt{cast} \int\ A_{\mathrm{rel}}\ B_{\mathrm{rel}}\ f_{\mathrm{rel}} : \mathrm{El}_{\mathrm{rel}}\ (\pi\, B^{\mathrm{d}}_{\mathrm{rel}}\, B^{\mathrm{c}}_{\mathrm{rel}})\ ([\mathtt{cast}]\, A\, B\, f)\ (\{\mathtt{cast}\}\, A'\, B'\, f').$$

Case $A_{\mathbf{rel}} = \pi_{\mathbf{rel}}\, A^{\mathbf{d}}_{\mathbf{rel}}\, A^{\mathbf{c}}_{\mathbf{rel}}$ and $B_{\mathbf{rel}} = ?_{\mathbf{rel}}$: By definition of the logical relation at $?^{i}_{\mathrm{rel}}$, we need to build a witness of type

$$\mathrm{El}_{\mathrm{rel}}\ (?^{c_\Pi(i)} \to ?^{c_\Pi(i)})\ ([\mathtt{cast}]\, A\, (? \to ?)\, f)\ (\downarrow^{?}_{?\to ?}\ (\{\mathtt{cast}\}\, A'\, ?\, f'))$$

We compute that

$$\downarrow^{?}_{?\to ?}\ (\{\mathtt{cast}\}\, A'\, ?\, f') = \downarrow^{?}_{?\to ?} \circ \downarrow^{?}_{?} \circ \uparrow^{?}_{A'}\ f' = \downarrow^{?}_{?\to ?} \circ \uparrow^{?}_{A'}\ f' = \{\mathtt{cast}\}\, A'\, (? \to ?)\, f'$$

So the result holds by induction hypothesis.

Other cases with $A_{\mathbf{rel}} = \pi_{\mathbf{rel}}\, A^{\mathbf{d}}_{\mathbf{rel}}\, A^{\mathbf{c}}_{\mathbf{rel}}$: 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 $\downarrow^{?}_{\mathrm{Germ}\ c} \uparrow^{?}_{\mathrm{Germ}\ c'}\ x = \maltese_{\mathrm{El}_{\mathbb{H}}\, c}$ whenever $c \neq c'$ and downcasts preserve $\maltese$.

Case $A_{\mathbf{rel}} = ?_{\mathbf{rel}}$, $B_{\mathbf{rel}} = \pi_{\mathbf{rel}}\, B^{\mathbf{d}}_{\mathbf{rel}}\, B^{\mathbf{c}}_{\mathbf{rel}}$ and $a = (\mathtt{pi}; f)$: By hypothesis, $a_{\mathrm{rel}} : \mathrm{El}_{\mathrm{rel}}\ (? \to ?)\ f\ (\downarrow^{?}_{?\to ?}\ a')$ and $\{\mathtt{cast}\}\ ?\ B'\ a' = \{\mathtt{cast}\}\ (? \to ?)\ B'\ (\downarrow^{?}_{?\to ?}\ a')$ so by induction hypothesis

$$\wr \mathtt{cast} \int\ (?_{\mathrm{rel}} \to_{\mathrm{rel}} ?_{\mathrm{rel}})\ B_{\mathrm{rel}}\ f\ (\downarrow^{?}_{?\to ?}\ a')\ a_{\mathrm{rel}} : \mathrm{El}_{\mathrm{rel}}\, B_{\mathrm{rel}}\ ([\mathtt{cast}]\, ?\, B\, (\mathtt{pi}; f))\ (\{\mathtt{cast}\}\, ?\, B'\, a')$$

The others cases with $A_{\mathrm{rel}} = ?_{\mathrm{rel}}$ 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 $A_{\mathrm{rel}} = \mathfrak{Nat}_{\mathrm{rel}}$ follow the same pattern as for π_{rel}. □

Translation of contexts

$$\wr \cdot \int := \cdot \qquad \wr \Gamma, x : A \int := \wr \Gamma \int, x_{\text{dis}} : [\![A]\!], x_{\text{mon}} : \{\!|A|\!\}, x_{\text{rel}} : \wr A \int\ x_{\text{dis}}\ x_{\text{mon}}$$

Logical relation on terms and types

$$\begin{array}{lcl}
\wr A \int & := & \text{El}_{\text{rel}}\ \wr A \int \\
\\
\wr x \int & := & x_{\text{rel}} \\
\wr \Box_i \int & := & \mathfrak{u}_{\text{rel},i} \\
\wr t\ u \int & := & \wr t \int\ [u]\ \{u\}\ \wr u \int \\
\wr \lambda x : A.t \int & := & \lambda(x_{\text{dis}} : [\![A]\!])(x_{\text{mon}} : \{\!|A|\!\})(x_{\text{rel}} : \wr A \int\ x_{\text{dis}}\ x_{\text{mon}}).\ \wr t \int \\
\wr \Pi x : A.B \int & := & \pi_{\text{rel}}\ \wr A \int\ (\lambda(x_{\text{dis}} : [\![A]\!])(x_{\text{mon}} : \{\!|A|\!\})(x_{\text{rel}} : \wr A \int\ x_{\text{dis}}\ x_{\text{mon}}).\wr B \int) \\
\wr \mathbb{N} \int & := & \mathfrak{nat}_{\text{rel}} \\
\wr ?_A \int & := & ?_{\wr A \int} : \wr A \int\ ?_{[\![A]\!]}\ ?_{\{A\}} \\
\wr \mathsf{err}_A \int & := & \maltese_{\wr A \int} : \wr A \int\ \maltese_{[\![A]\!]}\ \maltese_{\{A\}} \\
\wr \mathsf{cast} \int & := & \mathsf{cast}_{\text{rel}}
\end{array}$$

Inductive-recursive relational universe $\mathbb{U}_{\mathbf{rel}} : \mathbb{U}^{\mathbf{dis}} \to \mathbb{U}^{\mathbf{mon}} \to \Box$

$$\frac{A_{\text{rel}} \in \mathbb{U}_{\text{rel},i}\ A\ A' \qquad B \in \Pi(a : A)(a' : A').\text{El}_{\text{rel}}\ A_{\text{rel}}\ a\ a' \to \mathbb{U}_{\text{rel},j}\ (B\ a)\ (B'\ a')}{\pi_{\text{rel}}\ A_{\text{rel}}\ B_{\text{rel}} \in \mathbb{U}_{\text{rel},s_\Pi(i,j)}\ (\pi\ A\ B)\ (\pi\ A'\ B')}$$

$$\frac{j < i}{\mathfrak{u}_{\text{rel},j} \in \mathbb{U}_{\text{rel},i}\ \mathfrak{u}_j\ \mathfrak{u}_j} \qquad \mathfrak{nat}_{\text{rel}} \in \mathbb{U}_{\text{rel},i}\ \mathfrak{nat}\ \mathfrak{nat} \qquad ?_{\text{rel}} \in \mathbb{U}_{\text{rel},i}\ ?\ ? \qquad \maltese_{\text{rel}} \in \mathbb{U}_{\text{rel},i}\ \maltese\ \maltese$$

Decoding function $\text{El}_{\mathbf{rel}} : \mathbb{U}_{\mathbf{rel}}\ A\ A' \to \text{El}\ A \to \text{El}\ A' \to \Box$

$$\begin{array}{rcl}
\text{El}_{\text{rel}}\ \mathfrak{u}_{\text{rel},j}\ A\ A' & := & \mathbb{U}_{\text{rel},j}\ A\ A' \\
\text{El}_{\text{rel}}\ \mathfrak{nat}_{\text{rel}}\ n\ m & := & n = m \\
\text{El}_{\text{rel}}\ \maltese_{\text{rel}}\ *\ * & := & \top \\
\text{El}_{\text{rel}}\ ?_{\text{rel}}\ (c; x)\ y & := & \text{El}_{\text{rel}}\ (\text{Germ}_{\text{rel}}\ c)\ x\ (\mathsf{downcast}_{?,\text{Germ}\ c}\ y) \\
\text{El}_{\text{rel}}\ (\pi_{\text{rel}}\ A_{\text{rel}}\ B_{\text{rel}})\ f\ f' & := & \Pi(a : \text{El}\ A)(a' : \text{El}\ A')(a_{\text{rel}} : \text{El}_{\text{rel}}\ A_{\text{rel}}\ a\ a'). \\
& & \qquad \text{El}_{\text{rel}}\ (B_{\text{rel}}\ a\ a'\ a_{\text{rel}})\ (f\ a)\ (f'\ a')
\end{array}$$

Fig. 19. Logical relation between the discrete and monotone models