Gradualizing the Calculus of Inductive Constructions

Meven Bertrand, Kenji Maillard, Nicolas Tabareau, Éric Tanter

Gradualizing the Calculus of Inductive Constructions

MEVEN 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 canonicity, decidability 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 jointly 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 graduality. We then establish the metatheory of this cast calculus through both a syntactic model into CIC, which provides weak canonicity, confluence, and when applicable, normalization, and a monotone model that purports the study of the graduality of two of the three variants. This work informs and paves the way towards the development of malleable proof assistants and dependently-typed programming languages.

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 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 effectivity of gradual languages is measured 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]. 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] 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. 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. Approximate normalization is however not applicable to design a computationally relevant type theory, because a term that uses imprecision in any useful way necessarily normalizes to the unknown term, even when no errors would have ensued.

Authors' addresses: Meven 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.

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].

We make 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.
- We present an approach to gradualize 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.
- We give a novel, mutually-recursive and bidirectional elaboration of GCIC to a dependently-typed cast calculus CastCIC and prove that it satisfies a static notion of graduality akin to the static gradual guarantee [Siek et al. 2015] (§5).
- We give a bidirectional presentation of CIC (§4), which we could not readily find in the literature, and exploit it to define elaboration.
- We explain the challenging issue of equality in gradual type theories,[1] and propose an approach to handle equality in GCIC through elaboration.
- We develop two models of CastCIC (§6). First, in order to justify CastCIC as a meaningful type theory, 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, we provide a model of CastCIC that captures the notion of monotonicity with respect to precision, and prove graduality for the relevant choices of parameters.

We finally discuss related work (§7) and conclude (§8). Complete definitions and detailed proofs can be found in appendix.

2 FUNDAMENTAL TRADEOFFS IN GRADUAL DEPENDENT TYPE THEORY

Before exposing a specific approach to gradualize CIC, we present a general analysis of the main properties at stake and tensions that arise when gradualizing a dependent type theory. In particular, we establish a fundamental impossibility in the gradualization of CIC, which means that at least one of the desired properties has to be sacrificed.

To begin, let us recall that, as a logically consistent type theory, CIC enjoys **Normalization** ($\mathcal{N}$), which entails that conversion is decidable. In particular, $\mathcal{N}$ means that one can devise a sound and complete decision procedure (a.k.a. a reduction strategy) in order to decide conversion, and hence, typing. CIC also satisfies canonicity, meaning that the normal forms of closed terms of a given type are exactly the canonical forms of that type: for instance, any closed term of type $\mathbb{B}$ is convertible (and reduces) to either `true` or `false`.

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

2.1 The Axiomatic Approach and Conservativity

Let us first address the elephant in the room: why would one want to gradualize CIC instead of simply postulating any property one does not feel like proving (yet)?

Indeed, we can augment CIC with a general-purpose axiom: `Axiom ? : forall A, A`. The resulting theory, CIC+?, has an obvious practical benefit: we can use `(? A)`, hereafter noted $?_A$, as a "wildcard" whenever we are asked to exhibit an inhabitant of some type `A` and we do not want to. However, we cannot use $?_A$ in any meaningful way as a value *at the type level*. For instance, one might be tempted to give to the `filter` function on vectors (size-indexed lists) the type `forall A n (f:A→ 𝔹), Vect A n → Vect A ?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 (filter ℕ 4 even [0;1;2:3])` does not type check because `Vect A (S ?𝕟)` is not convertible to `Vect A ?𝕟`. So the axiomatic approach is not useful for making dependently-typed programming any more pleasing.

On the metatheoretical front, CIC+? has a good property: all pure (*i.e.* axiom-free) CIC terms behave as they would in CIC. In the gradual typing literature, this is often referred to as the **Conservativity** (C) of the "gradual language" (here CIC+?) with respect to the "static language" (here CIC). We parametrize this property with the base theory, so we say that CIC+? satisfies $C_{/\text{CIC}}$.[2] Also, CIC+? still satisfies $\mathcal{N}$, so that conversion remains decidable. However, CIC+? loses canonicity. Importantly, this does not just mean that, given a type `A`, $?_A$ inhabits `A`, but that an *infinite* number of normal forms (more adequately called *stuck terms*) inhabit `A`. For instance, in $\mathbb{B}$, we not only have the normal forms `true`, `false`, and $?_{\mathbb{B}}$, but an infinite number of terms stuck on eliminations of ?, such as `match` $?_A$ `with ...` or $?_{\mathbb{N}\to\mathbb{B}}$ 1. Additionally, these terms are not convertible to a single representative while they all represent the same wildcard, which really breaks conversion.

2.2 Exceptions and Weak Canonicity

Pédrot and Tabareau [2018] demonstrated that it is possible to extend a type theory with a wildcard while preserving a form of canonicity. In particular they present an exceptional type theory ExTT, which is essentially CIC+`err`. Like our axiom above, the error term $\texttt{err}_A$ can inhabit any type `A`, but instead of being treated as an axiom, `err` is endowed with computational content emulating exceptions in programming languages, which propagate instead of being stuck. For instance, eliminating $\texttt{err}_{\mathbb{B}}$ at type ℕ (such as `match` $\texttt{err}_{\mathbb{B}}$ `with true → 0 | false → 1`) is convertible to $\texttt{err}_{\mathbb{N}}$. Notably, exceptions in ExTT 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 $\texttt{err}_{A\to B}$ and λ `_ : A ⇒` $\texttt{err}_B$ are convertible, and the latter is considered to be in normal form. So $\texttt{err}_A$ is a normal form of `A` only if `A` is a positive type.

Consequently, ExTT satisfies **Weak Canonicity** ($\mathcal{W}$): any closed term of a positive type (*e.g.* $\mathbb{1}$) is convertible to either the constructors of that type (*e.g.* `tt`) or `err` at that type (*e.g.* $\texttt{err}_{\mathbb{1}}$). $\mathcal{W}$, together with $\mathcal{N}$, gives *weak logical consistency*: any closed proof of `False` is convertible to $\texttt{err}_{\texttt{False}}$, which is discriminable at `False`. In terms of a programming language, this means that we avoid stuck terms, and simply admit exceptions as a possible outcome of computation. We can still reason soundly in an exceptional type theory, using different techniques [Pédrot and Tabareau 2018; Pédrot et al. 2019]. Note that ExTT is also normalizing ($\mathcal{N}$), although $\mathcal{W}$ can still hold in a theory for which $\mathcal{N}$ does not: it then simply means that normal forms, *when they exist*, are canonical. It is important

[2] Conservativity here means that two CIC terms are convertible in the larger system iff they are convertible in CIC. This is the counterpart of the similar property for gradual languages in simple type systems [Siek et al. 2015], where conservativity means that typechecking and evaluation coincide on terms from the static language. Importantly, this does not mean that CIC+? is a conservative extension of CIC *as a logic*.

to highlight that not only is $\mathcal{W}$ the *least* one can hope for from a meaningful type theory, it is also the *most* one can expect in terms of canonicity in a theory with effects [Pédrot and Tabareau 2020].

So why cannot we use ExTT as a proto-gradual type theory? Unfortunately, while it solves the canonicity issue, using $\mathtt{err}_{\mathtt{A}}$ as a wildcard has the same practical limitation as the axiomatic approach: even though we can use it to inhabit any type, we cannot use it in any meaningful way as a value at the type level. In our example above, $\mathtt{Vect\ A\ (S\ err}_{\mathbb{N}}\mathtt{)}$ is not convertible to $\mathtt{Vect\ A\ err}_{\mathbb{N}}$.

2.3 Consistency, Precision, and Graduality

In non-dependent type systems and theories, **?** is only a type, not a term. Type-level relations, such as equality and subtyping, can be systematically relaxed to account for the unknown type [Garcia et al. 2016; Siek and Taha 2006, 2007]. The relaxing of equality is called *consistency*, not to be confused with logical consistency! With full dependent types, there is no longer any type/term syntactic distinction, so the unknown type is in fact an unknown *term*. The key relation to relax in this setting is conversion, so that $\mathtt{Vect\ A\ (S\ ?}_{\mathbb{N}}\mathtt{)}$ can be deemed consistent with (or consistently convertible to) $\mathtt{Vect\ A\ ?}_{\mathbb{N}}$. In essence, the unknown term can be seen as a dual form of exception: it should likewise propagate, but be optimistically comparable, *i.e.* consistent with, any other terms. More precisely, $\mathtt{?}_{\mathtt{A}}$ should be consistent with any term of type A while $\mathtt{err}_{\mathtt{A}}$ should not be consistent with any such term. The "unknown type" is simply $\mathtt{?}_{\square}$.[3] Note that because we now have two exceptional terms, $\mathcal{W}$ means that the normal forms of type $\mathbb{1}$ are tt, $\mathtt{err}_{\mathbb{1}}$, and $\mathtt{?}_{\mathbb{1}}$.

The early accounts of gradual typing emphasized consistency as the central idea. However, Siek et al. [2015] observed that this characterization left too much 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 *precision* as the key notion. Precision is a preorder that can be used to capture the intended *continuity* of the static-to-dynamic spectrum afforded by gradual typing. The unknown type **?** is considered the most imprecise type ($\mathtt{A} \sqsubseteq \mathtt{?}$ for any type A), and 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 naturally extended from types to terms, but importantly, imprecision only arises from types.

In an effort to give a more mathematically elegant presentation of the dynamic gradual guarantee, New and Ahmed [2018] argue that the fundamental property of gradual typing is that precision gives rise to embedding-projection (adjoint) pairs: going to a less precise type and back is the identity. Therefore, if we have $\mathtt{A} \sqsubseteq \mathtt{B}$, and a term t of type A, $\mathtt{t :: B :: A}$ is equal to t.[4] They introduce the term **Graduality** ($\mathcal{G}$) for this property, to highlight the parallel between parametricity as the key property of polymorphic typing and graduality as the key property of gradual typing. Graduality is also based on important underlying structural properties of precision on terms, namely that it is stable by reduction (if $\mathtt{t} \sqsubseteq \mathtt{t'}$ and t reduces to v and t' to 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 user-level interpretation of precision and the gradual guarantees, namely that precision is compositional and that losing precision is harmless [Siek et al. 2015].

We observe that, in a dependently-typed setting, the notion of graduality as embedding-projection pairs is not just an equivalent formulation of the dynamic gradual guarantee, but a much stronger property! Indeed, because of the presence of the unknown term, one could decide to produce $\mathtt{?}_{\mathtt{A}}$ as a result of going to $\mathtt{?}_{\square}$ and back. Doing so would satisfy the dynamic gradual guarantee formulated

[3] We use the notation $\square_i$ for the predicative universe of types $\mathtt{Type}_i$, omitting the universe level i when irrelevant.
[4] We write $t :: A$ for a type ascription, which is syntactic sugar for $(\lambda x : A.x)\ t$.

by Siek et al. [2015], but it would break the embedding-projection requirement of graduality stated by [New and Ahmed 2018]. In a non-dependent setting, where ? is only a type and not a term, the only possible values at type $\mathbb{N}$ are natural numbers (and the error), so the difference between both properties is not palpable.

This imprecise approach of producing $?_{\mathsf{A}}$ as a result of going through imprecision and back is used in Approximate Normalization (AN) [Eremondi et al. 2019]. For instance, with AN, $1 :: ?_{\square} :: \mathbb{N}$ is convertible to $?_{\mathbb{N}}$, not 1. This ensures that conversion is both total and decidable, but it is not meaningful computationally. This is why Eremondi et al. [2019] only use AN in the typechecking phase of the programming language GDTL, and not for executing GDTL programs—at runtime, GDTL uses a standard possibly-failing, possibly-diverging, and precise notion of reduction. Therefore, graduality as embedding-projection pairs characterizes *both* the smoothness of the static-to-dynamic checking spectrum, and the proper computational content of valid uses of imprecision.

Note that in a dependently-typed setting where invalid gains of precision raise errors, it is not possible to consider $\mathtt{err}_{\mathsf{A}}$ as a separate "runtime-only" term, because it can now appear at the type-level as well, as in: `forall (b : 𝔹), if b then ℕ else` $\mathtt{err}_{\square}$. Therefore, $\mathtt{err}_{\mathsf{A}}$ also needs to be considered in the precision partial order, and to satisfy graduality, it needs to be a *bottom* element.

Finally, observe that because for any type A, $?_{\mathsf{A}}$ is the most imprecise term at that type, then $?_{?_{\square}}$ is the least precise term of all; more exactly, there is one such term per type universe. We simply write ? for this term, omitting the universe level when not required. For clarity, we often still write $?_{\mathsf{A}}$, which has to be understood as sugar for $? :: \mathsf{A}$.

2.4 The Fire Triangle of Graduality

To sum up, we have seen three important properties that can be expected from a gradual type theory: conservativity with respect to theory X ($C_{/X}$), graduality ($\mathcal{G}$), and normalization ($\mathcal{N}$). Unfortunately, achieving them all is impossible in the case of CIC.

Theorem 1 (Fire Triangle of Graduality). *It is impossible to devise a gradual type theory that satisfies at the same time the properties $C_{/\mathsf{CIC}}$, $\mathcal{G}$ and $\mathcal{N}$.*

Proof. Let us assume a theory that satisfies all three properties. Consider the term `Omega` defined as $(\lambda\ \mathtt{x} :\ ?_{\square_i} \Rightarrow \mathtt{x}\ \mathtt{x})\ (\lambda\ \mathtt{x} :\ ?_{\square_i} \Rightarrow \mathtt{x}\ \mathtt{x})$. By $C_{/\mathsf{CIC}}$, universes are closed under $\rightarrow$, so that $?_{\square_i} \rightarrow ?_{\square_i}$ and $?_{\square_i}$ lives at the same universe, hence by maximality of $?_{\mathsf{A}}$ (for any A) with respect to precision, we have that $?_{\square_i} \rightarrow ?_{\square_i} \sqsubseteq\ ?_{\square_i}$. This means that $?_{\square_i} \rightarrow ?_{\square_i}$ and $?_{\square_i}$ are consistent, and so the self-applications in `Omega` (such as `x x`) are well-typed. Therefore, `Omega` is well-typed and has type $?_{\square_i}$. Given that $?_{\square_i} \rightarrow ?_{\square_i}$ and $?_{\square_i}$ are closed types, by $\mathcal{W}$ we know that the embedding-projection induced by precision ($\mathcal{G}$) is definitional. Indeed, if it were not the case, consider the `S` constructor of $\mathbb{N}$ of type $\mathbb{N} \rightarrow \mathbb{N}$, then the term $(\mathtt{S} :: ?_{\square_i} \rightarrow ?_{\square_i} :: ?_{\square_i})\ 0$ would be a stuck closed term of type $\mathbb{N}$, contradicting $\mathcal{W}$. So `Omega` must diverge, which is a violation of $\mathcal{N}$. □

This no-go theorem means that when designing a gradual version of CIC, we must pick at most two properties out of the three, and decide how to sacrifice the third. Note that Theorem 1 can be proven for type theories others than CIC (for instance for a theory with only universes and dependent functions), although there exists type theories for which all properties *can* simultaneously be achieved (we will present one).

The Fire Triangle of Graduality admits several solutions, and this work develops several of them. At the very least, one can devise theories that are degenerate in the sacrificed aspect. For instance a theory that diverges as soon as ? is involved trivially satisfies $C_{/\mathsf{CIC}}$ and $\mathcal{G}$. Likewise, a theory whose typing only accepts fully-precise terms would trivially satisfy $C_{/\mathsf{CIC}}$ and $\mathcal{N}$, but not $\mathcal{G}$. A more interesting example is approximate normalization [Eremondi et al. 2019], in which—as explained

above—$\mathcal{G}$ is sacrificed by producing ? for any tentative gain of precision, irrespective of its validity, in order to avoid both failure and non-termination.

Note that considering a non-universal notion of precision (where $?_{\mathtt{A}}$ is not maximal with respect to precision for any type A) is not a viable escape route in CIC if one wants to satisfy $\mathcal{G}$. Indeed, as soon as there exists at least one term of a positive type that is more precise than, and different from, ? (*e.g.* $\mathtt{tt} \sqsubseteq ?_{\mathbb{1}}$), then ? must be more imprecise than *any* term. This fact comes from the monotonicity of the large eliminator of the considered inductive. For instance, consider the function $\mathtt{f} := \lambda\, \mathtt{x} : \mathbb{1} \Rightarrow \mathtt{match}\ \mathtt{x}\ \mathtt{with}\ \mathtt{tt} \longrightarrow \mathtt{u}$ for any term $\mathtt{u} : \mathtt{A}$. By monotonicity and the exceptional-like propagation of $?_{\mathbb{1}}$, we have $\mathtt{f}\ \mathtt{tt} \sqsubseteq \mathtt{f}\ ?_{\mathbb{1}}$ and $\mathtt{f}\ ?_{\mathbb{1}}$ convertible to $?_{\mathtt{A}}$, so $\mathtt{u} \sqsubseteq ?_{\mathtt{A}}$. So unless precision is vacuously useless (*i.e.* coincides with conversion), it has to be universal.

2.5 A Last Challenge: 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 sized lists, a.k.a. vectors in Coq:

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

In a gradual dependent type theory, the monotonicity of constructors with respect to precision raises a non-trivial challenge: by monotonicity, we should have $\mathtt{vect\ A\ 0} \sqsubseteq \mathtt{vect\ A}\ ?_{\mathbb{N}}$, and by $\mathcal{G}$, the roundtrip $\mathtt{nil} :: \mathtt{vect\ A}\ ?_{\mathbb{N}} :: \mathtt{vect\ A\ 0}$ should be equal to nil. However, no constructor of vect can possibly inhabit $\mathtt{vect\ A}\ ?_{\mathbb{N}}$. Therefore, by $\mathcal{W}$, the only inhabitant of $\mathtt{vect\ A}\ ?_{\mathbb{N}}$ 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:

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

In CIC, the propositional equality = corresponds to the Martin-Löf identity type, 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 $\mathtt{J\ A\ P\ x\ t\ x\ (refl\ A\ x)} \equiv \mathtt{t}$.

By $\mathcal{G}$, whenever $\mathtt{x} \sqsubseteq \mathtt{y}$, we have $\mathtt{x = x} \sqsubseteq \mathtt{x = y}$, so going from $\mathtt{x = x}$ to $\mathtt{x = y}$ should not fail. This in turn means that there has to be a canonical inhabitant of $\mathtt{x = y}$ whenever $\mathtt{x} \sqsubseteq \mathtt{y}$. If precision were internalized in CIC, as equality is, this would mean that $\mathtt{x} \sqsubseteq \mathtt{y}$ iff $\mathtt{x = y}$, because by J, $\mathtt{x = y}$ would imply $\mathtt{x} \sqsubseteq \mathtt{y}$. 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` $\equiv$ `g n` for any closed term `n` does not imply that `f` $\equiv$ `g`.

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

3 GCIC: OVERALL APPROACH, MAIN CHALLENGES AND RESULTS

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. We end this section with a few concrete examples.

3.1 GCIC: 3-in-1

To explore the spectrum of possibilities enabled by the Fire Triangle of Graduality (Theorem 1), we develop a general approach to gradualize CIC, and use it to define three theories, corresponding to each of the possible resolutions of the triangular tension between normalization ($\mathcal{N}$), graduality ($\mathcal{G}$) and conservativity with respect to CIC ($\mathcal{C}_{/\mathrm{CIC}}$):

(1) GCIC$^{\mathcal{G}}$: a theory that satisfies both $\mathcal{C}_{/\mathrm{CIC}}$ and $\mathcal{G}$, but sacrifices $\mathcal{N}$
(2) GCIC$^{\mathcal{N}}$: a theory that satisfies both $\mathcal{C}_{/\mathrm{CIC}}$ and $\mathcal{N}$, but sacrifices $\mathcal{G}$
(3) GCIC$^{\uparrow}$: a theory that satisfies both $\mathcal{N}$ and $\mathcal{G}$, and supports $\mathcal{C}$ wrt to a variant of CIC, CIC$^{\uparrow}$

Instead of developing three theories independently, we expose a parametrized version of a gradual CIC, called GCIC, with two parameters (Fig. 2). 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. This latter option corresponds to what we call CIC$^{\uparrow}$ (read "CIC-shift"). We prove that we can satisfy all three properties $\mathcal{C}$, $\mathcal{N}$ and $\mathcal{G}$ for CIC$^{\uparrow}$. The downside is that CIC$^{\uparrow}$ rejects some terms that are well-typed in CIC.[5] The second parameter of GCIC is the reduction counterpart of the first parameter and needs to coincide for the theory to be gradual. Out of the four resulting possibilities, only three are meaningful (if one is strict for typing, then more flexibility for reduction would break subject reduction); these three possibilities correspond to GCIC$^{\mathcal{G}}$, GCIC$^{\mathcal{N}}$ and GCIC$^{\uparrow}$.

3.2 Typing, Cast Insertion, and Conversion

In CIC, typing appeals to conversion: any term of type `A` can also be given type `B` as long as both are convertible, *i.e.* `A` $\equiv$ `B`. Conversion is decided by a reduction strategy. Therefore, typing and reduction are fundamentally intertwined in dependently-typed theories.

In a gradual language, whenever we reclaim precision, we might be wrong and need to fail in order to satisfy subject reduction. Therefore, reduction needs to rely on *casts*.[6] 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.

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. The interplay between typing and cast insertion is more subtle in the context of a dependent type theory. Because typing needs computation, and imprecision needs safeguards to satisfy subject reduction, GCIC is likewise elaborated in a type-directed manner to a second calculus, named CastCIC (§5.1), but this cast calculus is used *as*

[5] A minimal example of a well-typed CIC term that is ill-typed in CIC$^{\uparrow}$ is `narrow` $: \mathbb{N} \to \square$, where `narrow n` is the type of functions that accept `n` arguments.
[6] Or some other form of runtime tracking and checking, such as evidence [Garcia et al. 2016].

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.

Note that this is similar to what happens in practice in proof assistants such as Coq, where terms that are input by the user are never typechecked as such, but are first elaborated in order to add implicit arguments, coercions, etc. The computation steps required by conversion are always performed on the elaborated terms, never on the raw input syntax. In fact, it would be impossible to define untyped conversion directly on the input syntax. And indeed, in Coq, when the programmer faces an error, it is often needed to use vernacular commands to make such inferred implicit arguments and coercions visible in order to understand the underlying problem.

In order to satisfy conservativity with respect to CIC ($C_{/\mathsf{CIC}}$) ascriptions 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 $\langle \mathbb{B} \Leftarrow ? \rangle\ \langle ? \Leftarrow \mathbb{N} \rangle\ 0$ 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. This ensures that subjection reduction holds, for instance when reducing $\langle \mathbb{B} \Leftarrow ? \rangle\ \langle ? \Leftarrow \mathbb{N} \rangle\ 0$ to $\langle \mathbb{B} \Leftarrow \mathbb{N} \rangle\ 0$.

Finally, standard presentations of CIC use a standalone conversion rule, usual in declarative presentations of type systems. In order to gradualize CIC, we have to move to an algorithmic presentation in order to forbid transitivity; otherwise all terms would be well-typed by way of a transition 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 is prior work on gradual dependent types [Eremondi et al. 2019], we adopt a bidirectional presentation of typing (§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

In order to justify that CastCIC makes sense as a type theory, we build a syntactic model of CastCIC by translation to a known variant of CIC (§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 $\mathtt{err}$ 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 $\mathtt{typerec}$, an elimination principle for the universe $\square$, which targets CIC augmented with induction-recursion [Dybjer and Setzer 2003; Ghani et al. 2015; Martin-Löf 1996].[7]

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 $?_{\square_i}$ behaves as a dependent sum $\Sigma \mathtt{A} : \square_i.\mathtt{A}$—the unknown type is inhabited by "hiding" the underlying type $\mathtt{A}$ of the term that is injected into the unknown type. Projecting out of the unknown type is realized through type analysis, and may fail with an error in the ExTT sense. This syntactic model informs the design and justifies the reduction rules provided for CastCIC: the reduction semantics enjoy confluence and, for the two variants CastCIC$^{\mathcal{N}}$ and CastCIC$^{\uparrow}$, strong normalization ($\mathcal{N}$). This model also gives us weak canonicity ($\mathcal{W}$), and hence weak logical consistency.

[7] In this work, we do not deal with the impredicative sort $\mathtt{Prop}$. The reason is that in the non-terminating setting, it can be interpreted just as $\mathtt{Type}$ following Palmgren [1998], so including it does not bring any interesting insight. For the terminating setting, it seems inherently impossible to avoid the problem of $\mathtt{Omega}$ with an impredicative sort; in particular, having an impredicative sort is incompatible with the level-shifting approach we use in CIC$^{\uparrow}$ to enforce normalization.

3.4 Varieties of Precision and Graduality

While the discrete model of CastCIC is enough to justify its computational content as a type theory, it does not allow us to deduce the main properties coming from the gradual typing literature. In our setting with GCIC that elaborates to CastCIC, graduality ($\mathcal{G}$) can be studied at different levels. First, at the source level (GCIC), we introduce a notion of *syntactic precision* that captures the syntactic intuition of a more imprecise term as "the same term with more ?", and is defined without any assumption of typing, and hence conversion, because the latter is not defined in GCIC proper. In CastCIC, we define a notion of *structural precision*, which is mostly syntactic except that, in order to account for cast insertion during elaboration, it tolerates precision-preserving casts (for instance, $\langle A \Leftarrow A\rangle\ t$ is related to t by structural precision. Armed with these two notions of precision, we prove *elaboration graduality* (Theorem 12): if a term $\tilde{t}$ of GCIC elaborates to a term t of CastCIC, then a term $\tilde{u}$ less syntactically precise than $\tilde{t}$ in GCIC elaborates to a term u less structurally precise than t in CastCIC.

Finally, to prove $\mathcal{G}$ for CastCIC (in its variants CastCIC$^{\mathcal{G}}$ and CastCIC$^{\uparrow}$), we essentially need to establish a bisimulation result for CastCIC wrt. structural precision. However, only a simulation can be established (if the more precise term reduces, then the less precise one reduces too, to a less precise term). The reverse does not hold because less precise terms may reduce more by "unblocking" neutral terms (stuck on variables). For instance, $\langle \mathbb{N} \Leftarrow X\rangle\ \langle X \Leftarrow \mathbb{N}\rangle\ 0$ is neutral due to the type variable X, while $\langle \mathbb{N} \Leftarrow ?_{\square}\rangle\ \langle ?_{\square} \Leftarrow \mathbb{N}\rangle\ 0$ reduces to 0 even though it is less precise. Structural precision gives a bisimulation only in closed contexts, but of course, to prove it inductively, one needs to reason about open contexts as well. Note that this difficulty is proper to dependent types and the presence of variables at the type level.

In order to overcome this problem, we build an alternative model of CastCIC (for the two variants where graduality holds), called the **monotone model** (§6.2 to 6.5), which endows types with the structure of an ordered set, or poset. In the monotone model, we can reason about (semantic) *propositional precision* and establish *propositional graduality*, which corresponds to graduality but only up-to propositional equality. Propositional precision subsumes structural precision and together with weak canonicity, allows us to establish *(computational) graduality* (Theorem 13) for CastCIC, which is the usual notion of graduality ($\mathcal{G}$).

3.5 Dealing with Equality

We now return to the equality/precision conundrum exposed in §2.5. We propose a resolution that allows us to remain close to CIC, and is compatible with all four properties, in particular $\mathcal{W}$ 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 $\mathtt{refl}\ \mathsf{a} :: \mathsf{a} = ?_{\mathsf{A}} :: \mathsf{a} = \mathsf{b}$, 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 when A is a closed type; otherwise it is just propositionally equal. This is because the embedding-projection pair induced by precision is definitional only for closed types.

CastCIC satisfies $\mathcal{W}$ in that the normal forms of type universal are either all, err or ?. However, this notion of $\mathcal{W}$ in CastCIC does not transfer fully to GCIC, because all normal forms of (the translation of) eq are not only (the translation of) refl, in addition to err and ?. So $\mathcal{W}$ in GCIC does not hold for eq, but it does hold for any other positive type. In particular, GCIC enjoys weak logical consistency: any closed proof of False is convertible to either $\mathtt{err}_{\mathtt{False}}$ or $?_{\mathtt{False}}$.

In fact, any use of a non-canonical equality proof to establish False reduces to $\mathtt{err}_{\mathtt{False}}$. For instance, consider the standard proof that 0 = 1 is a contradiction:

```
Definition contra : eq 0 1 → False :=
  λ e ⇒ J ℕ (λ n : ℕ ⇒ match n with 0 ⇒ True | S _ ⇒ False end) 0 I 1 e.
```

Using J' to interpret J, contra (refl 0 :: eq 0 $?_{\mathbb{N}}$ :: eq 0 1) reduces to ⟨False ⇐ True⟩ I in CastCIC, which in turn reduces to $\mathtt{err}_{\mathtt{False}}$.

Finally, we highlight that the conundrum of the identity type eq does not manifest with decidable forms of equality, thanks to their better computational behavior. To illustrate, consider an alternative proof that 0=1 is a contradiction, this time using the decidable equality on ℕ, $\mathtt{eq}_{\mathbb{N}}$:

```
Definition contra' : eqℕ 0 1 → False := λ e ⇒ e.
```

The proof is the identity, so contra' (I :: $\mathtt{eq}_{\mathbb{N}}$ 0 $?_{\mathbb{N}}$ :: $\mathtt{eq}_{\mathbb{N}}$ 0 1) is just I :: $\mathtt{eq}_{\mathbb{N}}$ 0 $?_{\mathbb{N}}$ :: $\mathtt{eq}_{\mathbb{N}}$ 0 1, which in CastCIC directly reduces to $\mathtt{err}_{\mathtt{False}}$.

So while eq and $\mathtt{eq}_{\mathbb{N}}$ are equivalent, choosing to work with $\mathtt{eq}_{\mathbb{N}}$ can be significantly better computationally.[8] For instance, going back to indexed inductive types, if we encode vect as $\mathtt{vect}_p$ using $\mathtt{eq}_{\mathbb{N}}$, we obtain the expected behavior:

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

3.6 GCIC in Action

To conclude this overview section, we present small examples that illustrate both the commonalities and differences between the three GCIC variants.

Indexed types. First of all, in all three systems, we can define filter by giving it the imprecise type forall A n (f : A → 𝔹), Vect A n → Vect A $?_{\mathbb{N}}$ in order to bypass the difficulty of precisely characterizing the size of the output vector. If we adopt an encoding of Vect with *decidable equality*, as explained in §3.5, then the term head $?_{\mathbb{N}}$ (filter ℕ 4 even [0;1;2;3]) typechecks and reduces to 2. Likewise, head $?_{\mathbb{N}}$ (filter ℕ 4 even [1;3]) typechecks and fails at runtime.

Large elimination. 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 for instance on their inputs, such as:

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

In a gradually-typed language, one can give this function the type ?, or even ℕ → ℕ → ? 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 A := forall (n m : ℕ), if (n > m) then ℕ else 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 implementor of the function and its clients. In all three variants of GCIC, one can explore the whole spectrum of type-level precision for such a function:

$$? \sqsupseteq \mathbb{N} \to \mathbb{N} \to ? \sqsupseteq \mathbb{N} \to \mathbb{N} \to \mathtt{if}\ ?\ \mathtt{then}\ \mathbb{N}\ \mathtt{else}\ ? \sqsupseteq \mathtt{forall}\ (\mathtt{n}\ \mathtt{m} : \mathbb{N}), \mathtt{if}\ (\mathtt{n} > \mathtt{m})\ \mathtt{then}\ \mathbb{N}\ \mathtt{else}\ ? \sqsupseteq \mathtt{A}$$

[8] This is not a novel observation, obviously, but the impact of decidable equality in the gradual setting is noteworthy.

At each stage from left to right, there is less flexibility for both the implementor of `foo` and its clients. Graduality ensures that if the function is actually faithful to the most precise type `A`, giving it any of the less precise types above does not introduce any new failure.

Omega. Let us come back to the term $\texttt{Omega} := (\lambda\ \texttt{x} :\ ?_{\square_i} \Rightarrow \texttt{x}\ \texttt{x})\ (\lambda\ \texttt{x} :\ ?_{\square_i} \Rightarrow \texttt{x}\ \texttt{x})$ used in the proof of Theorem 1. In both GCIC$^{\mathcal{G}}$ and GCIC$^{\mathcal{N}}$, this term is well-typed. In GCIC$^{\mathcal{G}}$, it even 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 on casts.

In GCIC$^{\uparrow}$, this term does not typecheck 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 subterm $\texttt{f} := \lambda\ \texttt{x} :\ ?_{\square_i} \Rightarrow \texttt{x}\ \texttt{x}$ does not typecheck: when attempting to use `x` as a function, its type $?_{\square_i}$ is elaborated to $?_{\square_{i-1}} \to ?_{\square_{i-1}}$, which cannot accept an argument of type $?_{\square_i}$.

Dependent arities. 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 the type of later arguments. This function brings out the limitation of GCIC$^{\uparrow}$: since the format string can specify and 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. Finally, in GCIC$^{\mathcal{G}}$ the function can be gradualized as much as one wants, without surprises.

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 avoids 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.

Our syntax for CIC featuring a predicative universe hierarchy $\square_i$ is:

$$t ::= x \mid \square_i \mid t\ t \mid \lambda x : t.t \mid \Pi x : t.t \mid I \mid c \mid \texttt{ind}_I(t, t, \mathbf{t})$$

When needed, we 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 ys for as. We present generic inductive types, although we restrict to strictly positive ones to preserve normalization, following [Giménez 1998]. At this point we consider only inductive types without indices: § 2.5 and 3.5 explained how to recover indexed inductive types using equality. Inductive types are formally annotated with a universe level $@i$, controlling the level of its parameters: for instance $\Sigma@i(A, B)$ expects A to be a type in $\square_i$. This level is hidden when unessential. An inductive type at level i with parameters $\mathbf{a} : \text{Params}(I, i)$ is noted $I@i(\mathbf{a})$. Similarly $c_k^I@i(\mathbf{a}, \mathbf{b})$ denotes the k-th constructor of the inductive I, taking parameters $\mathbf{a} : \text{Params}(I, i)$ and arguments $\mathbf{b} : \text{Args}(I, i, c_k)$.

The inductive eliminator $\texttt{ind}_I(s, z.P, f.\mathbf{y}.\mathbf{t})$ corresponds to a fixpoint immediately followed by a match: in Coq, one would write it `fix f s := match s as z return P with` $\mid c_1\ \mathbf{y} \Rightarrow t_1 \ldots \mid c_n\ \mathbf{y} \Rightarrow t_n$. The branches t_k are thus typed in a context where they are given access to a function f corresponding to a recursive call. Describing the exact guard condition to ensure termination is outside the scope of this presentation. We implicit assume in the rest of this paper that every fixpoint is guarded.

Fig. 1 presents bidirectional typing for CIC with several typing judgments. We reuse a discipline due to McBride [2018, 2019] in which the objects involved in these typing relations are split between

$\boxed{\vdash \Gamma}$

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

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

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

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

$$\frac{\Gamma \vdash a_m \triangleleft X_m[\mathbf{a}/\mathbf{x}]}{\Gamma \vdash I@i(\mathbf{a}) \triangleright \square_i} \qquad \frac{\Gamma \vdash a_m \triangleleft X_m[\mathbf{a}/\mathbf{x}] \Gamma \vdash b_n \triangleleft Y_n[\mathbf{a}/\mathbf{x}][\mathbf{b}/\mathbf{y}]}{\Gamma \vdash c_k^I@i(\mathbf{a}, \mathbf{b}) \triangleright I@i(\mathbf{a})}$$

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

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

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

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

$\boxed{\Gamma \vdash t \blacktriangleright_{\mathrm{h}} T}$

$$\frac{\Gamma \vdash t \triangleright T \qquad T \rightsquigarrow^* \Pi\, x : A.B}{\Gamma \vdash t \blacktriangleright_{\Pi} \Pi\, x : A.B} \qquad \frac{\Gamma \vdash t \triangleright T \qquad T \rightsquigarrow^* I@i\ \mathbf{a}}{\Gamma \vdash t \blacktriangleright_{\mathrm{I}} I@i\ \mathbf{a}} \qquad \frac{\Gamma \vdash t \triangleright T \qquad T \rightsquigarrow^* \square_i}{\Gamma \vdash t \blacktriangleright_{\square} \square_i}$$

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

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

$\boxed{t \equiv u}$

$$t \equiv u := \exists v, t \rightsquigarrow v \wedge u \rightsquigarrow v$$

Fig. 1. CIC: Bidirectional typing

three modes: inputs, subjects and outputs. In the *inference* judgment $\Gamma \vdash t \triangleright T$, the context is an input, the term is the subject, and the type is an output. In the *checking* judgment $\Gamma \vdash t \triangleleft T$, both context and type are inputs, and the term is the subject. Conversion is restricted to specific positions, namely to mediate between an inference and a checking judgment, and can thus never appear twice in a row without explicit annotation. In the context checking judgment $\vdash \Gamma$, the context is the subject.

We also appeal to a *constrained inference* judgment, $\Gamma \vdash t \blacktriangleright_{\mathrm{h}} T$, with the same modes as inference, which infers the type T under the constraint that its head constructor is h, where $h \in \mathbb{H}$:

$$\mathbb{H} := \square_i \mid \Pi \mid I \qquad \mathrm{head}\,(\Pi\, A\, B) := \Pi \qquad \mathrm{head}\,(\square_i) = \square_i \qquad \mathrm{head}\,(I\ \mathbf{a}) = I \qquad (1)$$

In usual presentations of CIC this judgment is kept implicit because a type has to reduce to a Π-type in order to validate the constraint. For instance, when typechecking an application $t\ u$, one starts by inferring a type for t, expecting a Π-type whose domain serves to check u. As we will see, making this judgment explicit is important when gradualizing CIC. Additionally, the input/subject/output distinction is particularly useful to turn typing into an elaboration procedure, as it clearly separates between inputs that can be used to elaborate the subject, and outputs that must be constructed. Each typing relation must ensure that its output are well-formed, under the

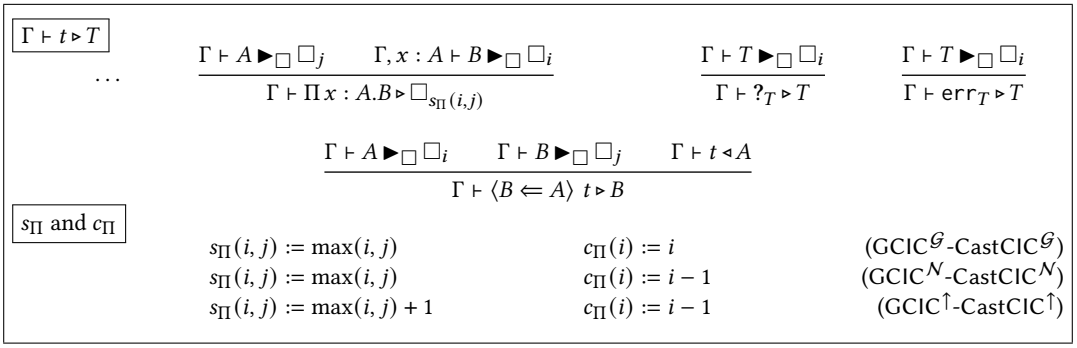


Fig. 2. CastCIC: Bidirectional typing (extending CIC Fig. 1), and parameters

hypothesis that its inputs are. Most properties hence need an explicit hypothesis that the involved contexts are well-formed, because contexts are never checked.

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, its typing, reduction and state its metatheoretical properties (§5.1). We then describe GCIC and its elaboration to CastCIC, along with few direct properties (§5.2). We then expose technical properties of the reduction of CastCIC (§5.3) used to prove the most important theorems on elaboration: conservativity over CIC or CIC$^\uparrow$, as well as elaboration and computational graduality (§5.4).

5.1 CastCIC

CastCIC is an extension of CIC 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. Casts keep track of the use of consistency during elaboration, implementing a form of runtime type-checking. We call *static* the terms of CastCIC without those, corresponding one-to-one with CIC terms.

CastCIC is parameterized by two functions to account for the three different variants of GCIC we consider. The first function s_Π computes the level of the universe of a dependent product, given the levels of its domain and codomain. The second function c_Π controls the universe levels in the reduction of casts between $? \to ?$ and $?$. The typing rules of CastCIC, extending those of CIC, and the three definitions of s_Π and c_Π are given in Fig. 2. The universe level of a Π-type is now provided by the parameter s_Π of the variant we are considering, rather than with a maximum. Note also that in this system no consistency appears when typing a cast, only plain conversion. When disambiguation is needed, we note this typing judgment as $\vdash_{\mathtt{cast}}$. Any derivation of $\vdash_{\mathtt{cast}}$ gives raise *mutatis mutandi* to a derivation without bidirectional information, closer to the judgment frequently employed in type theory.

The reduction rules associated to the new terms of CastCIC are presented in Fig. 3, omitting most congruences and using a let-form not present in the syntax as sugar. We reuse the notion of head constructor $h \in \mathbb{H}$ (see Eq. (1)). The germ $\mathrm{Germ}_i\ h$ constructs the least precise type with head h at level i, and is defined as (where $\mathbf{X} = \mathrm{Params}(I, i)$):

$$\mathrm{Germ}_i\ \square_j = \begin{cases} \square_j & (j < i) \\ \mathtt{err}_{\square_i} & (j \geq i) \end{cases} \qquad \mathrm{Germ}_i\ I = I\ ?_\mathbf{X} \qquad \mathrm{Germ}_i\ \Pi = \begin{cases} ?_{\square_{c_\Pi(i)}} \to ?_{\square_{c_\Pi(i)}} & (c_\Pi(i) \geq 0) \\ \mathtt{err}_{\square_i} & (c_\Pi(i) < 0) \end{cases}$$

The design of these reduction rules is enforced by the monotone model of CastCIC presented in §6. The rules for reducing casts fall under three categories: either the head constructor of both types

Reduction rules for cast:

$$\left\langle \Pi(x : B^{\mathrm{d}}).B^{\mathrm{c}} \Leftarrow \Pi(x : A^{\mathrm{d}}).A^{\mathrm{c}} \right\rangle\ (\lambda\, x : A^{\mathrm{d}}.t) \rightsquigarrow \lambda\, b : B^{\mathrm{d}}.\ \mathrm{let}\ a = \left\langle A^{\mathrm{d}} \Leftarrow B^{\mathrm{d}} \right\rangle\ b\ \mathrm{in}\ \left\langle (B^{\mathrm{c}}\ b) \Leftarrow (A^{\mathrm{c}}\ a) \right\rangle\ (t[a/x])$$

$$\left\langle ?_{\square_i} \Leftarrow \Pi(x : A^{\mathrm{d}}).A^{\mathrm{c}} \right\rangle\ f \rightsquigarrow \left\langle ?_{\square_i} \Leftarrow \mathrm{Germ}_i\ \Pi \right\rangle\ (\left\langle \mathrm{Germ}_i\ \Pi \Leftarrow \Pi(x : A^{\mathrm{d}}).A^{\mathrm{c}} \right\rangle\ f)$$

when $\Pi(x : A^{\mathrm{d}}).A^{\mathrm{c}} \neq \mathrm{Germ}_i\ \Pi$

$$\left\langle I(\mathbf{a}') \Leftarrow I(\mathbf{a}) \right\rangle\ c_k(\mathrm{a}, b_1, \ldots, b_n) \rightsquigarrow \begin{array}{l} \mathrm{let}\ b_1' = \langle Y_1\ \mathbf{a}' \Leftarrow Y_1\ \mathbf{a} \rangle\ b_1\ \mathrm{in}\ \ldots \\ \mathrm{let}\ b_n' = \langle (Y_n\ \mathbf{a}')[\mathbf{b}'/\mathbf{y}] \Leftarrow (Y_n\ \mathbf{a})[\mathbf{b}/\mathbf{y}] \rangle\ b_n\ \mathrm{in} \\ c_k(\mathrm{a}', b_1', \ldots, b_n') \end{array}$$

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

$$\langle \square_i \Leftarrow \square_i \rangle\ A \rightsquigarrow A \qquad \left\langle \square_j \Leftarrow \square_i \right\rangle\ A \rightsquigarrow \mathtt{err}_{\square_j} \quad \text{when } i \neq j$$

$$\left\langle I(\mathbf{a}') \Leftarrow I(\mathbf{a}) \right\rangle\ \star_{I(\mathbf{a})} \rightsquigarrow \star_{I(\mathbf{a}')} \qquad \left\langle ?_{\square_i} \Leftarrow \mathrm{Germ}_j\ h \right\rangle\ t \rightsquigarrow \mathtt{err}_{?_{\square_i}} \quad \text{when } j > i$$

$$\left\langle X \Leftarrow ?_{\square} \right\rangle\ \star_{?_{\square}} \rightsquigarrow \star_X \qquad \left\langle T' \Leftarrow T \right\rangle\ t \rightsquigarrow \mathtt{err}_{T'} \quad \text{when head } T \neq \text{head } T'$$

$$\left\langle X \Leftarrow ?_{\square_i} \right\rangle\ (\left\langle ?_{\square_i} \Leftarrow \mathrm{Germ}_i\ h \right\rangle\ t) \rightsquigarrow \langle X \Leftarrow \mathrm{Germ}_i\ h \rangle\ t \quad \text{when } \mathrm{Germ}_i\ h \neq \mathtt{err}_{\square_i}$$

$$\left\langle X \Leftarrow \mathtt{err}_{\square} \right\rangle\ t \rightsquigarrow \mathtt{err}_X \qquad \left\langle \mathtt{err}_{\square} \Leftarrow Z \right\rangle\ t \rightsquigarrow \mathtt{err}_{\mathtt{err}_{\square}} \quad \text{when head } Z \in \mathbb{H}$$

Reduction rules for ? and err:

$$\star_{\Pi(x:A).B} \rightsquigarrow \lambda(x : A).\ \star_B \qquad \mathtt{ind}_I(\star_{I(\mathbf{a})}, z.P, f.\mathbf{y}.\mathbf{t}) \rightsquigarrow \star_{P[\star_{I(\mathbf{a})}/z]} \qquad \frac{t \rightsquigarrow u}{\star_t \rightsquigarrow \star_u}$$

Notation: $\star_X$ stands for both $?_X$ and $\mathtt{err}_X$

Fig. 3. CastCIC: Reduction rules (congruence rules omitted)

match and the cast reduces recursively to casts on the argument of the types, or there is a mismatch leading to a failure err, or else an unknown type is involved. We denote the reflexive, transitive closure of this reduction as $\rightsquigarrow^*$. This presentation ensures the following properties on CastCIC:

Theorem 2 (Properties of CastCIC). CastCIC *enjoys:*

- ***Confluence:*** *if* $T \equiv U$ *there exists* S *such that* $T \rightsquigarrow^* S$ *and* $U \rightsquigarrow^* S$.
- ***Subject reduction:*** *if* $\Gamma \vdash_{\mathsf{cast}} t \triangleright A$ *and* $t \rightsquigarrow t'$ *then* $\Gamma \vdash_{\mathsf{cast}} t' \triangleleft A$
- ***Strong normalization*** *for* $\mathsf{CastCIC}^{\mathcal{N}}$ *and* $\mathsf{CastCIC}^{\uparrow}$
- ***Weak canonicity*** *(W)*

Proof sketch. We extend the notion of parallel reduction for CIC from [Sozeau et al. 2020] to account for our additional reduction rules and show that the triangle property (the existence of an optimal reduced term in one step) still holds. Subject reduction can be derived from the injectivity of type constructors, which is a direct consequence of confluence. The translation induced by the discrete model §6.1 maps each reduction step to at least one step, see Theorem 14. So strong normalization holds whenever the target calculus of the translation is normalizing. To get weak canonicity, we show a specific form of progress: if $\vdash_{\mathsf{cast}} t : \mathbb{B}$ then t takes a step or it is already a canonical form (true, false, the dynamic failure $\mathtt{err}_{\mathbb{B}}$ or the unknown term $?_{\mathbb{B}}$). □

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

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

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

Fig. 4. CastCIC: α-consistency

5.2 Elaboration from GCIC to CastCIC

GCIC simply extends the syntax of CIC with an unknown term $?_{@i}$ at each universe level. Its typing is *defined* by the elaboration judgment from GCIC to CastCIC. Elaboration is bidirectional, using three judgments that augment the typing ones of Fig. 1 with an extra output: the elaborated CastCIC term. This mutual typing-elaboration is required because of the intricate interdependence between typing and reduction (§3). The subject of the judgment is a source term in GCIC, and both inputs and outputs are target terms in CastCIC (colors should help with readability, but are not essential).

Instead of standard conversion, elaboration employs *consistent conversion* to compare CastCIC terms. Consistent conversion is essentially conversion modulo *α-consistency* $\sim_\alpha$ which is an extension of α-conversion that accommodates for imprecision, defined in Fig. 4. Apart from the standard rules making $?$ consistent with any term, α-consistency optimistically ignores casts that could interfere in the comparison, and does not consider errors consistent with any term.

Definition 1 (Consistent conversion). *Two terms are* consistently convertible*, or simply* consistent*, noted $s \sim t$, iff there exists s' and t' such that $s \rightsquigarrow^* s'$, $t \rightsquigarrow^* t'$ and $s' \sim_\alpha t'$.*

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 fundamental building block of the conservativity of GCIC wrt. CIC, is that it corresponds to conversion on static terms.

Proposition 3 (Properties of consistent conversion).

(1) two static terms are consistently convertible iff 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 reduction of static terms in CastCIC is the same as 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 from GCIC to CastCIC is given in Fig. 5, closely following the bidirectional presentation of CIC (Fig. 1). The salient features of this process are the two cast insertions to mediate between merely consistent but not convertible types: first, at transition between checking and inference; second, during constrained inference. Indeed, a term inferring unknown $?_{\square_i}$ is cast to $\mathrm{Germ}_i\ h$, the least precise type verifying the constraint of the head constructor h. Also note that $?$ is elaborated to $?_{?_\square}$ and the context is responsible for inserting the appropriate cast, *e.g.* $? :: T$ reduces

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

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

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

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

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

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

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

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

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

$\boxed{\Gamma \vdash t \rightsquigarrow t \blacktriangleright_\mathrm{h} T}$

$$\frac{\Gamma \vdash \tilde{t} \rightsquigarrow t \triangleright T \qquad T \rightsquigarrow^* \square_i}{\Gamma \vdash \tilde{t} \rightsquigarrow t \blacktriangleright_\square \square_i} \qquad \frac{\Gamma \vdash \tilde{t} \rightsquigarrow t \triangleright T \qquad T \rightsquigarrow^* \Pi x : A.B}{\Gamma \vdash \tilde{t} \rightsquigarrow t \blacktriangleright_\Pi \Pi x : A.B} \qquad \frac{\Gamma \vdash \tilde{t} \rightsquigarrow t \triangleright T \qquad T \rightsquigarrow^* I(\mathbf{a})}{\Gamma \vdash \tilde{t} \rightsquigarrow t \blacktriangleright_\mathrm{I} I(\mathbf{a})}$$

$$\frac{\Gamma \vdash \tilde{t} \rightsquigarrow t \triangleright T \qquad T \rightsquigarrow^* ?_{\square_i}}{\Gamma \vdash \tilde{t} \rightsquigarrow \langle \mathrm{Germ}_i\ h \Leftarrow T \rangle\ t \blacktriangleright_\mathrm{h} \mathrm{Germ}_i\ h}$$

Fig. 5. Type-directed elaboration from GCIC to CastCIC

to $?_T$ after elaboration. Thus only universe level annotations of $?@i$ are necessary to elaborate a term.

In order to obtain uniqueness of elaboration, we fix a reduction strategy for $\rightsquigarrow^*$ to unveil the head constructor in premises of constrained inference judgments, typically weak-head reduction. The reduction rules then immediately translate to an algorithm for elaboration. Coupled with the decidability of consistency (Prop. 3), this makes elaboration decidable in $\mathrm{GCIC}^{\mathcal{N}}$ and $\mathrm{GCIC}^{\uparrow}$, although the same algorithm might diverge in $\mathrm{GCIC}^{\mathcal{G}}$.

Let us already state two soundness properties of elaboration: it is correct, insofar as it produces well-typed terms, and functional, as we just hinted.

Theorem 4 (Correctness of elaboration). *The elaboration produces well-typed terms and contexts: if* $\vdash_{\mathsf{cast}} \Gamma$ *and* $\Gamma \vdash \tilde{t} \rightsquigarrow t \triangleright T$*, then* $\Gamma \vdash_{\mathsf{cast}} t \triangleright T$*.*

Theorem 5 (Uniqueness of elaboration). *Given* Γ *and* $\tilde{t}$*, there is at most one* t *and one* T *such that* $\Gamma \vdash \tilde{t} \rightsquigarrow t \triangleright T$*.*

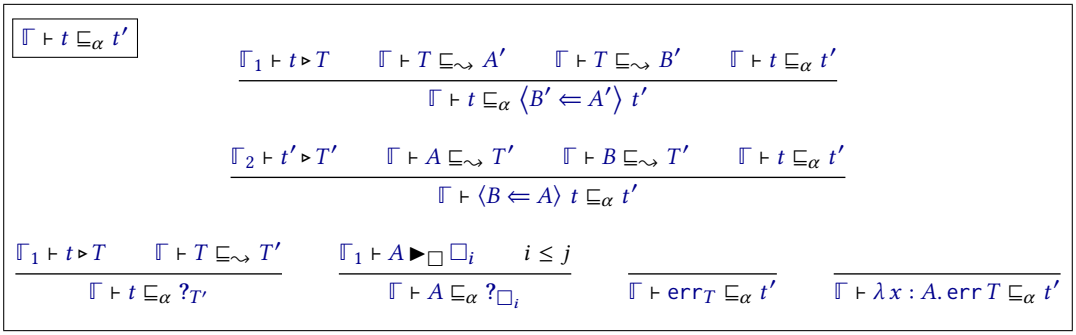


Fig. 6. CastCIC: Structural precision (Congruence rules omitted)

5.3 Precision and Reduction

To establish the graduality of elaboration, we need properties on reduction in CastCIC. For that, we need a notion of precision in CastCIC that is a simulation for reduction, *i.e.* that less precise terms reduce at least as well as more precise ones. To handle casts that might appear or disappear in one term but not the other during reduction, this notion of precision cannot be purely syntactical: it must give the possibility to ignore some casts, *i.e.* allow $\langle T \Leftarrow S \rangle\ t$ to be less precise than t' if $t \sqsubseteq_\alpha t'$, although we impose some restriction on the types involved to prevent unwanted failure. We call this notion *structural precision*, $\sqsubseteq_\alpha$, defined hereafter. This approach is similar to the proof of dynamic gradual guarantee in [Siek et al. 2015], although the presence of computation in the domain and codomain of casts makes the proof quite more challenging.

Because of typing, we need to record the two contexts of the term compared with precision. We do so by using double-struck letters, writing $\mathbb{\Gamma}, x : A \mid A'$ for context extensions. We use $\mathbb{\Gamma}_i$ for projections, *i.e.* $(\mathbb{\Gamma}, x : A \mid A')_1 := \mathbb{\Gamma}, x : A$, and write $\Gamma \mid \Gamma'$ for the converse pairing operation.

Definition 2 (Structural precision in CastCIC). Structural precision, *denoted* $\mathbb{\Gamma} \vdash t \sqsubseteq_\alpha t'$, *is defined in Fig. 6. We write* $\vdash \Gamma \sqsubseteq_\alpha \Gamma'$ *for the pointwise extension to contexts, and* $\sqsubseteq_\alpha\ \mathbb{\Gamma}$ *if* $\vdash \mathbb{\Gamma}_1 \sqsubseteq_\alpha \mathbb{\Gamma}_2$.

For the proof of simulation, we also need to consider structural reduction up-to reduction, that we call definitional precision, denoted $\mathbb{\Gamma} \vdash t \sqsubseteq_\rightsquigarrow t'$, and meaning that there exists s and s' such that $t \rightsquigarrow^* s$, $t' \rightsquigarrow^* s'$ and $\mathbb{\Gamma} \vdash s \sqsubseteq_\rightsquigarrow s'$. The proof of simulation relies on catch-up lemmas, uncovering head-constructors in less precise terms.

Lemma 6 (Type catch-up). *Under the hypothesis that* $\sqsubseteq_\alpha\ \mathbb{\Gamma}$, *if* $\mathbb{\Gamma} \vdash \Pi x : A.B \sqsubseteq_\rightsquigarrow T'$, $\mathbb{\Gamma}_1 \vdash \Pi x : A.B \triangleright \square_i$ *and* $\mathbb{\Gamma}_2 \vdash T' \blacktriangleright_\square \square_j$ *then* $T' \rightsquigarrow^* ?_{\square_j}$ *and* $i \leq j$, *or* $T' \rightsquigarrow^* \Pi x : A'.B'$ *for some* A' *and* B' *such that* $\mathbb{\Gamma} \vdash \Pi x : A.B \sqsubseteq_\rightsquigarrow \Pi x : A'.B'$.

We have similar catch-up properties for $\square_i$, $?_{\square_i}$ *and* $I(\mathbf{a})$ *replacing* $\Pi x : A.B$.

Now is the point where the difference between the three different variants of CastCIC manifest: the next lemma holds in full generality only in CastCIC$^{\mathcal{G}}$ and CastCIC$^{\uparrow}$. 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 we put on casts wrt. precision. This is the manifestation in the reduction of the embedding-projection property [New and Ahmed 2018]. The lemma is however still true in CastCIC$^{\mathcal{N}}$ if one restricts to terms without ?, where those casts never happen.

Lemma 7 (λ-abstraction catch-up). *If* $\mathbb{\Gamma} \vdash \lambda x : A.t \sqsubseteq_\alpha s'$, *where* t *is not an error,* $\mathbb{\Gamma}_1 \vdash \lambda x : A.t \triangleright \Pi x : A.B$ *and* $\mathbb{\Gamma}_2 \vdash s' \blacktriangleright_\Pi \Pi x : A'.B'$, *then* $s' \rightsquigarrow^* \lambda x : A'.t'$ *with* $\mathbb{\Gamma} \vdash \lambda x : A.t \sqsubseteq_\alpha \lambda x : A'.t'$.

A similar lemma can be proven for terms of the shape $?_{I(\mathbf{a})}$ and $c_k(\mathbf{a}, \mathbf{b})$, in all three variants of CastCIC.

$$\frac{}{x \sqsubseteq_\alpha x} \qquad \frac{}{\square_i \sqsubseteq_\alpha \square_i} \qquad \frac{A \sqsubseteq_\alpha A' \qquad t \sqsubseteq_\alpha t'}{\lambda\, x : A.t \sqsubseteq^{\mathrm{G}}_\alpha \lambda\, x : A.t} \qquad \frac{t \sqsubseteq_\alpha t' \qquad u \sqsubseteq_\alpha u'}{t\ u \sqsubseteq_\alpha t'\ u'} \qquad \cdots \qquad \frac{}{t \sqsubseteq_\alpha ?}$$

Fig. 7. GCIC: Syntactic precision (Extract)

We now come to the most important property of this section, the advertised simulation. As above, the proposition holds in CastCIC$^{\mathcal{G}}$, CastCIC$^{\uparrow}$ and for terms without ? is CastCIC$^{\mathcal{N}}$.

Theorem 8 (Simulation of reduction). *If $\Gamma_1 \vdash t \triangleright T$, $\Gamma_2 \vdash t' \triangleright T'$, $\Gamma \vdash t \sqsubseteq_\alpha t'$ and $t \rightsquigarrow^* s$ then there exists s' such that $t' \rightsquigarrow^* s'$ and $\Gamma \vdash t' \sqsubseteq_\alpha s'$.*

Proof. The hardest point is to simulate β and ι redexes. This is where we use Lemma 7, 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 in 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. □

Corollary 9 (Reduction and types). *If $\Gamma_1 \vdash T \blacktriangleright_\square \square_i$, $\Gamma_2 \vdash T' \blacktriangleright_\square \square_j$, $\Gamma \vdash T \sqsubseteq_\alpha T'$ and $T \rightsquigarrow^* \Pi\, x : A.B$, then either $T' \rightsquigarrow^* ?_{\square_j}$ with $i \leq j$, or $T' \rightsquigarrow^* \Pi\, x : A'.B'$ and $\Gamma \vdash \Pi\, x : A.B \sqsubseteq_\alpha \Pi\, x : A'.B'$. Similar properties hold for $\square_i$, $?_{\square_i}$ and $I(\mathbf{a})$ replacing $\Pi\, x : A.B$.*

Note that head reductions are simulated using head reductions and the reductions of Lemma 6 are also head reductions. Thus, Corollary 9 still holds when restricting to head reductions.

Corollary 10 (Monotony of consistency). *If $\Gamma \vdash T \sqsubseteq_\alpha T'$, $\Gamma \vdash S \sqsubseteq_\alpha S'$ and $T \sim S$ then $T' \sim S'$.*

Proof. We have $T \rightsquigarrow^* U$ and $S \rightsquigarrow^* V$ such that $U \sim_\alpha V$. Simulate those reductions to get some U' and V' such that $\Gamma_1 \vdash U \sqsubseteq_\alpha U'$ and $\Gamma_1 \vdash V \sqsubseteq_\alpha V'$. Now we only need to show that syntactic consistency is monotone wrt. structural precision, which is direct. □

5.4 Properties of GCIC

Elaboration systematically inserts casts during checking, thus even static terms are not elaborated to themselves. This is why the statement of conservativity uses a (partial) erasure function ε, that tries to take terms of CastCIC to term of CIC by erasing all casts. It holds in all three systems, typability being of course taken into the corresponding variant of CIC: full CIC for GCIC$^{\mathcal{G}}$ and GCIC$^{\mathcal{N}}$, and CIC$^{\uparrow}$ for GCIC$^{\uparrow}$.

Theorem 11 (Conservativity). *If $\vdash t \triangleright T$ then $\vdash t \rightsquigarrow t' \triangleright T'$ for some t' and T' such that $\varepsilon(t') = t$ and $\varepsilon(T') = T$. Conversely if t is a static term and $\vdash t \rightsquigarrow t' \triangleright T'$ for some t' and T', then $\vdash t \triangleright \varepsilon(T')$.*

Proof. The main difficulty is to ensure that the extra casts inserted by elaboration do not block reduction. For this we maintain the property that all terms considered in CastCIC are such that their erasure is both more and less precise than themselves, and never contains ?. This is enough, together with Theorem 8, to prove that they reduce in exactly the same way as their erasures. □

Next, we turn to elaboration graduality. It is stated wrt. *syntactic precision* of GCIC terms, defined in Fig. 7. Elaboration graduality holds in the two systems that (will be proven to) satisfy $\mathcal{G}$, *i.e.* GCIC$^{\mathcal{G}}$ and GCIC$^{\uparrow}$. Note that, with using universe polymorphism [Sozeau and Tabareau 2014], we could get rid of the extra assumption on universe levels using universe variables instead of integers, letting the typing of the elaborated term constrain these variables.

Theorem 12 (Elaboration Graduality). *If* $\vdash \tilde{t} \rightsquigarrow t \triangleright T$, $\tilde{t} \sqsubseteq^{\mathrm{G}}_{\alpha} \tilde{s}$ *and each subterm of* $\tilde{t}$ *that is against a* $?@i$ *in* $\tilde{s}$ *elaborates to a term whose type is in* $\square_i$, *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. Here again the technical difficulties arise in the rules involving reduction. This is where Corollary 9 for the constrained inference is useful, proving that the less structurally precise term obtained by induction outputs a less precise type. Similarly Corollary 10 proves that in the checking rule the less precise types are still consistent. □

Last but not least, the computational graduality, that again holds only in GCIC$^{\mathcal{G}}$ and GCIC$^{\uparrow}$. The proof is deferred to §6.5, where we develop the required monotone models. For simplicity it is stated for booleans, but it holds more generally on inductive types.

Theorem 13 (Computational graduality). *If* $\vdash_{\mathsf{cast}} s \triangleleft \mathbb{B}$, $\vdash_{\mathsf{cast}} t \triangleleft \mathbb{B}$, $\vdash s \sqsubseteq_{\alpha} t$ *and* $t \rightsquigarrow^{*} \mathtt{err}_{\mathbb{B}}$, *then* $s \rightsquigarrow^{*} \mathtt{err}_{\mathbb{B}}$.

6 REALIZING CastCIC

We realize CastCIC in two ways to obtain the missing steps towards confluence, normalization and graduality. First through a simple implementation of casts in §6.1 using case-analysis on types as well as exceptions resulting in the **discrete model**. Then, by building a more elaborate **monotone model** to define and reason about precision. 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 hierarchy of universes in §6.4 and put these pieces together in §6.5, culminating in a proof of graduality.. In both the discrete and monotone case, the choice of a normalizing model appears when building the hierarchy of universes and tying the knot with the unknown type. The models embed into a variant of CIC with induction-recursion [Dybjer and Setzer 2003] and function extensionality (for the monotone model), with judgement $\vdash_{\mathrm{IR}}$. We use Agda as a pratical counterpart to typecheck the components of the models (the code can be found as an anonymous supplementary material, with explanations in appendix) and assume standard metatheoretical properties (those still being open problems to our knowledge).

6.1 Discrete Models of CastCIC

The discrete model explains away CastCIC by translating it to CIC implementing the new constants of CastCIC 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.

The general theory of ExTT leads us to interpret each inductive type I by an extended inductive type $\tilde{I}$, containing two new constructors $?_{\tilde{I}}$ and $✠_{\tilde{I}}$, corresponding respectively to $?_I$ and $\mathtt{err}_I$ of CastCIC; see Fig. 11, left part, for an example. In the rest of this section, we only illustrate inductives on natural numbers. The definition of exceptions at all types is then defined by case analysis on the type in Fig. 8.

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. 9 containing dependent product (π), universes ($\mathfrak{u}$), inductives (*e.g.* $\mathfrak{nat}$) as well as codes $\tilde{?}$ for the unknown type and ✠ for the error type. Accompanying the inductive definition of $\mathbb{U}_i$, the recursively defined decoding function El provides a semantics for these codes. In particular, the error ✠ is decoded to the type $\top$ containing a unique element $* \equiv ?_{✠} \equiv \mathtt{err}_{✠}$. The unknown $\tilde{?}$ is decoded to the extended dependent sum consisting of

$$\star_{\pi\, A\, B} := \lambda\, x : \mathrm{El}\, A.\star_{B\, x} \qquad ?_{\mathfrak{u}_j} := ?^j \qquad ?_? := ?_{\tilde{\Sigma}\, \mathbb{H}\, \mathrm{Germ}} \qquad ?_{\mathfrak{nat}} := ?_{\mathbb{N}}$$
$$\star_{\maltese} := * \qquad \mathtt{err}_{\mathfrak{u}_j} := \maltese^j \qquad \mathtt{err}_? := \maltese_{\tilde{\Sigma}\, \mathbb{H}\, \mathrm{Germ}} \qquad \mathtt{err}_{\mathfrak{nat}} := \maltese_{\mathbb{N}}$$

Fig. 8. Realization of exceptions ($\star$ stands for either ? or err)

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

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

Fig. 9. Inductive-recursive encoding of the discrete universe hierarchy

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

$$\begin{array}{lcll}
\mathtt{cast}\ \mathfrak{u}_j\ \mathfrak{u}_j\ A & := & A & \\
\mathtt{cast}\ \mathfrak{u}_j\ ?^i\ A & := & (\square_j; A) & \text{if } j < i \\
\mathtt{cast}\ \mathfrak{u}_j\ X\ A & := & \maltese_X & \text{otherwise}
\end{array}
\qquad
\begin{array}{lcll}
\mathtt{cast}\ ?^i\ Z\ (c; x) & := & \mathtt{cast}\ (\mathrm{Germ}_i\, c)\ Z\ x & \\
\mathtt{cast}\ ?^i\ Z\ ?_? & := & ?_Z & \\
\mathtt{cast}\ ?^i\ Z\ \maltese_? & := & \maltese_Z & \text{otherwise}
\end{array}$$

$$\mathtt{cast}\ \maltese\ Z\ * := \maltese_Z \qquad \text{(Omitting inductives)}$$

Fig. 10. Implementation of cast (discrete models)

pairs $(h; t)$ of a head constructor $h \in \mathbb{H}_i$[9] and $t \in \mathrm{Germ}_i\, h$, with two additional constructors $?_{\tilde{\Sigma}}, \maltese_{\tilde{\Sigma}}$ freely added.

Crucially, the code for Π-types depends on the choice of parameter for $s_\Pi(i, j)$. For the system CastCIC$^{\mathcal{G}}$, the inductive-recursive definition of $\mathbb{U}_i$ is not well-founded: the decoding of ? depends through Germ on functions over itself, because $c_\Pi(s_\Pi(i, i)) = s_\Pi(i, i)$. Such arbitrary fixpoints are only achievable in a partial type theory. This is why CastCIC$^{\mathcal{G}}$ does not satisfy $\mathcal{N}$.

In order to maintain normalization, the construction of the unknown type and the universe thus needs to be stratified, which is possible when $c_\Pi(s_\Pi(i, i)) < s_\Pi(i, i)$. It is the case 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 universe $\mathbb{U}_i$ and the decoding function El at level i without trouble.

Equipped with these, we define $\mathtt{cast} : \Pi(A : \mathbb{U}_i)(B : \mathbb{U}_j).A \to B$ in Fig. 10 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 ?). We consider the translation defined by $[\![A]\!] := \mathrm{El}\, [A]$ on types, and recursively on terms using:

$$[\square_i] := \mathfrak{u}_i \qquad [\Pi\, x : A.B] := \pi\ [A]\ (\lambda\, x : [\![A]\!].[B]) \qquad [\langle B \Leftarrow A \rangle\ t] := \mathtt{cast}\ [A]\ [B]\ [t]$$

[9] We stratify the head constructors $\mathbb{H}$ (see Eq. (1)) according to the universe level i.

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

Fig. 11. Order structure on extended natural numbers

Theorem 14 (Discrete syntactic model). *The sketched translation* $[-]$ *(see Fig. 15) satisfies:*

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

Proof. 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 $\langle ?_{\square_i} \Leftarrow \mathrm{Germ}_i\, h \rangle\ t$ is in one-to-one correspondence with the one-step reduced form of its translation $(h; [t]) : \tilde{\Sigma}\mathbb{H}_i \mathrm{Germ}_i$. The second part is proved by a direct induction on the typing derivation of $\Gamma \vdash_{\mathsf{cast}} t : A$, using that $\vdash_{\mathrm{IR}} ?, \mathtt{err} : \Pi(A : \mathbb{U}_i)\, \mathrm{El}\, A$, $\vdash_{\mathrm{IR}} \mathtt{cast} : \Pi(A : \mathbb{U}_i)(B : \mathbb{U}_i) \to \mathrm{El}\, A \to \mathrm{El}\, B$, and the first part for the conversion rule. □

6.2 Poset-based Models of Dependent Type Theory

The simplicity of the discrete model is 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 $\sqsubseteq^A$ modeling precision between terms. Each term and type constructor is monotone with respect to these orders, providing a strong form of graduality.

As an illustration, the order on extended natural numbers (Fig. 11) 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, 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 3 (Embedding-projection pairs). *An ep-pair* $d : A \twoheadrightarrow B$ *between posets* A, B *consists of*

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

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 $d : A \twoheadrightarrow B$ an embedding-projection pair. Being an ep-pair is a property of the underlying relation.

By monotony, 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 $B_{a,a'} : B\, a \twoheadrightarrow B\, a'$ for each $a \sqsubseteq^A a'$. These ep-pairs need to satisfy a functoriality condition, which ensures that heterogeneous transitivity is well defined: $B_{a,a'}(b, b') \wedge B_{a',a''}(b', b'') \Rightarrow B_{a,a''}(b, b'')$.

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 $a \leq^A a' \Rightarrow B_{a,a'}\ (f\ a)\ (f\ a')$. Moreover, given ep-pairs $d_A : A \twoheadrightarrow A'$ and $d_B : B \twoheadrightarrow B'$[10] we can build an induced ep-pair $d_\Pi : \Pi^{\text{mon}}\ A\ B \twoheadrightarrow \Pi^{\text{mon}}\ A'\ B'$ with underlying relation $d_\Pi(f, f') = d_A(a, a') \Rightarrow d_B(f\ a, f'\ a')$, $\uparrow_{d_\Pi} f = \uparrow_{d_B} \circ f \circ \downarrow_{d_A}$ and $\downarrow_{d_\Pi} f = \downarrow_{d_B} \circ f \circ \uparrow_{d_A}$.

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) $?_I$ is above a constructor whenever the arguments of the constructor are themselves dominated by their respective $?$, and (4) each constructor is monotone with respect to the order on its arguments. Similarly to dependent product, an ep-pair $\mathbf{X} \twoheadrightarrow \mathbf{X}'$ between the parameters of an inductive type I induces an ep-pair $I\,\mathbf{X} \twoheadrightarrow I\,\mathbf{X}'$.

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 $\tilde{\Sigma}(h : \mathbb{H}_i)\,\text{Germ}_i\,h$ from the discrete model with the precision relation generated by the rules:

$$\maltese_? \sqsubseteq z \qquad ?_? \sqsubseteq ?_? \qquad \frac{h \equiv h' \qquad x\ {}_{\text{Germ}\,h}\sqsubseteq_{\text{Germ}\,h'}\ x'}{(h; x) \sqsubseteq (h'; x')} \qquad \frac{x \sqsubseteq ?_{\text{Germ}\,h}}{(h; x) \sqsubseteq ?_?} \tag{2}$$

In order to globally satisfy $\mathcal{G}$, $\overline{?}$ should admit an ep-pair $d_h : \text{Germ}_i\,h \twoheadrightarrow ?_{\mathfrak{u}_i}$ for any $h \in \mathbb{H}_i$ corresponding to $\text{Germ}_i\,h \sqsubseteq ?_{\mathfrak{u}_i}$. Embedding an element $x \in \text{Germ}\,h$ by $\uparrow_{d_h} x = (h; x)$ and projecting out of $\text{Germ}\,h$ by the following equations form a reasonable candidate.

$$\downarrow_{d_h} (h, x) = x, \qquad \downarrow_{d_h} (h', x) = \maltese_{\text{Germ}\,h}\ \ (h \neq h'), \qquad \downarrow_{d_h} ? = ?_{\text{Germ}\,h}, \qquad \downarrow_{d_h} \maltese = \maltese_{\text{Germ}\,h}.$$

Note that to compute the first two cases of $\downarrow_{d_h}$, we rely on $\mathbb{H}$ having decidable equality. However, for $\uparrow_{d_h} \dashv \downarrow_{d_h}$ to be adjoints, we need the following relation to hold:

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

Extending precision with the left-hand side imposes in turn that the now equivalent terms $\maltese_?$ and $(h, \maltese_{\text{Germ}\,h})$ must be quotiented to recover antisymmetry of the order $\sqsubseteq^{\overline{?}}$. We show that with this quotient, $\overline{?}$ effectively admits 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 $\text{El} : \mathbb{U}_i \to \square$ presented in Fig. 12. The relation of **precision** $\sqsubseteq$: $\mathbb{U}_i \to \mathbb{U}_j \to \square$ presented below is an order (Theorem 15) 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. This unique decomposition is at the heart of the reduction of the cast operator given in Fig. 3. Such a derivation of precision $A \sqsubseteq B$ gives rise through decoding to ep-pairs $\text{El}_\varepsilon\ (A \sqsubseteq B) : A \twoheadrightarrow B$, with underlying relation noted ${}_A\sqsubseteq_B : \text{El}\,A \to \text{El}\,B \to \square$.

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 $\pi\,A\,B, B : \text{El}\,A \to \mathbb{U}_i$ is a monotone function with respect to the order on $\text{El}\,A$ 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

[10] A larger amount of data is actually required to handle dependency of B over A; we refer to the accompanying Agda development

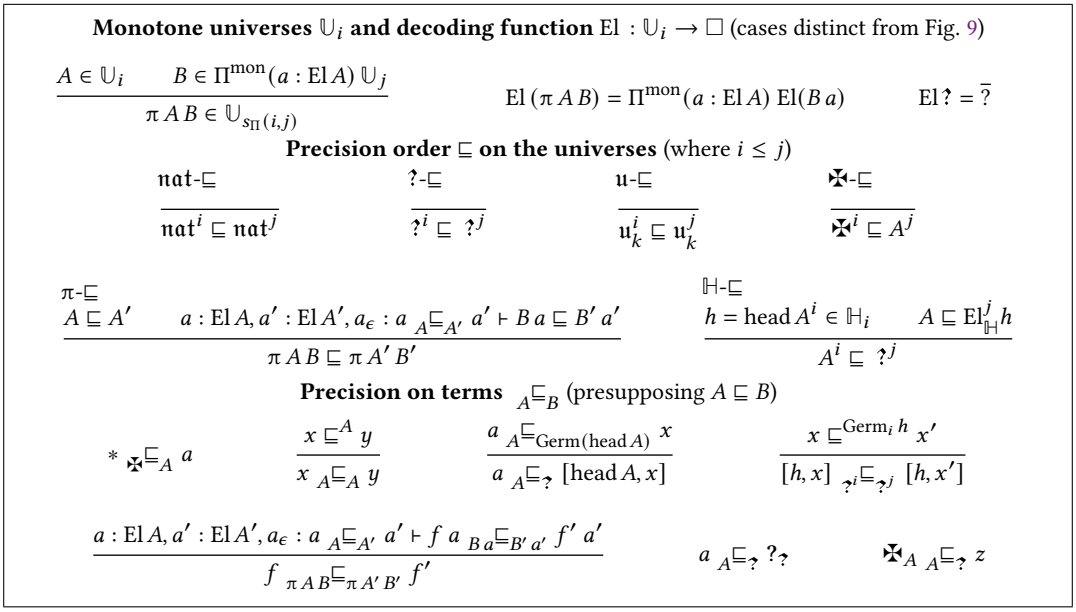


Fig. 12. Monotone universe of codes and precision

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 15 (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 $?^i$; in particular, $A \sqsubseteq ?^i$ for any $A : \mathbb{U}_i$.
(3) $\mathrm{El} : \mathbb{U}_i \to \square$ is an indexed-poset 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[11]: the equality on these types is irrelevant.

Proof. 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 depending on the particular property. □

6.5 Monotone Models of CastCIC

The monotone translation $\{-\}$ presented hereafter 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 15:

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

Note that this definition formally depends on a chosen universe level for $?$, 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 monotoncity of `cast`, thus hold by definition. However, as a consequence the translation does not validate the computation rules of CastCIC on the nose: `cast` can get stuck on type variables eagerly. They still hold propositionally so that we have at least a model in an extensional variant ECIC[12] of CIC.

[11] Uniqueness of Identity Proofs; in HoTT parlance, $\mathbb{U}_i$ and $\mathrm{El}\, A$ are hSets.
[12] ECIC enjoy *equality reflection*: two terms are *definitionally* equal whenever they are *propositionally* so.

$$\{\!|\ A\ |\!\} = \mathrm{El}\,\{A\} : \square \qquad \{\lambda\, x : A.\ t\} = (\lambda\, x : \{\!|\ A\ |\!\}.\{t\}, \{\lambda\, x : A.\ t\}_\varepsilon)$$
$$\{\!|\ A\ |\!\}_\varepsilon = \mathrm{El}_\varepsilon\,\{A\}_\varepsilon : \Pi(a\, a' : \{\!|\ A\ |\!\}).\square \qquad \{\lambda\, x : A.\ t\}_\varepsilon = \lambda(x\, x' : \{\!|\ A\ |\!\})(x_\varepsilon : \{\!|\ A\ |\!\}_\varepsilon\, x\, x').\ \{t\}_\varepsilon$$

Fig. 13. Translation of the monotone model (excerpt)

Lemma 16. *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 17. *The translation* $\{-\}$ *of Fig. 13 extends to a model of* CastCIC *into* CIC *extended with induction-recursion and functional extensionality: if* $\Gamma \vdash_{\mathsf{cast}} t : A$ *then* $\{\!|\ \Gamma\ |\!\} \vdash_{\mathrm{IR}} \{t\} : \{\!|\ A\ |\!\}$.

Connecting the discrete and monotone models. Looking at the two translations, they clearly coincide on ground types such as $\mathbb{N}$. On dependent products over ground types, for instance $\mathbb{N}\to\mathbb{N}$, the monotone interpretation is more conservative $\{\!|\ \mathbb{N}\to\mathbb{N}\ |\!\} \subset [\![\mathbb{N}\to\mathbb{N}]\!]$, keeping only the monotone functions. Extending the sketched correspondence at higher types, we obtain a logical relation $\wr - \int$, for which we can prove the basis lemma: if $\Gamma \vdash_{\mathsf{cast}} t : A$ 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.

Back to computational graduality. The precision induced by the monotone model can be reflected back to CastCIC. To this hand, we define the *propositional precision* judgment $\Gamma \vdash_{\mathsf{cast}} t\ {}_T\sqsubseteq_S\ u$ on typed CastCIC terms as the existence of a proof e witnessing $\{\!|\ \Gamma\ |\!\}_\varepsilon \vdash_{\mathrm{IR}} e : \{t\}\ {}_{\{T\}}\sqsubseteq_{\{S\}}\ \{u\}$.

Lemma 18 (Properties of propositional precision). *The propositional precision verifies:*

- *If* $\vdash t \sqsubseteq_\alpha u$ *then* $\vdash_{\mathsf{cast}} t \sqsubseteq u$.
- *If* $\vdash_{\mathsf{cast}} t \sqsubseteq \mathsf{err}_{\mathsf{nat}}$ *then* $t \equiv \mathsf{err}_{\mathsf{nat}}$.

A direct corollary of this lemma is that both $\mathrm{GCIC}^{\mathcal{G}}$ and $\mathrm{GCIC}^{\uparrow}$ satisfies computational graduality, which is the key missing point of §5 and the *raison d'etre* of the monotone model.

7 RELATED WORK

Bidirectional typing and unification. Bidirectional elaboration is a common feature in implemented proof assistants, 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. 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 [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].

Partial and Directed type theories. Interpretations of type theories in ordered structures goes back to various works on 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. In the restricted setting of types refined with decidable logical predicates, Lehmann and Tanter [2017] exploit the Abstracting Gradual Typing (AGT) methodology [Garcia et al. 2016] to design a language with imprecise formulas and implication. 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 valid 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.

8 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. In doing so, we bridge the gap between recent advances in syntactic models of type theory, including effectful ones, and in gradual typing, in particular thanks to the characterization of graduality as embedding-projection pairs.

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 weakly consistent gradual features together with the standard CIC. 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 Coq, possibly with different modes reflecting the different theories developed here (just as now one can run Coq with type-in-type, strict propositions, etc).

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

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.

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*, Yves Bertot and Viktor Vafeiadis (Eds.). ACM, 182–194. https://doi.org/10.1145/3018610.3018620

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.

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.

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.

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 (Lecture Notes in Computer Science, Vol. 1158)*, Stefano Berardi and Mario Coppo (Eds.). Springer, 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.

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

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. 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, Daniel R. Licata, and Amal Ahmed. 2019. Gradual Type Theory. See[POPL 2019 2019], 15:1–15:31.
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. 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.
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 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
Matthieu Sozeau and Nicolas Tabareau. 2014. Universe Polymorphism in Coq. In *Interactive Theorem Proving*, Gerwin Klein and Ruben Gamboa (Eds.). Springer International Publishing, Cham, 499–514.
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.
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.
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 of properties of CastCIC, that are useful for in the next sections. They are very standard technical building blocks of type theories, and should not be very surprising. The main interesting point 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 one variable. We prove by (mutual) induction on the typing derivation 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* $S \equiv T$.

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 an extra property to prove that a type convertible to a type with head constructor h reduces to a type with the same head constructor, which is a consequence of confluence. □

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

Proof. Once again, this is a routine induction on the inference derivation, using subject reduction to handle the reductions in the constrained inference rules, to ensure the reduced type is still well-formed. □

A.2 Elaboration from GCIC to CastCIC

The clear distinction between inputs and outputs forced by the bidirectional approach makes the proof of correction almost trivial, by distinguishing objects that are already well-formed from objects whose well-formedness we should ensure.

Proof of Theorem 4. The proof is by induction on the elaboration derivation, mutually with similar properties for all typing judgments.

The first key point is to ensure that context extensions we do in the rules are valid, *i.e.* we only consider $\Gamma', x : A'$ when we know that $\vdash \Gamma', x : A'$, so that we can use the induction hypothesis. The second is to use the validity property of CastCIC to ensure that all inserted casts are well-typed. □

As already hinted in the text, the uniqueness of elaboration is also eased by the bidirectional approach, once we fix a reduction strategy.

Proof of Theorem 5. Uniqueness of elaboration is proven by induction on the elaboration derivation, together with uniqueness for the other elaboration judgments.

The main argument is that there is always at most one rule that can apply to get the desired conclusion. This is true for all inference statement 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 constrained inference, by confluence of CastCIC the inferred type cannot at the same time reduce to $?_{\square}$ and $\square$ (resp. a Π-type or an inductive type), because those do not have a common reduct. Thus, only one of the two rules can apply. Because of the fixed reduction strategy, the inferred types is moreover unique. □

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

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

$$\frac{\Gamma \vdash A \sqsubseteq_\alpha A' \qquad \Gamma, x : A \mid A' \vdash t \sqsubseteq_\alpha \tilde{t}}{\Gamma \vdash \lambda x : A.t \sqsubseteq_\alpha \lambda x : \tilde{A}.\tilde{t}} \qquad \frac{\Gamma \vdash A \sqsubseteq_\alpha \tilde{A} \qquad \Gamma \vdash B \sqsubseteq_\alpha \tilde{B} \qquad \Gamma \vdash t \sqsubseteq_\alpha \tilde{t}}{\Gamma \vdash \langle B \Leftarrow A \rangle\ t \sqsubseteq_\alpha \left\langle \tilde{B} \Leftarrow \tilde{A} \right\rangle\ \tilde{t}}$$

$$\frac{\Gamma \vdash t \sqsubseteq_\alpha \tilde{t} \qquad \Gamma \vdash u \sqsubseteq_\alpha \tilde{u}}{\Gamma \vdash t\ u \sqsubseteq_\alpha \tilde{t}\ \tilde{u}} \qquad \frac{\Gamma \vdash \mathbf{a} \sqsubseteq_\alpha \tilde{\mathbf{a}} \qquad i = \tilde{i}}{\Gamma \vdash I(\mathbf{a}) \sqsubseteq_\alpha I(\tilde{\mathbf{a}})} \qquad \frac{\Gamma \vdash \mathbf{a} \sqsubseteq_\alpha \tilde{\mathbf{a}} \qquad \Gamma \vdash \mathbf{b} \sqsubseteq_\alpha \tilde{\mathbf{b}} \qquad i = \tilde{i}}{\Gamma \vdash c_k(\mathbf{a}, \mathbf{b}) \sqsubseteq_\alpha c_k(\tilde{\mathbf{a}}, \tilde{\mathbf{b}})}$$

$$\frac{\begin{array}{c}\Gamma \vdash s \sqsubseteq_\alpha \tilde{s} \qquad \Gamma_1 \vdash s \blacktriangleright_1 I(\mathbf{a}) \qquad \Gamma_2 \vdash \tilde{s} \blacktriangleright_1 I(\tilde{\mathbf{a}}) \qquad \Gamma, z : I(\mathbf{a}) \mid I(\tilde{\mathbf{a}}) \vdash P \sqsubseteq_\alpha \tilde{P} \\ \Gamma, f : (\Pi z : I(\mathbf{a}), P) \mid (\Pi z : I(\tilde{\mathbf{a}}), \tilde{P}), \mathbf{y} : \mathbf{Y_k}[\mathbf{a}/\mathbf{x}] \mid \mathbf{Y_k}[\tilde{\mathbf{a}}/\mathbf{x}] \vdash t_k \sqsubseteq_\alpha \tilde{t}_k \\ \Gamma_1, f : (\Pi z : I(\mathbf{a}), P), \mathbf{y} : \mathbf{Y_k}[\mathbf{a}/\mathbf{x}] \vdash t_k \triangleright T_k \\ \Gamma, f : (\Pi z : I(\mathbf{a}), P) \mid (\Pi z : I(\tilde{\mathbf{a}}), \tilde{P}), \mathbf{y} : \mathbf{Y_k}[\mathbf{a}/\mathbf{x}] \mid \mathbf{Y_k}[\tilde{\mathbf{a}}/\mathbf{x}] \vdash T_k \sqsubseteq_\rightsquigarrow \tilde{P}[?_{I(\tilde{\mathbf{a}})}/z]\end{array}}{\Gamma \vdash \mathtt{ind}_I(s, P, \mathbf{t}) \sqsubseteq_\alpha \mathtt{ind}_I(\tilde{s}, \tilde{P}, \tilde{\mathbf{t}})}$$

$$\frac{\Gamma_1 \vdash t \triangleright T \qquad \Gamma \vdash T \sqsubseteq_\rightsquigarrow \tilde{A} \qquad \Gamma \vdash T \sqsubseteq_\rightsquigarrow \tilde{B} \qquad \Gamma \vdash t \sqsubseteq_\alpha \tilde{t}}{\Gamma \vdash t \sqsubseteq_\alpha \left\langle \tilde{B} \Leftarrow \tilde{A} \right\rangle\ \tilde{t}}$$

$$\frac{\Gamma_2 \vdash \tilde{t} \triangleright \tilde{T} \qquad \Gamma \vdash A \sqsubseteq_\rightsquigarrow \tilde{T} \qquad \Gamma \vdash B \sqsubseteq_\rightsquigarrow \tilde{T} \qquad \Gamma \vdash t \sqsubseteq_\alpha \tilde{t}}{\Gamma \vdash \langle B \Leftarrow A \rangle\ t \sqsubseteq_\alpha \tilde{t}}$$

$$\frac{\Gamma_1 \vdash t \triangleright T \qquad \Gamma \vdash T \sqsubseteq_\rightsquigarrow \tilde{T}}{\Gamma \vdash t \sqsubseteq_\alpha ?_{\tilde{T}}} \qquad \frac{\Gamma_1 \vdash A \blacktriangleright_\square \square_i \qquad i \leq j}{\Gamma \vdash A \sqsubseteq_\alpha ?_{\square_i}}$$

$$\frac{}{\Gamma \vdash \mathtt{err}_T \sqsubseteq_\alpha \tilde{t}} \qquad \frac{}{\Gamma \vdash \lambda x : A.\, \mathtt{err}\, T \sqsubseteq_\alpha \tilde{t}}$$

$\boxed{\Gamma \vdash t \sqsubseteq_\rightsquigarrow \tilde{t}}$

$$\frac{\Gamma \vdash t \sqsubseteq_\alpha \tilde{t}}{\Gamma \vdash t \sqsubseteq_\rightsquigarrow \tilde{t}} \qquad \frac{\Gamma \vdash t \sqsubseteq_\alpha \tilde{t} \qquad s \rightsquigarrow t}{\Gamma \vdash s \sqsubseteq_\rightsquigarrow \tilde{t}} \qquad \frac{\Gamma \vdash t \sqsubseteq_\alpha \tilde{t} \qquad \tilde{s} \rightsquigarrow \tilde{t}}{\Gamma \vdash t \sqsubseteq_\rightsquigarrow \tilde{s}}$$

Fig. 14. Structural and definitional precisions

A.3 Precision and Reduction

First thing first, the extended definition of precision. The only point to note, apart from the non-congruence cases already present in the text, are the extra assumptions in the congruence ind rule. The typing assumptions are there to provide us with the context needed to type-check the predicate and branches, and the extra definitional precision, although a consequence of the other premises once the simulation is proven, is needed to provide a strong enough induction hypothesis to conclude when proving this simulation.

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

Lemma 19 (Weakening of precision). *If* $\Gamma \vdash t \sqsubseteq_\alpha \tilde{t}$*, then* $\Gamma, \Delta \vdash t \sqsubseteq_\alpha \tilde{t}$ *for any* Δ.

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

Lemma 20 (Substitution and precision). *If* $\Gamma, x : S \mid \tilde{S}, \Delta \vdash t \sqsubseteq_\alpha \tilde{t}$, $\Gamma \vdash u \sqsubseteq_\alpha \tilde{u}$, $\Gamma_1 \vdash u \triangleleft S$ *and* $\Gamma_2 \vdash \tilde{u} \triangleleft \tilde{S}$ *then* $\Gamma, \Delta[u \mid \tilde{u}/x] \vdash t[u/x] \sqsubseteq_\alpha \tilde{t}[\tilde{u}/x]$.

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

Next, the extended catch-up lemmas.

Lemma 21 (Type catch-up). *We have the following property, under the hypothesis that* $\sqsubseteq_\alpha \Gamma$*:*

1. *if* $\Gamma \vdash \square_i \sqsubseteq_\rightsquigarrow T'$ *and* $\Gamma_2 \vdash T' \blacktriangleright_\square \square_j$, *then* $T' \rightsquigarrow^* ?_{\square_j}$ *and* $i + 1 \leq j$ *or* $T' \rightsquigarrow^* \square_i$;
2. *if* $\Gamma \vdash ?_{\square_i} \sqsubseteq_\rightsquigarrow T'$ *and* $\Gamma_2 \vdash T' \blacktriangleright_\square \square_j$, *then* $T' \rightsquigarrow^* ?_{\square_j}$ *and* $i \leq j$;
3. *if* $\Gamma \vdash \Pi\, x : A.B \sqsubseteq_\rightsquigarrow T'$, $\Gamma_1 \vdash \Pi\, x : A.B \triangleright \square_i$ *and* $\Gamma_2 \vdash T' \blacktriangleright_\square \square_j$ *then* $T' \rightsquigarrow^* ?_{\square_j}$ *and* $i \leq j$, *or* $T' \rightsquigarrow^* \Pi\, x : A'.B'$ *for some* A' *and* B' *such that* $\Gamma \vdash \Pi\, x : A.B \sqsubseteq_\rightsquigarrow \Pi\, x : A'.B'$;
4. *if* $\Gamma \vdash I(\mathbf{a}) \sqsubseteq_\rightsquigarrow T'$, $\Gamma_1 \vdash I(\mathbf{a}) \triangleright \square_i$ *and* $\Gamma_2 \vdash T' \blacktriangleright_\square \square_j$ *then* $T' \rightsquigarrow^* ?_{\square_j}$ *and* $i \leq j$, *or* $T' \rightsquigarrow^* I(\mathbf{a}')$ *for some* a' *such that* $\Gamma \vdash I(\mathbf{a}) \sqsubseteq_\rightsquigarrow I(\mathbf{a}')$.

Proof. All four proofs follow the same structure, we detail the first one, which is done by induction on the precision derivation, mutually with the same property for structural precision.

Let us start with the proof of the property for structural precision. Using the precision hypothesis, we can decompose $\tilde{T}$ into $\langle S_n \Leftarrow T_{n-1}\rangle \ldots \langle S_2 \Leftarrow T_1\rangle\ U$, where $\tilde{T}$ is either $\square_i$ or $?_S$ for some S, and we have $\Gamma \vdash \square_{i+1} \sqsubseteq_\alpha S_k$, $\Gamma \vdash \square_{i+1} \sqsubseteq_\alpha S'_k$ and possibly $\Gamma \vdash \square_{i+1} \sqsubseteq_\alpha S$. 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 $\tilde{T}$ type-checks against $\square_j$, we must have $S_n \equiv \square_j$. This implies that it 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

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

we can reduce all casts. We thus get $\tilde{T} \rightsquigarrow^* \square_i$ or $\tilde{T} \rightsquigarrow^* ?_{\square_{i+1}}$, as expected.

For the definitional precision, if $\Gamma \vdash \square_i \sqsubseteq_\rightsquigarrow \tilde{T}$ then by decomposing the precision derivation there is an $\tilde{S}$ such that $\tilde{T} \rightsquigarrow^* \tilde{S}$, $\Gamma \vdash \square_i \sqsubseteq_\alpha \tilde{S}$, and by subject reduction $\Gamma_1 \vdash \tilde{S} \blacktriangleright_\square \square_j$. By induction hypothesis, either $\tilde{S} \rightsquigarrow^* \square_i$ or $\tilde{S} \rightsquigarrow^* ?_{\square_{i+1}}$, and composing both reductions we get the desired result.

The next three properties are proven in a similar fashion, apart from the fact that the lemma just proven is used instead of the induction hypothesis. □

Proof of Lemma 7. As for Lemma 21, decompose $\tilde{s}$ into $\left\langle S_n \Leftarrow S'_{n-1}\right\rangle \ldots \left\langle S_2 \Leftarrow S'_1\right\rangle\ \tilde{u}$, where $\tilde{u}$ is either $\lambda\, x : \tilde{A}.\tilde{t}$ or $?_S$ for some S. Because all of the S_k, S'_k and possibly S are definitionally less precise than $\Pi\, x : A.B$, by the previous lemma they all reduce all to either some $?_{\square_j}$ with $i \leq j$, or some $\Pi\, x : \tilde{A}.\tilde{B}$, and by typing it must be the second for S_n. By the rule $\langle X \Leftarrow ?_\square\rangle\ ?_{?_\square} \rightsquigarrow ?_X$, if S is not a Π-type, we can reduce the casts until it is, then use the rule $?_{\Pi\, x:\tilde{A}.\tilde{B}} \rightsquigarrow \lambda\, x : \tilde{A}.\tilde{B}$. Thus without loss of generality we can suppose $\tilde{u}$ is some $\lambda\, x : \tilde{A}.\tilde{t}$.

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

If it is $\left\langle ?_{\square_j} \Leftarrow \mathrm{Germ}_j\ \Pi\right\rangle\ \tilde{u}$, then by typing this cannot be the outermost cast, and thus we can use the rule $\left\langle X \Leftarrow ?_{\square_j}\right\rangle\ \left\langle ?_{\square_j} \Leftarrow \mathrm{Germ}_j\ \Pi\right\rangle\ X \rightsquigarrow \left\langle X \Leftarrow \mathrm{Germ}_j\ \Pi\right\rangle\ \tilde{u}$ to reduce it.

In the second case, the cast is some $\langle \Pi\, x : A_2.B_2 \Leftarrow \Pi\, x : A_1.B_1\rangle\ \lambda\, x : A_1.\tilde{t}$ that reduces to $\lambda\, x : A_2.\langle B_2 \Leftarrow B_1[a/x]\rangle\ \tilde{t}[a/x]$ where a is $\langle A_1 \Leftarrow A_2\rangle\ x$. Inverting the precision hypothesis, we obtain that $\Gamma \vdash A \sqsubseteq_\alpha A_1$, $\Gamma \vdash A \sqsubseteq_\alpha A_2$, $\Gamma, x : A \mid A_1 \vdash B \sqsubseteq_\alpha B_1$, $\Gamma, x : A \mid A_2 \vdash B \sqsubseteq_\alpha B_2$ and

$\Gamma, x : A \mid A_1 \vdash t \sqsubseteq_\alpha \tilde{t}$, and we can use weakening to deduce from those that the reduct is still less precise than $\lambda\, x : A.t$.

The last case corresponds to $\left\langle ?\square_j \Leftarrow \Pi\, x : \tilde{A}.\tilde{B} \right\rangle \tilde{u}$ when $\Pi\, x : \tilde{A}.\tilde{B}$ is not $\mathrm{Germ}_j\, h$, which is reduced to $\left\langle ?\square_j \Leftarrow ?\square_{c_\Pi(j)} \to ?\square_{c_\Pi(j)} \right\rangle \left\langle ?\square_{c_\Pi(j)} \to ?\square_{c_\Pi(j)} \Leftarrow \Pi\, x : \tilde{A}.\tilde{B} \right\rangle \tilde{u}$. 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 $\Pi\, x : A.B$. For $?\square_j$ and $\Pi\, x : \tilde{A}.\tilde{B}$ it is already an hypothesis. For $?\square_{c_\Pi(j)} \to ?\square_{c_\Pi(j)}$, 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 $s_\Pi(k, l) \leq j$, thus $k \leq c_\Pi(j)$ and $l \leq c_\Pi(j)$. Therefore $\Gamma \vdash A \sqsubseteq_\alpha ?\square_{c_\Pi(j)}$, and similarly for B.

Thus, all casts must reduce, and each of those reductions preserves precision, so we end up with a term $\lambda\, x : \tilde{A}.\tilde{t}$ such that $\Gamma \vdash \lambda\, x : A.t \sqsubseteq_\alpha \lambda\, x : \tilde{A}.\tilde{t}$, as expected. □

Lemma 22 (Constructors and inductive error catch-up). *If* $\Gamma \vdash c_k(\mathbf{a}, \mathbf{b}) \sqsubseteq_\alpha s'$, $\Gamma_1 \vdash c_k(\mathbf{a}, \mathbf{b}) \triangleright I(\mathbf{a})$ *and* $\Gamma_2 \vdash s' \blacktriangleright_{\mathrm{I}} I(\mathbf{a}')$, *then either* $s' \rightsquigarrow^* ?_{I(\mathbf{a}')}$ *or* $s' \rightsquigarrow^* c_k(\mathbf{a}', \mathbf{b}')$ *with* $\Gamma \vdash c_k(\mathbf{a}, \mathbf{b}) \sqsubseteq_\alpha c_k(\mathbf{a}', \mathbf{b}')$.

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

Proof. The proofs are very similar to the abstraction case. The main difference is that we must treat the case of $?_{I(\tilde{\mathbf{a}})}$ differently from $?_{\Pi\, x:\tilde{A}.\tilde{B}}$, because on an inductive type it does not reduce. However, the rule $\langle I(\mathbf{a_2}) \Leftarrow I(\mathbf{a_1}) \rangle\ ?_{I(\mathbf{a_1})} \rightsquigarrow ?_{I(\mathbf{a_2})}$ can be used instead. □

From this, the simulation follows. We state it in full, *i.e.* together with the corresponding property on definitions precision.

Theorem 23 (Simulation of reduction). *The two following properties are true:*

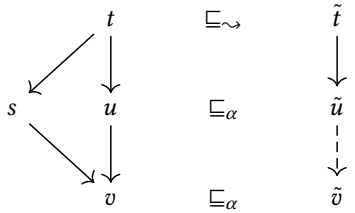
(1) if $\sqsubseteq_\alpha \Gamma$, $\Gamma_1 \vdash t \triangleright T$, $\Gamma_2 \vdash \tilde{t} \triangleright \tilde{T}$, $\Gamma \vdash t \sqsubseteq_\alpha \tilde{t}$ *and* $t \rightsquigarrow^* s$ *then there exists* $\tilde{s}$ *such that* $\tilde{t} \rightsquigarrow^* \tilde{s}$ *and* $\Gamma \vdash \tilde{t} \sqsubseteq_\alpha \tilde{s}$,

(2) if $\sqsubseteq_\alpha \Gamma$, $\Gamma_1 \vdash t \triangleright T$, $\Gamma_2 \vdash \tilde{t} \triangleright \tilde{T}$, $\Gamma \vdash t \sqsubseteq_\rightsquigarrow \tilde{t}$ *and* $t \rightsquigarrow^* s$ *then* $\Gamma \vdash s \sqsubseteq_\rightsquigarrow \tilde{t}$.

Proof. Both are shown by mutual induction on the precision derivation.

Denotational precision

The second point is the easiest. By definition of definitional precision, there exists u and $\tilde{u}$, reducts respectively of t and $\tilde{t}$, and such that $\Gamma \vdash u \sqsubseteq_\alpha \tilde{u}$. By confluence, there exists some v that is a reduct of both u and s. By subject reduction, t and $\tilde{t}$ are all well-typed, and thus by induction hypothesis, there exists some $\tilde{v}$ such that $v \rightsquigarrow^* \tilde{v}$ and $\Gamma \vdash v \sqsubseteq_\alpha \tilde{v}$. But then v is a reduct of s and $\tilde{v}$ is a reduct of $\tilde{t}$, and so $\Gamma \vdash s \sqsubseteq_\rightsquigarrow \tilde{t}$.



As a direct corollary, we have that if $\Gamma \vdash t \triangleright T$, $\Gamma \vdash T \sqsubseteq_\rightsquigarrow \tilde{T}$, $t \rightsquigarrow^* s$ and $\Gamma_1 \vdash s \triangleright S$, then $\Gamma \vdash S \sqsubseteq_\rightsquigarrow \tilde{T}$. Indeed $\Gamma_1 \vdash s \triangleleft T$ by subject reduction, thus S and T are convertible, and have a common reduct U by confluence. By what we just proved, this gives $\Gamma \vdash U \sqsubseteq_\rightsquigarrow \tilde{T}$, thus $\Gamma \vdash S \sqsubseteq_\rightsquigarrow \tilde{T}$.

Syntactical precision — Congruence rules

It is enough to show that one step of reduction can be simulated.

First, we consider the congruence rules. The CastCIC and cast congruence rules on t can be simulated by similar congruence rules on $\tilde{t}$. To handle the possibility of non-diagonal cast rules,

with hypothesis involving the inferred type of t, we use the previous corollary, that shows that any type definitionally less precise than the type of t is also definitionally less precise than the type of s. As for the error congruence rule it can be ignored, since an error is smaller than anything else. Thus, it suffices to consider the cases where the reduction happens at the head of t, and $\tilde{t}$ has the same head, *i.e.* we already got rid of all casts surrounding $\tilde{t}$ if they existed.

Syntactical precision – β and ι redexes

Next we consider the case of β or ι redexes. If the redex is an applied $(\lambda x : A.\, \mathtt{err}_T)\ u$, then the reduct is $\mathtt{err}_T$ and must be still smaller that $\tilde{t}$. Otherwise, Lemmas 7 and 22 apply, and all casts around the destructed term (*i.e.* the λ-abstraction or inductive constructor) can be reduced while keeping the syntactic precision relation between t and $\tilde{t}$. Then the β or ι reduction of t can then be simulated with another β or ι reduction in $\tilde{t}$, using the substitution property of precision to conclude. There is a special case, in the situation where the scrutinee of the less precise ι-redex is $?_{I(\tilde{\mathbf{a}})}$. In that case, this redex reduces to $?_{\tilde{P}[?_{I(\tilde{\mathbf{a}})}/z]}$, and we must show this term to be less precise that $t_k[\lambda x : I(\mathbf{a}).\, \mathtt{ind}_I(x, P, \mathbf{t})/z][\mathbf{b}/\mathbf{y}]$ of type T_k. But the precision rule for the inductor gives that $\mathbb{\Gamma} \vdash T_k \sqsubseteq_{\rightsquigarrow} \tilde{P}[?_{I(\mathbf{a})}/z]$, which is exactly what we need to conclude – this is the reason for that extraneous assumption: giving us the needed induction hypothesis here.

Syntactical precision – error and ? reductions

If the reduction of t is any reduction to an error, including some of the specific error reduction rules, then there is no need for a simulation because the error is more precise than anything. Similarly, if the reduction is $?_{\Pi x:A.B} \rightsquigarrow \lambda x : A.\, \mathtt{err}_B$, the reduct is more precise than anything again.

Conversely, let us consider the ? reduction rules. If t is $?_{\Pi x:A.B}$ and reduces to $\lambda x : A.B$, then $\tilde{t}$ must be $?_T$ with $\mathbb{\Gamma} \vdash \Pi x : A.B \sqsubseteq_{\rightsquigarrow} T$. Thus $T \rightsquigarrow^* ?_{?_\square}$ or $T_{\Pi x:\tilde{A}.\tilde{B}}$. In the first case, $\tilde{t}$ is still less precise than $\lambda x : A.B$, and in the second case, it can simulate the reduction with the same one. If t is $\mathtt{ind}_I(?_{I(\mathbf{a})}, P, \mathbf{b})$, reducing to $?_{P[?_{I((a))}/z]}$, we use the lemma on cast elimination around constructors to conclude that also $\tilde{t}$ must reduce to some $\mathtt{ind}_I(?_{I(\tilde{\mathbf{a}})}, \tilde{P}, \tilde{\mathbf{b}})$ that is less precise than t. From this, $\tilde{t} \rightsquigarrow ?_{\tilde{P}[?_{I((\tilde{a}))}/z]}$, which is less precise than the reduct of t.

Syntactical precision – non-diagonal cast

Next, let us turn to the case where t is $\langle B \Leftarrow A \rangle\ u$ with $\mathbb{\Gamma} \vdash u \sqsubseteq_\alpha \tilde{t}$ (*i.e.* the precision was obtained with a non-diagonal rule). There are four possibilities. The first one is when the cast fails, then the error is of course smaller than $\tilde{t}$.

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 $\tilde{t}$. In the case of cast expansion, we must use non-diagonal precision rules, and thus prove that the type of $\tilde{t}$ is less precise than the introduced germ. This is because the inferred type of $\tilde{t}$ is definitionally less precise than some $?_{\square_i}$, and the germ considered is $\mathrm{Germ}_i\ h$. It is enough to prove that $\mathbb{\Gamma} \vdash \mathrm{Germ}_i\ h \sqsubseteq_\alpha ?_{\square_i}$, which is true because for all h, if the germ is not an error (which it is here as it appeared in a reduction) we have $\mathbb{\Gamma}_1 \vdash \mathrm{Germ}_i\ h \triangleright j$ for some $j \leq i$.

The third case is when A and B are both Π-types or inductive types, and u starts with a λ or an inductive constructor. In that case, by the previous lemmas $\tilde{t}$ reduces to a term $\tilde{u}$ with the same head constructor as u, and by the substitution property of precision we have $\mathbb{\Gamma} \vdash s \sqsubseteq_\alpha \tilde{u}$.

In the fourth case, t is $\left\langle X \Leftarrow ?_{\square_i} \right\rangle \left\langle ?_{\square_i} \Leftarrow \mathrm{Germ}_i\ h \right\rangle\ u$ reducing to $\langle X \Leftarrow \mathrm{Germ}_i\ h \rangle\ u$. If $\mathbb{\Gamma} \vdash u \sqsubseteq_\alpha \tilde{t}$, then we directly have $\mathbb{\Gamma} \vdash \langle X \Leftarrow \mathrm{Germ}_i\ h \rangle\ u \sqsubseteq_\alpha \tilde{t}$. Otherwise, $\tilde{t}$ is some $\left\langle \tilde{B} \Leftarrow \tilde{A} \right\rangle\ \tilde{u}$ and we have $\mathbb{\Gamma} \vdash u \sqsubseteq_\alpha \tilde{u}$, $\mathbb{\Gamma}_1 \vdash \mathrm{Germ}_i\ h \sqsubseteq_{\rightsquigarrow} \tilde{A}$. Moreover, the non-diagonal left cast rule gives $\mathbb{\Gamma}_1 \vdash X \sqsubseteq_{\rightsquigarrow} \tilde{B}$ because $\mathbb{\Gamma}_2 \vdash \left\langle \tilde{B} \Leftarrow \tilde{A} \right\rangle\ \tilde{u} \triangleright \tilde{B}$. Thus $\mathbb{\Gamma} \vdash \langle X \Leftarrow \mathrm{Germ}_i\ h \rangle\ u \sqsubseteq_\alpha \left\langle \tilde{B} \Leftarrow \tilde{A} \right\rangle\ \tilde{u}$.

Syntactical precision – diagonal cast

This only leaves us with the diagonal rule for casts: we are given $\langle T \Leftarrow S \rangle\ t$ and $\left\langle \tilde{T} \Leftarrow \tilde{S} \right\rangle\ \tilde{t}$ that are pointwise comparable, and such that $\langle T \Leftarrow S \rangle\ t$ reduces, and we must show that $\left\langle \tilde{T} \Leftarrow \tilde{S} \right\rangle\ \tilde{t}$

simulates that reduction. We consider the reductions from top to bottom, ignoring all the ones that give an error, as those are trivial to simulate.

First, we are in the situation of $\langle \Pi\, x : A_2.B_2 \Leftarrow \Pi\, x : A_1.B_1 \rangle\ \lambda\, x : A_1.u$. If u is $\mathtt{err}_{B_1}$ then the reduct is more precise than any term. If S reduces to $?_\square$ then $\tilde{t}$ must reduce to $?_{?_\square}$ because it is less precise than $\lambda\, x : A_1.u$ and by typing it cannot start with a λ. In that case, $\left\langle \tilde{T} \Leftarrow \tilde{S} \right\rangle\ \tilde{t} \rightsquigarrow ?_{\tilde{T}}$, and since $\Gamma \vdash \Pi\, x : A_2.B_2 \sqsubseteq_\alpha \tilde{T}$, we have that $\Gamma \sqsubseteq_\alpha s \sqsubseteq_\alpha ?_{\tilde{T}}$. Otherwise S reduces to some $\Pi\, x : \tilde{A}_1.\tilde{B}_1$, and $\tilde{t}$ reduces to some $\lambda\, x : \tilde{A}_1.\tilde{u}$. If T reduces to some $\Pi\, x : \tilde{A}_2.\tilde{B}_2$, then the whole term can do the same reduction as t, and the substitution property of precision enables us to conclude. Thus, the only case left is that of $\left\langle ?_{\square_i} \Leftarrow \Pi\, x : \tilde{A}_1.\tilde{B}_1 \right\rangle\ \lambda\, x : \tilde{A}_1.\tilde{t}$. If $\Pi\, x : \tilde{A}_1.\tilde{B}_1$ is $\mathrm{Germ}_i\ \Pi$, then all of A_1, A_2, B_1 and B_2 are less precise than $?_{\square_{c_\Pi(i)}}$, and this is enough to conclude that s is less precise than $\left\langle \mathrm{Germ}_i\ \Pi \Leftarrow ?_{\square_i} \right\rangle\ \lambda\, x :?_{\square_{c_\Pi(i)}} \tilde{t}$. The last case is when $\Pi\, x : \tilde{A}_1.\tilde{B}_1$ is not a germ. Then there must first be an expansion through $\mathrm{Germ}_i\ \Pi$, followed by a reduction of the cast between $\Pi\, x : \tilde{A}_1.\tilde{B}_1$ and $\mathrm{Germ}_i\ \Pi$. As the case just before, we have enough precisions of the domains and codomains to get precision between the reducts by substitution. The reasoning is similar in the corresponding case for inductive types.

Next, let us consider $\left\langle ?_{\square_i} \Leftarrow \Pi\, x : A_1.B_1 \right\rangle\ f$. We have that $\tilde{T} \rightsquigarrow ?_{\square_j}$ with $i \leq j$, and thus $\Gamma \vdash \mathrm{Germ}_i\ \Pi \sqsubseteq_\rightsquigarrow \tilde{T}$. Thus, using a diagonal rule for the innermost cast, and a non-diagonal rule for the outermost one, we conclude $\Gamma \vdash \left\langle ?_{\square_i} \Leftarrow \Pi\, x : A_1.B_1 \right\rangle\ f \sqsubseteq_\alpha \left\langle \tilde{T} \Leftarrow \tilde{S} \right\rangle\ \tilde{f}$. The reasoning is similar is the corresponding case for inductive types.

As for $\langle \square_i \Leftarrow \square_i \rangle\ A$, then we can turn the diagonal rule into a non-diagonal one: indeed $\Gamma_1 \vdash A \triangleleft \square_i$ by typing, thus $\Gamma_1 \vdash A \triangleright T$ and $T \rightsquigarrow \square_i$. Thus, as $\Gamma \vdash \square_i \sqsubseteq_\rightsquigarrow \tilde{T}$, we have $\Gamma \vdash T \sqsubseteq_\rightsquigarrow \tilde{T}$ and similarly $\Gamma \vdash T \sqsubseteq_\rightsquigarrow \tilde{S}$. Therefore, $\Gamma \vdash A \sqsubseteq_\alpha \left\langle \tilde{T} \Leftarrow \tilde{S} \right\rangle\ \tilde{A}$.

Finally, the last case is $\left\langle X \Leftarrow ?_{\square_i} \right\rangle\ \left\langle ?_{\square_i} \Leftarrow \mathrm{Germ}_i\ h \right\rangle\ t$, compared with $\left\langle \tilde{X} \Leftarrow \tilde{S} \right\rangle\ \tilde{t}$. If $\Gamma \vdash t \sqsubseteq_\alpha \tilde{t}$ (*i.e.* there was a non-diagonal precision rule on the innermost cast), then we simply have $\Gamma \vdash \langle X \Leftarrow \mathrm{Germ}_i\ h \rangle\ t \sqsubseteq_\alpha \left\langle \tilde{X} \Leftarrow \tilde{S} \right\rangle\ \tilde{t}$ by a diagonal rule, as $\Gamma \vdash \mathrm{Germ}_i\ h \sqsubseteq_\rightsquigarrow \tilde{S}$ since $\Gamma \vdash ?_{\square_i} \sqsubseteq_\rightsquigarrow \tilde{S}$. Otherwise, we compare $\langle \mathrm{Germ}_i\ h \Leftarrow X \rangle\ t$ with $\left\langle \tilde{X} \Leftarrow ?_{\square_j} \right\rangle\ \left\langle ?_{\square_j} \Leftarrow \tilde{S} \right\rangle\ \tilde{t}$ (after reduction of the types less precise than $?_{\square_i}$ to some $?_{\square_j}$ with $i \leq j$). We can use a diagonal rule of the outermost cast, and a non-diagonal one on the innermost, as $\Gamma \vdash \mathrm{Germ}_i\ h \sqsubseteq_\rightsquigarrow ?_{\square_j}$ since $i \leq j$. □

Corollary 24 (Reduction and types). *If* $\Gamma_1 \vdash T \blacktriangleright_\square \square_i$, $\Gamma_2 \vdash T' \blacktriangleright_\square \square_j$, $\Gamma \vdash T \sqsubseteq_\alpha T'$ *then*

- *if* $T \rightsquigarrow^* ?_{\square_j}$ *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* $T' \rightsquigarrow^* ?_{\square_i}$ *with* $i \leq j$, *or* $T' \rightsquigarrow^* I(\mathbf{a}')$ *and* $\Gamma \vdash I(\mathbf{a}) \sqsubseteq_\alpha I(\mathbf{a}')$.

A.4 Properties of GCIC

Let us first define the erasure function that is used in the statement of conservativity.

Definition 4 (Erasure). *Erasure* ε *is a partial function from the syntax of* CastCIC *to the syntax of* CIC, *that is undefined on* $?$ *and* $\mathtt{err}$, *is such that* $\varepsilon(\langle B \Leftarrow A \rangle\ t) = \varepsilon(t)$ *and is a congruence for all other term constructors.*

We say that a context Γ *of* CastCIC *is erasable if both* $\vdash \Gamma \sqsubseteq_\alpha \varepsilon(\Gamma)$ *and* $\vdash \varepsilon(\Gamma) \sqsubseteq_\alpha \Gamma$. *When it is clear from the context, we write simply* $\Gamma \vdash t \sqsubseteq_\alpha s$ *instead of either* $\Gamma \mid \varepsilon(\Gamma) \vdash t \sqsubseteq_\alpha s$ *or* $\varepsilon(\Gamma) \mid \Gamma \vdash s \sqsubseteq_\alpha t$.

Similarly, given an erasable context Γ *we say that a term* t *is erasable if both* $\Gamma \vdash t \sqsubseteq_\alpha \varepsilon(t)$ *and* $\Gamma \vdash \varepsilon(t) \sqsubseteq_\alpha t$.

Now we can state and prove both halves of the conservativity theorems, in an open context and for the three different judgments.

Theorem 25 (GCIC is weaker than CIC – Open context).

- *If* Γ *is an erasable context of* CastCIC *and* $\varepsilon(\Gamma) \vdash_{\mathrm{CIC}} t \triangleright T$ *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* Γ *is an erasable context of* CastCIC, T' *is an erasable term of* CastCIC, *and* $\varepsilon(\Gamma) \vdash_{\mathrm{CIC}} t \triangleleft \varepsilon(T')$ *then* $\Gamma \vdash t \triangleleft T' \rightsquigarrow t'$ *for some erasable* t' *containing no* ? *such that* $\varepsilon(t') = t$.
- *If* Γ *is an erasable context of* CastCIC *and* $\varepsilon(\Gamma) \vdash_{\mathrm{CIC}} t \blacktriangleright_{\mathrm{h}} T$ *then* $\Gamma \vdash t \rightsquigarrow t' \blacktriangleright_{\mathrm{h}} T'$ *for some erasable* t' *and* T' *containing no* ? *such that* $\varepsilon(t') = t$ *and* $\varepsilon(T') = T$.

Proof. The inference steps are direct: one needs to combine the induction hypothesis together, using the substitution property of precision and the fact that $\varepsilon(t[u/x]) = \varepsilon(t)[\varepsilon(u)/x]$ 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 $\varepsilon(\Gamma) \vdash_{\mathrm{CIC}} t \triangleright T$ and $T \rightsquigarrow^* \Pi x : A.B$. By induction hypothesis, $\Gamma \vdash t \rightsquigarrow t' \blacktriangleright_\Pi T'$ with t' and T' erasable, and $\varepsilon(t') = t$, $\varepsilon(T') = T$. Because T' is erasable, it is less precise than T. We cannot have $T' \rightsquigarrow^* ?_\square$ because T' does not contain any ? as it is erasable, thus there is some A' and B' such that $T' \rightsquigarrow^* \Pi x : A'.B'$ and $\Gamma \vdash \Pi x : A.B \sqsubseteq_\alpha \Pi x : A'.B'$ by Corollary 24. Using that lemma again, there is also some $\tilde{A}$ and $\tilde{B}$ such that $T \rightsquigarrow^* \Pi x : \tilde{A}.\tilde{B}$ and $\Gamma \vdash \Pi x : A'.B' \sqsubseteq_\alpha \Pi x : \tilde{A}.\tilde{B}$. Now because T is static, so are $\Pi x : A.B$ and $\Pi x : \tilde{A}.\tilde{B}$, and because of the comparisons with $\Pi x : A'.B'$ we must have $\Pi x : A.B = \Pi x : \tilde{A}.\tilde{B} = \varepsilon(\Pi x : A'.B')$. Therefore we have $\Gamma \vdash t \rightsquigarrow t' \blacktriangleright_\Pi \Pi x : A'.B'$ and both t' and $\Pi x : A'.B'$ are erasable, and moreover $\varepsilon(t') = t$ and $\varepsilon(\Pi x : A'.B') = \Pi x : A.B$.

The other cases of constrained inference being very similar, let us turn to checking. We are given Γ and T erasable, and suppose that $\varepsilon(\Gamma) \vdash_{\mathrm{CIC}} t \triangleright S$ such that $\varepsilon(S) \equiv \varepsilon(T)$. By induction hypothesis, $\Gamma \vdash t \rightsquigarrow t' \blacktriangleright_\Pi S'$ with t' and S' erasable, and $\varepsilon(t') = t$, $\varepsilon(S') = S$. But convertibility implies consistency, so $\varepsilon(S) \sim \varepsilon(T)$, and the monotony of consistency gives $S \sim T$. 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 $\Gamma \vdash \langle T' \Leftarrow S' \rangle\ t' \sqsubseteq_\alpha t$ and $\Gamma \vdash t \sqsubseteq_\alpha \langle S' \Leftarrow T' \rangle\ t'$. Using a diagonal rule for cast, this reduces to showing that $\Gamma \vdash T' \sqsubseteq_\rightsquigarrow S'$ and $\Gamma \vdash S' \sqsubseteq_\rightsquigarrow T'$. The situation is symmetric, let us show only the first inequation. Because S and T are convertible, let U be a common reduct. Using the simulation, $T' \rightsquigarrow^* T''$ with $\Gamma \vdash U \sqsubseteq_\alpha T''$. Simulating the reduction then gives $S \rightsquigarrow^* \tilde{U}$ and $\Gamma \vdash T'' \sqsubseteq_\alpha \tilde{U}$. Because U and $\tilde{U}$ are static, this implies $U = \tilde{U}$, and thus $\Gamma \vdash T'' \sqsubseteq_\alpha U$. Simulating the reduction to U also gives some S'' such that $\Gamma \vdash U \sqsubseteq_\alpha S''$. From this we can deduce $\Gamma \vdash T'' \sqsubseteq_\alpha S''$. □

Theorem 26 (CIC is weaker than GCIC – Open context).

- *If* Γ' *is an erasable context of* CastCIC *and* t *is a static term such that* $\Gamma' \vdash t \rightsquigarrow t' \triangleright T'$, *then* t' *and* T' *are erasable and contain no* ?, $\varepsilon(t') = t$ *and* $\varepsilon(\Gamma') \vdash \varepsilon(t') \triangleright \varepsilon(T')$.
- *If* Γ' *is an erasable context of* CastCIC, T' *is an erasable term of* CastCIC *containing no* ?, *and* t *is a static term such that* $\Gamma' \vdash t \triangleleft T' \rightsquigarrow t'$, *then* t' *is erasable,* $\varepsilon(t') = t$ *and* $\varepsilon(\Gamma') \vdash \varepsilon(t') \triangleleft \varepsilon(T')$.
- *If* Γ' *is an erasable context of* CastCIC *and* t *is a static term such that* $\Gamma' \vdash t \rightsquigarrow t' \blacktriangleright_{\mathrm{h}} T'$, *then* t' *and* T' *are erasable and contain no* ?, $\varepsilon(t') = t$ *and* $\varepsilon(\Gamma') \vdash \varepsilon(t') \blacktriangleright_{\mathrm{h}} \varepsilon(T')$.

Proof. The proof is similar to the previous one, using the simulations again to handle reduction steps. □

As a direct corollary of those propositions, we get conservativity Theorem 11.

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

Theorem 27 (Elaboration graduality – Open context). *In all the properties below, we are given a context* $\mathbb{\Gamma}$ *such that* $\sqsubseteq_\alpha \mathbb{\Gamma}$ *and two term* t *and* $\tilde{t}$ *such that* $t \sqsubseteq^{\mathrm{G}}_\alpha \tilde{t}$.

- *If* $\Gamma_1 \vdash t \rightsquigarrow t' \triangleright T$ *and each subterm of* t *that is against a* $?^{@i}$ *in* $\tilde{t}$ *is in sort* i*, then there exists* $\tilde{t}'$ *and* $\tilde{T}$ *such that* $\tilde{\Gamma}_2 \vdash \tilde{t} \rightsquigarrow \tilde{t}' \triangleright \tilde{T}$, $\Gamma \vdash t' \sqsubseteq_\alpha \tilde{t}'$ *and* $\Gamma \vdash T \sqsubseteq_\alpha \tilde{T}$.
- *If* $\Gamma_1 \vdash t \triangleleft T \rightsquigarrow t'$ *and each subterm of* t *that is against a* $?^{@i}$ *in* $\tilde{t}$ *is in sort* i*, then for all* $\tilde{T}$ *such that* $\Gamma \vdash T \sqsubseteq_\alpha \tilde{T}$*, there exists* $\tilde{t}'$ *such that* $\tilde{\Gamma}_2 \vdash \tilde{t} \triangleleft \tilde{T} \rightsquigarrow \tilde{t}'$ *and* $\Gamma \vdash t' \sqsubseteq_\alpha \tilde{t}'$.
- *If* $\Gamma_1 \vdash t \rightsquigarrow t' \blacktriangleright_{\mathrm{h}} T$ *and each subterm of* t *that is against a* $?^{@i}$ *in* $\tilde{t}$ *is in sort* i*, then there exists* $\tilde{t}'$ *and* $\tilde{T}$ *such that* $\tilde{\Gamma}_2 \vdash \tilde{t} \rightsquigarrow \tilde{t}' \blacktriangleright_{\mathrm{h}} \tilde{T}$, $\Gamma \vdash t' \sqsubseteq_\alpha \tilde{t}'$ *and* $\Gamma \vdash T \sqsubseteq_\alpha \tilde{T}$.

Proof. Those are proven by mutual induction on the typing derivation of t, and case analysis on the precision relation between t and $\tilde{t}$.

For inference, there are two kinds of cases: the ones where the rule used for precision is the diagonal one (*i.e.* the one where t and $\tilde{t}$ have the same head), and the non-diagonal one, where the right term is some $?$. We treat some of the cases of the first kind first (the rest is similar), and treat the non-diagonal last.

Inference, diagonal variable case

The inference rule for a variable gives us $(x : T) \in \Gamma_1$. Because $\vdash \Gamma \sqsubseteq_\alpha \tilde{\Gamma}$, there exists some $\tilde{T}$ such that $(x : \tilde{T}) \in \tilde{\Gamma}$, and $\Gamma \vdash T \sqsubseteq_\alpha \tilde{T}$ using weakening. Thus, $\tilde{\Gamma} \vdash x \rightsquigarrow x \triangleright \tilde{T}$ and $\Gamma \vdash T \sqsubseteq_\alpha \tilde{T}$, and $\Gamma \vdash x \sqsubseteq_\alpha x$.

Inference, diagonal universe case

By reflexivity.

Inference, diagonal product case

The type inference rule for product gives $\Gamma_1 \vdash A \rightsquigarrow A' \blacktriangleright_\square \square_i$ and $\Gamma_1, x : A' \vdash B \rightsquigarrow B' \blacktriangleright_\square \square_j$, and the diagonal precision one gives $A \sqsubseteq^{\mathrm{G}}_\alpha \tilde{A}$ and $t \sqsubseteq^{\mathrm{G}}_\alpha \tilde{t}$. Applying the induction hypothesis, we get some $\tilde{A}'$ such that $\tilde{\Gamma}_2 \vdash \tilde{A} \rightsquigarrow \tilde{A}' \blacktriangleright_\square \square_i$ and $\Gamma \vdash A' \sqsubseteq_\alpha \tilde{A}'$. As for the type, it must be $\square_i$ because it is as the same time less precise than $\square_i$, and a sort (as it is the type of a constrained inference). From this, we also deduce that $\sqsubseteq_\alpha \Gamma, x : A' \mid \tilde{A}'$ and thus the induction hypothesis can be applied to B, giving that $\tilde{\Gamma}_2 \vdash \tilde{B} \rightsquigarrow \tilde{B}' \blacktriangleright_\square \square_j$. Therefore, combining those two statements we obtain $\tilde{\Gamma}_2 \vdash \Pi x : \tilde{A}.\tilde{B} \rightsquigarrow \Pi_{i,j}\, x : \tilde{A}'.\tilde{B}' \triangleright \square_{s_\Pi(i,j)}$. Moreover, $\Gamma \vdash \Pi x : A'.B' \sqsubseteq_\alpha \Pi x : \tilde{A}'.\tilde{B}'$ by combining the precision hypothesis on A' and B', and the equality of the indexes, and of course $\Gamma \vdash \square_{s_\Pi(i,j)} \sqsubseteq_\alpha \square_{s_\Pi(i,j)}$.

Inference, diagonal inductor case

In the case of the inductor, to be able to prove the precision, there are some extra assumptions. The ones on typability are obtained by the correctness of elaboration. More interesting is the last one. We pose $\mathbb{\Delta} := \Gamma, f : (\Pi z : I(\mathbf{a}), P) \mid (\Pi z : I(\tilde{\mathbf{a}})), \mathbf{y} : \mathbf{Y_k}[\mathbf{a}/\mathbf{x}] \mid \mathbf{Y_k}[\tilde{\mathbf{a}}/\mathbf{x}]$ the double context corresponding to branch k. We have by correctness of elaboration $\mathbb{\Delta}_1 \vdash t'_k \blacktriangleright_{\mathrm{P}} P'[c_k(\mathbf{a}, \mathbf{y})/z]$, and thus $\mathbb{\Delta}_1 \vdash t'_k \triangleright T$ for some T such that $T \equiv P'[c_k(\mathbf{a}, \mathbf{y})/z]$. Thus, T and $P'[c_k(\mathbf{a}, \mathbf{y})/z]$ have a common reduct U. Moreover, $\mathbb{\Delta} \vdash P'[c_k(\mathbf{a}, \mathbf{y})/z] \sqsubseteq_\alpha \tilde{P}'[?_{I(\tilde{\mathbf{a}})}/z]$ by the substitution property of precision. Thus, by simulation, there must exist some $\tilde{U}$ such that $\tilde{P}'[?_{I(\tilde{\mathbf{a}})}/z] \rightsquigarrow \tilde{U}$ and $\mathbb{\Delta} \vdash U \sqsubseteq_\alpha \tilde{U}$. Therefore, $\mathbb{\Delta} \vdash T_k \sqsubseteq_{\rightsquigarrow} \tilde{P}'[?_{I(\tilde{\mathbf{a}})}/z]$. From this, we get the expected conclusion that $\Gamma \vdash \mathtt{ind}_I(s, P, \mathbf{t}) \sqsubseteq_\alpha \mathtt{ind}_I(\tilde{s}, \tilde{P}, \tilde{\mathbf{t}})$, and by substitution property of the precision, also $\Gamma \vdash P[s/z] \sqsubseteq_\alpha \tilde{P}[\tilde{s}/z]$.

Inference, other diagonal cases

They are similar to the previous ones, basically combining together the induction hypothesis together to conclude.

Inference, non-diagonal case

We have $\Gamma_1 \vdash t \rightsquigarrow t' \triangleright T'$ at level i, and $\tilde{\Gamma}_2 \vdash ?^{@i} \rightsquigarrow ?_{?_{\square_i}} \triangleright ?_{\square_i}$. By correctness of elaboration, we have $\Gamma_1 \vdash t' \triangleright T'$, and by validity $\Gamma_1 \vdash T' \triangleright \square_i$. Thus we have $\Gamma \vdash T' \sqsubseteq_\alpha ?_{\square_i}$. Moreover this implies that $\Gamma \vdash t' \sqsubseteq_\alpha ?_{?_{\square_i}}$, and so we get the conclusions intended.

$$
\begin{array}{llll}
[\![A]\!] & := & \mathrm{El}\,[A] & [\Pi\, x : A.B] := \pi\,[A]\,(\lambda\, x : [\![A]\!].[B]) \\
[x] & := & x & [\mathbb{N}] := \mathfrak{nat} \\
[\square_i] & := & \mathfrak{u}_i & [?_A] := ?_{[A]} \\
[t\ u] & := & [t]\ [u] & [\mathrm{err}_A] := \mathrm{err}_{[A]} \\
[\lambda\, x : A.t] & := & \lambda\, x : [\![A]\!].[t] & [\langle B \Leftarrow A\rangle\ t] := \mathtt{cast}\,[A]\,[B]\,[t]
\end{array}
$$

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

Checking

For checking, we have the following hypothesis by typing: $[\![\Gamma]\!]_1 \vdash t \rightsquigarrow t' \triangleright S'$ and $S' \sim T'$. By induction hypothesis, $[\![\Gamma]\!]_2 \vdash \tilde{t} \rightsquigarrow \tilde{t}' \triangleright \tilde{S}'$ with $[\![\Gamma]\!] \vdash t' \sqsubseteq_\alpha \tilde{t}'$ and $[\![\Gamma]\!] \vdash T' \sqsubseteq_\alpha \tilde{T}'$. Using the extra hypothesis that $[\![\Gamma]\!] \vdash S' \sqsubseteq_\alpha \tilde{S}'$ and the monotony of consistency, we conclude that $\tilde{S}' \sim \tilde{T}'$, and thus $[\![\Gamma]\!]_2 \vdash \tilde{t} \triangleleft \tilde{T}' \rightsquigarrow \left\langle \tilde{T}' \Leftarrow \tilde{S}' \right\rangle \tilde{t}'$. A use of the diagonal rule for cast then ensures that $[\![\Gamma]\!] \vdash \langle T' \Leftarrow S' \rangle\ t' \sqsubseteq_\alpha \left\langle \tilde{T}' \Leftarrow \tilde{S}' \right\rangle \tilde{t}'$.

Π **constrained inference,** Π **rule**

By typing, we have the following hypothesis: $[\![\Gamma]\!]_1 \vdash t \rightsquigarrow t' \triangleright S'$ and $S' \rightsquigarrow^* \Pi x : A'.B'$. By induction hypothesis, $[\![\Gamma]\!]_2 \vdash \tilde{t} \rightsquigarrow \tilde{t}' \triangleright \tilde{S}'$ with $[\![\Gamma]\!] \vdash S' \sqsubseteq_\alpha \tilde{S}'$. By the previous section, we get that $\tilde{S}' \rightsquigarrow^* \Pi x : \tilde{A}'.\tilde{B}'$ such that $[\![\Gamma]\!] \vdash \Pi x : A'.B' \sqsubseteq_\alpha \Pi x : \tilde{A}'.\tilde{B}'$, or $\tilde{S}' \rightsquigarrow^* ?_{\square_i}$. In the first case, we get directly $[\![\Gamma]\!]_2 \vdash \tilde{t} \rightsquigarrow \tilde{t}' \blacktriangleright_\Pi \Pi x : \tilde{A}'.\tilde{B}'$ together with the precision inequalities with t' and $\Pi x : A'.B'$. In the second case, we get (using the other rule for constrained elaboration) $[\![\Gamma]\!]_2 \vdash \tilde{t} \rightsquigarrow \left\langle ?_{\square_{c_\Pi(i)}} \to ?_{\square_{c_\Pi(i)}} \Leftarrow \tilde{S}' \right\rangle \tilde{t}' \blacktriangleright_\Pi ?_{\square_{c_\Pi(i)}} \to ?_{\square_{c_\Pi(i)}}$, and $c_\Pi(i)$ is larger than the universe levels of both A' and B'. A use of the non-diagonal rule for precision, together with the fact that $[\![\Gamma]\!] \vdash A' \sqsubseteq_\alpha ?_{\square_{c_\Pi(i)}}$ and $[\![\Gamma]\!], x : A' \mid ?_{\square_{c_\Pi(i)}} \vdash B' \sqsubseteq_\alpha ?_{\square_{c_\Pi(i)}}$, gives that $[\![\Gamma]\!] \vdash t' \sqsubseteq_\alpha \left\langle ?_{\square_{c_\Pi(i)}} \to ?_{\square_{c_\Pi(i)}} \Leftarrow \tilde{S}' \right\rangle \tilde{t}'$.

Π **constrained inference,** $?$ **rule**

By typing, we have the following hypothesis: $[\![\Gamma]\!]_1 \vdash t \rightsquigarrow t' \triangleright S'$ and $S' \rightsquigarrow^* ?_{\square_i}$. By induction hypothesis, $[\![\Gamma]\!]_2 \vdash \tilde{t} \rightsquigarrow \tilde{t}' \triangleright \tilde{S}'$ with $[\![\Gamma]\!] \vdash S' \sqsubseteq_\alpha \tilde{S}'$. By the previous section, we get that $\tilde{S}' \rightsquigarrow^* ?_{\square_i}$. Thus $[\![\Gamma]\!]_2 \vdash \tilde{t} \rightsquigarrow \left\langle ?_{\square_{c_\Pi(i)}} \to ?_{\square_{c_\Pi(i)}} \Leftarrow \tilde{S}' \right\rangle \tilde{t}' \blacktriangleright_\Pi ?_{\square_{c_\Pi(i)}} \to ?_{\square_{c_\Pi(i)}}$. A use of the diagonal rule for precision is enough to conclude.

Other constrained inference

Similar to one of the two previous cases, the handling of the universe levels being actually simpler. □

B COMPLEMENT TO MODELS OF CastCIC

This section provides material supplementing §6. Appendix B.1 provides a correspondence between the notions developed in this paper and the formal development in Agda provided in the artefact. An alternative paper version of some of the proof around the monotone universe hierarchy not covered by the formalization are given in Appendix B.2 and an extended sketch of the alluded logical relation between the discrete and monotone models can be found in Appendix B.3. Figure 15 completes the missing cases in the presentation of the translation for the discrete model.

B.1 Mapping to the Agda files

The agda formalization covers most component of the discrete (`DiscreteModelPartial.agda`) and monotone model (`UnivPartial.agda`) in the 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 $\overline{?}$ 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`).

B.2 Properties of the monotone universe hierarchy

Since the agda code does not cover the total monotone model, we give a short paper proof of the main part of Theorem 15.

Lemma 28. *$\sqsubseteq$ is an order on $\Sigma_{j\leq i}\mathbb{U}_j$, with bottom element $\maltese^0$ and top element $?^i$.*

Proof. Reflexivity proceed by induction on the code and is immediate for all codes but $\pi\, A\, B$ for which it holds by induction hypothesis on A and by monotony of B.

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 $\mathbb{L}$.

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 apply the inductive hypothesis to $A \sqsubseteq \mathrm{El}_{\mathbb{H}}^{\mathrm{pred}\ j}(\mathrm{head}\, A)$ and $\mathrm{El}_{\mathbb{H}}^{\mathrm{pred}\ j}(\mathrm{head}\, A) \sqsubseteq \mathrm{El}_{\mathbb{H}}^{\mathrm{pred}\ j'}(\mathrm{head}\, A)$ 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

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

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

By induction hypothesis, $A^{\mathrm{d}} \sqsubseteq C^{\mathrm{d}}$ and 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 $A^{\mathrm{c}}\, a \sqsubseteq C^{\mathrm{c}}\, c$. This follows from the induction hypothesis applied to $A^{\mathrm{c}}\, a \sqsubseteq B^{\mathrm{c}}\, (\uparrow_{A^{\mathrm{d}}\sqsubseteq B^{\mathrm{d}}}\, a)$ and $B^{\mathrm{c}}\, (\uparrow_{A^{\mathrm{d}}\sqsubseteq B^{\mathrm{d}}}\, a) \sqsubseteq C^{\mathrm{c}}\, c$ where we use that $a\ {}_{A^{\mathrm{d}}}\sqsubseteq_{B^{\mathrm{d}}}\ (\uparrow_{A^{\mathrm{d}}\sqsubseteq B^{\mathrm{d}}}\, a)$ and $(\uparrow_{A^{\mathrm{d}}\sqsubseteq B^{\mathrm{d}}}\, a)\ {}_{B^{\mathrm{d}}}\sqsubseteq_{C^{\mathrm{d}}}\ c$ thanks to the adjoint properties of $\uparrow_{A^{\mathrm{d}}\sqsubseteq B^{\mathrm{d}}}$, reflexivity of ${}_{A^{\mathrm{d}}}\sqsubseteq_{A^{\mathrm{d}}}$ and $a\ {}_{A^{\mathrm{d}}}\sqsubseteq_{C^{\mathrm{d}}}\ c$.
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 $B \sqsubseteq \mathrm{El}_{\mathbb{H}}^{\mathrm{pred}\ j}(\mathrm{head}\, B)$ 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 is 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. □

In order to prove Lemma 18, we use the following lemma.

Lemma 29 (Compatibility of definitional and propositional precision). *If* $\Gamma \vdash t \sqsubseteq_\alpha u$, $\Gamma_1 \vdash_{\mathsf{cast}} t : T$, $\Gamma_2 \vdash_{\mathsf{cast}} u : U$ *then there exists a* CIC *term* e *such that* $\{\!|\ \Gamma\ |\!\} \vdash e : \{t\}\ {}_{\{T\}}\sqsubseteq_{\{U\}}\ \{u\}$.

Proof. We prove the lemma by induction on the derivation of syntactic precision. The variable and □ cases hold by reflexivity. The two cases involving `?` and `err` amount to $\{?\}$ and $\{\mathtt{err}\}$ being respectively interpreted as top and bottom elements. The cases $t \sqsubseteq_\alpha \langle B' \Leftarrow A' \rangle\ t'$ and $\langle B \Leftarrow A \rangle\ t \sqsubseteq_\alpha t'$ can be reduce to the diagonal case of cast because $\langle T \Leftarrow T \rangle\ t = t$ propositionally. All the other cases, being congruence rules with respect to some term constructor, are consequences of the monotonicity of said constructor with a direct application of the inductive hypothesis and inversion of the typing judgments. □

B.3 Sketch of the logical relation between the discrete and monotone model

We recall that $\{\cdot\}$ denote the monotone translation (with monotonicity proof given by $\{t : A\}_\varepsilon : \{\!|\ A\ |\!\}_\varepsilon\ \{t\}\ \{t\}$). We define a (binary) logical relation between the discrete and monotone translations that corresponds to forgetting the monotonicity information on base cases. 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 $\wr t \int : \wr A \int\ [t]\ \{t\}$. Context are translated standardly as $\wr \cdot \int = \cdot$ and $\wr \Gamma, x : A \int = \wr \Gamma \int, x : [\![A]\!], x' : \{\!|\ A\ |\!\}, x_\varepsilon : \wr A \int\ x\ x'$.

Logical relation on terms and types

$$
\begin{array}{lcl}
\wr A \int & := & \mathrm{El}_\varepsilon\ \wr A \int \\
\wr x \int & := & x_\epsilon \\
\wr \square_i \int & := & \mathfrak{u}_{\varepsilon,i} \\
\wr t\ u \int & := & \wr t \int\ [u]\ \{u\}\ \wr u \int \\
\wr \lambda x : A.t \int & := & \lambda(x : [\![A]\!])(x' : \{\!|\ A\ |\!\})(x_\varepsilon : \wr A \int\ x\ x').\ \wr t \int \\
\wr \Pi x : A.B \int & := & \pi_\varepsilon\ \wr A \int\ (\lambda(x : [\![A]\!])(x' : \{\!|\ A\ |\!\})(x_\varepsilon : \wr A \int\ x\ x').\wr B \int) \\
\wr \mathbb{N} \int & := & \mathfrak{nat}_\varepsilon \\
\wr \mathtt{raise} \int & := & \lambda(b\ b' : \mathbb{B})(\_ : b = b')(A\ A' : \mathbb{U}_i)(A_\varepsilon : \mathbb{U}_\varepsilon\ A\ A').\mathrm{rec}_{\mathbb{B}}\ A_\varepsilon\ ?_{\varepsilon,A_\varepsilon}\ \maltese_{\varepsilon,A_\varepsilon} \\
\wr \mathtt{cast} \int & := & \mathtt{cast}_\varepsilon
\end{array}
$$

Inductive-recursive relational universe $\mathbb{U}_\varepsilon : \mathbb{U}^{\mathbf{dis}} \to \mathbb{U}^{\mathbf{mon}} \to \square$ and decoding function $\mathrm{El}_\varepsilon : \mathbb{U}_\varepsilon\ A\ A' \to \mathrm{El}\ A \to \mathrm{El}\ A' \to \square$

$$
\frac{A_\varepsilon \in \mathbb{U}_{\varepsilon,i}\ A\ A' \qquad B \in \Pi(a : A)(a' : A').\mathrm{El}_\varepsilon\ A_\varepsilon\ a\ a' \to \mathbb{U}_{\varepsilon,j}\ (B\ a)\ (B'\ a')}{\pi_\varepsilon\ A_\varepsilon\ B_\varepsilon \in \mathbb{U}_{\varepsilon,s_\Pi(i,j)}\ (\pi\ A\ B)\ (\pi\ A'\ B')} \qquad \frac{j < i}{\mathfrak{u}_{\varepsilon,j} \in \mathbb{U}_{\varepsilon,i}\ \mathfrak{u}_j\ \mathfrak{u}_j}
$$

$$
\mathfrak{nat}_\varepsilon \in \mathbb{U}_{\varepsilon,i}\ \mathfrak{nat}\ \mathfrak{nat} \qquad ?_\varepsilon \in \mathbb{U}_{\varepsilon,i}\ ?\ ? \qquad \maltese_\varepsilon \in \mathbb{U}_{\varepsilon,i}\ \maltese\ \maltese
$$

$$
\begin{array}{rcl}
\mathrm{El}_\varepsilon\ (\pi_\varepsilon\ A_\varepsilon\ B_\varepsilon)\ f\ f' & := & \Pi(a : \mathrm{El}\ A)(a' : \mathrm{El}\ A')(a_\varepsilon : \mathrm{El}_\varepsilon\ A_\varepsilon\ a\ a'). \\
 & & \quad \mathrm{El}_\varepsilon\ (B_\varepsilon\ a\ a'\ a_\varepsilon)\ (f\ a)\ (f'.1\ a') \\
\mathrm{El}_\varepsilon\ \mathfrak{u}_{\varepsilon,j}\ A\ A' & := & \mathbb{U}_{\epsilon,j}\ A\ A' \\
\mathrm{El}_\varepsilon\ \mathfrak{nat}_\varepsilon\ n\ m & := & n = m \\
\mathrm{El}_\varepsilon\ ?_\varepsilon\ (c; x)\ y & := & \mathrm{El}_\varepsilon\ (\mathrm{Germ}_\varepsilon\ c)\ x\ (\mathtt{downcast}_{?,\mathrm{Germ}\ c}\ y) \\
\mathrm{El}_\varepsilon\ \maltese_\epsilon\ *\ * & := & \top
\end{array}
$$

The main difficulty of the relation lie in the relation between the `casts`

$$
\mathtt{cast}_\varepsilon : \wr\, \Pi(A\ B : \mathbb{U}).A \to B \int\ [\mathtt{cast}]\ \{\mathtt{cast}\}
$$

Expanding the type of $\mathsf{cast}_\varepsilon$, we need to provide a term

$$c_\varepsilon = \mathsf{cast}_\varepsilon\ A\,A'\,A_\varepsilon\,B\,B'\,B_\varepsilon\,a\,a'\,a_\varepsilon : \mathrm{El}_\varepsilon\,B_\varepsilon\ ([\mathsf{cast}]\,A\,B\,a)\ (\{\mathsf{cast}\}\,A'\,B'\,a')$$

where

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

We proceed by induction on $A_\varepsilon, B_\varepsilon$, following the defining cases for $[\mathsf{cast}]$.

Case $A_\varepsilon = \pi_\varepsilon\,A_\varepsilon^{\mathbf{d}}\,A_\varepsilon^{\mathbf{c}}$ **and** $B_\varepsilon = \pi_\varepsilon\,B_\varepsilon^{\mathbf{d}}\,B_\varepsilon^{\mathbf{c}}$**:** we compute with $A' = \pi\,A'^{\mathrm{d}}\,A'^{\mathrm{c}}$ and $B' = \pi\,B'^{\mathrm{d}}\,B'^{\mathrm{c}}$

$$\begin{aligned} \{\mathsf{cast}\}\,A'\,B'\,f' &= \Downarrow^?_{B'}\ (\Uparrow^?_{A'}\ f') \\ &= \Downarrow^{?\to ?}_{B'} \circ \Downarrow^?_{?\to ?} \circ \Uparrow^?_{?\to ?} \circ \Uparrow^{?\to ?}_{A'}\ (f) \\ &= \Downarrow^{?\to ?}_{B'} \circ \Uparrow^{?\to ?}_{A'}\ (f) \\ &= \lambda(b' : \mathrm{El}\,A'^{\mathrm{d}}).\ \mathrm{let}\ a' = \Downarrow^?_{B'^{\mathrm{d}}} \circ \Uparrow^?_{A'^{\mathrm{d}}}\ (b)\ \mathrm{in} \\ &\qquad \Downarrow^?_{B'^{\mathrm{c}}\,b'} \circ \Uparrow^?_{A'^{\mathrm{c}}\,a'}\ (f\,a') \\ &= \lambda(b' : \mathrm{El}\,A'^{\mathrm{d}}).\ \mathrm{let}\ a' = \{\mathsf{cast}\}\ B'^{\mathrm{d}}\,A'^{\mathrm{d}}\,b'\ \mathrm{in} \\ &\qquad \{\mathsf{cast}\}\ (A'^{\mathrm{c}}\,a')\ (B'^{\mathrm{c}}\,b')\ (f\,a') \end{aligned}$$

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

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

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

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

and by another application of the inductive hypothesis

$$\wr\,\mathsf{cast}\,\int\ (B_\varepsilon^{\mathrm{c}}\,b\,b'\,b_\varepsilon)\ (A_\varepsilon^{\mathrm{c}}\,a\,a'\,a_\varepsilon)\ (f_\varepsilon\,a\,a'\,a_\varepsilon) : \wr B_\varepsilon^{\mathrm{c}}\,b\,b'\,b_\varepsilon\int\ ([\mathsf{cast}]\,A\,B\,f\,a)\ (\{\mathsf{cast}\}\,A'\,B'\,f'\,a')$$

Packing these together, we built a term

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

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

$$\mathrm{El}_\varepsilon\ (?^{\mathrm{pred}\ i} \to ?^{\mathrm{pred}\ i})\ ([\mathsf{cast}]\,A\,(? \to ?)\,f)\ (\Downarrow^?_{?\to ?}\ (\{\mathsf{cast}\}\,A'\,?\,f'))$$

We compute that

$$\Downarrow^?_{?\to ?}\ (\{\mathsf{cast}\}\,A'\,?\,f') = \Downarrow^?_{?\to ?} \circ \Downarrow^?_? \circ \Uparrow^?_{A'}\ f' = \Downarrow^?_{?\to ?} \circ \Uparrow^?_{A'}\ f' = \{\mathsf{cast}\}\,A'\,(? \to ?)\,f'$$

So the result holds by induction hypothesis.

Other cases with $A_\varepsilon = \pi_\varepsilon\,A_\varepsilon^{\mathbf{d}}\,A_\varepsilon^{\mathbf{c}}$**:** It is enough to show that $\{\mathsf{cast}\}\,A'\,B'\,f' = \maltese_{B'}$ when $B' = \maltese$ (trivial) or head $B' \neq \mathtt{pi}$. The latter case holds because $\Downarrow^?_{\mathrm{Germ}\ c}\Uparrow^?_{\mathrm{Germ}\ c'}\ x = \maltese_{\mathrm{El}_{\Vdash}\ c}$ whenever $c \neq c'$ and downcasts preserve $\maltese$.

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

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