Bounded Cumulative Universe Polymorphism with Algebraic Universes

AGDA Implementors' Meeting XLI Angers, France Nov 24th 2025

Matthieu Sozeau, Inria & LS2N, University of Nantes, France

Marc Bezem, University of Bergen, Norway

ROCQ

Inria

Throwback to AIM XIX 2014

Program

(not etched in stone)

Thu 22nd	**Room 314**	
	10.00	Jesper Cockx, Eliminating dependent pattern matching without K
	10.45	Introduction of participants; code sprint proposals
	14.00	Andreas Abel, Introduction to Agda's internals and live bug fixing session
	18.00	Wrap-up meeting
Fri 23nd	**Room 421**	
	10.00	Nicolas Pouillard, Dependent Protocols for Communication
afternoon	**Amphiteater Perrin**	
	14.00	Ambrus Kaposi, A Syntax for Cubical Type Theory (slides at http://mazzo.li/dump/aim-kaposi-pres.pdf)
	18.00	Wrap-up meeting
Mon 26th	**Room 201**	
	10.00	Sergei Sinchuk, Coq-verified operational transformation for trees
	11.00	Christoph Senjak, Verfying Deflate Trees (Unzip)
	18.00	Wrap-up meeting
Tue 27th	**Room 201**	
	10.00	Matthieu Sozeau, Universe polymorphism in Coq

My links to Agda

- Homotopy Type Theory / Opetopic Type Theory

  [1] Antoine Allioux, Eric Finster, and Matthieu Sozeau. 2021. Types are Internal ∞-groupoids. *LICS* (January 2021). https://arxiv.org/abs/2105.00024

  [2] Antoine Allioux. 2023. Higher Structures in Homotopy Type Theory. https://inria.hal.science/tel-04335842v1

- Verified erasure / extraction: $\lambda_{\square}$

  [10] Matthieu Sozeau, Yannick Forster, Meven Lennon-Bertrand, Jakob Nielsen, Nicolas Tabareau, and Théo Winterhalter. 2025. Correct and Complete Type Checking and Certified Erasure for Coq, in Coq. *J. ACM* 72, 1 (January 2025). DOI: 10.1145/3706056

Outline

1. Universes in Type Theory: Typing and Elaboration
2. Mechanized Universe Checking for Algebraic Universes and Cumulativity
3. Algebraic Universes and Variances in Rocq

Type Theory with Universes

$\mathcal{U} : \mathcal{U}$ is inconsistent (System U/U$^-$, Girard's paradox, Hurkens)

Use a hierarchy $\mathcal{U}_i (i \in \mathbb{N})$ to stratify universes.

$$\vdash \mathcal{U}_i : \mathcal{U}_{i+1}$$

Type Theory with Universes

$\mathcal{U} : \mathcal{U}$ is inconsistent (System U/U$^-$, Girard's paradox, Hurkens)

Use a hierarchy $\mathcal{U}_i (i \in \mathbb{N})$ to stratify universes.

$$\vdash \mathcal{U}_i : \mathcal{U}_{i+1}$$

$$\frac{\vdash A : \mathcal{U}_i \qquad x : A \vdash B : \mathcal{U}_j}{\vdash \Pi x : A.B : \mathcal{U}_{i \sqcup j}}$$

Usually with a bottom universe $\mathcal{U}_0$ (≡ (predicative) `Set` in Rocq)

Type Theory with Polymorphic Universes

Universe polymorphism = universal quantification on universe variables

$$
\begin{aligned}
u, v ::= {} & l && \text{(universe level variable)} \\
\mid {} & 0 && \text{(bottom universe)} \\
\mid {} & u + 1 && \text{(successor universe)} \\
\mid {} & \max(u, v) && \text{(supremum, } \sqcup \text{)}
\end{aligned}
$$

0 is neutral for max, successor distributes on max, max is idempotent, associative, commutative and subsumptive: $\max(u, u + 1) = u + 1$

Unification with $(\max, +)$

During elaboration:

$$u, v ::= \ldots \mid ?l \text{ (level metavariables)}$$

Subject to unification constraints.

The theory is **not unitary**: no m.g.u.'s in general for the $(\max, +)$ algebra.

Thiago Felicissimo, PhD, "Generic Bidirectional Typing in a Logical Framework for Dependent Type Theories". (Theorem 17.1)

Example $?x + 1 = ?y \sqcup ?z$ solvable but has no m.g.u.

Stuck constraints in practice (Agda, Lean), forcing explicit annotations.

Cumulativity + Polymorphism

Cumulative hierarchy $\mathcal{U}_i (i \in \mathbb{N})$, definitional equality becomes a subtyping relation:

$$\vdash \mathcal{U}_i \leq \mathcal{U}_{i+1} \qquad \vdash \Pi x : A.\mathcal{U}_i \leq \Pi x : A.\mathcal{U}_j \quad \text{iff } i \leq j$$

With polymorphism $\Rightarrow$ **bounded** quantification on universe variables

(Harper and Pollack, TCS'91, Sozeau and Tabareau, ITP'14)

```
Definition le@{i j | i ≤ j} (A : Type@{i}) : Type@{j} := A.
```

Formally:

$$\frac{\Sigma \uplus \mathcal{L}, \Phi \Vdash \qquad \Sigma \uplus \mathcal{L}, \Phi; \varepsilon \vdash t : T}{\Sigma, \texttt{Definition}\ x@\{\mathcal{L} \mid \Phi\} : T := t \vdash_{\text{wf}}}$$

Consistency and satisfiability checking

$$(\text{declaration}) \quad d ::= \texttt{Inductive}\ i := b \mid \texttt{Definition}\ c : T := t$$

$$\Sigma ::= \left\{ \mathcal{L}_g : \{\text{level}\}; \Phi_g : \{\text{constraint}\}; \text{decls} : [d] \right\}$$

$$\frac{\mathcal{L} \cap \Sigma.\mathcal{L}_g = \emptyset \qquad \text{levels}(\Phi) \subset \mathcal{L} \cup \Sigma.\mathcal{L}_g \qquad \text{consistent}\big(\Sigma.\Phi_g \cup \Phi\big)}{\Sigma \uplus \mathcal{L}, \Phi \Vdash} \text{UCtx-Con}$$

Consistency checking on a set of constraints:

$$u_0 \leq v_0 \wedge \ldots \wedge u_i \leq v_i \vdash u \leq v$$

$$u \equiv v \leftrightarrow u \leq v \wedge v \leq u$$

Herbelin's Algebraic Universes

Previous type inference algorithm of Rocq restricted to:

$$u_0 \leq \boldsymbol{l_0} \wedge \ldots \wedge u_i \leq \boldsymbol{l_i} \vdash u \leq \boldsymbol{l}$$

$\max(i+1, j)$ expressions could only appear in inferred types of terms / "on the left" of $\Gamma \vdash T \leq U$, as $l \leq \max(j, k)$ could not be handled.

Herbelin's Algebraic Universes

Previous type inference algorithm of Rocq restricted to:

$$u_0 \leq \boldsymbol{l_0} \wedge \ldots \wedge u_i \leq \boldsymbol{l_i} \vdash u \leq \boldsymbol{l}$$

$\max(i+1, j)$ expressions could only appear in inferred types of terms / "on the left" of $\Gamma \vdash T \leq U$, as $l \leq \max(j, k)$ could not be handled.

In general, algebraic expressions could **not** appear in terms themselves $\Rightarrow$ need a "freshening" operation.

Herbelin's Algebraic Universes

Example Elaboration of `@id ?X Type@{0}`

We have `Type@{0} : Type@{0+1}`, but to solve `Type@{0+1} ≤ ?X`:

`?X := Type@{0+1}` forbidden as `0 + 1` is an algebraic expression.

`?X := Type@{?i}, 1 ⩽ ?i` for a fresh `?i` is ok as `?i` is a level variable.

Herbelin's Algebraic Universes

Example Elaboration of `@id ?X Type@{0}`

We have `Type@{0} : Type@{0+1}`, but to solve `Type@{0+1} ≤ ?X`:

`?X := Type@{0+1}` forbidden as `0 + 1` is an algebraic expression.

`?X := Type@{?i}, 1 ⩽ ?i` for a fresh `?i` is ok as `?i` is a level variable.

✅ Pro: efficient satisfiability and validity checking is possible.

Implemented by J.H. Jourdan based on a state-of-the-art graph algorithm [3]

❌ Cons: restricted algebra, error-prone API, proliferation of fresh levels and constraints, esp. to avoid `x < max(u, v)` constraints.

Mechanized Universe Checking for Algebraic Universes and Cumulativity

Bezem and Coquand's algorithm

"Loop-checking and the uniform word problem for join-semilattices with an inflationary endomorphism", Bezem and Coquand, TCS, 2022 [4]

Satisfiability is decidable in **weakly polynomial** time
for the $(\max, +, \leq)$ algebra.
Actually, for **any** semilattice with an inflationary
(and not necessarily injective) endomorphism.

Bezem and Coquand's algorithm

"Loop-checking and the uniform word problem for join-semilattices with an inflationary endomorphism", Bezem and Coquand, TCS, 2022 [4]

Satisfiability is decidable in **weakly polynomial** time
for the $(\max, +, \leq)$ algebra.
Actually, for **any** semilattice with an inflationary
(and not necessarily injective) endomorphism.

Subtly enough, **no strongly polynomial** time algorithm is known for this problem (after 30+ years of research). The so called Max-Atom problem, restricting to variables in $\mathbb{Z}$ is in NP $\cap$ co-NP: its complement is in NP [5].

Semilattice definition

```
Class Semilattice (carrier : Type) (incr : Type) `{CM : IsCommMonoid incr} :=
{ eq : carrier → carrier → Prop where "x ≡ y" := (eq x y) : sl_scope;
  eq_equiv :: Equivalence eq;
  add : incr → carrier → carrier where "x + k" := (add k x) : sl_scope;
  join : carrier → carrier → carrier where "x ⊔ y" := (join x y) : sl_scope;
  add_neutral x      : x + 0 ≡ x;
  add_distr n m x    : (x + m) + n ≡ x + (CommutativeMonoid.add m n);
  add_congr n x y    : x ≡ y → x + n ≡ y + n;
  add_inj n x y      : x + n ≡ y + n → x ≡ y; (* optional *)
  join_assoc x y z   : (x ⊔ y) ⊔ z ≡ x ⊔ (y ⊔ z);
  join_comm x y      : x ⊔ y ≡ y ⊔ x;
  join_congr x x' y  : x ≡ x' → x ⊔ y ≡ x' ⊔ y;
  join_idem x        : x ⊔ x ≡ x;
  join_sub x         : x ⊔ (x + 1) ≡ x + 1;
  add_join n x y     : (x ⊔ y) + n ≡ (x + n) ⊔ (y + n) }.
```

Derived theory

⚠️ No 0, and no no-confusion axiom, the semilattices on zero points and on one point are models.

```
(* Order relation induced by the equational theory *)
Definition le {SL : Semilattice carrier incr} x y := x ⊔ y ≡ y.
Notation "x ≤ y" := (le x y) : sl_scope.

Instance semi_po : PartialOrder eq le.
Proof. ... Qed.

(* This allows [rewrite] to work on this theory.  *)
Instance semi_join_mor : Proper (eq ==> eq ==> eq) join.
Proof. ... Qed.
```

Equivalence to Horn clauses

Going back to Lorentzen ('51): "Algebraische und logistische Untersuchungen über freie Verbände" (translated [7]). The free semilattice theory is equivalent to a deductive entailment system:

$$x_1 \wedge .. \wedge x_n \leq y \equiv x_1, .., x_n \vdash y \text{ (for meet semilattices)}$$

We rather consider join-semilattices and hypothetical judgments, based on clauses:

$$\text{(atom, conclusion) } a ::= l + k \quad \text{(level + increment in } \mathbb{Z}\text{)}$$

$$\text{(body, premises) } p ::= a \mid a, p \qquad \text{(non empty sets)}$$

$$\text{(clause) } c ::= p \to a \qquad \text{(clauses) } \mathcal{C} ::= \{c\}$$

For a historical perspective see [6] by Thierry Coquand, Henri Lombardi, and Stefan Neuwirth.

Equivalence to Horn clauses

Deduction system:

$$\frac{a \in p}{\mathcal{C} \vdash_{\mathcal{H}} p \to a}\ \text{Axiom} \qquad \frac{\mathcal{C} \vdash_{\text{pred}} p' \to a \quad p' \subset p \quad \mathcal{C} \vdash_{\mathcal{H}} a, p \to c}{\mathcal{C} \vdash_{\mathcal{H}} p \to c}\ \text{Forward}$$

Equivalence to Horn clauses

Deduction system:

$$\frac{a \in p}{\mathcal{C} \vdash_{\mathcal{H}} p \to a}\ \text{Axiom} \qquad \frac{\mathcal{C} \vdash_{\text{pred}} p' \to a \quad p' \subset p \quad \mathcal{C} \vdash_{\mathcal{H}} a, p \to c}{\mathcal{C} \vdash_{\mathcal{H}} p \to c}\ \text{Forward}$$

$$\frac{(p \to c) \in \mathcal{C} \quad z \in \mathbb{Z}}{\mathcal{C} \vdash_{\text{pred}} p + z \to c + z}\ \text{In-Shift} \qquad \frac{z \in \mathbb{Z}}{\mathcal{C} \vdash_{\text{pred}} l + (z + 1) \to l + z}\ \text{Down}$$

Equivalence to Horn clauses

Deduction system:

$$\frac{a \in p}{\mathcal{C} \vdash_{\mathcal{H}} p \to a}\ \text{Axiom} \qquad \frac{\mathcal{C} \vdash_{\text{pred}} p' \to a \quad p' \subset p \quad \mathcal{C} \vdash_{\mathcal{H}} a, p \to c}{\mathcal{C} \vdash_{\mathcal{H}} p \to c}\ \text{Forward}$$

$$\frac{(p \to c) \in \mathcal{C} \quad z \in \mathbb{Z}}{\mathcal{C} \vdash_{\text{pred}} p + z \to c + z}\ \text{In-Shift} \qquad \frac{z \in \mathbb{Z}}{\mathcal{C} \vdash_{\text{pred}} l + (z+1) \to l + z}\ \text{Down}$$

Define $\mathcal{C} \vdash_{\mathcal{H}} \mathcal{C}' := \forall c \in \mathcal{C}', \mathcal{C} \vdash_{\mathcal{H}} c$ and $\mathcal{C} \dashv\vdash_{\mathcal{H}} \mathcal{C}' := \mathcal{C} \vdash_{\mathcal{H}} \mathcal{C}' \wedge \mathcal{C}' \vdash_{\mathcal{H}} \mathcal{C}$.

$\vdash_{\mathcal{H}}$ has reflexivity (tautology), weakening and transitivity / cut-elimination. It is closed under shifting upwards and downwards: $\{x \to y\} \dashv\vdash_{\mathcal{H}} \{x + 1 \to y + 1\}$

(B & C only had downward shifting, with increments in $\mathbb{N}$ and not assuming an injective morphism).

Equivalence to Horn clauses: the initial semilattice

We implement universes (`NES.t`) as non-empty finite sets of atoms, i.e. `Level.t × Z` pairs (using sorted, non-empty lists).

Define the equational theory of the free semilattice constructed from a set of generators (variables) $\mathcal{L}$ and relations (equalities) $\mathcal{R}$ using judgment $\mathcal{R} \vdash_{\mathcal{F}} u \equiv v$:

```
Inductive entails_free (p : rels) : NES.t × NES.t → Prop :=
| entails_c {l r} : List.In (l, r) p → p ⊢_ℱ l ≡ r
| entails_refl {x} : p ⊢_ℱ x ≡ x
| entails_sym {x y} : p ⊢_ℱ x ≡ y → p ⊢_ℱ y ≡ x
| entails_trans {x y z} : p ⊢_ℱ x ≡ y → p ⊢_ℱ y ≡ z → p ⊢_ℱ x ≡ z
| entails_add_congr x y n : p ⊢_ℱ x ≡ y → p ⊢_ℱ x + n ≡ y + n ..
 where " p ⊢_ℱ r " := (entails_free p r%_rel).
```

It is **initial**: equality of syntax, up-to the obvious quotient, under equalities in $\mathcal{R}$.

Equivalence to Horn Clauses

We can prove:

$$\text{clauses\_of\_relations}(\mathcal{R}) \vdash_{\mathcal{H}} \text{clauses\_of\_relation}\ (u \equiv v) \leftrightarrow \mathcal{R} \vdash_{\mathcal{F}} u \equiv v$$

clauses_of_relation transforms an equality into $|u| + |v|$ clauses, where $|\_|$ measures the number of distinct atoms in a join, using the equivalences:

$$u \equiv v \leftrightarrow u \leq v \wedge v \leq u$$

$$a_1 \sqcup \ldots \sqcup a_n \leq v \leftrightarrow (v \to a_1) \wedge \ldots \wedge v \to a_n$$

The original algorithm

Satisfiability checking based on a forward search algorithm on the Horn clauses:

$$[\![\Phi]\!] : \mathcal{C}$$

$$[\![w \leq \max(u+1, v) \wedge s < w]\!] = \{u+1, v \to w; w \to s+1\}$$

The original algorithm

Satisfiability checking based on a forward search algorithm on the Horn clauses:

$$[\![\Phi]\!] : \mathcal{C}$$

$$[\![w \leq \max(u+1, v) \wedge s < w]\!] = \{u+1, v \to w; w \to s+1\}$$

Given a set of clauses, the algorithm constructs a minimal model of the variables and clauses in $\mathbb{Z}^\infty$:

$$u \mapsto 0, v \mapsto 0, w \mapsto 1, s \mapsto 2 \quad \text{for the clauses above}$$

Intuitively, the lowest universe (s here) has highest value and we maintain minimal distances between universes, ∞ represents a loop.

I.o.w., if ∞ is reached during inference, there is no finite model.

The new algorithm, incremental and with early stop

Given finite sets of levels $\mathcal{L}$ and clauses $\mathcal{C}$, and a **partial** model $m_{\text{init}} : \mathcal{L} \to \mathbb{Z}^\infty$ (where some variables but not all might start at $+\infty$), return a minimal model in $\mathbb{Z}^\infty$ or report a loop.

The algorithm has two possible outcomes:

- a loop on a finite level was found, i.e. $\exists u, \mathcal{C} \vdash_{\mathcal{H}} u \to u + 1 \land m(u) \downarrow k \in \mathbb{Z}$ (trivial contradiction when interpreting in $\mathbb{Z}$).
- a minimal model was found, i.e. $\exists m, m \Vdash \mathcal{C}$. In addition, $\mathcal{C} \vdash_{\mathcal{H}} [\![m_{\text{init}}]\!] \to [\![m]\!]$:

$$[\![\_]\!] : (\mathcal{L} \to \mathbb{Z}^\infty) \to \text{premises}$$

$$[\![l \mapsto z \uplus m']\!] = \begin{cases} (l + k, [\![m']\!]) \text{ when } k \neq +\infty \\ [\![m']\!] \text{ otherwise} \end{cases}$$

The new algorithm in Rocq: specification

```
Definition model := LevelMap.t (option Z).
```

The new algorithm in Rocq: specification

```
Definition model := LevelMap.t (option Z).
```

Bodies are just universes.

```
Definition Clause.t := NES.t × (Level.t × Z).
Definition Clauses.t := Set(Clause.t).t.
```

Restriction operations on clauses:

- `cls ↓ W`: subset of clauses in `cls` with conclusion in `W`
- `cls ↓ W`: subset of clauses in `cls` with **all premises** and conclusion in `W`

Specification in Rocq: checking clauses

```
Definition check (cls : clauses) (m : model) : option (LevelSet.t × model) :=
  ..

Lemma check_model : check cls m = None ⟷ is_model cls m.

Lemma check_update : check cls m = Some (W, m') →
  strictly_updates cls W m m'.
```

Intuition of the algorithm: check if some updates in `m` are necessary to satisfy the clauses, until we find a model of all the clauses or a loop.

```
Lemma min_premise_spec (m : model) (s : NES.t) :
  (∃ (l, k) ∈ s, min_premise m s = option_map (fun v ⇒ v - k) m[l]) ∧
  (forall (l, k) ∈ s, min_premise m s ≼ option_map (fun v ⇒ v - k) m[l]).
```

Specification in Rocq: strict update relation

```
Definition strict_update m '(prems, (concl, k)) m' :=
  ∃ v, [/\ min_premise m prems = Some v, ¬ m[concl] ≺ (k + v) &
    m' =m (LevelMap.add concl (Some (k + v)) m)].
(* The "greatest" level in the premises is not greater than the conclusion *)

Inductive strictly_updates cls (s : LevelSet.t) : model → model → Prop :=
| update_one m cl m' : Clauses.In cl cls →
  strict_update m cl m' →
  strictly_updates cls (LevelSet.singleton (clause_conclusion cl)) m m'

| update_trans {ls ls' m m' m''} :
  strictly_updates cls ls m m' →
  strictly_updates cls ls' m' m'' →
  strictly_updates cls (LevelSet.union ls ls') m m''.
```

`strictly_updates` models **necessary** updates in `m` due to `cls` to get to model `m'`.

The new algorithm in Rocq: specification

```
Variant result (ls : LevelSet.t) (cls : Clauses.t) (m : model) :=
| Loop mext m' ls' loop :
  strictly_updates cls ls' m mext →
  strictly_updates (cls ⌊ loop) loop mext m' → result ls cls m

| Model ls' m' : ls' ⊊ ls → strictly_updates cls ls' m m' →
  is_model cls m' → result ls cls m
```

The new algorithm in Rocq: specification

```
Variant result (ls : LevelSet.t) (cls : Clauses.t) (m : model) :=
| Loop mext m' ls' loop :
  strictly_updates cls ls' m mext →
  strictly_updates (cls ⌊ loop) loop mext m' → result ls cls m

| Model ls' m' : ls' ⊊ ls → strictly_updates cls ls' m m' →
  is_model cls m' → result ls cls m
```

- `Loop`: There is a subset of the clauses that is looping, i.e.

$$\mathcal{C} \lfloor \text{loop} \vdash_{\mathcal{H}} [\![m_{\text{ext}}]\!] \to [\![m']\!] \land m_{\text{ext}} < m'$$
$$\Rightarrow \mathcal{C} \lfloor \text{loop} \vdash_{\mathcal{H}} [\![m_{\text{ext}}]\!] \to [\![m_{\text{ext}}]\!] + 1$$

- `Model`: `m'` is the minimal model above `m`; there is no loop on the finite levels in `m`.

A nested, well-founded recursive structure

```
Equations inner_loop (V W : LevelSet.t) (cls : Clauses.t) (m : model)
  (hconcl : clauses_conclusions cls ⊆ W) : result W cls m :=
 by (measure W cls m) lt := .. loop W LevelSet.empty (cls ⌊ W) m ..

Equations loop V W cls m : result V cls m :=
 by wf (loop_measure ls W) lexprod_rel :=
  .. inner_loop W (cls ↓ W) m .. .. loop V W' cls m' ..
```

Using the measures described by Bezem & Coquand, we can show these are terminating (computation in Rocq is possible).

- the inner loop is bounded by the **values** in `m` of premises **not in** `W`, which can force increase of values **in W** only a finite number of times.
- in the main loop the `W` set strictly increases until it reaches `V` in which case it stops with a loop.

Reaping the fruits: satisfiability

Satisfiability checking for clauses $(\mathcal{L}, \mathcal{C})$ in $\mathbb{Z}$:

$$(\mathcal{L}, \mathcal{C}) \Vdash \iff \exists \mathcal{M} : \mathcal{L} \to \mathbb{Z}, \mathcal{M} \Vdash \mathcal{C}$$

- Launch `loop` on a fully defined model with initial `0` value for each level in $\mathcal{L}$
- Correct and complete for interpretation in $\mathbb{Z}$

Valuation is a flipping of the model:

$$\{\!|\mathcal{M}|\!\} : (\mathcal{L} \to \mathbb{Z}) \to (\mathcal{L} \to \mathbb{N})$$
$$\{\!|\mathcal{M}|\!\}\ l = \max \mathcal{M} - \mathcal{M}\ l - \min \mathcal{M}$$

Constraint satisfiability: $\mathcal{M} \Vdash (\mathcal{L}, [\![\Phi]\!]) \iff \Phi[l \mapsto \{\!|\mathcal{M}|\!\}\ l]$ (in $\mathbb{Z}$)

Reaping the fruits: validity

Validity checking: $(\mathcal{L}, \Phi) \vdash u \geq v$?

Two possible algorithms:

1. Launch `loop` on a **partial** model $\{l \mapsto 0 \mid l \in \mathcal{L}\}$ check if v holds:
   - correct and complete for $\mathbb{Z}^\infty$ (i.e. the initial semilattice)
   - correct but **not** complete for $\mathbb{Z}$. Counter-example: $x \sqcup y \geq y + 1 \vdash x \geq y$?

     Minimal model of $\mathcal{M} := \{x \mapsto 0, y \mapsto \infty\}$ is $\mathcal{M}$ not $\{x \mapsto 0, y \mapsto 1\}$ as

     $$x \sqcup \infty \geq \infty + 1$$
2. Check satisfiability of $[\![\Phi]\!] \wedge [\![v + 1 \geq u]\!]$: if satisfiable, the initial clause is not valid, otherwise (loop) it is ($x \leq y \vee y < x$ in $\mathbb{Z}$)

   Correct and complete for $\mathbb{Z}$, but more expensive (might update the whole model).

Integration in MetaRocq

Rocq's universe system supposes a 0/bottom element and additionally:

$$l > 0 \text{ for global universes}$$

$$l \geq 0 \text{ for local (polymorphic) universes}$$

Also constraints are expressed using **natural number** increments only.

```
Definition init_model : univ_model. (* Only 0 is declared *)
(* Try to extend the model with a consistent set of constraints *)
Definition push_uctx : univ_model → ContextSet.t → option univ_model := ..
Definition check : univ_model → UnivConstraintSet.t → bool := ..
```

MetaRocq PR #704

Algebraic Universes and Variances in Rocq

Using the algorithm in Rocq

- Almost direct translation of the verified Rocq algorithm to OCaml.
- Efficiency concerns: smart representation of clauses, fast checking of clause sets.

Using the algorithm in Rocq

- Almost direct translation of the verified Rocq algorithm to OCaml.
- Efficiency concerns: smart representation of clauses, fast checking of clause sets.
- Model "benign" loops $l \leq l' + k \wedge l' + k \leq l$ as substitutions $l := l' + k$, reducing the number of universes and constraints.

Using the algorithm in Rocq

- Almost direct translation of the verified Rocq algorithm to OCaml.
- Efficiency concerns: smart representation of clauses, fast checking of clause sets.
- Model "benign" loops $l \leq l' + k \wedge l' + k \leq l$ as substitutions $l := l' + k$, reducing the number of universes and constraints.
- Generalizes Rocq to handle all constraints, simplifying the APIs (closer to Agda and Lean now).

Using the algorithm in Rocq

- Almost direct translation of the verified Rocq algorithm to OCaml.
- Efficiency concerns: smart representation of clauses, fast checking of clause sets.
- Model "benign" loops $l \leq l' + k \wedge l' + k \leq l$ as substitutions $l := l' + k$, reducing the number of universes and constraints.
- Generalizes Rocq to handle all constraints, simplifying the APIs (closer to Agda and Lean now).
- General substitution operation $\mathcal{M}[l := u]$, supporting universe metavariable instantiation.

Minimization

Elaboration with cumulativity ⇒ large number of unnecessary constraints.

Example `id@{?i} Type@{0} nat ▷ Type@{0} | 1 ≤ ?i` rather than `?i := 1`

W.l.o.g. `?i := 1`, same principal type and convertibility properties:

`id@{?i} Type@{0} nat` $\rightarrow^*$ `nat` $^*\leftarrow$ `id@{1} Type@{0} nat`

💡 Record **where** and **how** bound universes appear to perform simplifications.

⚒️ Operations on constraint sets/models $\Sigma_{\mathcal{U}} ::= (\mathcal{L}, \Phi, \mathcal{M})$:

- $(\mathcal{L}, \Phi, \mathcal{M})[?\, i := u] \triangleq (\mathcal{L} \setminus i, \Phi[i := u], \mathcal{M}[i := u])$ ($?\, i = u$ must be consistent)
- $\mathrm{glb}_{\Sigma_{\mathcal{U}}}\, l \triangleq \sqcup \{a \mid a \leq l \in \Sigma_{\mathcal{U}}.\Phi\}$

In this example: `?i` is irrelevant, so solving $\Sigma_{\mathcal{U}}\left[?\, i := \mathrm{glb}_{\Sigma_{\mathcal{U}}}\, ?\, i\right]$ is sound.

Universe Variances

Record for each definition the occurrences of bound universe variables in their body and type and their associated variance (compositionally).

A variance is either: irrelevant ($*$), covariant ($+$), contravariant ($-$) or invariant ($=$).

Universe Variances

Record for each definition the occurrences of bound universe variables in their body and type and their associated variance (compositionally).

A variance is either: irrelevant ($*$), covariant ($+$), contravariant ($-$) or invariant ($=$).

Cumulative Inductive Types already use this notion.

```
Inductive eq@{i} (A : Type@{i}) (a : A) : A → Type@{i} :=
| eq_refl : eq A a a.
```

For `i`:

- contravariant in the first binder
- covariant in the conclusion type
- irrelevant in the constructor arguments

Universe Variances

```
Inductive eq@{i} : (A : Type@{i}), A → A → Type@{i} := ..
```

Timany and Sozeau have shown that this justifies treating the i universe as **irrelevant** when eq is applied to its parameter.

`eq@{i} nat 0 1` ≡ `eq@{j} nat 0 1 : Type@{max(i,j)}` for all `i`, `j`

⇒ Justification behind so-called `template-polymorphic` inductives in Rocq

Idea: Lift this analysis to all definitions.

Universe Variances for Definitions

```
Definition relation@{i} (A : Type@{i}) := A → A → Prop.

Class Proper@{i} (A : Type@{i}) (R : relation A) := ∀ m, R m m.

Definition respectful@{i j} (A : Type@{i}) (B : Type@{j})
  (RA : relation A) (RB : relation B) : relation@{max(i,j)} (A → B) :=
  fun f g ⇒ ∀ x y, RA x y → RB (f x) (g y).
```

All universe binders here are irrelevant, when defs are fully applied.

$$\texttt{relation@\{i\}}\ A \equiv A \to A \to \texttt{Prop} \equiv \texttt{relation@\{j\}}\ A \equiv \texttt{relation@\{max(i,j)\}}\ A$$

- Sound first-order unification: enforce constraints only on invariant or covariant universes before unifying actual arguments.
- Enables simplifications: instances for the `i` binder can be lowered w.l.o.g.

Universe Variances for Definitions

Polymorphic functions on containers, e.g.:

```
map@{*i *j} (A : Type@{i}) (B : Type@{j}) : (A → B) → list@{i} A → list@{j} B
```

HoTT path algebra: groupoid laws use the equality's universe level irrelevantly, e.g.:

```
concat1p@{*i} (A : Type@{i}) (x y : A) (p : x = y) : 1 @ p = p.
```

User Interface

Mixed inference mode

Declaring only quantified universes:

```
Lemma foo@{i j} : forall X : Type@{i}, predicate@{i j} X. Proof. …. Qed.
```

Solves all metavariables in terms of `i` and `j`, but can add constraints.

Checking mode

```
Lemma foo@{i j| } : Type@{j}.
Proof. Fail exact Type@{i}. (* i < j not implied by the constraints. *)
```

- Porting the HoTT library: $\sim$ 500LoC diff, adding explicit universe bindings.
- Uses checking for fine-grained cumulativity constraints in a few cases.

Summary

- A **mechanized** variant of Bezem and Coquand's algorithm for full universe inference and checking with algebraic universes and cumulativity
- Variances for **sound and efficient** unification, conversion and minimization
- Supports both **explicit and implicit** universes and constraints in Rocq

Summary

- A **mechanized** variant of Bezem and Coquand's algorithm for full universe inference and checking with algebraic universes and cumulativity
- Variances for **sound and efficient** unification, conversion and minimization
- Supports both **explicit and implicit** universes and constraints in Rocq

WIP:

- Porting of Rocq developments: done for HoTT, elpi/Hierarchy Builder, next is math-comp
- Launch of Rocq 10 branch integrating this in a new, level polymorphic core library with bounded sort polymorphism [8, 9].

Open questions

- Is there a **strongly** polynomial algorithm?
- First-class bounded polymorphism? Type preservation issues if done naïvely.

$$\mathcal{U}_w \ ?$$

$$\mathcal{U}_\omega : \mathcal{U}_{\omega+1} : \ldots \ ?$$

- Alternative hierarchies: $\mathbb{Q}$?

Bibliography

[1] Antoine Allioux, Eric Finster, and Matthieu Sozeau. 2021. Types are Internal ∞-groupoids. *LICS* (January 2021). https://arxiv.org/abs/2105.00024

[2] Antoine Allioux. 2023. Higher Structures in Homotopy Type Theory. https://inria.hal.science/tel-04335842v1

[3] Michael A. Bender, Jeremy T. Fineman, Seth Gilbert, and Robert E. Tarjan. 2016. A New Approach to Incremental Cycle Detection and Related Problems. *ACM Trans. Algorithms* 12, 2 (2016), 1–22. DOI: 10.1145/2756553

[4] Marc Bezem and Thierry Coquand. 2022. Loop-checking and the uniform word problem for join-semilattices with an inflationary endomorphism. *Theoretical Computer Science* 913, (2022), 1–7. DOI: 10.1016/j.tcs.2022.01.017

[5] Marc Bezem, Robert Nieuwenhuis, and Enric Rodríguez-Carbonell. 2008. The Max-Atom Problem and Its Relevance. In *Logic for Programming, Artificial Intelligence, and Reasoning*, 2008. Springer Berlin Heidelberg, 47–61. DOI: 10.1007/978-3-540-89439-1_4

[6] Thierry Coquand, Henri Lombardi, and Stefan Neuwirth. 2024. An introduction to Lorenzen's "Algebraic and logistic investigations on free lattices" (1951). https://arxiv.org/abs/1711.06139

[7] Paul Lorenzen. 2023. Algebraic and logistic investigations on free lattices. https://arxiv.org/abs/1710.08138

[8] Josselin Poiret, Gaëtan Gilbert, Kenji Maillard, Pierre-Marie Pédrot, Matthieu Sozeau, Nicolas Tabareau, and Éric Tanter. 2025. All Your Base Are Belong to Us: Sort Polymorphism for Proof Assistants. *Proc. ACM Program. Lang.* 9, POPL (January 2025). DOI: 10.1145/3704912

[9] Johann Rosain, Tomás Díaz, Kenji Maillard, Matthieu Sozeau, Nicolas Tabareau, Éric Tanter, and Théo Winterhalter. 2026. Bounded Sort Polymorphism with Elimination Constraints. *PACMPL* POPL (January 2026). https://hal.science/hal-05372721

[10] Matthieu Sozeau, Yannick Forster, Meven Lennon-Bertrand, Jakob Nielsen, Nicolas Tabareau, and Théo Winterhalter. 2025. Correct and Complete Type Checking and Certified Erasure for Coq, in Coq. *J. ACM* 72, 1 (January 2025). DOI: 10.1145/3706056