The formal verification of the ctm approach to forcing[⋆]

Emmanuel Gunther[1], Miguel Pagano[1],
Pedro Sánchez Terraf[1,2], and Matías Steinberg[1]

[1] Universidad Nacional de Córdoba.
Facultad de Matemática, Astronomía, Física y Computación.
[2] Centro de Investigación y Estudios de Matemática (CIEM-FaMAF), Conicet.
Córdoba. Argentina.
{gunther,sterraf}@famaf.unc.edu.ar
{miguel.pagano,matias.steinberg}@unc.edu.ar

Abstract. We discuss some highlights of our computer-verified proof of the construction, given a countable transitive set-model M of ZFC, of generic extensions satisfying $ZFC + \neg CH$ and $ZFC + CH$. Moreover, let $\mathcal{R}$ be the set of instances of the Axiom of Replacement. We isolated a 21-element subset $\Omega \subseteq \mathcal{R}$ and defined $\mathcal{F} : \mathcal{R} \to \mathcal{R}$ such that for every $\Phi \subseteq \mathcal{R}$ and M-generic G, $M \models ZC \cup \mathcal{F}``\Phi \cup \Omega$ implies $M[G] \models ZC \cup \Phi \cup \{\neg CH\}$, where ZC is Zermelo set theory with Choice.

To achieve this, we worked in the proof assistant *Isabelle*, basing our development on the Isabelle/ZF library by L. Paulson and others.

Keywords: forcing · Isabelle/ZF · countable transitive models · continuum hypothesis · proof assistants · interactive theorem provers · generic extension

1 Introduction

This paper is the culmination of our project on the computerized formalization of the undecidability of the Continuum Hypothesis (CH) from Zermelo-Fraenkel set theory with Choice (ZFC), under the assumption of the existence of a countable transitive model (ctm) of ZFC. In contrast to our reports of the previous steps towards this goal [19, 20, 21], we intend here to present our development to the mathematical logic community. For this reason, we start with a general discussion around the formalization of mathematics.

1.1 Formalized mathematics

The use of computers to assist the creation and verification of mathematics has seen a steady grow. But the general awareness on the matter still seems to be a

⋆ Supported by Secyt-UNC projects 33620180100855CB and 33620180100465CB, and Conicet.

bit scant (even among mathematicians involved in foundations), and the venues devoted to the communication of formalized mathematics are, mainly, computer science journals and conferences: JAR, ITP, IJCAR, CPP, CICM, and others.

Nevertheless, the discussion about the subject in central mathematical circles is increasing; there were some hints on the ICM2018 panel on "machine-assisted" proofs [12] and a lively promotion by Kevin Buzzard, during his ICM2022 special plenary lecture [5].

Before we start an in-depth discussion, a point should be made clear: A formalized proof is not the same as an *automatic proof*. The reader surely understands that, aside from results of a very specific sort, no current technology allows us to write a reasonably complex (and correct) theorem statement in a computer and expect to obtain a proof after hitting "Enter", at least not after a humanly feasible wait. On the other hand, it is quite possible that the same reader has some mental image that formalizing a proof requires making each application of Modus Ponens explicit.

The fact is that *proof assistants* are designed for the human prover to be able to decompose a statement to be proved into smaller subgoals which can actually be fed into some automatic tool. The balance between what these tools are able to handle is not easily appreciated by intuition: Sometimes, "trivial" steps are not solved by them, which can result in obvious frustration; but they would quickly solve some goals that do not look like a "mere computation."

To appreciate the extent of mathematics formalizable, it is convenient to recall some major successful projects, such as the Four Color Theorem [16], the Odd Order Theorem [17], and the proof the Kepler's Conjecture [24]. There is a vast mathematical corpus at the Archive of Formal Proofs (AFP) based on Isabelle; and formalizations of brand new mathematics like the Liquid Tensor Experiment [53, 54] and the definition of perfectoid spaces [6] have been achieved using Lean.

We will continue our description of proof assistants in Section 2. We kindly invite the reader to enrich the previous exposition by reading the apt summary by A. Koutsoukou-Argyraki [30] and the interviews therein; some of the experts consulted have also discussed in [3] the status of formalized versus standard proof in mathematics.

1.2 Our achievements

We formalized a model-theoretic rendition of forcing (Sect. 4), showing how to construct proper extensions of ctms of ZF (respectively, with AC), and we formalized the basic forcing notions required to obtain ctms of $ZFC + \neg CH$ and of $ZFC + CH$ (Sect. 6.2). No metatheoretic issues (consistency, FOL calculi, etc) were formalized, so we were mainly concerned with the mathematics of forcing. Nevertheless, by inspecting the foundations underlying our proof assistant Isabelle (Section 2.1) it can be stated that our formalization is a bona fide proof in ZF of the previous constructions.

In order to reach our goals, we provided basic results that were missing from Isabelle's ZF library, starting from ones involving cardinal successors, countable sets, etc. (Section 3.2). We also extended the treatment of relativization of set-theoretical concepts (Section 3.1).

One added value that is obtained from the present formalization is that we identified a handful of instances of Replacement which are sufficient to set the forcing machinery up (Section 6.1), on the basis of Zermelo set theory. The eagerness to obtain this level of detail might be a consequence of "an unnatural tendency to investigate, for the most part, trivial minutiae of the formalism" on our part, as it was put by Cohen [9], but we would rather say that we were driven by curiosity.

The code of our formalization can be accessed at the AFP site, via the following link:

https://www.isa-afp.org/entries/Independence_CH.html

2 Proof assistants and Isabelle/ZF

Let us briefly introduce Isabelle [43] in the large landscape of proof assistants ("assistants" for short; also known as "interactive theorem provers"); we refer to the excellent chapter by Harrison et al. [27] for a more thorough reconstruction of the history of assistants.

It is expected that an assistant aids the human user while mechanizing some piece of mathematics; the interaction varies from system to system, but a common interface consists of a display showing the current goal and assumptions. The user instructs the assistant to modify them by means of *tactics*; a proof is completed when the (current) goal is an instance of one of the assumptions.

In that dialog, the user produces a script of tactics that can be later reproduced step-by-step by the system (to check, for example, that an imported theory is correct) or by the user to understand the proof.

To have any value at all, the system should only allow the application of sound tactics. Edinburgh LCF [18] was an influential proof assistant in which the critical code (that constructs proofs in response to user scripts or other modules) was reduced to a small *kernel*. Hence, by verifying the correctness of the kernel, one achieves confidence on the whole system.

The metalogic of Isabelle, as well as that of LCF, is based on higher-order logic. In contrast, some of the other prominent assistants of today are based on some (dependent) type theory. Both Coq [10] and Lean [38] are based on the Calculus of Inductive Constructions [50, 11], while Agda [1] is an extension of Martin-Löf type theory [32]. Mizar [36] is the oldest assistant still used today but is far away both in terms of foundations and architecture from Isabelle; Mizar inspired, though, the Isar [56] dialect used in Isabelle nowadays, which aims at the production of proof scripts that are closer to mathematical texts.[3]

Isabelle also inherited from LCF the possibility for the user to define tactics to encapsulate common patterns used to solve goals. In fact, this is the origin of

[3] We recommend the survey [2] by J. Avigad for details about the different logical foundations on which assistants are based.

the ML family of languages: a Meta-Language for programming tactics. In the case of Isabelle, *Standard* ML is the first of the four layers on which we worked in this assistant. Both the kernel and the automation of proofs are coded in ML, sometimes as a substitute for induction on formulas, as the next section explains.

2.1 Isabelle metalogic $\boldsymbol{\mathcal{M}}$

The second layer of Isabelle is an intuitionistic fragment of higher-order logic (or simple type theory) called $\mathcal{M}$; its original version was described in [42], and the addition of "sorts" was reported in [39].

The only predefined type is `prop` ("propositions"); new base types can be postulated when defining objects logics. Types of higher order can be assembled using the function space constructor $\Rightarrow$.

The type of propositions `prop` is equipped with a binary operation $\Longrightarrow$ ("meta-implication") and a universal "meta-quantifier" $\bigwedge$, that are used to represent the object logic rules. As an example, the axiomatization of first-order logic postulates a type `o` of booleans, and Modus Ponens is written as

$$\bigwedge P\,Q.\ \ [P \longrightarrow Q] \implies ([P] \implies [Q]). \tag{1}$$

The square brackets (which are omitted in Isabelle theories, as well as the outermost universal quantifiers) represent an injection from `o` into `prop`. A consequence of this representation is that every formula of the object logic appears atomic to $\mathcal{M}$.

Types in Isabelle are organized into *classes* and *sorts*; for ease of exposition, we will omit the former. The axiomatization of first-order logic postulates a sort {`term`} (of "individuals," or elements of a first-order universe of discourse) and stipulates that every further type variable α must be of that sort. In particular, Isabelle/ZF only postulates one new type `i` ("sets") of sort {`term`}. Hence, from the type of the universal quantifier functional $\forall :: (\alpha \Rightarrow \mathtt{o}) \Rightarrow \mathtt{o}$, it follows that it may only be applied to predicates with a variable of type `i`. This ensures that the object logic is effectively first-order.

Paulson [42] carried out a proof that the encoding $\mathcal{M}_{\mathrm{IFOL}}$ of intuitionistic first-order logic IFOL without equality in the original $\mathcal{M}$ is conservative (there is a correspondence between provable φ in IFOL and provable $[\varphi]$ in $\mathcal{M}_{\mathrm{IFOL}}$) by putting $\mathcal{M}_{\mathrm{IFOL}}$ proofs in *expanded normal form* [52]; atomicity as stated after Equation (1) plays a role in this argument. Passing to classical logic does not present difficulties, but the addition of meta-equality must be taken care of. Even more so, since the treatment of equality differs between the original and the present incarnation of $\mathcal{M}$; details for the latter are exhaustively expounded in the recent formalization by Nipkow and Roßkof [40].

The meta-logic $\mathcal{M}$ is rather weak; it has no induction/recursion principles. Types are not inductively presented and, in particular, it is not possible to prove by induction statements about object-logic formulas (which are construed as terms of type $\mathtt{i} \Rightarrow \ldots \Rightarrow \mathtt{i} \Rightarrow \mathtt{o}$). Two ways to overcome this limitation are:

1. to construct the proof of each instance of the statement by hand or by programming on ML; or
2. to encode formulas as sets and prove an internal version statement using induction of ZF.

For recursive definitions, only the second option is available, and that is the way the definition of the forcing relation is implemented in our formalization.

2.2 Isabelle/ZF

For the most part, the development of set theory in Isabelle is carried out using its ZF object logic [49], which is the third logical layer of the formalization and the most versatile one, since Isabelle's native automation is available at this level. Apart from the type and sort declarations detailed above, it features a finite axiomatization, with a predicate for membership, constants for the empty set and an infinite set, and functions $\texttt{Pow} :: \texttt{i} \Rightarrow \texttt{i}$, $\bigcup :: \texttt{i} \Rightarrow \texttt{i}$, and $\texttt{PrimReplace} :: \texttt{i} \Rightarrow (\texttt{i} \Rightarrow \texttt{i} \Rightarrow \texttt{o}) \Rightarrow \texttt{i}$ (for Replacement). The Axiom of Replacement has a free predicate variable P:

$$(\forall x \in A.\ \forall y\, z.\ P(x,y) \wedge P(x,z) \longrightarrow y = z) \implies \\ b \in \texttt{PrimReplace}(A,P) \longleftrightarrow (\exists x \in A.\ P(x,b))$$

The restrictions on sorts described above ensure that it is not possible that higher-order quantification gets used in P. The statement of AC also uses a free higher-order variable to denote an indexed family of nonempty sets.

Isabelle/ZF reaches essentially Hessenberg's $|A| \cdot |A| = |A|$. Our decision (during 2017) to use this assistant was triggered by its constructibility library, *ZF-Constructible* [45], which contains the development of L, the proof that it satisfies AC, and a version of the Reflection Principle. The latter was actually encoded as a series of instructions to Isabelle automatic proof tools that would prove each particular instance of reflection: This is an example of what was said at the end of Section 2.1.

The development of relativization and absoluteness for classes $C :: \texttt{i} \Rightarrow \texttt{o}$ follows the same pattern. Each particular concept was manually written in a relational form and relativized. Here, the contrast between the usual way one regards ZF as a first-order theory in the language $\{\in\}$ and the mathematical practice of freely using defined concepts comes to the forefront. Assistants have refined mechanisms to cope with defined concepts and the introduction of new notation (which also make their foundations more complicated than plain first-order logic), and this is the only way that nontrivial mathematics can be formalized. But when studying relative interpretations, one usually assumes a spartan syntax and defines relativization by induction on formulas of the more succinct language. The approach taken in *ZF-Constructible* is to consider relativizations of the formulas that define each concept. For instance, in the case of unions, we find a relativization $\texttt{big\_union} :: (\texttt{i} \Rightarrow \texttt{o}) \Rightarrow \texttt{i} \Rightarrow \texttt{i} \Rightarrow \texttt{o}$ of the statement "$\bigcup A = z$":

$$\texttt{big\_union}(M,A,z) \equiv \forall x[M].\ x \in z \longleftrightarrow (\exists y[M].\ y \in A \wedge x \in y)$$

where $\forall x[M] \ldots$ stands for $\forall x.\ M(x) \longrightarrow \ldots$, etc. The need to work with *relational* presentations of defined concepts stems from the fact that the model-theoretic definition of L requires working with set models and satisfaction, which is defined for ("codes" of) formulas in the language $\{\in\}$ (viz. next Section 2.3).

There is one further point concerning the organization of *ZF-Constructible*. Isabelle provides a very convenient way to define "contexts," called *locales*, in which some variables are fixed and assumptions are made. In the case of the constructibility library, several locales are defined where the variable M is assumed to denote a class satisfying certain finite amount of *ZF*; the weakest one, `M_trans` [21, Sect. 3], just assumes that M is transitive and nonempty. Inside such context, many absoluteness results are proved. In order to quote those results for a particular class C, one has to *interpret* the locale at C, which amounts to prove that C satisfies the assumptions made by the context.

2.3 Internalized formulas

ZF-Constructible defines the set `formula` of first-order formulas in the language $\{\in\}$, internalized as sets.[4] Its atomic formulas have the form $\cdot x \in y \cdot$ and $\cdot x = y \cdot$. (We use dots as a visual aid signaling internalized formulas.) Variables are represented by de Bruijn indices [4], so in those formulas $x, y \in \omega$; for $\varphi \in$ `formula` and $z \in \omega$, z is free in φ if it occurs under at most z quantifiers. The `arity` function on φ is one plus the maximum free index occurring in φ.

The satisfaction predicate `sats` $::$ `i` $\Rightarrow$ `i` $\Rightarrow$ `i` $\Rightarrow$ `o` takes as arguments a set M, a list $env \in \mathtt{list}(M)$ for the assignment of free indices, and $\varphi \in \mathtt{formula}$, and it is written $M, env \models \varphi$ in our formalization. This completes the description of the fourth and last formal layer of the development.

Internalized formulas for most (but not all) of the relational concepts can be obtained by guiding the automatic tactics. But in the early development of *ZF-Constructible*, most of the concepts were internalized by hand; this is the case for union,

$$\mathtt{big\_union\_fm}(A, z) \equiv \\ ((\cdot\forall\cdot \cdot 0 \in succ(z)\cdot \longleftrightarrow (\cdot\exists\cdot \cdot 0 \in succ(succ(A))\cdot \wedge \cdot 1 \in 0\cdot \cdot\cdot)\cdot\cdot)$$

for which we have the following satisfaction lemma:

$$A \in \omega \implies z \in \omega \implies env \in \mathtt{list}(M) \implies \\ \big(M, env \models \mathtt{big\_union\_fm}(A, z)\big) \longleftrightarrow \\ \mathtt{big\_union}(\#\#M, \mathtt{nth}(A, env), \mathtt{nth}(z, env)) \quad (2)$$

Above, $\mathtt{nth}(x, env)$ is the xth element of env and $\#\#M$ $::$ `i` $\Rightarrow$ `o` is the class corresponding to the set M $::$ `i`.

[4] These, alongside with lists, are instances of Isabelle/ZF treatment of inductively defined (internal) datatypes [44, Sect. 4]; induction and recursion theorems for them are proved automatically (this is in contrast to general well-founded recursion, for which one has to work with the fundamental recursor `wfrec`).

3 Relative versions of non-absolute concepts

The treatment of relativization/internalization described in the previous sections was enough for Paulson's treatment of constructibility. This is the case because essentially all the concepts in the way of proving the consistency of AC are absolute, and the treatment of relational versions and relativized notions could be minimized after proving the relevant absoluteness results: For example, the lemma `Union_abs`,

$$M(A) \implies M(z) \implies \texttt{big\_union}(M, A, z) \longleftrightarrow z = \bigcup A$$

proved under the assumption that M is transitive and nonempty.

Our first attempt to relativize cardinal arithmetic proceeded in the same way and we rapidly found out that stating and proving statements like $(||A|| = |A|)^M$ in a completely relational language was extremely cumbersome. This observation lead to the discovery of the discipline expounded in the next subsection.

3.1 Discipline and tools for relativization

The missing step, that naturally appears in the literature, consists of having relative *functions* like $\mathcal{P}^M$, and the ability to translate between the different presentations discussed so far.

To achieve this, we provide automatic tools to ease the definitions of such relative versions, their fully relational counterparts, and the internalized formulas. For instance, consider the `cardinal` :: `i` ⇒ `i` function defined in *Isabelle/ZF*. Then the commands

```
relativize functional "cardinal" "cardinal_rel" external
relationalize "cardinal_rel" "is_cardinal"
synthesize "is_cardinal" from_definition assuming "nonempty"
```

define the relative cardinal function `cardinal_rel` :: (`i` ⇒ `o`) ⇒ `i` ⇒ `i` (denoted $|\cdot|^M$, as expected), the relational version `is_cardinal` of the latter, the internalized formula `is_cardinal_fm` whose satisfaction by a set is equivalent to the relational version, and prove the previous statement (analogous to (2)). The proof that $\texttt{is\_cardinal}(M, x, z)$ encodes the statement $|x|^M = z$ must still be done by hand, since the definition of `cardinal_rel` already involves some tacit absoluteness results ("*the least* $z \in \mathsf{Ord}$ *such that* $z \approx^M x$" instead of "*the least* $z \in \mathsf{Ord}^M$ *such that* $z \approx^M x$", and the like).

3.2 Extension of Isabelle/ZF

We extended [55] the material formalized in Isabelle, from basic results involving function spaces and the definition of cardinal exponentiation, to a treatment of cofinality and the Delta System Lemma for ω_1-families. We also included a concise treatment of the axiom of Dependent Choices DC and the general version of Rasiowa-Sikorski Lemma [19] and a choiceless one for countable preorders.

This material was subsequently put in relative form in our formal development on transitive class models [23] using as an aid the tools from Section 3.1. We also relativized many original theories appearing in Isabelle/ZF, including the fundamentals of cardinal arithmetic, the cumulative hierarchy, and the definition of the $\aleph$ function.

4 Set models and forcing

4.1 The *ZFC* axioms as locales

The description of set models of fragments of *ZFC* was performed using locales that fix a variable $M :: \mathtt{i}$ and pack assumptions stating that $\langle M, \in\rangle$ satisfy some axioms; for example, the locale `M_Z_basic` states that Zermelo set theory holds in M. It would be natural to state those assumptions directly as the corresponding satisfactions, as in

$$M, [\,] \models \cdot\mathtt{Union\ Ax}\cdot$$

where $\cdot$`Union Ax`$\cdot$ is the `formula` code for the Union Axiom. We actually decided to express the axioms other than the infinite schemes in relational form, by using terms already available in *ZF-Constructible* and for which useful lemmas had already been proved (and, as it was mentioned in Section 2.2, this third layer of the formalization has more tools at our disposal); the Union Axiom (`Union_ax`), for instance, is defined as follows:

$$\forall x[\mathtt{\#\#}M].\ \exists z[\mathtt{\#\#}M].\ \mathtt{big\_union}(\mathtt{\#\#}M, x, z)$$

Both assumptions are then shown to be equivalent:

$$\mathtt{Union\_ax}(\mathtt{\#\#}M) \longleftrightarrow M, [\,] \models \cdot\mathtt{Union\ Ax}\cdot$$

For stating the axiom schemes, *ZF-Constructible* defines the expressions

$$\mathtt{separation}(N, Q) \text{ and } \mathtt{strong\_replacement}(N, R)$$

whose first argument N is a class and their second arguments Q and R are predicates of types $Q :: \mathtt{i} \Rightarrow \mathtt{o}$ and $R :: \mathtt{i} \Rightarrow \mathtt{i} \Rightarrow \mathtt{o}$, respectively. The Separation Axiom appears in `M_Z_basic` as follows, where φ is free and `@` denotes list concatenation:

```
separation_ax: "φ ∈ formula ⟹ env ∈ list(M) ⟹
                    arity(φ) ≤ 1 +ω length(env) ⟹
                    separation(##M,λx. (M, [x] @ env ⊨ φ))"
```

Note that the predicate Q mentioned above corresponds to the satisfaction of φ.

In contrast to Separation, we stated each instance of the Replacement Axiom separately by means of the $\mathtt{replacement\_assm}(M, env, \varphi)$ predicate:

```
φ ∈ formula ⟶ env ∈ list(M) ⟶
  arity(φ) ≤ 2 +ω length(env) ⟶
    strong_replacement(##M,λx y. (M , [x,y]@env ⊨ φ))"
```

In turn, the ·`Replacement`· function takes a formula code and returns the corresponding replacement instance:

```
φ ∈ formula ⟹
(M, [] ⊨ ·Replacement(φ)·) ⟷ (∀env. replacement_assm(M, env, φ))
```

Starting from `M_Z_basic`, stronger locales are defined by assuming more replacement instances. These assumptions are then invoked to interpret at the class `##`M the relevant locales appearing in *ZF-Constructible*, and further ones required for the relative results from Section 3.2. See Section 6.1 for details.

4.2 The fundamental theorems

At this point, we work inside the locale `M_ctm1` that assumes M to be countable and transitive, and satisfies some fragment of *ZFC*[5]. This is further extended by assuming a forcing notion $\langle \mathbb{P}, \preceq, \mathbf{1}\rangle \in M$. The actual implementation reads:

```
locale forcing_notion =
  fixes P (‹ℙ›) and leq and one (‹1›)
  assumes one_in_P:       "1 ∈ ℙ"
    and leq_preord:       "preorder_on(ℙ,leq)"
    and one_max:          "∀p∈ℙ. ⟨p,1⟩∈leq"

locale forcing_data1 = forcing_notion + M_ctm1 +
  assumes P_in_M:          "ℙ ∈ M"
    and leq_in_M:         "leq ∈ M"
```

The version of the Forcing Theorems that we formalized follows the considerations on the $\Vdash^*$ relation as discussed in Kunen's new *Set Theory* [31, p. 257ff]. We defined forcing for atomic formulas by recursion on names in an analogous fashion. But, in contrast to the point made on p. 260 of this book, the structural recursion used to define the forcing relation was replaced by one involving codes for formulas. Thus, the metatheoretic formula transformer $\varphi \mapsto \mathit{Forces}_\varphi$ was replaced by a set-theoretic class function `forces` :: `i` ⇒ `i`, which was defined by using Isabelle/ZF facilities for primitive recursion.

Next, we state this version of the fundamental theorems in a compact way. For any $G \subseteq \mathbb{P}$, our notation for the extension of M by G is the customary one: $M[G] := \{\mathit{val}(G,\tau) : \tau \in M\}$, where the interpretation $\mathit{val}(G,\tau)$ is defined by well-founded recursion on τ.

Theorem 1. *For every* $\varphi \in$ `formula` *with* `arity`$(\varphi) \leq n$ *and* $\tau_1, \dots, \tau_n \in M$*,*

1. *(Definability)* `forces`$(\varphi) \in$ `formula`*,*

where the arity of `forces`(φ) *is at most* `arity`$(\varphi)+4$*; and if "*$p \Vdash \varphi\ [\tau_1, \dots, \tau_n]$*" denotes "*$M, [p, \mathbb{P}, \preceq, \mathbf{1}, \tau_1, \dots, \tau_n] \models$ `forces`(φ)*", we have:*

[5] Namely, Zermelo set theory plus the 7 replacement instances included in the locales `M_ZF1` and `M_ZF_ground`.

2. *(Truth Lemma) for every M-generic G,*

$$\exists p \in G.\ \ p \Vdash \varphi\ [\tau_1, \ldots, \tau_n]$$

is equivalent to

$$M[G], [\mathit{val}(G,\tau_1), \ldots, \mathit{val}(G,\tau_n)] \models \varphi.$$

3. *(Density Lemma)* $p \Vdash \varphi\ [\tau_1, \ldots, \tau_n]$ *if and only if* $\{q \in \mathbb{P} : q \Vdash \varphi\ [\tau_1, \ldots, \tau_n]\}$ *is dense below* p.

The items in Theorem 1 appear in our *Independence_CH* session [22] as three separate lemmas (located in the theory `Forcing_Theorems`). For instance, the Truth Lemma is stated as follows:

```
lemma truth_lemma:
  assumes
    "φ∈formula"
    "env∈list(M)" "arity(φ)≤length(env)"
  shows
    "(∃p∈G. p ⊩ φ env)   ⟷   M[G], map(val(G),env) ⊨ φ"
```

where the $\Vdash$ notation (and its precedence) had already been set up in the `Forces_Definition` theory.

Kunen first describes forcing for atomic formulas using a mutual recursion but then [31, p. 257] it is cast as a single recursively defined function F over a well-founded relation R. In our formalization, these are called `frc_at` and `frecR`, respectively, and are defined on tuples $\langle \mathit{ft}, t_1, t_2, p\rangle$ (where $\mathit{ft} \in \{0,1\}$ indicates whether the atomic formula being forced is an equality or a membership, respectively). Forcing for general formulas is then defined by recursion on the datatype `formula` as indicated above. Technical details on the implementation and proofs of the Forcing Theorems have been spelled out in our [21].

5 A sample formal proof

We present a fragment of the formal version of the proof that the Powerset Axiom holds in a generic extension, which also serves to illustrate the Isar dialect of Isabelle.

We quote the relevant paragraph of Kunen's [31, Thm. IV.2.27]:

> For Power Set (similarly to Union above), it is sufficient to prove that whenever $a \in M[G]$, there is a $b \in M[G]$ such that $\mathcal{P}(a) \cap M[G] \subseteq b$. Fix $\tau \in M^{\mathbb{P}}$ such that $\tau_G = a$. Let $Q = (\mathcal{P}(\mathrm{dom}(\tau) \times \mathbb{P}))^M$. This is the set of all names $\vartheta \in M^{\mathbb{P}}$ such that $\mathrm{dom}(\vartheta) \subseteq \mathrm{dom}(\tau)$. Let $\pi = Q \times \{\mathbf{1}\}$ and let $b = \pi_G = \{\vartheta_G : \vartheta \in Q\}$. Now, consider any $c \in \mathcal{P}(a) \cap M[G]$; we need to show that $c \in b$. Fix $\chi \in M^{\mathbb{P}}$ such that $\chi_G = c$, and let $\vartheta = \{\langle \sigma, p\rangle : \sigma \in \mathrm{dom}(\tau) \wedge p \Vdash \sigma \in \chi\}$; $\vartheta \in M$ by the Definability Lemma. Since $\vartheta \in Q$, we are done if we can show that $\vartheta_G = c$.

The assumption $a \in M[G]$ appears in the lemma statement, and the goal involving b in the first sentence will appear below (signaled by "(**)"); formalized material necessarily tends to be much more linear than usual prose. In what follows, we will intersperse the relevant passages of the proof.

```
lemma Pow_inter_MG:
  assumes "a∈M[G]"
  shows "Pow(a) ∩ M[G] ∈ M[G]"
proof -
```

Fix $\tau \in M^{\mathbb{P}}$ such that $\tau_G = a$.

```
  from assms
  obtain τ where "τ ∈ M" "val(G, τ) = a"
    using GenExtD by auto
```

Let $Q = (\mathcal{P}(\mathrm{dom}(\tau) \times \mathbb{P}))^M$. This is the set of all names $\vartheta \in M^{\mathbb{P}}$ [...]

```
  let ?Q="Pow^M(domain(τ)×ℙ)"
```

Let $\pi = Q \times \{\mathbf{1}\}$ and let $b = \pi_G = \{\vartheta_G : \vartheta \in Q\}$.

```
  let ?π="?Q×{1}"
  let ?b="val(G,?π)"
```

(Recall: *...there is a $b \in M[G]$ such that...*)

```
  from ‹τ∈M›
  have "domain(τ)×ℙ ∈ M" "domain(τ) ∈ M"
    by simp_all
  then
  have "?b ∈ M[G]"
    by (auto intro!:GenExtI)
```

Now, consider any $c \in \mathcal{P}(a) \cap M[G]$; we need to show that $c \in b$.

```
  have "Pow(a) ∩ M[G] ⊆ ?b"                                    (**)
  proof
    fix c
    assume "c ∈ Pow(a) ∩ M[G]"
```

Fix $\chi \in M^{\mathbb{P}}$ such that $\chi_G = c$,

```
    then
    obtain χ where "c∈M[G]" "χ ∈ M" "val(G,χ) = c"
      using GenExt_iff by auto
```

and let $\vartheta = \{\langle\sigma, p\rangle : \sigma \in \mathrm{dom}(\tau) \wedge p \Vdash \sigma \in \chi\}$;

```
    let ?ϑ="{⟨σ,p⟩ ∈domain(τ)×ℙ . p ⊩ ·0 ∈ 1· [σ,χ] }"
```

$\vartheta \in M$ by the Definability Lemma.

```
have "arity(forces( ·0 ∈ 1· )) = 6"
  using arity_forces_at by auto
with ‹domain(τ) ∈ M› ‹χ ∈ M›
have "?ϑ ∈ M"
  using sats_fst_snd_in_M
  by simp
```

Since $\vartheta \in Q$,

```
with ‹domain(τ)×ℙ ∈ M›
have "?ϑ ∈ ?Q"
  using Pow_rel_char by auto
```

we are done if we can show that $\vartheta_G = c$.

```
have "val(G,?ϑ) = c"
proof  [...]
```

This cherry-picked example shows that the formalization can be close to the mathematical exposition and might be useful to reconstruct the proof from the book; nonetheless, it also has significantly more details than the mathematical prose, even with some indications to direct the automatic tools.

There has been some progress on assistants where one writes statements and proofs in natural language; recently P. Koepke and his team achieved magnificent results by using Isabelle/Naproche [13] to formalize proofs of several results (particularly, the proof of König's Theorem). The input language of Isabelle/Naproche is a *controlled* natural language that presents the result being formalized as a deduction in first-order logic, where every assumption and the "whole logical scenario" are explicitly given. From the input language, Isabelle/Naproche builds "proof tasks" that are handled to automatic theorem provers. As far as we can tell, Isabelle/Naproche is promising but still unsuitable for a project of the magnitude of ours.

6 Main achievements of the formalization

6.1 A sufficient set of replacement instances

We isolated 22 instances of Replacement that are sufficient to force CH or $\neg CH$. Many of these were already present in relational form in the *ZF-Constructible* library.

The first 4 instances, collected in the subset `instances1_fms` of `formula`, consist of basic constructions:

- 2 instances for transitive closure: one to prove closure under iteration of $X \mapsto \bigcup X$ and an auxiliary one used to show absoluteness.
- 1 instance to define $\in$-rank.
- 1 instance to construct the cumulative hierarchy (rank initial segments).

The next 4 instances (gathered in `instances2_fms`) are needed to set up cardinal arithmetic in M:

- 2 instances for the definition of ordertypes: The relevant well-founded recursion and a technical auxiliary instance.
- 2 instances for Aleph: Replacement through the ordertype function (for Hartogs' Theorem) and the well-founded recursion using it.

We also need a one extra replacement instance ψ on M for each φ of the previous ones to have them in $M[G]$:

$$\psi(x, \alpha, y_1, \ldots, y_n) := \cdot\alpha = \min\big\{\beta \mid \exists \tau \in V_\beta.\ \mathit{snd}(x) \Vdash \varphi\ [\mathit{fst}(x), \tau, y_1, \ldots, y_n]\big\}\cdot$$

Here, $\mathit{fst}(\langle a, b\rangle) = a$ and $\mathit{snd}(\langle a, b\rangle) = b$. The map $\varphi \mapsto \psi$ is the function $\mathcal{F}$ referred to in the abstract. All such "ground" replacement instances appear in the locale `M_ZF3` and form the set `instances3_fms`.

That makes 16 instances up to now. For the setup of forcing, we require the following 3 instances, which form the set `instances_ground_fms`:

- Well-founded recursion to define check-names.
- Well-founded recursion for the definition of forcing for atomic formulas.
- Replacement through $x \mapsto \langle x, \check{x}\rangle$ (for the definition of $\dot{G}$).

The proof of the Δ-System Lemma requires 2 instances which form the set `instances_ground_notCH_fms`, that are used for the recursive construction of sets using a choice function (as in the construction of a wellorder of X given a choice function on $\mathcal{P}(X)$), and to show its absoluteness.

The 21 formulas up to this point are collected into the set `overhead_notCH` (called Ω in the abstract), which is enough to force $\neg CH$. To force CH, we required one further instance for the absoluteness of the recursive construction in the proof of Dependent Choices from AC. A listing with the names of all the formulas can be found in Appendix F.

The particular choice of some of the instances above arose from Paulson's architecture on which we based our development. This applies every time a locale from *ZF-Constructible* has to be interpreted (`M_eclose` and `M_ordertype`, respectively, for the "auxiliary" instances).

On the other hand, we replaced the original proof of the Schröder-Bernstein Theorem by Zermelo's one [37, Exr. x4.27], because the former required at least one extra instance arising from an iteration. We also managed to avoid 12 further replacements by restructuring some of original theories in *ZF-Constructible*, so these modifications are included as part of our project.

It is to be noted that the proofs of the Forcing Theorems do not require any extra replacement on the ground model; actually, they only need the 7 instances appearing in `instances1_fms` and `instances_ground_fms`. But this seems not be the case for Separation, at least by inspecting our formalization: More instances holding in M are needed as the complexity of φ grows. One point where this is apparent is in the proof of Theorem 1(2), that appears as the `truth_lemma` in our development; it depends on `truth_lemma'` and `truth_lemma_Neg`, which explicitly invoke `separation_ax`. In any case, our intended grounds (v.g., the transitive collapse of countable elementary submodels of a rank initial segment V_α or an $H(\kappa)$) all satisfy full Separation.

6.2 Models for *CH* and its negation

The statements of the existence of models of $ZFC + \neg CH$ and of $ZFC + CH$ appear in our formalization as follows:

```
corollary ctm_ZFC_imp_ctm_not_CH:
  assumes
    "M ≈ ω" "Transset(M)" "M ⊨ ZFC"
  shows
    "∃N.
      M ⊆ N ∧ N ≈ ω ∧ Transset(N) ∧ N ⊨ ZFC ∪ {·¬·CH··} ∧
      (∀α. Ord(α) ⟶ (α ∈ M ⟷ α ∈ N))"
```

```
corollary ctm_ZFC_imp_ctm_CH:
  assumes
    "M ≈ ω" "Transset(M)" "M ⊨ ZFC"
  shows
    "∃N.
      M ⊆ N ∧ N ≈ ω ∧ Transset(N) ∧ N ⊨ ZFC ∪ {·CH·} ∧
      (∀α. Ord(α) ⟶ (α ∈ M ⟷ α ∈ N))"
```

where $\approx$ is equipotency, and the predicate `Transset` holds for transitive sets. Both results are proved without using Choice.

As the excerpts indicate, these results are obtained as corollaries of two theorems in which only a subset of the aforementioned replacement instances are assumed of the ground model. We begin the discussion of these stronger results by considering extensions of ctms of fragments of *ZF*.

```
theorem extensions_of_ctms:
  assumes
    "M ≈ ω" "Transset(M)"
    "M ⊨ ·Z· ∪ {·Replacement(p)· . p ∈ overhead}"
    "Φ ⊆ formula"
    "M ⊨ { ·Replacement(ground_repl_fm(φ))· . φ ∈ Φ}"
  shows
    "∃N.
      M ⊆ N ∧ N ≈ ω ∧ Transset(N) ∧ M≠N ∧
      (∀α. Ord(α) ⟶ (α ∈ M ⟷ α ∈ N)) ∧
      ((M, []⊨ ·AC·) ⟶ N, [] ⊨ ·AC·) ∧
      N ⊨ ·Z· ∪ { ·Replacement(φ)· . φ ∈ Φ}"
```

Here, the 7-element set `overhead` is enough to construct a proper extension. It is the union of `instances1_fms` and `instances_ground_fms`. Also, ·Z· denotes Zermelo set theory and one can use the parameter Φ to ensure those replacement instances in the extension.

In the next theorem, the relevant set of formulas is `overhead_notCH`, defined above in Section 6.1, and `ZC` denotes Zermelo set theory plus Choice:

```
theorem ctm_of_not_CH:
  assumes
    "M ≈ ω" "Transset(M)"
    "M ⊨ ZC ∪ {·Replacement(p)· . p ∈ overhead_notCH}"
    "Φ ⊆ formula"
    "M ⊨ { ·Replacement(ground_repl_fm(φ))· . φ ∈ Φ}"
  shows
    "∃N.
      M ⊆ N ∧ N ≈ ω ∧ Transset(N) ∧
      N ⊨ ZC ∪ {·¬·CH··} ∪ { ·Replacement(φ)· . φ ∈ Φ} ∧
      (∀α. Ord(α) ⟶ (α ∈ M ⟷ α ∈ N))"
```

Finally, `overhead_CH` is the union of `overhead_notCH` with the *DC* instance `dc_abs_fm`:

```
theorem ctm_of_CH:
  assumes
    "M ≈ ω" "Transset(M)"
    "M ⊨ ZC ∪ {·Replacement(p)· . p ∈ overhead_CH}"
    "Φ ⊆ formula"
    "M ⊨ { ·Replacement(ground_repl_fm(φ))· . φ ∈ Φ}"
  shows
    "∃N.
      M ⊆ N ∧ N ≈ ω ∧ Transset(N) ∧
      N ⊨ ZC ∪ {·CH·} ∪ { ·Replacement(φ)· . φ ∈ Φ} ∧
      (∀α. Ord(α) ⟶ (α ∈ M ⟷ α ∈ N))"
```

7 Related work

There is another formalization of forcing in Lean by Han and van Doorn, under the name *Flypitch* [25, 26]. When our project started, we were unaware of this initiative, and the same as them, we were deeply influenced by Wiedijk's list of 100 theorems [58].

Many aspects make their formalization different from ours. Their presentation of the mathematics is somewhat more elegant and cohesive, since they go for the Boolean valued approach; they also set up the calculus of first-order logic, and *en route* to forcing they formalized the basic model theory of Boolean valued models and Gödel's Completeness Theorem. They also provided the treatment of the regular open algebra, and the general version of the Delta System Lemma. Putting this together they readily obtain a proof that $ZFC \nvdash CH$ [25] and after formalizing collapse forcing they show $ZFC \nvdash \neg CH$ [26, Sect. 5.6].

It should be emphasized, however, that the Flypitch project was carried out assuming a rather strong metatheory. Carneiro [7] reports that Werner's results in [57] can be adapted to show that the base logic of Lean (restricted to n type universes) proves the consistency of *ZFC* plus n inaccessibles. Han and van Doorn did use universes in their implementation; for instance, ordinals are "defined as equivalence classes of (well-ordered) types, [. . .] one universe level higher than the types used to construct them" [25]. It is not clear to us if they are able to avoid such strength: At least, Con(ZFC) is provable in their context. On a lesser note, in order to prove AC in the generic extension, Flypitch requires choice in the metatheory [25, p. 11], while our formalization works entirely in ZF.

This is perhaps an appropriate time to insist that we have *not* formalized the relative consistency of $\neg CH$, and we are actually not aiming for that in the short term. In our context, going for the plain consistency result seems off the mark, since we can not weaken our base theory (which is essentially equivalent to ZF). Even if we intended to do so, the standard route to use ctms for a relative consistency proof (through the Reflection Theorem) is prohibitive for us, since our metatheory does not have the required induction principles (on Isabelle/ZF formulas—in contrast to formulas coded as sets, as in our presentation).

We believe that our formalization using the ctm approach over Isabelle/ZF might be more appealing to set-theorists because of the type-theoretic machinery used in Flypitch, and since absoluteness grants us extra naturality. This last point may also be illustrated by the treatment of ordinals; in our formalization, as it is expected intuitively, the following are equivalent for every x in a ctm M:

– $\texttt{Ord}(x)$;
– $\texttt{ordinal}(\texttt{\#\#}M, x)$ (the relational definition relativized to M);
– $M, [x] \models \cdot 0\ \texttt{is ordinal}\cdot$.

where $\cdot n\ \texttt{is ordinal}\cdot$ is the code for the appropriate first-order formula (0 is a de Bruijn index above!). In contrast, Han and van Doorn require an injection from the ordinals of the corresponding type universe into their encoding of a model of ZF, and a further necessary injection into the Boolean valued model using checks—this last step obviously appears in our presentation, but the *val* function used to construct $M[G]$ will turn check-names into the corresponding argument, as expected.

This faithfulness to set-theoretical practice does not come for free. Recursive constructions and inductive definitions are far easier to perform in the Calculus of Inductive Constructions on which Lean is based, and in Isabelle/ZF are rather cumbersome. Also, a typed discipline provides aid to write succinctly and many assumptions are satisfied by mere notation. To be clear, those benefits come from doing set theory in a non set-theoretical language. On the other hand, Isar proofs, as the one shown in Sec. 5, are easier to understand than the language of tactics of Lean.

A sweet spot combining the best of both worlds is to be found on developments in Isabelle/HOL based on the AFP entry *ZFC_in_HOL* by Paulson [46]. There is a range of results in combinatorics and other set-theoretical material that was swiftly formalized in this setting: Erdős-Milner partition theorem [47], Nash-Williams theorem and Larson's $\forall k\ \omega^\omega \longrightarrow (\omega^\omega, k)$ [14], Design Theory [15], and Wetzel's problem [48]; this last paper describes in a brief and clear way the convenience of the interaction with Isabelle/HOL (which is also paid in consistency-strength currency [41, Sect. 3]).

Concerning the minimum amount of Replacement needed to construct forcing extensions, only recently we learned about Mathias' work on the subject (for which a summary is offered in [29, Sect. 6]). In [33, Sect. 1], models M of Zermelo set theory are constructed for which each of the inclusions $M \subseteq M[G]$ and $M \supseteq M[G]$ fail, where the poset $\mathbb{P}$ is the trivial $\{\mathbf{1}\}$. In one of them, K, we have $\omega \in K \setminus K[G]$, hence the ordinals do not coincide.

Also, in the reference [34], a reasonably minimal fragment $\textsc{Prov}$ of *ZF* that allows to do set forcing is identified, and transitive sets satisfying it are called *provident*. Existence of rank and of transitive closure are implied by $\textsc{Prov}$; hence their appearance in our list seems justified. Nevertheless, $\textsc{Prov}$ is far weaker than the fragments of *ZF* considered here, since it is a restriction of Kripke-Platek set theory, and thus it does not include neither Powerset nor full Separation. The detailed theory of provident sets is developed in [35].

8 Some lessons

We want to finish this report by gathering some of the conclusions we reached after the experience of formalizing the basics of forcing in a proof assistant.

8.1 Aims of a formalization and planning

We believe that in every project of formalization of mathematics, there is a tension between the haste to verify the target results and the need to obtain a readable, albeit extremely detailed, corpus of statements and proofs. This tension is mirrored in two different purposes of formalization: Developing new mathematics from scratch and producing verified results on the way, versus verifying and documenting material that has already been produced on paper.

Our present project clearly belongs to the second category, so we prioritized trying to obtain formal proofs that mimicked standard prose (as can be seen in the sample proof in Section 5). We feel that the Isar language provided with Isabelle has the right balance between elegance and efficacy. Another crucial aspect to achieve this goal is the level of detail of the blueprint for the formalization. We must however confess that we learned many of the subtleties of Isabelle in the making, and many engineering decisions were also taken before it was clear the precise way things would develop in the future.

A similar experience, but on an opposite side of the formalization spectrum happened to the Liquid Tensor Experiment [8] as described by Scholze in [54]. People involved in the formalization simply pushed their way to reach the summit, formalizing lemma after lemma. They actually wrote the blueprint for that formalization *afterwards* it was complete! From time to time, we were also frenziedly trying to get the results formalized, going beyond what we had planned.

As a result from this, some design choices that seemed reasonable at first were proved to be inconvenient. For instance, we should had better used predicates (of type $\mathtt{i} \Rightarrow \mathtt{i} \Rightarrow \mathtt{o}$) for the forcing posets' order relations; this is the way they are presented in the *Delta_System_Lemma* session. A similar problem, which can be traced to our reading of Kunen's suggestion on how to formalize the forcing relation [31, p. 260], is that we require the forcing poset to be an element of M, so the present infrastructure does not allow class forcing out of the box. (The latter change seems to be rather straightforward, but the former does not.)

Nearly the final stage of the project, we decided to go for the minimal set of definitions and versions of lemmas that were needed to obtain our target results. For example,

– we only proved the Delta System Lemma for $\aleph_1$-sized families; thus limiting us to the case of the $\aleph_1$-chain condition, and avoiding the relativization of the material on cofinalities [55];
– we showed preservation of sequences by considering countably closed forcings (in fact, we formalized the bare minimum requirement of being $(<\omega{+}1)$-closed, that is, closure under ω-sequences and not δ-sequences for every countable δ).

In doing this we went against the conventional wisdom that one should formalize the most general version of the results available. Another shortcut we took was to simplify some proofs by appealing to the countability of the ground model; this is the case of `definition_of_forces` and the result on forcing values of a function.

8.2 How to believe in the formalization

This is a rather tricky question, that was addressed by Pollack in his [51]. There is little point to discuss that, after an assistant has accepted some input successfully, *some mathematics* has been formally verified. What might not be apparent is if the claimed theorems are indeed the results that have been checked. One key aspect of this is the logical foundation of the assistant (Section 2.1). But the weakest link in the chain is the laying down of definitions building up to the concepts needed to state the target results.[6]

We took care of this matter by providing, as an entry point for our whole development, the theory `Definitions_Main` in which a path from some the fundamental concepts from Isabelle/ZF reaching to our main theorems is expounded. Cross-references to major milestones (which can be navigated by using Isabelle) are provided there. A curated version can be found as Appendix A to this paper.

Frequently, we formalized material by directly typing the proof we knew by heart, and in so doing we assumed that some definitions accommodated some of our preconceptions. It is significant that in a few such occasions, we were doubly surprised by the fact that some supposedly trivial lemma would not go through, because the definitions addressed something different (think of *restriction* of a function to a set versus that of a relation), and also that we were able to prove the adjacent results. The takeaway is that intuition may drive proofs even if you are not working on what you think you are.

[6] Another related aspect, concerning the way results are printed and parsed by assistant versus their internal meaning, was studied by Wiedijk [59].

A final aspect on this topic concerns automated methods. In the Introduction we hinted at the fact that a proof can actually be *obscured* by automation. Specifically, proof steps that were solved automatically give no information for someone who wants to understand the details of the argument; by the same token, automatic methods might silently exploit inconsistencies in the definitions, and this will only be apparent in a later stage of the development.

8.3 Bureaucracy and scale factors

It is noteworthy that although the "math" of the construction of a model of $ZFC + \neg CH$ was already in place by the end of November 2020, it was only 9 months later that we were able to finish the formalization of that result. The missing pieces were essentially bureaucracy. Some of the material filed under this category comprises:

- permutation of indices and calculation of arities of internalized formulas;
- proving that certain constructions belong to the relevant models;
- (required for the above) showing that particular instances of separation and replacement hold in the ground model.

Some of those proofs were almost copy-pasted once and again with minor variants; this would usually be relegated to some function in the meta-language, but we were unable to do this due to our limitations in programming Isabelle/ML[7].

Nevertheless, experience in software engineering is invaluable in large projects like the present one. For example, it is (mathematically) misleading when automatic tools (`simp`, `auto`, etc) stop working just because of the sheer size of the goal (v.g., the same statement with 7 variables succeeds but with 8 variables does not). Scale issues are very easily disregarded in the abstract but, as a colorful example, the formula `forces`$(\cdot 0 \in 1 \cdot)$ can be explicitly printed by Isabelle2021-1 (it spans nearly 20k symbols), but `forces`$(\cdot\neg\cdot\neg\cdot 0 \in 1 \cdots)$ can not.

Another point where computer science expertise was a prime asset was the very definition of `forces`. As a proof of concept, one of us tried to obtain its definition by using formula synthesis exclusively, which was supposed to be as trivial as in the usual mathematical development (similarly to the case of Equation (2)). But in fact, some early minor mistake rendered the whole effort useless. We then turned to a more informed programming discipline, which involved decomposing the definition in stages, each of which was checked for correctness, and in that way we were able to reach our objective.

9 Future directions

There are many possibilities for further work starting from this formalization. We will mention just a few.

[7] On the other hand, our inability to automate proofs of replacement instances paved the way for identifying which were the ones needed for forcing!

Obvious missing pieces would be proving the standard properties of general Cohen posets $\mathrm{Fn}_\kappa(I, J)$, and to modify the core definitions to allow for class forcing. We would also like to try the Boolean valued approach to compare the (un)ease of formalization using Isabelle/ZF.

Another desirable goal is to construct transitive set models of *ZFC* from a large cardinal. There is some work to be done for that: Even the definition of inaccessibles, and of the transitive collapse (for that matter), are still missing.

As we did for Replacement, we would like to pinpoint an (almost) minimal set of instances of Separation needed to use forcing. A necessary ingredient will certainly be an implementation of Gödel operations [28, Thm. 13.4].

In our previous landmark [21], we contributed with some modifications to *ZF-Constructible*; this is now part of the official Isabelle distribution. We intend ask Isabelle maintainers to consider the more modular versions of some of those theories that we are presenting in this project.

As final words about our journey, we believe that, as in mathematics in general, the experience of working in a formal environment can be daunting, but at the same time extremely rewarding: The feeling of accomplishment after seeing your own writings validated beyond doubt is in some ways comparable to that of finding a proof of an important lemma. It also allows subtly different ways of reasoning (with their own merits and pitfalls—it is easy to forget how easy a proof on paper is once you are fully engaged in directing your assistant). We hope that at some point these experiences are shared by our community at large.

Acknowledgments

We are indebted to Timothy Chow for insightful comments on a draft of this paper. We also thank Z. Vidnyánszky for some discussions where the idea of building the summary in `Definitions_Main` took shape. Mikhail Mandrykin helped us with technical assistance regarding the rendering of formulas in the Isabelle/jEdit interface, which improved our presentation of this material during the Latin American Congress of Mathematicians (CLAM 2021); we extend our gratitude to the rest of the Isabelle community, for their kind support through the Users mailing list. We also want to warmly thank Larry Paulson for his work and encouragement. Last, and definitely not least, we are deeply grateful to Ken Kunen for his inspiring expositions. This formalization is a tiny homage to his memory.

Bibliography

[1] Agda Development Team, “Agda”, version 2.6.2 (2022). https://agda.readthedocs.io/en/v2.6.2.1/index.html.

[2] J. Avigad, Foundations, *arXiv e-prints* **2009.09541** (2020). For the forthcoming Handbook of Proof Assistants and Their Applications in Mathematics and Computer Science, edited by Jasmin Blanchette and Assia Mahboubi.

[3] J. Bayer, C. Benzmüller, K. Buzzard, M. David, L. Lamport, Y. Matiyasevich, L. Paulson, D. Schleicher, B. Stock, E. Zelmanov, Mathematical proof between generations, *arXiv e-prints* **2207.04779** (2022).

[4] N.G. de Bruijn, Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem, *Nederl. Akad. Wetensch. Proc. Ser. A* **75**=*Indag. Math.* **34**: 381–392 (1972).

[5] K. Buzzard, What is the point of computers? A question for pure mathematicians, *arXiv e-prints* **2112.11598** (2021).

[6] K. Buzzard, J. Commelin, P. Massot, Formalising perfectoid spaces, in: Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2020, Association for Computing Machinery, New York, NY, USA: 29–312 (2020).

[7] M. Carneiro, “The Type Theory of Lean”, Master’s thesis, Carnegie Mellon University (2019).

[8] D. Castelvecchi, Mathematicians welcome computer-assisted proof in ‘grand unification’ theory, *Nature* **595**: 18–19 (2021).

[9] P.J. Cohen, The discovery of forcing, *Rocky Mt. J. Math.* **32**: 1071–1100 (2002).

[10] Coq Development Team, “The Coq Proof Assistant”, version 8.7.0 (2017). https://doi.org/10.5281/zenodo.1028037.

[11] T. Coquand, G.P. Huet, The calculus of constructions, *Inf. Comput.* **76**: 95–120 (1988).

[12] J. Davenport, B. Poonen, J. Maynard, H. Helfgott, P.H. Tiep, L. Cruz-Filipe, Machine-assisted proofs (ICM 2018 Panel), (2018).

[13] A. De Lon, P. Koepke, A. Lorenzen, A. Marti, M. Schütz, E. Sturzenhecker, Beautiful formalizations in isabelle/naproche, in: F. Kamareddine, C. Sacerdoti Coen (Eds.), Intelligent Computer Mathematics: 14th International Conference, CICM 2021, Timisoara, Romania, July 26–31, 2021, Proceedings, Springer International Publishing, Cham: 19–31 (2021).

[14] M. Džamonja, A. Koutsoukou-Argyraki, L.C. Paulson, Formalizing ordinal partition relations using isabelle/hol, *Experimental Mathematics* **0**: 1–18 (2021).

[15] C. Edmonds, L.C. Paulson, A modular first formalisation of combinatorial design theory, in: F. Kamareddine, C. Sacerdoti Coen (Eds.), Intelligent Computer Mathematics: 14th International Conference, CICM 2021, Timisoara, Romania, July 26–31, 2021, Proceedings, Springer-Verlag, Cham: 3–18 (2021).

[16] G. Gonthier, Formal proof—the four-color theorem, *Notices Amer. Math. Soc.* **55**: 1382–1393 (2008).

[17] G. Gonthier, A. Asperti, J. Avigad, Y. Bertot, C. Cohen, F. Garillot, S. Le Roux, A. Mahboubi, R. O’Connor, S.O. Biha, I. Pasca, L. Rideau, A. Solovyev, E. Tassi, L. Théry, A machine-checked proof of the odd order theorem, in: Proceedings of the 4th International Conference on Interactive Theorem Proving, ITP’13, Springer-Verlag, Berlin, Heidelberg: 163–179 (2013).

[18] M. Gordon, R. Milner, C.P. Wadsworth, "Edinburgh LCF", Lecture notes in computer science, Springer, Berlin, Germany (1979), 1979 edition.

[19] E. Gunther, M. Pagano, P. Sánchez Terraf, First steps towards a formalization of forcing, in: B. Accattoli, C. Olarte (Eds.), Proceedings of the 13th Workshop on Logical and Semantic Frameworks with Applications, LSFA 2018, Fortaleza, Brazil, September 26-28, 2018, Electronic Notes in Theoretical Computer Science **344**, Elsevier: 119–136 (2018).

[20] E. Gunther, M. Pagano, P. Sánchez Terraf, Mechanization of Separation in Generic Extensions, *arXiv e-prints* **1901.03313** (2019).

[21] E. Gunther, M. Pagano, P. Sánchez Terraf, Formalization of Forcing in Isabelle/ZF, in: N. Peltier, V. Sofronie-Stokkermans (Eds.), Automated Reasoning. 10th International Joint Conference, IJCAR 2020, Paris, France, July 1–4, 2020, Proceedings, Part II, Lecture Notes in Artificial Intelligence **12167**, Springer International Publishing: 221–235 (2020).

[22] E. Gunther, M. Pagano, P. Sánchez Terraf, M. Steinberg, The independence of the continuum hypothesis in Isabelle/ZF, *Archive of Formal Proofs* (2022). `https://isa-afp.org/entries/Independence_CH.html`, Formal proof development.

[23] E. Gunther, M. Pagano, P. Sánchez Terraf, M. Steinberg, Transitive models of fragments of ZFC, *Archive of Formal Proofs* (2022). `https://isa-afp.org/entries/Transitive_Models.html`, Formal proof development.

[24] T. Hales, M. Adams, G. Bauer, T.D. Dang, J. Harrison, L.T. Hoang, C. Kaliszyk, V. Magron, S. Mclaughlin, T.T. Nguyen, et al., A formal proof of the Kepler conjecture, *Forum Math. Pi* **5**: e2, 29 (2017).

[25] J.M. Han, F. van Doorn, A formalization of forcing and the unprovability of the continuum hypothesis, in: J. Harrison, J. O'Leary, A. Tolmach (Eds.), 10th International Conference on Interactive Theorem Proving (ITP 2019), Leibniz International Proceedings in Informatics (LIPIcs) **141**, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany: 19:1–19:19 (2019).

[26] J.M. Han, F. van Doorn, A formal proof of the independence of the continuum hypothesis, in: J. Blanchette, C. Hritcu (Eds.), Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2020, New Orleans, LA, USA, January 20-21, 2020, ACM (2020).

[27] J. Harrison, J. Urban, F. Wiedijk, History of interactive theorem proving, in: J.H. Siekmann (Ed.), Computational Logic, Handbook of the History of Logic **9**, pp. 135–214, Elsevier: 135–214 (2014).

[28] T. Jech, "Set Theory. The Millennium Edition", Springer Monographs in Mathematics, Springer-Verlag (2002), third edition. Corrected fourth printing, 2006.

[29] A. Kanamori, Mathias and set theory, *Mathematical Logic Quarterly* **62**: 278–294 (2016).

[30] A. Koutsoukou-Argyraki, History of interactive theorem proving, in: B. Löwe, D. Sarikaya (Eds.), 60 Jahre DVMLG (Special Issue for the 60th Anniversary of the DVMLG), Tributes, College Publications (2022).

[31] K. Kunen, "Set Theory", Studies in Logic, College Publications (2011), second edition. Revised edition, 2013.

[32] P. Martin-Löf, "Intuitionistic type theory", Studies in proof theory **1**, Bibliopolis (1984).

[33] A.R.D. Mathias, Set forcing over models of Zermelo or Mac Lane, in: Libert, Thierry, Hinnion, Roland (Eds.), One hundred years of axiomatic set theory, Cahiers Centre Logique **17**, Acad.-Bruylant, Louvain-la-Neuve, Bruxelles, Belgium: 41–66 (2008).

[34] A.R.D. Mathias, Provident sets and rudimentary set forcing, *Fundamenta Mathematicae* **230**: 99–148 (2015).
[35] A.R.D. Mathias, N.J. Bowler, Rudimentary recursion, gentle functions and provident sets, *Notre Dame Journal of Formal Logic* **56**: 3–60 (2015).
[36] Mizar, "Mizar", version 8.1.11 (2022). http://mizar.uwb.edu.pl/.
[37] Y.N. Moschovakis, "Notes on Set Theory", Springer Texts in Electrical Engineering, Springer-Verlag (1994).
[38] L. de Moura, S. Ullrich, The Lean 4 theorem prover and programming language, in: A. Platzer, G. Sutcliffe (Eds.), Automated Deduction - CADE 28 - 28th International Conference on Automated Deduction, Virtual Event, July 12-15, 2021, Proceedings, Lecture Notes in Computer Science **12699**, Springer: 625–635 (2021).
[39] T. Nipkow, Order-sorted polymorphism in Isabelle, in: G. Huet, G. Plotkin (Eds.), Logical Environments, Cambridge University Press: 164–188 (1993).
[40] T. Nipkow, S. Rosskopf, Isabelle's metalogic: Formalization and proof checker, in: A. Platzer, G. Sutcliffe (Eds.), Automated Deduction – CADE 28, Springer International Publishing, Cham: 93–110 (2021).
[41] S. Obua, Partizan games in Isabelle/HOLZF, in: K. Barkaoui, A. Cavalcanti, A. Cerone (Eds.), Theoretical Aspects of Computing - ICTAC 2006, Third International Colloquium, Tunis, Tunisia, November 20-24, 2006, Proceedings, Lecture Notes in Computer Science **4281**, Springer: 272–286 (2006).
[42] L.C. Paulson, The foundation of a generic theorem prover, *Journal of Automated Reasoning* **5**: 363–397 (1989).
[43] L.C. Paulson, "Isabelle - A Generic Theorem Prover (with a contribution by T. Nipkow)", Lecture Notes in Computer Science **828**, Springer (1994).
[44] L.C. Paulson, Set theory for verification. II: Induction and recursion, *Journal of Automated Reasoning* **15**: 167–215 (1995).
[45] L.C. Paulson, The relative consistency of the axiom of choice mechanized using Isabelle/ZF, *LMS Journal of Computation and Mathematics* **6**: 198–248 (2003).
[46] L.C. Paulson, Zermelo Fraenkel set theory in higher-order logic, *Archive of Formal Proofs* (2019). http://isa-afp.org/entries/ZFC_in_HOL.html, Formal proof development.
[47] L.C. Paulson, A formalised theorem in the partition calculus, *Annals of Pure and Applied Logic* (2021). In press. Available at https://doi.org/10.17863/CAM.88991 and https://arxiv.org/abs/2104.11613.
[48] L.C. Paulson, Wetzel: Formalisation of an undecidable problem linked to the Continuum Hypothesis, in: K. Buzzard, T. Kutsia (Eds.), Intelligent Computer Mathematics, Springer International Publishing, Cham: 92–106 (2022).
[49] L.C. Paulson, K. Grabczewski, Mechanizing set theory, *J. Autom. Reasoning* **17**: 291–323 (1996).
[50] F. Pfenning, C. Paulin-Mohring, Inductively defined types in the calculus of constructions, in: M.G. Main, A. Melton, M.W. Mislove, D.A. Schmidt (Eds.), Mathematical Foundations of Programming Semantics, 5th International Conference, Tulane University, New Orleans, Louisiana, USA, March 29 - April 1, 1989, Proceedings, Lecture Notes in Computer Science **442**, Springer: 209–228 (1989).
[51] R. Pollack, How to believe a machine-checked proof, in: Twenty-five years of constructive type theory (Venice, 1995), Oxford Logic Guides **36**, pp. 205–220, Oxford Univ. Press, New York: 205–220 (1998).
[52] D. Prawitz, Ideas and results in proof theory, in: Proceedings of the Second Scandinavian Logic Symposium (Univ. Oslo, Oslo, 1970), Studies in Logic and the Foundations of Mathematics **63**, pp. 235–307 (1971).

[53] P. Scholze, Liquid Tensor Experiment, Xena Project blog post, (2020). https://xenaproject.wordpress.com/2020/12/05/liquid-tensor-experiment/.
[54] P. Scholze, Half a year of the Liquid Tensor Experiment: Amazing developments, Xena Project blog post, (2021). https://xenaproject.wordpress.com/2021/06/05/half-a-year-of-the-liquid-tensor-experiment-amazing-developments/.
[55] P. Sánchez Terraf, Cofinality and the delta system lemma, *Archive of Formal Proofs* (2020). https://isa-afp.org/entries/Delta_System_Lemma.html, Formal proof development.
[56] M. Wenzel, Isar - A generic interpretative approach to readable formal proof documents, in: Y. Bertot, G. Dowek, A. Hirschowitz, C. Paulin-Mohring, L. Théry (Eds.), Theorem Proving in Higher Order Logics, 12th International Conference, TPHOLs'99, Nice, France, September, 1999, Proceedings, Lecture Notes in Computer Science **1690**, Springer: 167–184 (1999).
[57] B. Werner, Sets in types, types in sets, in: Proceedings of the Third International Symposium on Theoretical Aspects of Computer Software, TACS '97, Springer-Verlag, Berlin, Heidelberg: 530–346 (1997).
[58] F. Wiedijk, Formalizing 100 theorems, Web page, (2005). https://www.cs.ru.nl/~freek/100/ (retrieved 29 May 2022).
[59] F. Wiedijk, Pollack-inconsistency, in: Proceedings of the 9th international workshop on user interfaces for theorem provers (UITP10), Edinburgh, UK, July 15, 2010, pp. 85–100, Amsterdam: Elsevier: 85–100 (2012).

A Main definitions of the formalization

This section, which appears almost verbatim as the theory `Definitions_Main` in [22], might be considered as the bare minimum reading requisite to trust that our development indeed formalizes the theory of forcing.

The reader trusting all the libraries on which our development is based, might jump directly to Section A.3, which treats relative cardinal arithmetic as implemented in `Transitive_Models`. But in case one wants to dive deeper, the following sections treat some basic concepts of the ZF logic (Section A.1) and in the *ZF-Constructible* library (Section A.2) on which our definitions are built.

A.1 ZF

For the basic logic ZF we restrict ourselves to just a few concepts (for its axioms, consult Appendix D).

```
bij(A, B) ≡
{f ∈ A → B . ∀w∈A. ∀x∈A. f ‘ w = f ‘ x ⟶ w = x} ∩
{f ∈ A → B . ∀y∈B. ∃x∈A. f ‘ x = y}

A ≈ B ≡ ∃f. f ∈ bij(A, B)

Transset(i) ≡ ∀x∈i. x ⊆ i

Ord(i) ≡ Transset(i) ∧ (∀x∈i. Transset(x))
```

```
i < j ≡ i ∈ j ∧ Ord(j)
i ≤ j ⟷ i < j ∨ (i = j ∧ Ord(j))
```

With the concepts of empty set and successor in place,

```
lemma empty_def': "∀x. x ∉ 0"
lemma succ_def': "succ(i) = i ∪ {i}"
```

we can define the set of natural numbers ω. In the sources, it is defined as a fixpoint, but here we just write its characterization as the first limit ordinal.

```
Ord(ω) ∧ 0 < ω ∧ (∀y. y < ω ⟶ succ(y) < ω)
Ord(i) ∧ 0 < i ∧ (∀y. y < i ⟶ succ(y) < i) ⟹ ω ≤ i
```

Then, addition and predecessor on ω are inductively characterized as follows:

```
m +ω succ(n) = succ(m +ω n)
m ∈ ω ⟹ m +ω 0 = m

pred(0) = 0
pred(succ(y)) = y
```

Lists on a set `A` can be characterized by being recursively generated from the empty list `[]` and the operation `Cons` that adds a new element to the left end; the induction theorem for them shows that the characterization is “complete”. (Mind the `⟦P; Q⟧ ⟹ R` abbreviation for `P ⟹ Q ⟹ R`.)

```
[] ∈ list(A)
⟦a ∈ A; l ∈ list(A)⟧ ⟹ Cons(a, l) ∈ list(A)

⟦x ∈ list(A); P([]); ⋀a l. ⟦a ∈ A; l ∈ list(A); P(l)⟧ ⟹
  P(Cons(a, l))⟧ ⟹ P(x)
```

Length, concatenation, and `nth` element of lists are recursively characterized as follows.

```
length([]) = 0
length(Cons(a, l)) = succ(length(l))

[] @ ys = ys
Cons(a, l) @ ys = Cons(a, l @ ys)

nth(0, Cons(a, l)) = a
n ∈ ω ⟹ nth(succ(n), Cons(a, l)) = nth(n, l)
```

We have the usual Haskell-like notation for iterated applications of `Cons`:

```
lemma Cons_app: "[a,b,c] = Cons(a,Cons(b,Cons(c,[])))"
```

Relative quantifiers restrict the range of the bound variable to a class `M` of type `i ⇒ o`; that is, a truth-valued function with set arguments.

```
lemma "∀x[M]. P(x) ≡ ∀x. M(x) ⟶ P(x)"
      "∃x[M]. P(x) ≡ ∃x. M(x) ∧ P(x)"
```

Finally, a set can be viewed ("cast") as a class using the following function of type `i ⇒ i ⇒ o`.

```
(##A)(x) ⟷ x ∈ A
```

A.2 Relative concepts

A list of relative concepts (mostly from the *ZF-Constructible* library) follows next.

```
big_union(M, A, z) ≡ ∀x[M]. x ∈ z ⟷ (∃y[M]. y ∈ A ∧ x ∈ y)

upair(M, a, b, z) ≡ a ∈ z ∧ b ∈ z ∧ (∀x[M]. x ∈ z ⟶ x = a ∨ x = b)

pair(M, a, b, z) ≡ ∃x[M]. upair(M, a, a, x) ∧
                     (∃y[M]. upair(M, a, b, y) ∧ upair(M, x, y, z))

successor(M, a, z) ≡
∃x[M]. upair(M, a, a, x) ∧ (∀xa[M]. xa ∈ z ⟷ xa ∈ x ∨ xa ∈ a)

empty(M, z) ≡ ∀x[M]. x ∉ z

transitive_set(M, a) ≡ ∀x[M]. x ∈ a ⟶ (∀xa[M]. xa ∈ x ⟶ xa ∈ a)

ordinal(M, a) ≡
transitive_set(M, a) ∧ (∀x[M]. x ∈ a ⟶ transitive_set(M, x))

image(M, r, A, z) ≡
∀y[M]. y ∈ z ⟷ (∃w[M]. w ∈ r ∧ (∃x[M]. x ∈ A ∧ pair(M, x, y, w)))

is_apply(M, f, x, y) ≡
∃xs[M].
   ∃fxs[M]. upair(M, x, x, xs) ∧ image(M, f, xs, fxs) ∧
     big_union(M, fxs, y)

is_function(M, r) ≡
∀x[M].
   ∀y[M].
      ∀y'[M].
         ∀p[M].
            ∀p'[M].
               pair(M, x, y, p) ⟶
               pair(M, x, y', p') ⟶ p ∈ r ⟶ p' ∈ r ⟶ y = y'

is_relation(M, r) ≡ ∀z[M]. z ∈ r ⟶ (∃x[M]. ∃y[M]. pair(M, x, y, z))

is_domain(M, r, z) ≡
∀x[M]. x ∈ z ⟷ (∃w[M]. w ∈ r ∧ (∃y[M]. pair(M, x, y, w)))
```

```
typed_function(M, A, B, r) ≡
is_function(M, r) ∧
is_relation(M, r) ∧
is_domain(M, r, A) ∧
(∀u[M]. u ∈ r ⟶ (∀x[M]. ∀y[M]. pair(M, x, y, u) ⟶ y ∈ B))
```

```
is_function_space(M, A, B, fs) ≡
M(fs) ∧ (∀f[M]. f ∈ fs ⟷ typed_function(M, A, B, f))
```

```
A →^M B ≡ THE d. is_function_space(M, A, B, d)
```

```
surjection(M, A, B, f) ≡
typed_function(M, A, B, f) ∧
(∀y[M]. y ∈ B ⟶ (∃x[M]. x ∈ A ∧ is_apply(M, f, x, y)))
```

Relative version of the *ZFC* axioms

```
extensionality(M) ≡ ∀x[M]. ∀y[M]. (∀z[M]. z ∈ x ⟷ z ∈ y) ⟶ x = y
```

```
foundation_ax(M) ≡
∀x[M]. (∃y[M]. y ∈ x) ⟶ (∃y[M]. y ∈ x ∧ ¬ (∃z[M]. z ∈ x ∧ z ∈ y))
```

```
upair_ax(M) ≡ ∀x[M]. ∀y[M]. ∃z[M]. upair(M, x, y, z)
```

```
Union_ax(M) ≡ ∀x[M]. ∃z[M]. big_union(M, x, z)
```

```
power_ax(M) ≡ ∀x[M]. ∃z[M]. ∀xa[M]. xa ∈ z ⟷
                (∀xb[M]. xb ∈ xa ⟶ xb ∈ x)
```

```
infinity_ax(M) ≡
∃I[M].
   (∃z[M]. empty(M, z) ∧ z ∈ I) ∧
   (∀y[M]. y ∈ I ⟶ (∃sy[M]. successor(M, y, sy) ∧ sy ∈ I))
```

```
choice_ax(M) ≡ ∀x[M]. ∃a[M]. ∃f[M]. ordinal(M, a) ∧
                 surjection(M, a, x, f)
```

```
separation(M, P) ≡ ∀z[M]. ∃y[M]. ∀x[M]. x ∈ y ⟷ x ∈ z ∧ P(x)
```

```
univalent(M, A, P) ≡
∀x[M]. x ∈ A ⟶ (∀y[M]. ∀z[M]. P(x, y) ∧ P(x, z) ⟶ y = z)
```

```
strong_replacement(M, P) ≡
∀A[M].
   univalent(M, A, P) ⟶ (∃Y[M]. ∀b[M]. b ∈ Y ⟷
      (∃x[M]. x ∈ A ∧ P(x, b)))
```

Internalized formulas "Codes" for formulas (as sets) are constructed from natural numbers using `Member`, `Equal`, `Nand`, and `Forall`.

```
⟦x ∈ ω; y ∈ ω⟧ ⟹ ·x ∈ y· ∈ formula
⟦x ∈ ω; y ∈ ω⟧ ⟹ ·x = y· ∈ formula
⟦p ∈ formula; q ∈ formula⟧ ⟹ ·¬(p ∧ q)· ∈ formula
p ∈ formula ⟹ (·∀p·) ∈ formula

⟦x ∈ formula; ⋀x y. ⟦x ∈ ω; y ∈ ω⟧ ⟹ P(·x ∈ y·);
 ⋀x y. ⟦x ∈ ω; y ∈ ω⟧ ⟹ P(·x = y·);
 ⋀p q. ⟦p ∈ formula; P(p); q ∈ formula; P(q)⟧ ⟹ P(·¬(p ∧ q)·);
 ⋀p. ⟦p ∈ formula; P(p)⟧ ⟹ P((·∀p·))⟧
⟹ P(x)
```

Definitions for the other connectives and the internal existential quantifier are also provided. For instance, negation:

```
·¬p· ≡ ·¬(p ∧ p)·
```

The `arity` function strictly bounding the free de Bruijn indices of a formula is defined below:

```
arity(·x ∈ y·) = succ(x) ∪ succ(y)
arity(·x = y·) = succ(x) ∪ succ(y)
arity(·¬(p ∧ q)·) = arity(p) ∪ arity(q)
arity((·∀p·)) = pred(arity(p))
```

We have the satisfaction relation between ∈-models and first order formulas (given a "environment" list representing the assignment of free variables),

```
⟦nth(i, env) = x; nth(j, env) = y; env ∈ list(A)⟧
⟹ x ∈ y ⟷ A, env ⊨ ·i ∈ j·

⟦nth(i, env) = x; nth(j, env) = y; env ∈ list(A)⟧
⟹ x = y ⟷ A, env ⊨ ·i = j·

env ∈ list(A) ⟹ (A, env ⊨ ·¬(p ∧ q)·) ⟷ ¬ ((A, env ⊨ p) ∧
  (A, env ⊨ q))

env ∈ list(A) ⟹ (A, env ⊨ (·∀p·)) ⟷ (∀x∈A. A, Cons(x, env) ⊨ p)
```

as well as the satisfaction of an arbitrary set of sentences.

```
A ⊨ Φ ≡ ∀φ∈Φ. A, [] ⊨ φ
```

The internalized (viz. as elements of the set `formula`) versions of the axioms are checked next against the relative statements.

```
Union_ax(##A) ⟷ A, [] ⊨ ·Union Ax·
power_ax(##A) ⟷ A, [] ⊨ ·Powerset Ax·
upair_ax(##A) ⟷ A, [] ⊨ ·Pairing·
foundation_ax(##A) ⟷ A, [] ⊨ ·Foundation·
```

```
extensionality(##A) ⟷ A, [] ⊨ ·Extensionality·
infinity_ax(##A) ⟷ A, [] ⊨ ·Infinity·

φ ∈ formula ⟹
(M, [] ⊨ ·Separation(φ)·) ⟷
(∀ env∈list(M).
    arity(φ) ≤ 1 +ω length(env) ⟶
    separation(##M, λx. M, [x] @ env ⊨ φ))

φ ∈ formula ⟹
(M, [] ⊨ ·Replacement(φ)·) ⟷ (∀ env. replacement_assm(M, env, φ))

choice_ax(##A) ⟷ A, [] ⊨ ·AC·
```

Finally, the axiom sets are defined as follows.

```
ZF_fin ≡
{·Extensionality·, ·Foundation·, ·Pairing·, ·Union Ax·, ·Infinity·,
 ·Powerset Ax·}

ZF_schemes ≡
{·Separation(p)· . p ∈ formula} ∪ {·Replacement(p)· . p ∈ formula}

·Z· ≡ ZF_fin ∪ {·Separation(p)· . p ∈ formula}
ZC ≡ ·Z· ∪ {·AC·}
ZF ≡ ZF_schemes ∪ ZF_fin
ZFC ≡ ZF ∪ {·AC·}
```

A.3 Relativization of infinitary arithmetic

In order to state the defining property of the relative equipotency relation, we work under the assumptions of the locale M_cardinals. They comprise a finite set of instances of Separation and Replacement to prove closure properties of the transitive class M.

```
lemma (in M_cardinals) eqpoll_def':
  assumes "M(A)" "M(B)" shows "A ≈^M B ⟷ (∃f[M]. f ∈ bij(A,B))"
```

Below, μ denotes the minimum operator on the ordinals.

```
lemma cardinalities_defs:
  fixes M::"i⇒o"
  shows
    "|A|^M ≡ μ i. M(i) ∧ i ≈^M A"
    "Card^M(α) ≡ α = |α|^M"
    "κ^↑ν,M ≡ |ν →^M κ|^M"
    "(κ⁺)^M ≡ μ x. M(x) ∧ Card^M(x) ∧ κ < x"
```

Analogous to the previous Lemma eqpoll_def', the next lemma holds under the assumptions of the locale M_aleph. The axiom instances included are sufficient to state and prove the defining properties of the relativized Aleph function (in particular, the required ability to perform transfinite recursions).

```
context M_aleph
begin

ℵ₀^M = ω
⟦Ord(α); M(α)⟧ ⟹ ℵ_succ(α)^M = (ℵ_α^M+)^M
⟦Limit(α); M(α)⟧ ⟹ ℵ_α^M = (⋃j∈α. ℵ_j^M)

end — M_aleph

lemma ContHyp_rel_def':
  fixes N::"i⇒o"
  shows
    "CH^N ≡ ℵ₁^N = 2^↑ℵ₀^N,N"
```

Under appropriate hypotheses (this time, from the locale `M_ZF_library`), $\texttt{CH}^M$ is equivalent to its fully relational version `is_ContHyp`. As a sanity check, we see that if the transitive class is indeed $\mathcal{V}$, we recover the original *CH*.

```
M_ZF_library(M) ⟹ is_ContHyp(M) ⟷ CH^M
is_ContHyp(𝒱) ⟷ ℵ₁ = 2^↑ℵ₀
```

In turn, the fully relational version evaluated on a nonempty transitive `A` is equivalent to the satisfaction of the first-order formula `·CH·` (since it actually is a sentence, it does not depend on `env`, which appears only because the definition of $\models$ requires that argument).

```
⟦env ∈ list(A); 0 ∈ A⟧ ⟹ is_ContHyp(##A) ⟷ A, env ⊨ ·CH·
```

B Discipline for relativization

As we said in Sec. 3, in order to force *CH* and its negation we depended on having relativized versions of cardinals, Alephs, etc. It was clear for us that our efforts would be more efficient if we set up a discipline for relativizing sets (terms of type `i`) and predicates/relations (terms of type `o`).

Paulson only had, for each set, the relational version. It seemed clearer to us to have a functional version of the relativized concept. Going back to our example in 3.1, for the concept `cardinal` :: `i` ⇒ `i` we want its relative version `cardinal_rel` :: (`i` ⇒ `o`) ⇒ `i` ⇒ `i` and the relational version of the latter `is_cardinal` :: (`i` ⇒ `o`) ⇒ `i` ⇒ `i` ⇒ `o`.

Our first attempt of defining a discipline was inspired by mathematical considerations: if we might prove that `is_cardinal` is functional and also prove the existence of a witness `c` such that `M(c)` and `is_cardinal(M,x,c)` then `cardinal_rel(M,x)` can be obtained by the operator of definite descriptions.

Soon we realized that resorting to definite descriptions was needed only for the most primitive concepts. In fact, once we have a relativized concept, we can use it to define other relativizations. For instance, `cardinal_rel` depends on having relative versions of `bij`. Instead of relationalizing `bij` to get `is_bij` and then prove uniqueness and existence of a witness, we define `bij_rel` using `inj_rel` and `surj_rel`.

C Recursions in cofinality

As we mentioned near the end of Section 8.1, we decided to minimize the requirements being formalized in order to achieve our immediate goal. In particular, the treatment of cofinality in the companion project [55] was left behind.

We already observed that well-founded, and in particular transfinite, recursion is not easily dealt with in Isabelle/ZF. Nevertheless, and mainly as a curiosity, we found out that only one recursive construction is needed for the development of the basic theory of cofinality (as in [31, Sect. I.13]), which is used in the proof of the following "factorization" lemma:

Lemma 1. *Let $\delta, \gamma \in \mathsf{Ord}$ and assume $f : \delta \to \gamma$ is cofinal. There exists a strictly increasing $g : \mathrm{cf}(\gamma) \to \delta$ such that $f \circ g$ is strictly increasing and cofinal in γ. Moreover, if f is strictly increasing, then g must also be cofinal.*

It turns out that the rest of the basic results on cofinality (namely, idempotence of cf, that regular ordinals are cardinals, the cofinality of Alephs, König's Theorem) follow easily from the previous Lemma by "algebraic" reasoning only.

We therefore expect that the relativization of these results be straightforward, when time permits.

D Axioms of Isabelle/ZF

In this appendix we list the complete set of axioms of Isabelle's metatheory and logic.

D.1 The metatheory Pure

```
Pure.abstract_rule: (⋀x. ?f(x) ≡ ?g(x)) ⟹ λx. ?f(x) ≡ λx. ?g(x)
Pure.combination: ?f ≡ ?g ⟹ ?x ≡ ?y ⟹ ?f(?x) ≡ ?g(?y)
Pure.equal_elim: PROP ?A ≡ PROP ?B ⟹ PROP ?A ⟹ PROP ?B
Pure.equal_intr: (PROP ?A ⟹ PROP ?B) ⟹ (PROP ?B ⟹ PROP ?A) ⟹
        PROP ?A ≡ PROP ?B
Pure.reflexive: ?x ≡ ?x
Pure.symmetric: ?x ≡ ?y ⟹ ?y ≡ ?x
Pure.transitive: ?x ≡ ?y ⟹ ?y ≡ ?z ⟹ ?x ≡ ?z
```

D.2 IFOL and FOL

In the axioms `refl, subst, allI, spec, exE, exI,eq_reflection` there is a constraint for the *type of* the variables `a, b, x` to be in the class `term_class`.

```
IFOL.FalseE: ⋀P. False ⟹ P
IFOL.refl: (⋀a. a = a)
IFOL.subst: (⋀a b P. a = b ⟹ P(a) ⟹ P(b))
IFOL.allI: (⋀P. (⋀x. P(x)) ⟹ ∀x. P(x))
IFOL.spec: (⋀P x. ∀x. P(x) ⟹ P(x))
IFOL.exE: (⋀P R. ∃x. P(x) ⟹ (⋀x. P(x) ⟹ R) ⟹ R)
IFOL.exI: (⋀P x. P(x) ⟹ ∃x. P(x))
IFOL.conjI: ⋀P Q. P ⟹ Q ⟹ P ∧ Q
IFOL.conjunct1: ⋀P Q. P ∧ Q ⟹ P
IFOL.conjunct2: ⋀P Q. P ∧ Q ⟹ Q
IFOL.disjE: ⋀P Q R. P ∨ Q ⟹ (P ⟹ R) ⟹ (Q ⟹ R) ⟹ R
IFOL.disjI1: ⋀P Q. P ⟹ P ∨ Q
IFOL.disjI2: ⋀P Q. Q ⟹ P ∨ Q
IFOL.eq_reflection: (⋀x y. x = y ⟹ x ≡ y)
IFOL.iff_reflection: ⋀P Q. P ↔ Q ⟹ P ≡ Q
IFOL.impI: ⋀P Q. (P ⟹ Q) ⟹ P ⟶ Q
IFOL.mp: ⋀P Q. P ⟶ Q ⟹ P ⟹ Q
FOL.classical: ⋀P. (¬ P ⟹ P) ⟹ P
```

D.3 ZF_Base

The following symbols are introduced in this theory:

```
axiomatization
    mem :: "[i, i] ⇒ o" (infixl <∈> 50) — membership relation
and zero :: "i" (<0>) — the empty set
and Pow :: "i ⇒ i" — power sets
and Inf :: "i" — infinite set
and Union :: "i ⇒ i" (<⋃_> [90] 90)
and PrimReplace :: "[i, [i, i] ⇒ o] ⇒ i"
```

After the definitions of ∉, ⊆, `succ`, and relative quantifications are presented, the following axioms are postulated:

```
ZF_Base.Pow_iff: ⋀A B. A ∈ Pow(B) ↔ A ⊆ B
ZF_Base.Union_iff: ⋀A C. A ∈ ⋃C ↔ (∃B∈C. A ∈ B)
ZF_Base.extension: ⋀A B. A = B ↔ A ⊆ B ∧ B ⊆ A
ZF_Base.foundation: ⋀A. A = 0 ∨ (∃x∈A. ∀y∈x. y ∉ A)
ZF_Base.infinity: 0 ∈ Inf ∧ (∀y∈Inf. succ(y) ∈ Inf)
ZF_Base.replacement: ⋀A P b. ∀x∈A. ∀y z. P(x, y) ∧ P(x, z) ⟶ y = z
         ⟹ b ∈ PrimReplace(A, P) ↔ (∃x∈A. P(x, b))
```

D.4 AC

The theory `AC` is only imported in the theory `Absolute_Versions`; none of the main results depends on AC. The latter theory shows that some absolute results can be obtained from the relativized versions on $\mathcal{V}$.

```
AC.AC: ⋀a A B. a ∈ A ⟹ (⋀x. x ∈ A ⟹ ∃y. y ∈ B(x)) ⟹
      ∃z. z ∈ Pi(A, B)
```

E Lambda replacements

The development of the locale structure of the project was a dynamical process. As further properties of closure of the ground M were required, we gathered the relevant instances of Separation and Replacement into a new locale (always assuming a class model, for added generality), and proceeded to apply them to those closure proofs.

This procedure lead to a steady grow in the number of interpretation obligations and therefore, of formula synthesis (since the two axiom schemes were postulated using codes for formulas). That number would easily surpass the hundred, and the automatic tools at our disposal for that task were rudimentary (as discussed in Section 8.3).

Facing this situation, we decided that we needed some sort of *compositionality* in order to obtain new instances from the ones already proved: Having Replacement for class functions F and G does not entail immediately replacement under $F \circ G$ (unless you use one further instance of Separation, and the net gain is zero). The solution was to postulate for the relevant Fs, instead of replacement through $x \mapsto F(x)$, a *lambda replacement* through $x \mapsto \langle x, F(x)\rangle$. The name “lambda” corresponds to the fact that this type of replacement is equivalent to closure under $(\lambda x \in A.\ F(x)) := \{\langle x, F(x)\rangle : x \in A\}$ for every $A \in M$.

Now, a fixed set of six replacements and one separation (apart from those in `M_basic`, which also assumes the Powerset Axiom for the class M) is sufficient to obtain the lambda replacement under $x \mapsto \langle x, F(G(x))\rangle$ given those for F and G. To obtain compositions with binary class functions H, it is enough to assume the lambda replacement $x \mapsto \langle x, H(\mathit{fst}(x), \mathit{snd}(x))\rangle$. We summarize the assumptions in Table 1.

No.	Name	Instance
Replacement Instances		
1.	`lam_replacement_fst`	$x \mapsto \langle x, \mathit{fst}(x)\rangle$
2.	`lam_replacement_snd`	$x \mapsto \langle x, \mathit{snd}(x)\rangle$
3.	`lam_replacement_Union`	$x \mapsto \langle x, \bigcup(x)\rangle$
4.	`lam_replacement_Image`	$x \mapsto \langle x, \mathit{fst}(x)``\mathit{snd}(x)\rangle$
5.	`lam_replacement_middle_del`	$x \mapsto \langle x, \langle \mathit{fst}(\mathit{fst}(x)), \mathit{snd}(\mathit{snd}(x))\rangle\rangle$
6.	`lam_replacement_prodRepl`	$x \mapsto \langle x, \langle \mathit{snd}(\mathit{fst}(x)), \langle \mathit{fst}(\mathit{fst}(x)), \mathit{snd}(\mathit{snd}(x))\rangle\rangle\rangle$
Separation Instances		
7.	`middle_separation`	$\mathit{snd}(\mathit{fst}(x)) = \mathit{fst}(\mathit{snd}(x))$
8.	`separation_fst_in_snd`	$\mathit{fst}(\mathit{snd}(x)) \in \mathit{snd}(\mathit{snd}(x))$

Table 1. Replacement and Separation instances of the locale `M_replacement`

F 22 replacement instances to rule them all

In Table 2 we show the name of the fourteen formulas involved in the twenty two instances of replacement needed in our mechanization. The formulas marked with (†) are needed twice: one by themselves and the other as the argument for `ground_repl_fm`. The `ground_repl_fm` function maps φ to ψ as described in Sect. 6.1. These eight instances form the set `instances3_fms`.

No.	Formula's name	Comment
instances1_fms		
1.	`eclose_closed_fm` †	Closure under iteration of $X \mapsto \bigcup X$.
2.	`eclose_abs_fm` †	Absoluteness of the previous construction.
3.	`wfrec_rank_fm` †	For $\in$-rank.
4.	`transrec_VFrom_fm` †	For rank initial segments.
instances2_fms		
5.	`wfrec_ordertype_fm` †	Well-founded recursion for the construction of ordertypes.
6.	`omap_replacement_fm` †	Auxiliary instance for the definition of ordertypes.
7.	`ordtype_replacement_fm` †	Replacement through the ordertype function, for Hartogs' Theorem.
8.	`wfrec_Aleph_fm` †	Well-founded recursion to define Aleph.
instances_ground_fms		
9.	`wfrec_Hcheck_fm`	Well-founded recursion to define check.
10.	`wfrec_Hfrc_at_fm`.	Well-founded recursion for the definition of forcing for atomic formulas.
11.	`lam_replacement_check_fm`	Replacement through $x \mapsto \langle x, \check{x}\rangle$, for $\dot{G}$.
instances_ground_notCH_fms		
12.	`rec_constr_fm`	Recursive construction of sets using a choice function.
13.	`rec_constr_abs_fm`	Absoluteness of the previous construction.
instances_ground_CH_fms		
14.	`dc_abs_fm`	Absoluteness of the recursive construction in the proof of Dependent Choice from AC.
instances3_fms		
15-22.	`ground_repl_fm`(φ)	one for each formula φ marked with †

Table 2. Replacement Instances used in our mechanization