Formalizing the Metatheory of Logical Calculi and Automatic Provers in Isabelle/HOL (Invited Paper)

Jasmin Christian Blanchette
Department of Computer Science
Vrije Universiteit Amsterdam
Amsterdam, the Netherlands
j.c.blanchette@vu.nl

Abstract

IsaFoL (Isabelle Formalization of Logic) is an undertaking that aims at developing formal theories about logics, proof systems, and automatic provers, using Isabelle/HOL. At the heart of the project is the conviction that proof assistants have become mature enough to actually help researchers in automated reasoning when they develop new calculi and tools. In this paper, I describe and reflect on three verification subprojects to which I contributed: a first-order resolution prover, an imperative SAT solver, and generalized term orders for λ-free higher-order logic.

1 Introduction

At programming language conferences such as POPL and ICFP, submissions are often accompanied by formalizations. Proof assistants are even used in the classroom to teach language semantics and type systems [69, 79]. Paradoxically, the automated reasoning community has largely stood on the sidelines of these developments.

Like the shoemaker's children who go barefoot, we reflexively turn to "pen and paper"—by which we usually mean LaTeX—to define our logics, specify our proof systems, and establish their soundness and completeness. The automatic/ interactive divide of our community is part of the reason. Few automatic prover developers have first-hand experience with a proof assistant. Nevertheless, it stands to reason that the members of the automated reasoning community should be early adopters of proof assistants. If we cannot convince these close colleagues of the value of tools called "theorem provers," how are we going to seduce mainstream mathematicians, philosophers, and engineers?

The IsaFoL (Isabelle Formalization of Logic) effort[1] aims at changing this situation. This "coalition of the willing" was inaugurated in 2015, initially as a Bitbucket repository that would enable Mathias Fleury in Saarbrücken and Anders Schlichtkrull in Copenhagen to carry out their respective Ph.D. projects while avoiding duplicated work. Their foresighted and well-funded bosses, Christoph Weidenbach and Jørgen Villadsen, have made this project possible.

My motto for the project is "Coq at POPL, why not Isabelle at CADE?" But in fact, proof assistants, including Isabelle, have been represented at CADE and IJCAR for several years, thanks to Tobias Nipkow, Lawrence Paulson, Christian Urban, and others. Moreover, René Thiemann, Christian Sternagel, and their colleagues have been using Isabelle to formalize term rewriting for over a decade. Their IsaFoR library directly inspired IsaFoL.[2]

Our main objective with IsaFoL is to develop libraries and a methodology to support modern research in automated reasoning, especially about propositional and first-order logic. Proof assistants can help when developing proofs, but also when modifying them, to study generalizations and variants. Reviewing becomes much easier when a formalization exists; reviewers can then focus on checking the definitions and theorem statements. Although my primary interest is in metatheory per se, once we have formal libraries, we can build verified provers and proof checkers on top of them.

The project has grown to include contributions from researchers and students in Amsterdam, Copenhagen, Gothenburg, Grenoble, Munich, Saarbrücken, and Zurich. Contributors include Heiko Becker, Alexander Bentkamp, Andreas Halkjær From, Alexander Birch Jensen, Peter Lammich, John Bruntse Larsen, Julius Michaelis, Tobias Nipkow, Nicolas Peltier, Andrei Popescu, Simon Robillard, Sophie Tourret, Dmitriy Traytel, and Petar Vukmirović. The IsaFoL repository on Bitbucket is where much of the development takes place. Once individual entries have become mature enough, they tend to migrate to Isabelle's *Archive of Formal Proofs*, which is continuously tested and updated to work with the most recent version of Isabelle. The maintenance burden largely falls onto the Isabelle developers, following the "you break it, you fix it" principle.

The use of Isabelle/HOL was initially motivated by personal preference, but it has turned out to be a fortunate

[1] https://bitbucket.org/isafol/isafol

[2] Font aficionados will notice the divergent preferences concerning serifs and kernings. Pronunciation is also crucial: to avoid confusion, Japanophiles should clearly articulate *izaforu* (IsaFoL) and *izafō* (IsaFoR).

CPP'19, January 14-15, 2019, Cascais/Lisbon, Portugal
2019.

choice. Isar proofs [100], parameterized modules [4], (co)datatypes [10], the code generator [38], the Refinement Framework [52], Imperative HOL [26], and the Sledgehammer [75] and Nitpick [16] tools have all played important roles. Higher-order logic's lack of expressiveness is often cited as a drawback of the system, but it has not been an issue for me in this project. Using Isabelle has been an opportunity to "eat my own dog food," gaining insights into how the subsystems I am responsible for—the (co)datatypes, Sledgehammer, and Nitpick—could be improved.

In his invited SAS 2018 paper [72], Peter O'Hearn put forward the notion that

> in most research papers one gets a description of where a project got to, but not how it got there. We get the results of science and/or engineering, but not a picture of how it developed. For the practice of doing science and engineering I've always thought the issue of "how it's done" is an important one, and that examples describing how can have some value.

In this spirit, I describe three subprojects of IsaFoL to which I have contributed: a first-order prover based on ordered resolution (Section 2), an optimized SAT solver based on conflict-driven clause learning (Section 3), and generalized term orders for λ-free higher-order logic (Section 4). I reflect on the motivation and rewards of such work, both scientific and metascientific, and use this opportunity to survey related work, including the other IsaFoL entries (Section 5).

All along, my primary metascientific motivation has been to continue carrying out research at the intersection of interactive and automatic proving, enabling me to remain in the CADE orbit. After the Sledgehammer success story—automatic provers at the service of interactive ones [75]—I was stimulated by the thought of reversing the formula. There is an undeniable self-referential thrill to the project. *Formalize logic in logic. Employ automatic provers to formalize their own metatheory.* But as the project advanced, I started to appreciate its deeper value. The overwhelmingly positive response at CADE, CPP, IJCAR, and *JAR* has convinced me that this research program was overdue.

I am excited about recent developments, where flaws and limitations revealed in formalization motivate new research. We are gradually moving from carrying out case studies, where the proof assistant aspect is predominant, to writing metatheory papers that only briefly mention the formal development in their introduction.

2 A First-Order Ordered Resolution Prover

Nearly two decades after its publication, the *Handbook of Automated Reasoning* remains an invaluable resource for researchers in the area. In 2014, I started formalizing its Chapter 2, written by Bachmair and Ganzinger [3]. The first four sections present the metatheory of ordered resolution with selection, culminating in Section 4.3 with a refutationally complete first-order prover, RP, based on this calculus. At the time, I had little experience using Isabelle, so it was fortunate that Traytel agreed to act as my mentor.

The chapter had been haunting me as a *tsundoku* for a couple of years. Formalizing it meant I would finally have to take it from my reading pile and read it thoroughly. But there were also sound scientific reasons to choose this target for formalization, as we remarked later [87, Section 1]:

> The text is a standard introduction to superposition-like calculi (together with *Handbook* Chapters 7 and 27). It offers perhaps the most detailed treatment of the lifting of a resolution-style calculus's static completeness to a saturation prover's dynamic completeness. It introduces a considerable amount of general infrastructure, including different types of inference systems (sound, reductive, counterexample-reducing, etc.), theorem proving processes, and an abstract notion of redundancy. The resolution calculus, extended with a term order and literal selection, captures most of the insights underlying ordered paramodulation and superposition, but with a simple notion of model.

Traytel and I made quick progress in two intense weeks, reaching the crucial Section 4.3. This is where the resolution calculus is lifted from ground (propositional) to nonground (first-order) clausal logic and where the RP prover is introduced and shown to be refutationally complete.

At the ground level, ordered resolution consists of the single $(n+1)$-ary inference rule

$$\frac{(C_i \vee A_i \vee \cdots \vee A_i)_{i=1}^{n} \quad \neg A_1 \vee \cdots \vee \neg A_n \vee D}{C_1 \vee \cdots \vee C_n \vee D}$$

with multiple side conditions that restrict the search space. This rule is *refutationally complete*, meaning that any unsatisfiable set that is closed under applications of the rule will contain the empty clause $\bot$. Remarkably, replacing the subclause $A_i \vee \cdots \vee A_i$ with a single literal A_i is enough to make the rule incomplete [3].

A *redundancy criterion* identifies clauses and inferences that can be ignored. For example, $\mathrm{p(a)} \vee \mathrm{q(b)}$ is made redundant by $\mathrm{p(a)}$, which in turn is made redundant by $\mathrm{p}(x)$.

Next, Bachmair and Ganzinger introduce the concept of a *theorem proving process*: a transition system that starts with an initial clause set N_0 and where each transition $\rhd$ corresponds either to an inference or the removal of redundant clauses. Under a fairness assumption, the calculus's completeness theorem characterizes the limit of a derivation $N_0 \rhd N_1 \rhd N_2 \rhd \cdots$.

Section 4.3 is where the trouble starts. For nonground clauses, the resolution rule takes the form

$$\frac{(C_i \vee A_{i1} \vee \cdots \vee A_{ik_i})_{i=1}^{n} \quad \neg A_1 \vee \cdots \vee \neg A_n \vee D}{(C_1 \vee \cdots \vee C_n \vee D)\sigma}$$

where σ is the most general unifier of the set of constraints $\{A_{i1} \stackrel{?}{=} \cdots \stackrel{?}{=} A_{ik_i} \stackrel{?}{=} A_i \mid i \in \{1,\ldots,n\}\}$. Side conditions block inferences where $A_i\sigma$ is smaller than $C\sigma$ or $D\sigma$ with respect to a fixed order. A literal selection mechanism further prunes the search space.

Traytel and I initially stopped here. By a stroke of good fortune, Schlichtkrull decided to resume the proof two years later. After months of labor, and with expert help from Waldmann, he reached the final **qed**. The resulting IJCAR 2018 paper [87] was well received by the reviewers, reaping a "strong accept" and two "accepts." One of them wrote:

> The authors convinced me that their development is a great tool for exploring/developing calculus extensions. It will enable us to "*extend/hack without fear*."

(The italics are mine throughout this paper.)

The RP prover is naturally formulated in Isabelle as an inductive predicate. Bachmair and Ganzinger's transition rules correspond directly to introduction rules:

inductive $\leadsto$ :: *'a state* $\Rightarrow$ *'a state* $\Rightarrow$ *bool* **where**
tautology_delete: Neg $A \in C \wedge$ Pos $A \in C \Longrightarrow$ $(\mathcal{N} \cup \{C\}, \mathcal{P}, \mathcal{O}) \leadsto (\mathcal{N}, \mathcal{P}, \mathcal{O})$
| *forward_subsume*: $D \in \mathcal{P} \cup \mathcal{O} \wedge$ subsumes $D\ C \Longrightarrow$ $(\mathcal{N} \cup \{C\}, \mathcal{P}, \mathcal{O}) \leadsto (\mathcal{N}, \mathcal{P}, \mathcal{O})$
| *backward_subsume_$\mathcal{P}$*: $D \in \mathcal{N} \wedge$ ssubsumes $D\ C \Longrightarrow$ $(\mathcal{N}, \mathcal{P} \cup \{C\}, \mathcal{O}) \leadsto (\mathcal{N}, \mathcal{P}, \mathcal{O})$
| *backward_subsume_$\mathcal{O}$*: $D \in \mathcal{N} \wedge$ ssubsumes $D\ C \Longrightarrow$ $(\mathcal{N}, \mathcal{P}, \mathcal{O} \cup \{C\}) \leadsto (\mathcal{N}, \mathcal{P}, \mathcal{O})$
| *forward_reduce*: $D \in \mathcal{P} \cup \mathcal{O} \wedge$ reduces $D\ C\ L \Longrightarrow$ $(\mathcal{N} \cup \{C \uplus \{L\}\}, \mathcal{P}, \mathcal{O}) \leadsto (\mathcal{N} \cup \{C\}, \mathcal{P}, \mathcal{O})$
| *backward_reduce_$\mathcal{P}$*: $D \in \mathcal{N} \wedge$ reduces $D\ C\ L \Longrightarrow$ $(\mathcal{N}, \mathcal{P} \cup \{C \uplus \{L\}\}, \mathcal{O}) \leadsto (\mathcal{N}, \mathcal{P} \cup \{C\}, \mathcal{O})$
| *backward_reduce_$\mathcal{O}$*: $D \in \mathcal{N} \wedge$ reduces $D\ C\ L \Longrightarrow$ $(\mathcal{N}, \mathcal{P}, \mathcal{O} \cup \{C \uplus \{L\}\}) \leadsto (\mathcal{N}, \mathcal{P} \cup \{C\}, \mathcal{O})$
| *clause_process*: $(\mathcal{N} \cup \{C\}, \mathcal{P}, \mathcal{O}) \leadsto (\mathcal{N}, \mathcal{P} \cup \{C\}, \mathcal{O})$
| *inference_compute*: $(\emptyset, \mathcal{P} \cup \{C\}, \mathcal{O}) \leadsto$ (concl_of ‘ infers_between $\mathcal{O}\ C, \mathcal{P}, \mathcal{O} \cup \{C\})$

We faced various difficulties when formalizing Section 4.3. It did not help that the text contains dozens of small mistakes. Even the statement of the nonground resolution rule suffers from typos and ambiguities. While we agree with the reviewer who wrote that "most of us unconsciously autocorrect it, and read it with the intended meaning," on several occasions we found ourselves blindly trusting the text, only to be disappointed later.

Reasoning about the $(n+1)$-premise resolution rule was particularly tedious. Ellipsis is a convenient pen-and-paper device that lacks a counterpart in proof assistants. We ended up keeping the clauses C_i and atoms $\mathcal{A}_i = \{A_{i1},\ldots,A_{ik_i}\}$ in parallel lists. Ideally, the first n premises of the rule would be regarded as a multiset; there is no need to consider $n!$ inferences all yielding the same clause. However, there is no such things as "parallel multisets," and our attempts at storing tuples $(C_i, \mathcal{A}_i)$ in multisets only made matters worse.

Another general difficulty with the chapter was that it is not always clear which hypotheses apply where. The text both presents a general framework and applies it, but dependencies are not tracked precisely, and many lemmas are never invoked explicitly. It was a challenge to understand the informal proof well enough to organize the modules and state the definitions and lemmas precisely and correctly.

Finally, some of the arguments are incomplete or misleading. The proof of Lemma 4.11 relies on the observation that "D' cannot be deleted by backward reduction or backward subsumption." However, in principle, D' could be deleted due to the presence of a more general clause D'', which in turn could be deleted due to D''', and so on. The key missing observation is that this process can be iterated at most finitely many times, generalization being a well-founded relation.

With the module hierarchy, definitions, and key lemma statements in place, carrying out the proofs was mostly straightforward. We relied heavily on Sledgehammer to discharge the proof obligations.

We did find a significant mistake in the chapter. Theorem 4.13 states that RP is refutationally complete, but this does not hold due to the improper treatment of inferences containing duplicate premises. Remarkably, we discovered the mistake several months *after* reaching the final **qed**, while reviewing our definitions. We had inadvertently "autocorrected" Bachmair and Ganzinger's definition of RP.

Formalization helps track assumptions and dependencies precisely. It helps us answer questions such as, "Is Lemma 3.13 actually needed, and if so, where?" Indeed, such a question recently arose in the course of Bentkamp's research on superposition [8]. He wanted to understand why the literal selection function S_M is defined so that "(ii) $S_M(C) = S(C)$, if C is not in K." He quickly got two replies. Waldmann wrote:

> *As far as I can see*, S_M is really only needed for ground instances, and then case (ii) is irrelevant. *I guess* they just wanted to define S_M as a total function.

Thanks to Isabelle, Schlichtkrull could be more confident:

> I tried to change the definition in the formalization to return the empty multiset if C is not in K. With this definition the above properties also hold, *and the proof goes through.*

This anecdote nicely illustrates how formal proofs help generate knowledge and understanding. Here, they helped Bentkamp "extend/hack without fear."

The second half of Bachmair and Ganzinger's chapter, starting with Section 5, focuses on variations, such as nonstandard clauses and hyperresolution. Most of these are not implemented in modern provers, and we did not attempt to formalize this material. Instead, Schlichtkrull, Traytel, and I further refined the abstract prover RP to obtain an executable functional program. This work is described in a manuscript [86] that was originally written for an audience interested in the principles of programming languages.

We started by defining the inductive predicate $\mathrm{RP_w}$, which resembles RP but adds a timestamp to clauses and a weight function. Bachmair and Ganzinger [3, p. 44] mention this idea in a footnote, but they require the weight function to be monotone in both the timestamp and the clause size, claiming that this is necessary to ensure fairness. Although it often makes sense to prefer smaller clauses, our proof reveals that this is not necessary to ensure fairness.

Next, we defined $\mathrm{RP_d}$ as a deterministic functional program. $\mathrm{RP_d}$ simply calls the auxiliary function $\mathrm{RP_d}$_step repeatedly until a final state (with $\mathcal{N} = \mathcal{P} = \emptyset$) is reached. $\mathrm{RP_d}$_step is a function of about 40 lines of code that is loosely modeled after Vampire's main loop [97]. To introduce this possibly nonterminating function in Isabelle, we defined $\mathrm{RP_d}$ by means of an option monad, using the **partial_function** command [48], so that it returns a value of the form Some R if the computation terminates and None otherwise.

Finally, we made $\mathrm{RP_d}$ executable by connecting it to IsaFoR, which provides first-order terms and operations on them, such as unification and the Knuth–Bendix order. We invoked Isabelle's code generator [38] to export the prover to Standard ML. The resulting program, $\mathrm{RP_x}$, consists of about 1000 lines of functional ML code, including dependencies.

After working hard to obtain an executable prover, we evaluated it on a representative subset of 1000 TPTP benchmarks against two leading provers, E [89] and Vampire [97], as well as Metis [45], which is written in Standard ML. The table below gives the number of problems solved (proved or disproved) by each prover [86]:

Vampire	E	Metis	$\mathrm{RP_x}$
833	770	527	353

Although our prover cannot yet challenge the state of the art, its performance is respectable and could be improved further using refinement. In his presentations, Natarajan Shankar often stresses the view that

$$\text{algorithm} = \text{inference} + \text{strategy} + \text{indexing}$$

Indeed, the performance of an automatic proof procedure comes largely from three sources: the calculus, the heuristics, and the data structures. $\mathrm{RP_x}$ implements an excellent calculus, but mediocre heuristics and data structures. In the next section, we will look at a verification project that takes these two aspects seriously.

The work on formalizing RP and $\mathrm{RP_x}$ opens exciting perspectives. First, it paves the way for the formalization of superposition-based provers. Peltier [77] took us by surprise when he announced, in 2016, his Isabelle formalization of the superposition calculus. Based on his work, it should be possible to define an SP prover analogous to RP, but which could reason efficiently about equality, implementing most of the simplification rules and heuristics described in, for example, the E paper [89]. Verifying data structures such as discrimination trees, feature-vector indices, and fingerprint indices would pose interesting challenges. In unpublished work, Vukmirović has verified the metatheory of fingerprints for λ-free higher-order logic [98] using Isabelle.

Another perspective is to improve Bachmair and Ganzinger's framework. In unpublished work, Waldmann, Tourret, Robillard, and I have conceived, with pen and paper, a framework that captures abstractly the lifting from a ground calculus's completeness result to a nonground RP-like prover. An Isabelle formalization is underway, which should culminate with a streamlined proof of Bachmair and Ganzinger's Theorem 4.13. The framework will also support infinitary inferences; these are useful for automating higher-order logic, where the unification procedure may yield an infinite stream of incomparable unifiers.

Around 2012, Vampire was extended with a SAT solver, using a novel architecture called AVATAR [97]. The empirical results were sensational, but a fundamental question was left unanswered: is AVATAR refutationally complete? The literature contains contradictory, erroneous definitions of the architecture [12, 80, 97], but following discussions with Giles Reger and his colleagues, I believe I have reached a precise definition, for which refutational completeness holds, while isolating a few potential sources of incompleteness in Vampire. Furthermore, Vukmirović and I have shown that labeled splitting [32], as implemented in SPASS, can be seen as an instance of a slightly generalized AVATAR. I hope to establish all of this formally in Isabelle, using the new framework by Waldmann et al. described above.

3 A CDCL SAT Solver

As part of a 2015 M.Sc. internship in Saarbrücken, Fleury started formalizing Weidenbach's textbook draft, tentatively called *Automated Reasoning—the Art of Generic Problem Solving*. He continued as a Ph.D. student, focusing largely on SAT solving and the conflict-driven clause learning (CDCL) calculus implemented in most modern SAT solvers.

Inconveniently for us, there already existed several verified CDCL-based solvers [58, 60, 61, 71, 91]. We found a niche by emphasizing the stepwise refinement methodology and the connection between calculi variants, and by considering some aspects that had not been the focus of formalization before: clause forgetting, solver restarts, and incremental solving. We hoped this would suffice to get our submission accepted at IJCAR 2016; little did we expect to receive the best paper award. In the jury's words, our paper [15]

> formalizes a modern SAT solver via a chain of refinements in a proof assistant, contributing to the program of formalizing highly-technical research in the field of automated reasoning using tools developed in this field.

We considered both the abstract CDCL calculus described by Nieuwenhuis, Oliveras, and Tinelli [68] and a refined implementation-oriented calculus proposed by Weidenbach [99]. The calculi are represented by inductive predicates on state tuples, roughly along the lines of the RP prover described in the previous section.

The proofs are largely elementary, relying on basic results about multisets and well-founded relations. We generally found Nieuwenhuis et al.'s arguments easy to follow, with the notable exception of a gap in their termination proof. Weidenbach's concise proofs were often more challenging to formalize. To give a flavor of his text, I quote a passage that argues why it is impossible for the same clause to be derived, or "learned," twice:

> By contradiction. Assume CDCL learns the same clause twice, i.e., it reaches a state $(M, N, U, D \vee L)$ where Jump is applicable and $D \vee L \in N \uplus U$. More precisely, the state has the form $(K_n \cdots K_2 K_1^{\dagger} M_2 K^{\dagger} M_1, N, U, D \vee L)$ where the K_i, $i > 1$ are propagated literals that do not occur complemented in D, as otherwise D cannot be of level i. Furthermore, one of the K_i is the complement of L. But now, because $D \vee L$ is false in $K_n \cdots K_2 K_1^{\dagger} M_2$ $K^{\dagger} M_1$ and $D \vee L \in N \uplus U$ instead of deciding $K_1^{\dagger}$ the literal L should be propagated by a reasonable strategy. A contradiction. Note that none of the K_i can be annotated with $D \vee L$.

Fleury needed over 700 lines of Isabelle to capture this paragraph. By changing the argument, he was later able to reduce the formal proof down to 250 lines. The key was to exploit invariance: by first establishing a strong invariant on SAT solver states (namely, that all unit propagations have been performed before deciding a new literal), he could easily perform the key step of the argument ("the literal L should be propagated by a reasonable strategy") without having to refer to earlier states or transitions.

Although Fleury did not discover any significant flaw in the metatheory of CDCL (which would have been most surprising), he did find a critical mistake in an extension, described in Weidenbach's draft, of CDCL with the branch-and-bound principle. He came up with a counterexample that invalidates the main correctness theorem, whose proof confused partial and total models. Interestingly, he later noticed the same flaw in a SAT 2009 paper by Larrosa et al. [56]; the issue is silently repaired in a subsequent article [57].

As a proof of concept, Fleury implemented a deterministic SAT solver and extracted functional Standard ML code from it, preserving the formal guarantees established about CDCL: soundness, completeness, and termination. The resulting program was very inefficient; it could solve *none* of the 2009 SAT Competition problems in reasonable time.

This was good enough for me but not for Fleury. With Lammich's help, he proceeded to specify the two-watched-literal (2WL) scheme and other imperative data structures [65], gradually departing from Weidenbach's draft. This work is described in a *JAR* article [13] and a CPP 2018 paper [33].

The 2WL scheme makes it possible to efficiently identify candidate clauses for unit propagation and conflict detection, which are the core CDCL operations. In each clause, the solver distinguishes two *watched* literals—literals that can possibly influence their clauses' truth value in the solver's current state. The solver maintains the "2WL invariant" for each clause. Unfortunately, the literature is imprecise about the nature of this invariant and about when it is required to hold. Fleury quickly found himself studying MiniSat's source code [31], and eventually came up with a precise formulation. He specified 2WL as an abstract calculus, following Weidenbach's draft, and proved that its transitions preserve the 2WL invariant. To get an executable program, he refined the calculus in several correctness-preserving steps. The refinement methodology enabled him to inherit invariants, correctness, and termination from previous layers.

The next refinement step implements the rules of the 2WL calculus in a more algorithmic fashion, using the nondeterministic programming language provided by Lammich's Refinement Framework [52]. The language is built on top of a nondeterminism monad. To give a flavor of it, I present the code of a function that implements four calculus rules:

```
definition PCUI_algo :: 'v lit × 'v clause ⟶ state ⟶ state
where
  PCUI_algo LC S = do {
    let (M, N, U, D, NP, UP, WS, Q) = S;
    let (L, C) = LC;
    L' ← RES {L' | L' ∈ watched C − {L}};
    if L' ∈ M then
      RETURN S
    else
      if ∀L ∈ unwatched C. −L ∈ M then
        if −L' ∈ M then
          RETURN (M, N, U, C, NP, UP, ∅, ∅)
        else
          RETURN (L'^C · M, N, U, D, NP, UP, WS,
              {−L'} ⊎ Q)
      else do {
        K ← RES {K | K ∈ unwatched C ∧ −K ∉ M};
        (N', U') ← RES {(N', U') |
            update_clss (N, U) C L K (N', U')};
        RETURN (M, N', U', D, NP, UP, WS, Q)
      }
  }
```

The subsequent refinement steps optimize the data structures and specify heuristics that reduce the nondeterminism, following ideas from the SAT literature and actual implementations. Multisets give way to lists, and clauses are now referenced by indices into a list of clauses instead of stored directly. For each literal L, the clauses that contain a watched L are chained together in a linked list, called a *watch list*.

The Sepref tool [53], which is based on separation logic, can be used to synthesize imperative code from a monadic Isabelle program. It replaces the functional data structures by imperative implementations, while leaving the algorithmic structure unchanged. The resulting program, called IsaSAT, is expressed using Imperative HOL's heap monad [26]. Using Isabelle's code generator [38], we can extract a self-contained program in imperative Standard ML.

Since the CPP 2018 publication, Fleury has improved IsaSAT further by implementing four optimizations: restarts, forgetting, blocking literals, and machine integers.

Restarts is a technique that enables the solver to explore another part of the search space. We can keep completeness by gradually increasing the interval between restarts. Our calculi included an optional Restart rule all along to allow such behavior, but it was not implemented in the first IsaSAT.

Forgetting removes some learned clauses—consequences of the initial problem clauses that are derived during solving. Helpfully, the abstract CDCL calculus included a Forget rule from the start. The main difficulty is that each clause now needs to store a Boolean flag indicating whether it is deleted. This requires adding a header to the clause data structure for storing this and other useful information that guides the heuristics, and adapting all the refinement proofs.

Blocking literals are literals stored directly in the watch list, next to a pointer (or index) to the clause, that can be checked directly without dereferencing the pointer. The information is redundant, but it often saves a pointer dereference.

Machine (*64-bit*) *integers* are large enough to store clause indices and other numbers for all SAT problems arising in practice, and they are more efficient than ML's unbounded integers. To use them without losing the formal guarantees about the program, we generate two versions of the prover's body and connect them with code of the form

```
while ¬ done ∧ ¬ overflow do
  ⟨invoke the 64-bit version of the solver's body⟩;
if ¬ done then
  ⟨convert the state from 64-bit to unbounded integers⟩;
  while ¬ done do
    ⟨invoke the unbounded version of the solver's body⟩
```

In a preliminary evaluation, Fleury ran IsaSAT against four other solvers on a collection of 3313 benchmark problems, consisting of all the SAT Competition problems from the 2009, 2011, 2013, 2014, 2016, and 2017 editions of the SAT Competition and the 2015 edition of the SAT-Race. The solvers Glucose [1] and CaDiCaL [11] represent the state of the art; MiniSat [31] is a well-known reference solver; and `versat` [71] is the only other efficient verified solver we know of. Since IsaSAT does not implement preprocessing techniques yet, CryptoMiniSat [92] was used to simplify all problems before benchmarking. The tests were run with a time limit of 30 minutes per problem. The table below shows the number of solved problems, whether satisfiable or unsatisfiable, for each system:

Glucose	CaDiCaL	MiniSat	IsaSAT	`versat`
1670	1645	1361	731	357

Given that competitive SAT solvers are extremely optimized programs, these results are very encouraging. However, we must bear in mind that it took two years of hard labor, and tens of thousands of Isabelle lines, to get from 0 to 731. An obvious avenue for future work would be to add optimizations such as pre- and inprocessing.

A benefit of having a verified SAT solver is that it can be employed as a backend in other verified tools. We have started looking into extending IsaSAT with theory reasoning, as in an SMT (satisfiability-modulo-theories) solver, with the goal of integrating it in CeTA [24], a verified safety and termination proof checker developed as part of IsaFoR.

4 Lambda-Free Higher-Order Terms Orders

The last subproject has a different flavor from the other two, in that the formalization arose as a side effect of carrying out research in automated reasoning and not as an end in itself. The Matryoshka project,[3] which started in 2017 and funds a collaboration between Amsterdam, Nancy, Saarbrücken, and Stuttgart, aims at extending superposition provers and SMT solvers with higher-order features. As a stepping stone towards full higher-order logic, we started by focusing on a λ-free fragment. Unlike in first-order logic, variables may be applied, and function symbols may be given fewer arguments than they can take. This language is sometimes called "applicative first-order" in the term-rewriting community.

Bentkamp developed, under Waldmann's and my supervision, a refutationally complete superposition calculus for λ-free higher-order logic and implemented it in a prototype prover developed with Cruanes [8]. He wrote his proofs directly in LaTeX, which was possible only because he is extremely rigorous and could count on two experts to check his proofs—namely, Waldmann and our colleague Tourret. In principle, he could have started from Peltier's Isabelle formalization of superposition [77], but it seemed more difficult than working in LaTeX, especially given that he was more familiar with Waldmann's informal proof, which has a different structure from Peltier's.

Superposition, like ordered resolution (Section 2), relies on a well-founded term order. Together with Becker, Waldmann, and Wand, I designed two (families of) orders that generalize the familiar Knuth–Bendix order (KBO) and recursive path order (RPO). Becker formalized most of KBO during an internship; I completed his work and proceeded with RPO. This research was presented at FoSSaCS 2017 [22] and CADE 2017 [6].

The term orders are comparatively simple mathematical objects that lend themselves well to mechanization. I worked directly in Isabelle to define them and state their desired properties, starting with the ground case. The following fragment corresponds to my first attempt at defining a lexicographic path order (LPO), a special case of RPO, on ground terms:

[3] http://matryoshka.gforge.inria.fr/

```
datatype 'c tm = F 'c ('c tm list)

context
  fixes <c :: 'c ⇒ 'c ⇒ bool
  assumes irreflp (<c) and transp (<c) and
    f <c g ∨ g <c f ∨ f = g
begin

definition chksubs :: ('c tm ⇒ 'c tm ⇒ bool) where
  chksubs R s t ⟷
  (case (s, t) of (F f ss, F g ts) ⇒
    (∀s' ∈ ss. R s' t ∨ s' = t) ∧
    (ss ≠ [] ∧ ts ≠ [] ∧ last ss = last ts ⟶
     R (F f (butlast ss)) (F g (butlast ts))))

inductive ≺ :: 'c tm ⇒ 'c tm ⇒ bool where
  s ≺ t ∨ s = t ⟹ s ≺ F f (ts @ [t]))
| s = F f ss ∧ t = F g ts ∧ s ≺ t ⟹
  F f (ss @ [u]) ≺ F g (ts @ [u])
| s = F f ss ∧ t = F g ts ∧ f <c g ∧
  chksubs (≺) s t ⟹ s ≺ t
| s = F f ss ∧ t = F f ts ∧ chksubs (≺) s t ∧
  lexordp (≺) ss ts ⟹ s ≺ t

lemma irrefl: ¬ s ≺ s
lemma trans: s ≺ t ∧ t ≺ u ⟹ s ≺ u
lemma total: s ≺ t ∨ t ≺ s ∨ s = t
lemma compat_fun:
  s ≺ t ⟹ F f (ss @ s · ss') ≺ F f (ss @ t · ss')
lemma compat_arg:
  F f ss ≺ F g ts ⟹ F f (ss @ [s]) ≺ F g (ts @ [s])
...
end
```

The specification is very short and depends on no background theory beyond list operations (cons ·, append @, butlast, lexordp). It is a perfect match for my counterexample generator Nitpick [16]. And indeed, the tool quickly finds a counterexample to the *irrefl* conjecture: given a type *'c* with two distinct symbols f, g such that $\mathsf{f} \prec_c \mathsf{g}$, we have f (g f) ≺ f (g f). (Sadly, Nitpick's rival Quickcheck [25] fails with the error "No type arity *tm* :: *full_exhaustive*.")

Using Nitpick, I was able to converge to an almost correct design. A major flaw had escaped my attention, namely: without arities or typing constraints, LPO is not well founded, because it allows infinite descending chains such as

$$\mathsf{f\ b} \succ \mathsf{f\ a\ b} \succ \mathsf{f\ a\ a\ b} \succ \mathsf{f\ a\ a\ a\ b} \succ \cdots$$

This was noticed early on by Waldmann. Nitpick is helpless here, because it is based on finite model finding. It can be used to detect cycles but not acyclic divergence.

For the proofs, we drew our inspiration mostly from Baader and Nipkow's textbook [2]. However, they cover only LPO and not RPO. When formalizing RPO, I faced a chicken-and-egg problem that took me several days to untangle. The issue is related to the multiset order, which is used to define RPO. There exist two main formulations of the multiset order: Dershowitz–Manna [30] and Huet–Oppen [43]. They are equivalent on partial orders. Since RPO is a partial order, at first I chose Huet–Oppen, which we had used for KBO; however, until we have proved irreflexivity and transitivity of RPO, we cannot assume it is a partial order, so the choice between the two multiset orders is crucial. Zantema [101] was well aware of this, but I came across his work too late. And regardless, I could not think clearly while under the charm of the myth "Dershowitz–Manna = Huet–Oppen."

As is often the case, once the main ideas were clarified, the formal proofs were straightforward to develop. According to generated logs, on some days I invoked Sledgehammer over 100 times. The first-order nature of most proof obligations (with the notable exception of well-foundedness) was a good match for automatic provers. The following Isar fragment, where each subproof (highlighted in gray) was produced by Sledgehammer, is fairly representative:

```
have arity_hd (head s) = 1
  by (metis One_nat_def arity.wary_AppE
    dual_order.order_iff_strict eSuc_enat
    enat_defs(1,2) ileI1 linorder_not_le not_iless0
    wary_st wt_gt_δ_if_superunary wt_s)
hence nargs_s: num_args s = 0
  by (metis enat_ord_simps(2) less_one nargs_lt
    one_enat_def)
have s = Hd (head s)
  by (simp add: Hd_head_id nargs_s)
then obtain f where
  f ∈ ground_heads (head s) and
  wt_sym f + the_enat (δ ∗ arity_sym f) = δ
  using exists_wt_sym wt_s by fastforce
```

The first proof above relies on 12 lemmas. Developing such a proof manually could easily have taken half an hour without the help of Sledgehammer (to find the proof) or the *metis* proof method (to reconstruct it), and would have required searching for lemmas or memorizing their names.

In our two papers and the associated technical reports, we presented informal versions of the Isabelle proofs, for human consumption. This has been an opportunity to clean up and restructure the Isabelle proofs, to emphasize the important steps. We also used Nitpick to create examples to illustrate fine points in the papers [6, Example 12; 22, Example 9]. Given two orders $\prec_1$ and $\prec_2$, we could ask the tool to generate terms s, t such that $s \prec_1 t$ but $s \succ_2 t$.

Proof assistants really come into their own when we start modifying existing developments. Late in the project, we asked ourselves, "Could we generalize definition so-and-so in such-and-such a way?" After spending one hour with Isabelle, I was convinced the answer was yes, and a few hours later I was done repairing the proofs. There was very little to change in the technical report, but I would have had a hard time locating the passages that needed modifications and convincing myself that I had found them all.

The two papers received a lukewarm welcome. The term orders were not implemented in any termination tool, nor (at the time) in a superposition prover, and some related work was not treated. Nevertheless, the reviewers were ostensibly impressed by the formalization, which probably saved the submissions. I realize I may be preaching to the choir, but let me quote two of the reviews:

> Statements are very precise. *Only necessary conditions are made.* In particular, orders are characterized by 9 properties, and for each main statement, the authors discuss which of them are necessary.

> [The submission] incorporates a comprehensible structure, *precise assumptions and thorough proofs* for all the KBO's essential properties. All the presented proofs (irreflexivity, transitivity, subterm property, compatibility with functions and arguments, stability under substitution, totality on ground terms, well-foundedness) have been formalized within Isabelle/HOL and published as part of the Archive of Formal Proofs. Together with the careful presentation of these proofs within the submission, this establishes a *high overall reliability* of the presented results.

Isabelle helped in other ways during the CADE rebuttal phase. A reviewer had asked, "Doesn't X2 follow from X1 by taking $x >_2 y$ iff $h(x) >_1 h(y)$?" Using Nitpick, we were able to provide a simple counterexample.

Our λ-free higher-order RPO suffers for a theoretical blemish: it does not satisfy *compatibility with arguments*, the requirement that $s \prec t$ implies $s\ u \prec t\ u$ for all terms s, t, u. In fact, this requirement cannot be met while maintaining the subterm property and coincidence with the standard RPO on the first-order fragment of higher-order terms. In 2017, I challenged Bentkamp to design an RPO-like order that satisfies compatibility with arguments, by giving up full coincidence with the standard RPO. He succeeded brilliantly. He even used Isabelle to design his new order, the *embedding path order* (EPO) [7], and reports that proving the properties formally was easier than it would have been with pen and paper. On the other hand, he struggled with my tool Nitpick and ended up testing his conjectures by coding them in Haskell and using Lazy SmallCheck [83].

At some point, I generalized our definition of KBO to use ordinals instead of natural numbers, resulting in a transfinite KBO (TKBO). For this work, I collaborated with Fleury and Traytel. Using Isabelle's new (co)datatype package [10], we could define the *syntactic ordinals*—the ordinals below ϵ_0, representable in Cantor normal form—as the following datatype of hereditarily finite multisets:

datatype *hmultiset* = HMSet (*hmultiset multiset*)

The recursion through a nondatatype, *multiset*, would be problematic in other systems, or in pre-2013 versions of Isabelle/HOL. An ordinal $\alpha = \omega^{\alpha_1} \cdot c_1 + \cdots + \omega^{\alpha_n} \cdot c_n$ in Cantor normal form is identified with the hereditarily finite multisets

$$\alpha = \{\overbrace{\alpha_1, \ldots, \alpha_1}^{c_1 \text{ copies}}, \ldots, \overbrace{\alpha_n, \ldots, \alpha_n}^{c_n \text{ copies}}\}$$

We used this opportunity to also introduce a type of nested finite multisets and defined Dershowitz and Manna's nested multiset order on it [30]:

datatype *'a nmultiset* =
Elem *'a* | MSet (*'a nmultiset multiset*)

This enabled us to finally give a positive answer to Paulson, who in 2014 had asked on the Isabelle mailing list:

> I wonder whether anybody is aware of a formalisation (in any system) of the nested multiset ordering, as described in the classic paper "Proving Termination With Multiset Orderings"?

Using Isabelle's Lifting and Transfer tools [44], we established a bijection between *hmultiset* and the Elem-free fragment of *'a nmultiset* and exploited it to lift definitions and properties. Notably, lifting the nested multiset order gives the familiar < operator on ordinals. The order's well-foundedness proof can be transferred as well. Ordinal arithmetic operations such as addition and multiplication can be defined directly in terms of multiset operations.

Overall, we were able to quickly develop a versatile, practical library of syntactic ordinals, which we used not only for our TKBO variant but also in a formal proof of Goodstein's theorem. This research was presented at FSCD 2017 [14].

5 Related Work

IsaFoL consists of many more subprojects beyond those described above. The first two listed below predate IsaFoL, but they are very much in its spirit and are mentioned on its web page. The entries are listed in rough chronological order:

- equisatisfiability of sort encodings for first-order logic, by Popescu and myself [17, 18];
- abstract soundness and completeness results for first-order logics using coinductive methods, by Popescu, Traytel, and myself [19–21];
- soundness and refutational completeness of first-order unordered resolution, by Schlichtkrull [84, 85];
- soundness and refutational completeness of a generalization of the superposition calculus, by Peltier [77];
- soundness and completeness of resolution-based prime implicate generation, by Peltier [76];
- metatheoretical results about a paraconsistent propositional logic, by Schlichtkrull and Villadsen [88, 95];
- soundness of a small-kernel first-order prover with equality described in Harrison's textbook [41], by Jensen, Larsen, Schlichtkrull, and Villadsen [46, 47];
- correctness of an optimized tool chain for checking SAT solver certificates, by Lammich [54, 55];
- various metatheoretical results about a wide range of proof systems for classical propositional logic (sequent

calculus, natural deduction, Hilbert systems, and resolution), by Michaelis and Nipkow [63, 64];
- extensions of Berghofer's formalization [9] of a first-order natural deduction calculus, by From [34];
- soundness of a substitutionless first-order proof system, by From, Larsen, Schlichtkrull, and Villadsen [36];
- soundness and completeness of an epistemic logic with countably many agents, by From [35];
- modernization of Ridge and Margetson's formalization [81, 82] of a sequent calculus for a term-free first-order logic, by Villadsen, Schlichtkrull, and From [96].

Formalizing metatheoretical results about logic, proof systems, and reasoning tools is an attractive proposition for many researchers in our field. Landmark achievements in the 1980s and 1990s include Shankar's proof of Gödel's first incompleteness theorem in Nqthm [90], Persson's completeness proof for intuitionistic predicate logic in ALF [78], and Harrison's formalization of basic first-order model theory in HOL Light [39].

Following Shankar's 1984 proof, Gödel's first incompleteness theorem has been formalized in Coq by O'Connor [70], in HOL Light by Harrison (in unpublished work), and in Isabelle/HOL by Paulson [73, 74]. Paulson also succeeded at verifying the second incompleteness theorem.

The completeness theorem for first-order logic has been mechanized many times in proof assistants. In Isabelle/HOL, Berghofer [9] proved the completeness of a natural deduction calculus, and Margetson and Ridge [81, 82] proved soundness, completeness, and cut-elimination of a sequent calculus for a term-free first-order logic. I refer to a recent article I wrote with Popescu and Traytel [21] for a discussion of such work.

Term rewriting is another popular target of formalization. The CoLoR library by Blanqui and Koprowski [23] and the C*i*ME3 toolkit by Contejean et al. [27], both in Coq, as well as IsaFoR [94] in Isabelle, have explored this territory. They include not only formalized metatheory but also verified (non)termination and (non)confluence checkers built on it.

SAT solving also lends itself particularly well to formalization. Marić [59, 60] verified a CDCL-based SAT solver in Isabelle/HOL, including watched literals, as a purely functional program. He also formalized the abstract CDCL calculus by Nieuwenhuis et al. and, together with Janičić [61], the more implementation-oriented calculus by Krstić and Goel [49]. As a milestone towards verified SMT solvers, Spasić and Marić [93] formalized the simplex algorithm in Isabelle. Thiemann extended this work to support incremental solving and provide unsatisfiable cores [62]. In Coq, Lescuyer [58] formalized the CDCL calculus and the core of an SMT solver. Another verification of a CDCL-based SAT solver, including termination but excluding two watched literals, is by Shankar and Vaucher [91] in PVS. Finally, Oe et al. [71] used Guru to specify and verify `versat`. The generated C program consists of 15 000 lines of code. Optimized data structures are implemented. However, termination is not guaranteed, and model soundness is established through a run-time check.

A pragmatic approach to obtain trustworthy unsatisfiability judgments from a SAT solver is to have it produce a certificate, which can be given to an independent checker. An efficient format for this is DRAT [42]. The standard DRAT checker is an unverified C program, but verified checkers have now been developed by Cruz-Filipe et al. [28] and Lammich [54] using ACL2, Coq, and Isabelle/HOL.

I alluded, in the introduction, to the self-referential thrill of formalizing theorem provers. Barras [5] took this idea to its logical extreme with his "Coq in Coq" Ph.D. project: a verification in Coq of a type checker for the calculus of inductive constructions underlying Coq. Analogously, Harrison [40] verified HOL Light's inference kernel in HOL Light. To circumvent the impossibility of defining higher-order logic's semantics in itself, he carried out two distinct proofs: one where the formalized logic has no infinity axiom, and one where HOL Light is extended with an axiom to increase its strength. This formalization was ported to HOL4 and extended by Kumar et al. [50] to include definitional mechanisms and to exploit CakeML [51], a verified ML environment. In another line of work, Davis built an ACL2-style prover called Milawa [29]. The development consists of a stack of provers, each used to verify the one above it. Together with Davis, Myreen [67] connected Milawa to a verified Lisp implementation [66] that was developed for hosting Milawa. A noteworthy feature of the prover is its `switch` command, which lets the user replace the inference kernel by an arbitrary kernel that has been proved sound, enabling powerful, highly optimized extensions that would be impossible using a traditional LCF architecture [37].

6 Conclusion

In this paper, I reported on some of the steps my colleagues and I have taken to help drive the adoption of proof assistants in the automated reasoning community. Far from following a definite plan, at every turn we focused on topics that appealed to us and for which we could perceive clear value in formalization. We have barely scratched the surface; a lot of exciting work still awaits us.

Automated reasoning is near ideal territory for proof assistants. Compared with other application areas, the proof obligations are manageable, and little background theory needs to be formalized before we can get started. Conveniently, researchers in the area are not afraid of logic, although they often lack familiarity with proof assistants and their higher-order formalisms.

Isabelle/HOL has been a very suitable vehicle for this kind of work, and we will continue using it. It is comparatively easy to use, has a simple but expressive logic, is based on a trustworthy LCF-style inference kernel, and includes rich libraries developed by a large, and growing, user base.

Acknowledgments

I am grateful to all contributors to the IsaFoL effort, listed in the introduction, and to our colleagues Maximilian Kirchmeier, Christian Sternagel, Uwe Waldmann, Daniel Wand, and Christoph Weidenbach. This paper would not have existed without the dedicated labor of Heiko Becker, Alexander Bentkamp, Mathias Fleury, Peter Lammich, Anders Schlichtkrull, and Dmitriy Traytel; and the research program might never have started without the early support and advice of Tobias Nipkow, Andrei Popescu, Dmitriy Traytel, Jørgen Villadsen, and Christoph Weidenbach. Discussions with Franz Baader, Carsten Fuhs, Marijn Heule, Cynthia Kop, Leonardo de Moura, Michael Norrish, Lawrence Paulson, Giles Reger, Stephan Schulz, Natarajan Shankar, Cesare Tinelli, Andrei Voronkov, and many others have influenced this research. The Isabelle development team and the editors of the *Archive of Formal Proofs* have made the work possible.

I thank Assia Mahboubi and Magnus Myreen for the invitation to write in these pages and for their useful comments. Alexander Bentkamp, Mathias Fleury, Pascal Fontaine, Robert Lewis, Anders Schlichtkrull, Mark Summerfield, Sophie Tourret, and Dmitriy Traytel have suggested many textual improvements. I should also acknowledge that parts of the text have appeared in a similar form in my earlier papers [13, 22, 33, 86, 87].

The research has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No. 713999, Matryoshka). My work was also partially supported by a Deutsche Forschungsgemeinschaft (DFG) project (grant Ni 491/14-1, Hardening the Hammer).

References

[1] Gilles Audemard and Laurent Simon. 2009. Predicting Learnt Clauses Quality in Modern SAT Solvers. In *IJCAI 2009*, Craig Boutilier (Ed.). ijcai.org, 399–404.

[2] Franz Baader and Tobias Nipkow. 1998. *Term Rewriting and All That*. Cambridge University Press.

[3] Leo Bachmair and Harald Ganzinger. 2001. Resolution Theorem Proving. In *Handbook of Automated Reasoning*, Alan Robinson and Andrei Voronkov (Eds.). Vol. I. Elsevier and MIT Press, 19–99.

[4] Clemens Ballarin. 2014. Locales: A Module System for Mathematical Theories. *J. Autom. Reasoning* 52, 2 (2014), 123–153.

[5] Bruno Barras. 1999. *Auto-validation d'un système de preuves avec familles inductives*. Ph.D. Dissertation. Université Paris 7.

[6] Heiko Becker, Jasmin Christian Blanchette, Uwe Waldmann, and Daniel Wand. 2017. A Transfinite Knuth–Bendix Order for Lambda-Free Higher-Order Terms. In *CADE-26*, Leonardo de Moura (Ed.). LNCS, Vol. 10395. Springer, 432–453.

[7] Alexander Bentkamp. 2018. Formalization of the Embedding Path Order for Lambda-Free Higher-Order Terms. *Archive of Formal Proofs* (2018). http://isa-afp.org/entries/Lambda_Free_EPO.html.

[8] Alexander Bentkamp, Jasmin Christian Blanchette, Simon Cruanes, and Uwe Waldmann. 2018. Superposition for Lambda-Free Higher-Order Logic. In *IJCAR 2018*, Didier Galmiche, Stephan Schulz, and Roberto Sebastiani (Eds.). LNCS, Vol. 10900. Springer, 28–46.

[9] Stefan Berghofer. 2007. First-Order Logic According to Fitting. *Archive of Formal Proofs* (2007). http://isa-afp.org/entries/FOL-Fitting.html.

[10] Julian Biendarra, Jasmin Christian Blanchette, Aymeric Bouzy, Martin Desharnais, Mathias Fleury, Johannes Hölzl, Ondřej Kunčar, Andreas Lochbihler, Fabian Meier, Lorenz Panny, Andrei Popescu, Christian Sternagel, René Thiemann, and Dmitriy Traytel. 2017. Foundational (Co)datatypes and (Co)recursion for Higher-Order Logic. In *FroCoS 2017*, Clare Dixon and Marcelo Finger (Eds.). LNCS, Vol. 10483. Springer, 3–21.

[11] Armin Biere. 2017. CaDiCaL, Lingeling, Plingeling, Treengeling, YalSAT Entering the SAT Competition 2017. In *SAT Competition 2017: Solver and Benchmark Descriptions*, Tomáš Balyo, Marijn Heule, and Matti Järvisalo (Eds.). University of Helsinki, 14–15.

[12] Nikolaj Bjø[r]ner, Giles Reger, Martin Suda, and Andrei Voronkov. 2016. AVATAR Modulo Theories. In *GCAI 2016*, Christoph Benzmüller, Geoff Sutcliffe, and Raúl Rojas (Eds.). EPiC Series in Computing, Vol. 41. EasyChair, 39–52.

[13] Jasmin Christian Blanchette, Mathias Fleury, Peter Lammich, and Christoph Weidenbach. 2018. A Verified SAT Solver Framework with Learn, Forget, Restart, and Incrementality. *J. Autom. Reasoning* 61, 3 (2018), 333–366.

[14] Jasmin Christian Blanchette, Mathias Fleury, and Dmitriy Traytel. 2017. Nested Multisets, Hereditary Multisets, and Syntactic Ordinals in Isabelle/HOL. In *FSCD 2017*, Dale Miller (Ed.). LIPIcs, Vol. 84. Schloss Dagstuhl—Leibniz-Zentrum für Informatik, 11:1–11:18.

[15] Jasmin Christian Blanchette, Mathias Fleury, and Christoph Weidenbach. 2016. A Verified SAT Solver Framework with Learn, Forget, Restart, and Incrementality. In *IJCAR 2016*, Nicola Olivetti and Ashish Tiwari (Eds.). LNCS, Vol. 9706. Springer, 25–44.

[16] Jasmin Christian Blanchette and Tobias Nipkow. 2010. Nitpick: A Counterexample Generator for Higher-Order Logic Based on a Relational Model Finder. In *ITP 2010*, Matt Kaufmann and Lawrence C. Paulson (Eds.). LNCS, Vol. 6172. Springer, 131–146.

[17] Jasmin Christian Blanchette and Andrei Popescu. 2013. Mechanizing the Metatheory of Sledgehammer. In *FroCoS 2013*, Pascal Fontaine, Christophe Ringeissen, and Renate A. Schmidt (Eds.). LNCS, Vol. 8152. Springer, 245–260.

[18] Jasmin Christian Blanchette and Andrei Popescu. 2013. Sound and Complete Sort Encodings for First-Order Logic. *Archive of Formal Proofs* (2013). http://isa-afp.org/entries/Sort_Encodings.html.

[19] Jasmin Christian Blanchette, Andrei Popescu, and Dmitriy Traytel. 2017. Abstract Completeness. *Archive of Formal Proofs* (2017). http://isa-afp.org/entries/Abstract_Completeness.html.

[20] Jasmin Christian Blanchette, Andrei Popescu, and Dmitriy Traytel. 2017. Abstract Soundness. *Archive of Formal Proofs* (2017). http://isa-afp.org/entries/Abstract_Soundness.html.

[21] Jasmin Christian Blanchette, Andrei Popescu, and Dmitriy Traytel. 2017. Soundness and Completeness Proofs by Coinductive Methods. *J. Autom. Reasoning* 58, 1 (2017), 149–179.

[22] Jasmin Christian Blanchette, Uwe Waldmann, and Daniel Wand. 2017. A Lambda-Free Higher-Order Recursive Path Order. In *FoSSaCS 2017*, Javier Esparza and Andrzej S. Murawski (Eds.). LNCS, Vol. 10203. Springer, 461–479.

[23] Frédéric Blanqui and Adam Koprowski. 2011. CoLoR: A Coq Library on Well-founded Rewrite Relations and Its Application to the Automated Verification of Termination Certificates. *Math. Struct. Comput. Sci.* 21, 4 (2011), 827–859.

[24] Marc Brockschmidt, Sebastiaan J. C. Joosten, René Thiemann, and Akihisa Yamada. 2017. Analyzing Program Termination and Complexity Automatically with AProVE. LNCS, Vol. 10395. Springer, 454–471.

[25] Lukas Bulwahn. 2012. The New Quickcheck for Isabelle: Random, Exhaustive and Symbolic Testing under One Roof. In *CPP 2012*, Chris

Hawblitzel and Dale Miller (Eds.). LNCS, Vol. 7679. Springer, 92–108.

[26] Lukas Bulwahn, Alexander Krauss, Florian Haftmann, Levent Erkök, and John Matthews. 2008. Imperative Functional Programming with Isabelle/HOL. In *TPHOLs 2008*, Otmane Aït Mohamed, César A. Muñoz, and Sofiène Tahar (Eds.). LNCS, Vol. 5170. Springer, 134–149.

[27] Évelyne Contejean, Pierre Courtieu, Julien Forest, Olivier Pons, and Xavier Urbain. 2011. Automated Certified Proofs with CiME3. In *RTA '11*, Manfred Schmidt-Schauß (Ed.). LIPIcs, Vol. 10. Schloss Dagstuhl—Leibniz-Zentrum für Informatik, 21–30.

[28] Luís Cruz-Filipe, Marijn J. H. Heule, Warren A. Hunt Jr., Matt Kaufmann, and Peter Schneider-Kamp. 2017. Efficient Certified RAT Verification. In *CADE-26*, Leonardo de Moura (Ed.). LNCS, Vol. 10395. Springer, 220–236.

[29] Jared C. Davis. 2009. *A Self-Verifying Theorem Prover.* Ph.D. Dissertation. University of Texas at Austin.

[30] Nachum Dershowitz and Zohar Manna. 1979. Proving Termination with Multiset Orderings. *Commun. ACM* 22, 8 (1979), 465–476.

[31] Niklas Eén and Niklas Sörensson. 2003. An Extensible SAT-Solver. In *SAT 2003*, Enrico Giunchiglia and Armando Tacchella (Eds.). LNCS, Vol. 2919. Springer, 502–518.

[32] Arnaud Fietzke and Christoph Weidenbach. 2009. Labelled Splitting. *Ann. Math. Artif. Intell.* 55, 1-2 (2009), 3–34.

[33] Mathias Fleury, Jasmin Christian Blanchette, and Peter Lammich. 2018. A Verified SAT Solver with Watched Literals using Imperative HOL. In *CPP 2018*, June Andronick and Amy P. Felty (Eds.). ACM, 158–171.

[34] Andreas Halkjær From. 2017. First-Order Logic According to Berghofer. *Isabelle Formalization of Logic* (2017). https://bitbucket.org/isafol/isafol/src/master/FOL_Berghofer/.

[35] Andreas Halkjær From. 2018. Epistemic Logic. *Archive of Formal Proofs* (2018). http://isa-afp.org/entries/Epistemic_Logic.shtml.

[36] Andreas Halkjær From, John Bruntse Larsen, Anders Schlichtkrull, and Jørgen Villadsen. 2018. Substitutionless First-Order Logic: A Formal Soundness Proof. In *Isabelle Workshop 2018*, Makarius Wenzel Tobias Nipkow, Larry Paulson (Ed.).

[37] Mike Gordon. 2000. From LCF to HOL: A Short History. In *Proof, Language, and Interaction, Essays in Honour of Robin Milner*, Gordon D. Plotkin, Colin Stirling, and Mads Tofte (Eds.). MIT Press.

[38] Florian Haftmann and Tobias Nipkow. 2010. Code Generation via Higher-Order Rewrite Systems. In *FLOPS 2010*, Matthias Blume, Naoki Kobayashi, and Germán Vidal (Eds.). LNCS, Vol. 6009. Springer, 103–117.

[39] John Harrison. 1998. Formalizing Basic First Order Model Theory. In *TPHOLs '98*, Jim Grundy and Malcolm C. Newey (Eds.). LNCS, Vol. 1479. Springer, 153–170.

[40] John Harrison. 2006. Towards Self-Verification of HOL Light. In *IJCAR 2006*, Ulrich Furbach and Natarajan Shankar (Eds.). LNCS, Vol. 4130. Springer, 177–191.

[41] John Harrison. 2009. *Handbook of Practical Logic and Automated Reasoning.* Cambridge University Press.

[42] Marijn Heule, Warren A. Hunt Jr., and Nathan Wetzler. 2014. Bridging the Gap between Easy Generation and Efficient Verification of Unsatisfiability Proofs. *Softw. Test. Verif. Reliab.* 24, 8 (2014), 593–607.

[43] Gérard Huet and Derek C. Oppen. 1980. Equations and Rewrite Rules: A Survey. In *Formal Language Theory: Perspectives and Open Problems*, R. V. Book (Ed.). Academic Press, 349–405.

[44] Brian Huffman and Ondřej Kunčar. 2013. Lifting and Transfer: A Modular Design for Quotients in Isabelle/HOL. In *CPP 2013*, Georges Gonthier and Michael Norrish (Eds.). LNCS, Vol. 8307. Springer, 131–146.

[45] Joe Hurd. 2003. First-Order Proof Tactics in Higher-Order Logic Theorem Provers. In *Design and Application of Strategies/Tactics in Higher Order Logics*, Myla Archer, Ben Di Vito, and César Muñoz (Eds.). Number CP-2003-212448 in NASA Technical Reports. 56–68.

[46] Alexander Birch Jensen, John Bruntse Larsen, Anders Schlichtkrull, and Jørgen Villadsen. 2018. Programming and Verifying a Declarative First-Order Prover in Isabelle/HOL. *AI Commun.* 31, 3 (2018), 281–299.

[47] Alexander Birch Jensen, Anders Schlichtkrull, and Jørgen Villadsen. 2017. First-Order Logic According to Harrison. *Archive of Formal Proofs* (2017). http://isa-afp.org/entries/FOL_Harrison.html.

[48] Alexander Krauss. 2010. Recursive Definitions of Monadic Functions. *EPTCS* 43 (2010), 1–13.

[49] Sava Krstić and Amit Goel. 2007. Architecting Solvers for SAT Modulo Theories: Nelson-Oppen with DPLL. In *FroCoS 2007*, Boris Konev and Frank Wolter (Eds.). LNCS, Vol. 4720. Springer, 1–27.

[50] Ramana Kumar, Rob Arthan, Magnus O. Myreen, and Scott Owens. 2016. Self-Formalisation of Higher-Order Logic: Semantics, Soundness, and a Verified Implementation. *J. Autom. Reasoning* 56, 3 (2016), 221–259.

[51] Ramana Kumar, Magnus O. Myreen, Michael Norrish, and Scott Owens. 2014. CakeML: A Verified Implementation of ML. In *POPL 2014*, Suresh Jagannathan and Peter Sewell (Eds.). ACM, 179–192.

[52] Peter Lammich. 2013. Automatic Data Refinement. In *ITP 2013*, Sandrine Blazy, Christine Paulin-Mohring, and David Pichardie (Eds.). LNCS, Vol. 7998. Springer, 84–99.

[53] Peter Lammich. 2015. Refinement to Imperative/HOL. In *ITP 2015*, Christian Urban and Xingyuan Zhang (Eds.). LNCS, Vol. 9236. Springer, 253–269.

[54] Peter Lammich. 2017. Efficient Verified (UN)SAT Certificate Checking. In *CADE-26*, Leonardo de Moura (Ed.). LNCS, Vol. 10395. Springer, 237–254.

[55] Peter Lammich. 2017. The GRAT Tool Chain—Efficient (UN)SAT Certificate Checking with Formal Correctness Guarantees. In *SAT 2017*, Serge Gaspers and Toby Walsh (Eds.). LNCS, Vol. 10491. Springer, 457–463.

[56] Javier Larrosa, Robert Nieuwenhuis, Albert Oliveras, and Enric Rodríguez-Carbonell. 2009. Branch and Bound for Boolean Optimization and the Generation of Optimality Certificates. In *SAT 2009*, Oliver Kullmann (Ed.). LNCS, Vol. 5584. Springer, 453–466.

[57] Javier Larrosa, Robert Nieuwenhuis, Albert Oliveras, and Enric Rodríguez-Carbonell. 2011. A Framework for Certified Boolean Branch-and-Bound Optimization. *J. Autom. Reasoning* 46, 1 (2011), 81–102.

[58] Stephane Lescuyer. 2011. *Formalizing and Implementing a Reflexive Tactic for Automated Deduction in Coq.* Ph.D. Dissertation. Université Paris-Sud.

[59] Filip Marić. 2008. Formal Verification of Modern SAT Solvers. *Archive of Formal Proofs* (2008). http://isa-afp.org/entries/SATSolverVerification.html.

[60] Filip Marić. 2010. Formal Verification of a Modern SAT Solver by Shallow Embedding into Isabelle/HOL. *Theor. Comput. Sci.* 411, 50 (2010), 4333–4356.

[61] Filip Marić and Predrag Janičić. 2011. Formalization of Abstract State Transition Systems for SAT. *Log. Meth. Comput. Sci.* 7, 3 (2011).

[62] Filip Marić, Mirko Spasić, and René Thiemann. 2018. An Incremental Simplex Algorithm with Unsatisfiable Core Generation. *Archive of Formal Proofs* (2018). http://isa-afp.org/entries/Simplex.shtml.

[63] Julius Michaelis and Tobias Nipkow. 2017. Proof Systems for Propositional Logic. *Archive of Formal Proofs* (2017). http://isa-afp.org/entries/Propositional_Proof_Systems.html.

[64] Julius Michaelis and Tobias Nipkow. 2018. Formalized Proof Systems for Propositional Logic. In *TYPES 2017*, Andreas Abel, Fredrik Nordvall Forsberg, and Ambrus Kaposi (Eds.). LIPIcs, Vol. 104. Schloss Dagstuhl—Leibniz-Zentrum für Informatik, 6:1–6:16.

[65] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. 2001. Chaff: Engineering an Efficient SAT Solver. In *DAC 2001*. ACM, 530–535.

[66] Magnus O. Myreen and Jared Davis. 2011. A Verified Runtime for a Verified Theorem Prover. In *ITP 2011*, Marko C. J. D. van Eekelen, Herman Geuvers, Julien Schmaltz, and Freek Wiedijk (Eds.). LNCS, Vol. 6898. Springer, 265–280.

[67] Magnus O. Myreen and Jared Davis. 2014. The Reflective Milawa Theorem Prover Is Sound (Down to the Machine Code That Runs It). In *ITP 2014*, Gerwin Klein and Ruben Gamboa (Eds.). LNCS, Vol. 8558. Springer, 421–436.

[68] Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. 2006. Solving SAT and SAT Modulo Theories: From an Abstract Davis–Putnam–Logemann–Loveland Procedure to DPLL(T). *J. ACM* 53, 6 (2006), 937–977.

[69] Tobias Nipkow. 2012. Teaching Semantics with a Proof Assistant: No More LSD Trip Proofs. In *VMCAI 2012*, Viktor Kuncak and Andrey Rybalchenko (Eds.). LNCS, Vol. 7148. Springer, 24–38.

[70] Russell O'Connor. 2005. Essential Incompleteness of Arithmetic Verified by Coq. In *TPHOLs 2005*, Joe Hurd and Thomas F. Melham (Eds.). LNCS, Vol. 3603. Springer, 245–260.

[71] Duckki Oe, Aaron Stump, Corey Oliver, and Kevin Clancy. 2012. versat: A Verified Modern SAT Solver. In *VMCAI 2012*, Viktor Kuncak and Andrey Rybalchenko (Eds.). LNCS, Vol. 7148. Springer, 363–378.

[72] Peter W. O'Hearn. 2018. Experience Developing and Deploying Concurrency Analysis at Facebook. In *SAS 2018*, Andreas Podelski (Ed.). LNCS, Vol. 11002. Springer, 56–70.

[73] Lawrence C. Paulson. 2013. Gödel's Incompleteness Theorems. *Archive of Formal Proofs* (2013). http://isa-afp.org/entries/Incompleteness.html.

[74] Lawrence C. Paulson. 2015. A Mechanised Proof of Gödel's Incompleteness Theorems using Nominal Isabelle. *J. Autom. Reasoning* 55, 1 (2015), 1–37.

[75] Lawrence C. Paulson and Jasmin Christian Blanchette. 2012. Three Years of Experience with Sledgehammer, a Practical Link Between Automatic and Interactive Theorem Provers. In *IWIL-2010*, Geoff Sutcliffe, Stephan Schulz, and Eugenia Ternovska (Eds.). EPiC Series in Computing, Vol. 2. EasyChair, 1–11.

[76] Nicolas Peltier. 2016. Propositional Resolution and Prime Implicates Generation. *Archive of Formal Proofs* (2016). http://www.isa-afp.org/entries/PropResPI.html.

[77] Nicolas Peltier. 2016. A Variant of the Superposition Calculus. *Archive of Formal Proofs* (2016). http://isa-afp.org/entries/SuperCalc.html.

[78] Henrik Persson. 1996. *Constructive Completeness of Intuitionistic Predicate Logic: A Formalisation in Type Theory*. Licentiate Thesis. Chalmers tekniska högskola and Göteborgs universitet.

[79] Benjamin C. Pierce. 2009. Lambda, the Ultimate TA: Using a Proof Assistant to Teach Programming Language Foundations. In *ICFP 2009*, Graham Hutton and Andrew P. Tolmach (Eds.). ACM, 121–122.

[80] Giles Reger, Martin Suda, and Andrei Voronkov. 2015. Playing with AVATAR. In *CADE-25*, Amy P. Felty and Aart Middeldorp (Eds.). LNCS, Vol. 9195. Springer, 399–415.

[81] Tom Ridge. 2004. A Mechanically Verified, Efficient, Sound and Complete Theorem Prover For First Order Logic. *Archive of Formal Proofs* (2004). http://isa-afp.org/entries/Verified-Prover.shtml.

[82] Tom Ridge and James Margetson. 2005. A Mechanically Verified, Sound and Complete Theorem Prover for First Order Logic. In *TPHOLs 2005*, Joe Hurd and Tom Melham (Eds.). LNCS, Vol. 3603. Springer, 294–309.

[83] Colin Runciman, Matthew Naylor, and Fredrik Lindblad. 2008. SmallCheck and Lazy SmallCheck: Automatic Exhaustive Testing for Small Values. In *Haskell 2008*, Andy Gill (Ed.). ACM, 37–48.

[84] Anders Schlichtkrull. 2016. The Resolution Calculus for First-Order Logic. *Archive of Formal Proofs* (2016). http://www.isa-afp.org/entries/Resolution_FOL.html.

[85] Anders Schlichtkrull. 2018. Formalization of the Resolution Calculus for First-Order Logic. *J. Autom. Reasoning* 61, 1–4 (2018), 455–484.

[86] Anders Schlichtkrull, Jasmin Christian Blanchette, and Dmitriy Traytel. 2018. *A Verified Prover Based on Ordered Resolution.* Unpublished Draft. http://matryoshka.gforge.inria.fr/pubs/fun_rp_paper.pdf.

[87] Anders Schlichtkrull, Jasmin Christian Blanchette, Dmitriy Traytel, and Uwe Waldmann. 2018. Formalizing Bachmair and Ganzinger's Ordered Resolution Prover. In *IJCAR 2018*, Didier Galmiche, Stephan Schulz, and Roberto Sebastiani (Eds.). LNCS, Vol. 10900. Springer, 89–107.

[88] Anders Schlichtkrull and Jørgen Villadsen. 2016. Paraconsistency. *Archive of Formal Proofs* (2016). http://www.isa-afp.org/entries/Paraconsistency.html.

[89] Stephan Schulz. 2002. E—a Brainiac Theorem Prover. *AI Commun.* 15, 2-3 (2002), 111–126.

[90] Natarajan Shankar. 1994. *Metamathematics, Machines, and Gödel's Proof.* Cambridge Tracts in Theoretical Computer Science, Vol. 38. Cambridge University Press.

[91] Natarajan Shankar and Marc Vaucher. 2011. The Mechanical Verification of a DPLL-Based Satisfiability Solver. *Electr. Notes Theor. Comput. Sci.* 269 (2011), 3–17.

[92] Mate Soos, Karsten Nohl, and Claude Castelluccia. 2009. Extending SAT Solvers to Cryptographic Problems. In *SAT 2009*, Oliver Kullmann (Ed.). LNCS, Vol. 5584. Springer, 244–257.

[93] Mirko Spasić and Filip Marić. 2012. Formalization of Incremental Simplex Algorithm by Stepwise Refinement. In *FM 2012*, Dimitra Giannakopoulou and Dominique Méry (Eds.). LNCS, Vol. 7436. Springer, 434–449.

[94] René Thiemann and Christian Sternagel. 2009. Certification of Termination Proofs using CeTA. In *TPHOLs 2009*, Stefan Berghofer, Tobias Nipkow, Christian Urban, and Makarius Wenzel (Eds.). LNCS, Vol. 5674. Springer, 452–468.

[95] Jørgen Villadsen and Anders Schlichtkrull. 2017. Formalizing a Paraconsistent Logic in the Isabelle Proof Assistant. *T. Large-Scale Data- and Knowledge-Centered Systems* 34 (2017), 92–122.

[96] Jørgen Villadsen, Anders Schlichtkrull, and Andreas Halkjær From. 2018. Simple Prover. *Isabelle Formalization of Logic* (2018). https://bitbucket.org/isafol/isafol/src/master/Simple_Prover/.

[97] Andrei Voronkov. 2014. AVATAR: The Architecture for First-Order Theorem Provers. In *CAV 2014*, Armin Biere and Roderick Bloem (Eds.). LNCS, Vol. 8559. Springer, 696–710.

[98] Petar Vukmirović, Jasmin Christian Blanchette, Simon Cruanes, and Stephan Schulz. 2018. *Extending a Brainiac Prover to Lambda-Free Higher-Order Logic (Technical Report)*. Technical Report. http://matryoshka.gforge.inria.fr/pubs/ehoh_report.pdf.

[99] Christoph Weidenbach. 2015. Automated Reasoning Building Blocks. In *Correct System Design: Symposium in Honor of Ernst-Rüdiger Olderog on the Occasion of His 60th Birthday*, Roland Meyer, André Platzer, and Heike Wehrheim (Eds.). LNCS, Vol. 9360. Springer, 172–188.

[100] Makarius Wenzel. 2007. Isabelle/Isar—a Generic Framework for Human-Readable Proof Documents. In *From Insight to Proof: Festschrift in Honour of Andrzej Trybulec*, Roman Matuszewski and Anna Zalewska (Eds.). Studies in Logic, Grammar, and Rhetoric, Vol. 10(23). University of Białystok.

[101] Hans Zantema. 2003. Termination. In *Term Rewriting Systems*, Marc Bezem, Jan Willem Klop, and Roel de Vrijer (Eds.). Cambridge Tracts in Theoretical Computer Science, Vol. 55. Cambridge University Press, 181–259.