EPiC Series in Computing

Volume 99, 2024, Pages 15–20

Proceedings of the 7th and 8th Vampire Workshop

The Spawns of the Saturation Framework – Extended Abstract

Sophie Tourret[1,2]

[1] Université de Lorraine, CNRS, Inria, LORIA, Nancy, France
sophie.tourret@inria.fr
[2] Max-Planck-Institut für Informatik, Saarland Informatics Campus, Saarbrücken, Germany

In 2001, Leo Bachmair and Harald Ganzinger authored "Resolution Theorem Proving", a chapter of the *Handbook of Automated Reasoning* [1]. In section 4, they introduce a framework for saturation-based theorem proving. It lifts the refutational completeness results of a calculus from static to dynamic, i.e., the fact that "a saturated unsatisfiable set necessarily includes $\bot$" (static) is enough to ensure that "$\bot$ is necessarily generated from an unsatisfiable set during the saturation under some fairness conditions" (dynamic). They also introduce RP, a resolution theorem prover for first-order logic, and show its dynamic completeness. However, they do not rely on their own framework for this proof because redundancy as provided by their framework is too restrictive in that it does not cover subsumption deletion, a technique used by all realistic provers, including RP. Moreover, their proof of completeness for RP mixes properties of the derivations, of the underlying calculus and of the groundings of this calculus at every step, so that adapting this proof is non-trivial.

Nevertheless, since then this chapter has been the primary reference for proving the completeness of saturation-based calculi. However, due to the non-modular nature of the proof of RP, most subsequent works stop at proving the static completeness of their calculi, remaining vague on how these transfer to the completeness of the actual implemented provers. In 2021, Uwe Waldmann, Simon Robillard, Jasmin Blanchette and myself proposed an extension of the framework [10] generalising the original one, so as to include subsumption deletion as a legitimate redundancy deletion operation. Our extended framework is modular in that it cleanly separates static completeness of a ground calculus, lifting of the calculus from ground to non-ground, lifting from static to dynamic completeness and lifting to the dynamic completeness of a concrete given-clause prover architecture built on top of the calculus.

Since then, our extended framework (subsequently denoted simply as *the framework*) has been verified in the Isabelle/HOL proof assistant and successfully used in various completeness proofs. In my invited talk I presented this formalisation as well as the different calculi whose completeness has been proved using the framework, highlighting relevant features of the framework along the way. Here, I will focus on the calculi and features of the framework. Indeed the mechanisation of the framework and of a refinement into variants of the given clause loop—the main loop of saturation-based theorem provers—have already been described in papers [6, 9].

The framework is organised in several layers. The first one is the so-called *ground* layer. It assumes given a set of formulas and a consequence relation. Beyond that, the underlying logic

does not matter. It models a calculus using a set of inferences and functions that provide the set of formulas (resp. inferences) redundant with respect to a given set of formulas (resp. inferences). Note that the set of inferences is really just a collection of tuples made of the premisses and conclusion of each inference. There is no notion of inference *rule* in the framework. If the calculus is proven statically complete, then its dynamic completeness can be obtained via the framework. The second layer, called *non-ground*, is defined on top of the first one. It only requires a set of inferences and grounding functions for formulas and inferences, describing how to over-approximate the ground calculus from the inferences at the non-ground level. Using the grounding functions, the framework defines a consequence relation as well as the redundancy functions at the non-ground level. From the static completeness of the ground calculus, the framework can then derive the static and dynamic completeness of the non-ground calculus. Finally, a labelling scheme is added on top of the calculi, to simulate the active and passive sets found in an abstract given clause prover. Transition rules covering the manipulations realised by provers are given in this layer and completeness of the prover under fairness assumptions can be derived from the static completeness of the ground layer's calculus.

Currently, the calculi that were proved complete using the framework can be organised in two main categories, depending on the logic they target: first-order or higher-order. In first-order logic there are:

Resolution This is the calculus underlying the RP prover. Its completeness proof is nothing new. The version using the framework is described in [9]. It was formalised in Isabelle/HOL as a sanity check for the formalisation of the framework.

Superposition The standard superposition calculus can also be integrated in the framework. Again, the result in itself is nothing new. Currently, the detailed version of the proof using the framework exists only as a draft in a private repository. It is being used to define an instance of the superposition calculus in Isabelle/HOL. The ground version of the calculus, with a proof of static completeness, is already publicly available on the IsaFoL repository[1] and should at some point move to the Isabelle *Archive of Formal Proofs*[2] along with the calculus for full first-order logic that is ongoing work.

Superposition with delayed unification This calculus is fairly recent [5] and it should be noted that its authors have no intersection with the framework's authors. The main difference between this calculus and standard superposition is that only the outermost part of the unification is performed at each inference, while the rest of the unification is added as constraints to the resolvent. However, these constraints differ from the ones used in constrained superposition in that they are not isolated from the rest of the clause. On the contrary, they appear as standard literals, later eligible for performing further inferences. The result is a calculus whose efficiency is currently far below that of standard superposition in first-order logic, but that achieves state-of-the-art performances when lifted to higher-order,[3] as demonstrated at the CADE ATP System Competition (CASC) in 2023.[4] This calculus is not yet formalised in Isabelle/HOL, but it could be done on top of the version of ground superposition previously described, that is available in IsaFoL.

The main features of the framework used by these calculi are

[1] https://bitbucket.org/isafol/isafol/src/master/Superposition_Calculus/

[2] https://www.isa-afp.org/

[3] Note that the higher-order version has not been proven complete.

[4] Vampire with this calculus received first place in the higher-order division at CASC 2023 (https://www.tptp.org/CASC/29/WWWFiles/DivisionSummary1.html)

- a well-founded tiebreaker ordering in the definition of redundancy over formulas at the non-ground level, which allows it to handle subsumption; and
- the use of a family of ground redundancy criteria upon which the non-ground calculus is lifted to handle the selection function.

Here is how the tiebreaker ordering $\prec$ affects the definition of redundancy over formulas. It replaces the standard lifted definition:

$$\mathsf{Red}_{\mathsf{F}}(N) = \{C \mid \forall D \in \mathcal{G}_{\mathsf{F}}(C).\ D \in \mathsf{Red}_{\mathsf{FG}}(\mathcal{G}_{\mathsf{Fset}}(N))\}$$

by

$$\mathsf{Red}_{\mathsf{F}}(N) = \{C \mid \forall D \in \mathcal{G}_{\mathsf{F}}(C).\ D \in \mathsf{Red}_{\mathsf{FG}}(\mathcal{G}_{\mathsf{Fset}}(N)) \vee \exists E \in N.\ E \prec C \wedge D \in \mathcal{G}_{\mathsf{F}}(E)\}.$$

In these statements, N is a set of formulas, C, D and E are formulas, $\mathcal{G}_{\mathsf{F}}$ denotes the grounding function over formulas and $\mathcal{G}_{\mathsf{Fset}}$ is its natural extension over sets of formulas. Finally $\mathsf{Red}_{\mathsf{FG}}$ provides the set of all formulas that are redundant with regard to a given set of formulas. The introduction of the tiebreaker $\prec$ ensures that, instead of only relying on the lifting of the redundancy at the ground level, a formula C is also considered redundant at the non-ground level if for each of its groundings D there is a clause in the set N that has the same grounding D and is smaller using $\prec$ than C. Note that it is possible to parametrise the tiebreaking order, so that for each of the groundings D in the above definition, a different $\prec$ relation is considered. This is unneeded for most variants of resolution and superposition but there are a few exceptions [10, Ex. 49] and so this is how it is done in the actual framework. Returning to subsumption, defined as "D subsumes C if there exists a substitution σ such that $D\sigma \subseteq C$", only the case where $D\sigma = C$ is problematic with the first definition of $\mathsf{Red}_{\mathsf{F}}$. To cover it, the tiebreaker ordering must be instantiated with the instantiation ordering (i.e., $D \prec C$ if there is σ such that $D = C\sigma$ and no γ such that $D\gamma = C$) which is well-founded in first-order logic.

The family of ground redundancy criteria is introduced as a means of avoiding the choice of a selection function to lift from the ground level, at a point where it is not possible to know which one is appropriate. Indeed the choice of the proper grounding of the non-ground selection function is only possible in the static case, because it depends of the saturated set. With the family, instead of having to guarantee fairness with regard to the appropriate (but unknown) grounding, we guarantee it with regard to every grounding, so there is no need to choose.

Another interesting feature, used only by superposition with delayed unification among the first-order calculi, is the definition of a non-standard grounding function that omits the extra constraints in the ground resolvent, so that the standard superposition calculus at the ground level can still be used as the ground layer of the framework for this calculus.

In higher-order logic and logics in between first- and higher-order, the following calculi were proved complete using the framework.

Boolean-free λ-free superposition (λfSup) This calculus [2], or rather, this family of calculi, can handle applied variables such as x in $x\,t\,t'$ where t and t' are arguments of x, and partial applications such as add in "add a" that is the binary addition with only one argument a. The main problem faced by these calculi is that term orderings are usually not compatible with arguments, i.e., $s \succ s'$ does not imply $s\,t \succ s'\,t$. To compensate this, the calculi allow some superpositions at variable positions, but otherwise behave like first-order superposition. In particular, inferences can only apply on *green* contexts, i.e., contexts that look first-order, as $g\,a$ in $f\,(g\,a)\,b$ but not as g. In addition, the congruence and extensionality axioms must be handled and whereas extensionality is kept as an axiom, congruence becomes a rule of the calculus.

Boolean-free λ-superposition (λSup) This calculus [4] is built on top of λfSup. It additionally handles λ-expressions such as $\lambda x.\, x\, a\, b$. A first consequence of this added expressiveness is that unification becomes infinitary. The notion of a most general unifier from first-order logic must be replaced by that of a possibly infinite family of unifiers that subsume all others. This is handled at the prover level by only pulling a finite amount of unifiers at a time and dovetailing this process with the standard given-clause loop operations. The framework can mirror this behaviour thanks to its flexible labelling scheme [6]. Another consequence of the presence of λ-terms is that terms may look very different before and after a unification. For example consider the term $s = x\, a\, b\, c$ and the substitution $\theta = \{x \mapsto \lambda y.\lambda y'.\lambda y''.\, y''\, (y'\, y)\}$. Then the term $s\theta$ is $c\, (b\, a)$. In this new term, a superposition at $b\, a$ is possible, while it is not directly possible in s. Such terms are called *fluid* and to make inferences on them possible, the calculus includes a rule for fluid superposition that adds an arbitrary context around a term before the unification. This context permits the appearance of arbitrary positions below a term, even if they are not immediately accessible, such as in s before applying θ.

Boolean λ-superposition (λoSup) This calculus [3] works in full higher-order logic. It builds upon λSup and a superposition variant for first-order logic with interpreted Booleans that was proved complete without using the framework [8]. It handles interpreted Booleans as the latter, including the symbols $\bot$, $\top$, $\neg$, $\wedge$, $\vee$, $\Longrightarrow$, $\simeq$, $\not\simeq$, $\forall$ and $\exists$. The outer layer of the formula stays in clausal form. Dedicated rules are added for simplifying the inner Boolean structure (e.g., to replace $\top \vee \bot$ with $\top$), and to hoist innermost Boolean terms as equational literals of the outermost disjunction. For example, given a formula $C[u]$ where u is a Boolean, one can infer $C[\bot] \vee u \simeq \top$. This in turn may allow for further inferences on u and simplifications in $C[\bot]$. As in the case of superposition with lambdas, it is also necessary to allow for inferences below an arbitrary context.

For all of these calculi, a pen-and-paper proof of refutational completeness with Henkin semantics[5] using the framework has been provided. There is no Isabelle/HOL formalisation.

Henkin semantics is a workaround for the incompleteness of higher-order logic with classical semantics. It makes it possible to have refutationally complete procedures for higher-order logic. A good explanation of this workaround can be found in Chapter 2 of Melvin Fitting's book *Types, Tableaus and Gödel's God* [7]. A noteworthy point is that this semantics requires the use of *Tarski* entailment, defined as "$M \models N$ if any model of M is a model of N" while the framework, at the non-ground level, provides results using *Herbrand* entailment, i.e., defined by lifting from the ground level. Fortunately, these entailments coincide for entailing the false formula, which is enough to fix the discrepancy [4].

The proofs of completeness of the higher-order calculi above all have a structure in three layers. First, a ground first-order calculus is proven statically complete (**GF**) by providing a method to generate a model for satisfiable formulas. Then a ground higher-order calculus that over-approximates the one in **GF** is considered (**GH**) and the model from **GF** is manually lifted to the **GH** level, proving the corresponding calculus complete. Finally, the framework provides the dynamic completeness of the non-ground higher-order calculus (**H**) from **GH**. For λfSup and λSup, the **GF** layer is the usual superposition calculus for ground first-order logic. For λoSup, the **GF** calculus is the one designed for first-order logic with interpreted Booleans [8].

In their completeness proofs, λfSup, λSup and λoSup use the two main features presented earlier. The "purifying" variants of λfSup use a non-standard grounding function on inferences,

[5] For λfSup, a simpler semantics resembling Henkin prestructures is used instead [2] thanks to the absence of λ-terms.

due to a purifying mechanism only available at **H**. The other calculi do not need this feature. In addition the following features of the framework are used:

- ignoring inferences from **H** to **GH**,
- mapping an inference from **GH** to several inferences from **H**, and
- allowing inferences with no premisses.

The first two features are simple consequences of the fact that the non-ground calculus in the framework only has to be an over-approximation of the ground one. Hence it is not a problem to have inferences that cannot be grounded, or to have a single ground inference covered by several non-ground inferences, as is the case for the superposition rule for λSup and λoSup. In these calculi, the superposition rule has two variants (Sup and FluidSup) in **H** but only one in **GH**. The only thing that matters is that all inferences from **GH** are covered by grounded inferences from **H**.

The last feature, allowing inferences with no premisses, is critical for handling extensionality in the three calculi since in **H** they use the axiom directly to ensures that $(\forall x.f\,x = g\,x) \to f = g$ for all functions f and g of the same type. This introduces new clauses as instances of this axiom that are subsequently used in derivations. Thus, it is necessary to introduce in **GH** all the groundings of these new clauses to be able to mirror the non-ground derivations, which is done via an inference rule without premisses.

In summary, the framework has allowed various researchers, including myself, to develop modular completeness proofs of resolution- and superposition-like calculi on paper and in Isabelle/HOL. The core of the argument is performed at the ground level and lifted to the non-ground level. Then, using the framework's tiebreaker ordering, (static) completeness of the calculus is extended to dynamic completeness of an abstract prover. This results in streamlined, more intelligible, and more complete completeness proofs.

Acknowledgments. Ahmed Bhayat, Michael Rawson and Johannes Schoisswohl gave advice on how best to represent their work. Alexander Bentkamp, Jasmin Blanchette and Uwe Waldmann suggested textual improvements and technical clarifications. I thank them all.

References

[1] Leo Bachmair and Harald Ganzinger. Resolution theorem proving. In *Handbook of Automated Reasoning*, pages 19–99. Elsevier and MIT Press, 2001.

[2] Alexander Bentkamp, Jasmin Blanchette, Simon Cruanes, and Uwe Waldmann. Superposition for lambda-free higher-order logic. *Log. Methods Comput. Sci.*, 17(2), 2021.

[3] Alexander Bentkamp, Jasmin Blanchette, Sophie Tourret, and Petar Vukmirovic. Superposition for higher-order logic. *J. Autom. Reason.*, 67(1):10, 2023.

[4] Alexander Bentkamp, Jasmin Blanchette, Sophie Tourret, Petar Vukmirovic, and Uwe Waldmann. Superposition with lambdas. *J. Autom. Reason.*, 65(7):893–940, 2021.

[5] Ahmed Bhayat, Michael Rawson, and Johannes Schoisswohl. Superposition with delayed unification. In *CADE*, Lecture Notes in Computer Science. Springer, 2023.

[6] Jasmin Blanchette, Qi Qiu, and Sophie Tourret. Verified given clause procedures. In *CADE*, Lecture Notes in Computer Science. Springer, 2023.

[7] Melvin Fitting. *Types, tableaus, and Gödel's god*, volume 12. Springer Science & Business Media, 2002.

[8] Visa Nummelin, Alexander Bentkamp, Sophie Tourret, and Petar Vukmirovic. Superposition with first-class booleans and inprocessing clausification. In *CADE*, volume 12699 of *Lecture Notes in Computer Science*, pages 378–395. Springer, 2021.

[9] Sophie Tourret and Jasmin Blanchette. A modular isabelle framework for verifying saturation provers. In *CPP*, pages 224–237. ACM, 2021.

[10] Uwe Waldmann, Sophie Tourret, Simon Robillard, and Jasmin Blanchette. A comprehensive framework for saturation theorem proving. In *IJCAR (1)*, volume 12166 of *Lecture Notes in Computer Science*, pages 316–334. Springer, 2020.