Mechanizing the Metatheory of Sledgehammer

Jasmin Christian Blanchette and Andrei Popescu

Fakultät für Informatik, Technische Universität München, Germany

Abstract. This paper presents an Isabelle/HOL formalization of recent research in automated reasoning: efficient encodings of sorts in unsorted first-order logic, as implemented in Isabelle's Sledgehammer proof tool. The formalization provides the general-purpose machinery to reason about formulas and models, emulating the theory of institutions. Quantifiers are represented using a nominal-like approach designed for interpreting syntax in semantic domains.

1 Introduction

Despite steady progress in the usability of proof assistants, paper proofs reign supreme in the automated reasoning community. Myreen and Davis's verification of an ACL2-like prover in HOL4 [17] and Harrison's partial self-verification of HOL Light [13] are exceptions rather than the rule. Important metamathematical results have been formalized (e.g., Shankar's Gödel proof [26]), but new research is still carried out almost exclusively on paper, with all the risks this entails.

This paper presents a formalization in Isabelle/HOL [18] of the proofs for translations from many-sorted to unsorted first-order logic (FOL). Claessen et al. [10] designed lightweight encodings that eliminate much of the clutter associated with traditional schemes. Blanchette et al. [3, 4] introduced even lighter encodings in a sequel. Central to these new encodings is the notion of monotonicity. Informally, a sort is monotonic if its domain can be extended with new elements without compromising satisfiability. Nonmonotonic sorts can be made monotonic by introducing protector functions or predicates, and monotonic sorts can be merged into a single sort.

Sorts are trivially monotonic in FOL without equality. The addition of interpreted equality makes it possible to encode upper cardinality bounds on the models, breaking monotonicity. Like other interesting semantic properties, monotonicity is undecidable but can often be inferred in practice. Monotonicity has many applications in theorem provers and model finders [5,10]. It is also roughly equivalent to smoothness, a criterion that arises when combining decision procedures in SMT solvers [28].

The Sledgehammer [19] proof tool for Isabelle/HOL relies on the monotonicity-based encodings to apply state-of-the-art unsorted provers to sorted problems. The tool translates interactive proof goals along with relevant lemmas and invokes the external automatic theorem provers to find proofs, which are reconstructed through Isabelle's inference kernel. Early versions of Sledgehammer relied on unsound sort encodings; as a result, they would often find spurious, unreconstructable proofs, which annoyed users and could conceal sound proofs. Whereas Sledgehammer reconstructs the external proofs, tools such as Monotonox [10] and the fully-automatic competition version of Isabelle [27] do not perform such checks; soundness is crucial for them.

The mechanization of the sort encodings fully covers the correctness proofs from Claessen et al. [10] and the monomorphic half of its sequel [3, 4], as well as a theorem by Bouillaguet et al. [9]. This formalization work arose from a desire to provide more solid assurance to this recent research. Even if the intuition is clear, a paper proof offers many opportunities for flaws, especially because of the variety of encodings.

The mechanization effort partly coincided with the development of the informal proofs [4]. The two proofs largely follow the same conventions, with one major difference: The core of the formal proof (Sections 3 to 5) assumes quantifier-free clausal normal form (CNF) rather than negation normal form (NNF). This reduces the exposure to name binders, which are notoriously difficult to reason about. The results are lifted to NNF using a classification theorem (Sections 6 to 8). This organization is reminiscent of the architecture of automatic reasoners that combine a clausifier and a CNF core.

Isabelle's higher-order logic (HOL) might not be as expressive as set or type theory, but it can cope with the statements and proofs of classical metatheorems (as shown by Harrison and others [2, 12, 25]) and practical results. The proof assistant offers many conveniences; two features have been particularly useful:

- *Locales* [1,15] parameterize theories over constants and assumptions, with the usual benefits associated with modularity. Locales are particularly suited to expressing logic translations abstractly as in the theory of institutions [11].
- A framework for *syntax with bindings* [23,24] eases reasoning about quantified formulas. It lies at the intersection of first-order nominal approaches [21] and higher-order abstract syntax [20]. The framework is designed specifically for interpreting syntax in semantic domains.

Locales have been part of Isabelle for many years and are widely used. The syntax with bindings is a newer addition; the current application is among the first case studies that feature it. The formal proofs are available online [6, 7].

Although sort encodings are the focus of this paper, our infrastructure is designed to be reusable for other applications of many-sorted FOL. Many important metatheories are awaiting formalization, such as the completeness of paramodulation and tableaux.

2 An Isabelle View of Logic Translations

The formalization covers a variety of translations, including not only the sort encodings but also clausification. The guiding principles, described below, originate from the theory of institutions; their Isabelle materialization relies on locales.

Institutions. A *logic* $\mathcal{L}$ provides a category of signatures $\mathcal{Sig}$ and, for each signature $\Sigma \in \mathcal{Sig}$, a set of sentences $\mathcal{Sen}(\Sigma)$, a class of structures (interpretations) $\mathcal{Str}(\Sigma)$, and a satisfaction relation $\models_\Sigma$ between structures and sentences. A signature morphism $k : \Sigma \to \Sigma'$ is equipped with a forward sentence translation $k : \mathcal{Sen}(\Sigma) \to \mathcal{Sen}(\Sigma')$ and a backward structure translation $\restriction_k : \mathcal{Str}(\Sigma') \to \mathcal{Str}(\Sigma)$. An *institution* is a logic whose signature morphisms enjoy the property that "truth is invariant under change of notation": $\mathcal{M}' \models_{\Sigma'} k\,\varphi \longleftrightarrow \mathcal{M}'\restriction_k \models_\Sigma \varphi$ for all $k : \Sigma \to \Sigma'$, $\mathcal{M}' \in \mathcal{Str}(\Sigma')$, and $\varphi \in \mathcal{Sen}(\Sigma)$.

A translation of $\mathcal{L}$-problems (sets of sentences) into $\mathcal{L}'$-problems consists of a function \$ between $\mathcal{L}$'s and $\mathcal{L}'$'s signature classes and, for each $\Sigma \in \mathrm{dom}(\$)$ and Σ-problem Φ, a sentence translation $\mathsf{enc}_\Phi : \mathcal{S}en(\Sigma) \to \mathcal{S}en(\Sigma^\$)$ and a set of axioms $\mathcal{A}x_\Phi \subseteq \mathcal{S}en(\Sigma^\$)$. The translation of Φ is defined as $\mathsf{enc}\ \Phi = \{\mathsf{enc}_\Phi\ \varphi \mid \varphi \in \Phi\} \cup \mathcal{A}x_\Phi$. Thus, $\mathcal{L}$-problems are mapped to $\mathcal{L}'$-problems by joining an elementwise translation and additional axioms. Given a class $\mathcal{C}$ of $\mathcal{L}$-problems, the translation is *sound* w.r.t. $\mathcal{C}$ if satisfiability of Φ implies satisfiability of $\mathsf{enc}\ \Phi$ for all $\Phi \in \mathcal{C}$, and *complete* if the converse holds.

The institution literature focuses on "uniform" encodings. For these, the sentence translation depends only on Φ's signature Σ, and there exists a backward translation $\mathsf{dec} : \mathcal{S}tr(\Sigma^\$) \to \mathcal{S}tr(\Sigma)$ for which an inter-institution version of the institutional condition holds: $\mathcal{M}' \models_{\Sigma^\$} \mathsf{enc}_\Sigma\ \varphi \longleftrightarrow \mathsf{dec}\ \mathcal{M}' \models_\Sigma \varphi$. This condition implies completeness.

The source logic $\mathcal{L}$ for all the translations considered in this paper is many-sorted FOL; the target logic $\mathcal{L}'$ is either many-sorted or unsorted FOL. Sentences are either CNF clauses or NNF formulas. Most of the translations are nonuniform.

Isabelle. Isabelle/HOL is based on polymorphic HOL, which can be thought of as a fragment of Standard ML enriched with logical constructs and a proof system. Type variables are identified by a leading prime (e.g., $'a$). The type $\sigma \to \tau$ is interpreted as the set of (total) functions from σ to τ. Propositions are terms of type bool, and predicates are functions to bool. Function applications are written without parentheses (e.g., $f\ x\ y$) or in infix notation (e.g., $x + y$). Constants and variables can be functions.

The type $'a\ \mathsf{list}$ of finite lists over $'a$ is generated freely from the empty list $[]$ and the infix constructor $\# : {'a} \to {'a}\ \mathsf{list} \to {'a}\ \mathsf{list}$. The notation $[x_1, x_2, \ldots, x_n]$ abbreviates $x_1 \# (x_2 \# (\cdots \# (x_n \# []) \cdots))$. The higher-order constant $\mathsf{map} : ({'a} \to {'b}) \to {'a}\ \mathsf{list} \to {'b}\ \mathsf{list}$ applies a unary function to each element in a list, and $\mathsf{set} : {'a}\ \mathsf{list} \to {'a}\ \mathsf{set}$ returns the set of elements in a list. Sets are written using traditional mathematical notation. Type parameters of polymorphic types are sometimes omitted (e.g., set for $'a\ \mathsf{set}$).

Locales. Isabelle locales are a structuring mechanism provided on top of basic HOL. They fix types, constants, and assumptions, as in the following schematic examples:

$\mathtt{locale}\ \mathsf{X} = \mathtt{fixes}\ {'a}\ \mathtt{fixes}\ \mathsf{c} : \sigma_{'a}\ \mathtt{assumes}\ P_{'a,\mathsf{c}}$
$\mathtt{locale}\ \mathsf{Y} = \mathtt{fixes}\ {'b}\ \mathtt{fixes}\ \mathsf{d} : \tau_{'b}\ \mathtt{assumes}\ Q_{'b,\mathsf{d}}$

The definition of locale X fixes a type $'a$ and a constant c whose type $\sigma_{'a}$ may depend on $'a$, and states an assumption $P_{'a,\mathsf{c}} : \mathsf{bool}$ over $'a$ and c. Lemmas proved within the locales can rely on them. In general, a single locale can introduce several types, constants, and assumptions. The definition of X also produces a polymorphic *locale predicate* $\mathsf{X} = (\lambda c.\ P_{'a,c})$. Seen from outside the locale, the lemmas proved in locale X are polymorphic in type variable $'a$, universally quantified over variable c, and conditional on $\mathsf{X}\ c$.

Locales support inheritance, union, and embedding. To embed X into Y, one needs to indicate how an arbitrary instance of X can be regarded as an instance of Y, by providing, in the context of X, definitions of the types and constants of Y together with proofs of Y's assumptions. The command

$\mathtt{sublocale}\ \mathsf{X} < \mathsf{Y}\ \mathtt{where}\ {'b} = \upsilon\ \mathtt{and}\ \mathsf{d} = t$

emits the goal $Q_{\upsilon,t}$, where υ and t may depend on types and constants from X. After the proof, all the lemmas proved in the Y become available in X, with υ and t in place of $'b$ and d. Homonymous constants d in X and Y are instantiated as $\mathsf{d} = \mathsf{d}$ by default.

The sublocale relationship is sometimes abbreviated to $\mathsf{X}_{'a,\mathsf{c}} < \mathsf{Y}_{v,t}$ or $\mathsf{X} < \mathsf{Y}$.

Locales provide a shallow realization of institutions in Isabelle. The institutional methodology serves as an inspiration and guidance to formulate results about specific logic translations in a consistent style. Given a logic $\mathcal{L}$, its signatures $\mathcal{Sig}$ are captured by a locale $\mathcal{L}$.Signature, which fixes Isabelle constants for the signature components (e.g., sorts and symbols) and defines a notion of sentence (e.g., clauses or formulas). A locale $\mathcal{L}$.Problem extends $\mathcal{L}$.Signature with a fixed set of sentences Φ. Structures $\mathcal{M}$ are represented by a locale $\mathcal{L}$.Structure that also defines a notion of satisfaction. Finally, satisfiable problems are represented by a locale $\mathcal{L}$.Model that joins $\mathcal{L}$.Problem and $\mathcal{L}$.Structure and further requires satisfaction between Φ and $\mathcal{M}$.

In this setting, translations between logics $\mathcal{L}$ and $\mathcal{L}'$ and their properties are captured via locale embedding mechanisms in four steps.

SIG: Define \$ as a sublocale relationship $\mathcal{L}$.Signature $<$ $\mathcal{L}'$.Signature with suitable parameter instantiations reflecting the definition of $\Sigma^{\$}$ in terms of Σ.

TRANS: Define enc_Φ inside $\mathcal{L}$.Problem (where Σ and the Σ-problem Φ are fixed).

SOUND: To prove soundness, define a $\Sigma^{\$}$-structure $\mathcal{M}'$ inside $\mathcal{L}$.Model (where the signature Σ, the Σ-problem Φ, and the structure $\mathcal{M}$ such that $\Phi \models_\Sigma \mathcal{M}$ are fixed) and show $\mathcal{L}.\mathsf{Model}_{\mathcal{M}} < \mathcal{L}'.\mathsf{Model}_{\mathcal{M}'}$.

COMPLETE: To prove completeness, define a locale Problem_Model$'$ = $\mathcal{L}$.Problem + $\mathcal{L}'$.Model that joins a Σ-problem Φ and a $\Sigma^{\$}$-model $\mathcal{M}'$ of enc Φ, define inside Problem_Model$'$ a Σ-structure $\mathcal{M}$, and show $\mathsf{Problem\_Model}'_{\mathcal{M}'} < \mathcal{L}.\mathsf{Model}_{\mathcal{M}}$.

3 Clausal First-Order Logic

The terms, atoms, and literals of (quantifier-free) CNF are represented in HOL by ML-style free datatypes, parameterized by types $'f$ and $'p$ of function and predicate symbols:

```
datatype 'f tm =           datatype ('f,'p) atm =       datatype ('f,'p) lit =
  Var var |                  Pr 'p ('f tm list) |         Pos (('f,'p) atm) |
  Fn 'f ('f tm list)         Eq ('f tm) ('f tm)           Neg (('f,'p) atm)
```

The type var is countably infinite. An atom is either an applied predicate (e.g., $p(t)$) or equality (e.g., $t \approx u$). A clause is a list of literals (interpreted disjunctively), and a problem is a set of clauses (interpreted conjunctively). Formally, $('f,'p)$ clause $=$ $('f,'p)$ lit list and $('f,'p)$ problem $=$ $('f,'p)$ clause set. The CNF representation involves no name binders, unlike (quantified) NNF (Section 6).

Many-sorted signatures (for CNF and NNF) are captured by the following locale:

```
locale Signature =
  fixes 's and 'f and 'p
  fixes arity_F : 'f → 's list and res : 'f → 's and arity_P : 'p → 's list
  assumes countable UNIV_'s and countable UNIV_'f and countable UNIV_'p
```

The locale is parameterized by types for sorts ($'s$), function symbols ($'f$), and predicate symbols ($'p$), all required to be countable (i.e. finite or countably infinite). The locale attaches to each symbol a sort arity ($\mathsf{arity_F}$ or $\mathsf{arity_P}$) and, for functions, a result sort (res). The sort arity can be empty. Symbols cannot be overloaded. The polymorphic constant $\mathsf{UNIV}_{'a} : {'a}$ set is predefined in Isabelle as the set of all values of type $'a$.

The Signature locale defines an underspecified function sort : var $\rightarrow$ $'s$ that arbitrarily assigns sorts to variables. Whereas the formalization consistently refers to FOL's sorts as types (in view of a possible extension to n-ary type constructors and polymorphism), in this paper they are more precisely called sorts. Wellsortedness and wellformedness of terms and the other syntactic categories are defined in the usual way. Wellformedness is a precondition to many operations, but such details are omitted here.

The Problem locale joins a signature Σ and a CNF Σ-problem Φ. The Structure locale combines a signature, a universe $'u$, and a triple of functions $(\mathsf{int_S}, \mathsf{int_F}, \mathsf{int_P})$ that interpret sorts, function symbols, and predicate symbols:

```
locale Problem = Signature$_{'s,'f,'p}$ arity$_F$ res arity$_P$ +
  fixes $\Phi$ : $('f, 'p)$ problem

locale Structure = Signature$_{'s,'f,'p}$ arity$_F$ res arity$_P$ +
  fixes $'u$
  fixes int$_S$ : $'s \rightarrow 'u \rightarrow$ bool and int$_F$ : $'f \rightarrow 'u$ list $\rightarrow 'u$ and
        int$_P$ : $'p \rightarrow 'u$ list $\rightarrow$ bool
```

A few wellformedness assumptions are made on the triple $(\mathsf{int_S}, \mathsf{int_F}, \mathsf{int_P})$, such as inhabitation of all sorts $(\forall\sigma.\ \exists d.\ \mathsf{int_S}\ \sigma\ d)$. The Structure locale also defines the interpretation of terms and satisfaction of clauses. A related locale, Model, represents satisfiable CNF problems by combining a Problem and a Structure it satisfies.

4 Monotonicity and Its Inference

This section focuses on monotonicity in its own right; Section 5 discusses the associated sort encodings. To simplify the monotonicity arguments, both sections assume a fixed infinitely countable type ω as the universe $'u$ of structures, thus working implicitly with the instances $\mathsf{Structure}_\omega$ and Model_ω. This limitation is lifted in Section 8 by appealing to the downward Löwenheim–Skolem theorem.

Claessen et al. [10, § 2] define monotonicity on single sorts. Blanchette et al. [3, § 3] generalized the notion to sets of sorts S, making it more useful. The sorts S are collectively *monotonic* in the problem Φ if for all models $\mathcal{M}$ of Φ, there exists a model $\mathcal{M}'$ such that for all sorts σ, $\mathcal{M}'$ interprets σ by an infinite domain if $\sigma \in S$ and by a domain of the same cardinality as in $\mathcal{M}$ otherwise.

In the formalization, the Mono_Problem locale enriches Problem with a monotonicity assumption on all sorts, expressed using locale predicates:

$$\big(\exists int_\mathsf{S}\ int_\mathsf{F}\ int_\mathsf{P}.\ \mathsf{Model}\ \mathsf{arity_F}\ \mathsf{res}\ \mathsf{arity_P}\ \Phi\ int_\mathsf{S}\ int_\mathsf{F}\ int_\mathsf{P}\big) \longrightarrow \\ \exists int_\mathsf{S}\ int_\mathsf{F}\ int_\mathsf{P}.\ \mathsf{Infinite\_Model}\ \mathsf{arity_F}\ \mathsf{res}\ \mathsf{arity_P}\ \Phi\ int_\mathsf{S}\ int_\mathsf{F}\ int_\mathsf{P}$$

The Infinite_Model locale is itself an enrichment of Model with the assumption that for each sort σ, the expression $\mathsf{int_S}\ \sigma\ d$ is true for infinitely many elements d.

First Criterion. Claessen et al. designed two syntactic criteria to infer monotonicity. The first one is defined as a predicate $\rhd$ that checks the absence of naked variables of a given sort σ in a clause c or a problem Φ:

$$\sigma \rhd c \longleftrightarrow \forall x \in \mathsf{nv}\ c.\ \mathsf{sort}\ x \neq \sigma \qquad\qquad \sigma \rhd \Phi \longleftrightarrow \forall c \in \Phi.\ \sigma \rhd c$$

A *naked variable* is a variable that occurs directly on either side of a positive equality, such as X in the literal $X \approx \mathsf{f}(Y)$. Formally:

$$\begin{array}{lll} \mathsf{nv}\,(\mathsf{Var}\,x) = \{x\} & \mathsf{nv}\,(\mathsf{Eq}\,t_1\,t_2) = \mathsf{nv}\,t_1 \cup \mathsf{nv}\,t_2 & \mathsf{nv}\,(\mathsf{Pos}\,a) = \mathsf{nv}\,a \\ \mathsf{nv}\,(\mathsf{Fn}\,f\,ts) = \emptyset & \mathsf{nv}\,(\mathsf{Pr}\,p\,ts) = \emptyset & \mathsf{nv}\,(\mathsf{Neg}\,a) = \emptyset \end{array}$$

with $\mathsf{nv}\,c = \bigcup\,\mathsf{set}\,(\mathsf{map}\,\mathsf{nv}\,c)$ for clauses. The criterion $\rhd$ soundly infers monotonicity. This is expressed as a sublocale inclusion Problem_Crit1 < Mono_Problem, where Problem_Crit1 enriches Problem with the assumption $\forall\sigma.\ \sigma \rhd \Phi$. The inclusion holds because a model of a problem whose sorts pass $\rhd$ can be extended into an infinite model by replicating elements. For each finite sort σ, the extended model contains infinitely many copies of some element pick σ, all interpreted as in the original model.

Blanchette et al. strengthened the criterion by injecting "infinity knowledge": Any sort that is interpreted by an infinite domain in all models is monotonic, regardless of naked variables [3, §3]. This aspect is part of the formalization but omitted here.

Second Criterion. The improved criterion is parameterized by an assignment of a per-sort *extension policy*—which may be *true*, *false*, or *copy*—to each predicate symbol. In the model construction, the true-extended (resp. false-extended) predicates are interpreted as true (resp. false) for new domain elements of the given sort, whereas the copy-extended predicates are treated as in the simple criterion.

Implementations can enumerate the possible policy combinations (e.g., using a SAT solver). In the formalization, the policies are supplied along with the problem as a curried function policy that maps pairs σ, p to T, F, or C. A function guard associates each variable x in need of protection with its guarding literal. The criterion is defined as

$$\begin{array}{l} \sigma \blacktriangleright c \longleftrightarrow \forall l\,x.\ l \in \mathsf{set}\,c \wedge x \in \mathsf{nv}\,l \wedge \mathsf{sort}\,x = \sigma \longrightarrow \mathsf{isGuard}\,x\,(\mathsf{guard}\,c\,l\,x) \\ \sigma \blacktriangleright \Phi \longleftrightarrow \forall c \in \Phi.\ \sigma \blacktriangleright c \end{array}$$

where isGuard determines whether the given literal actually protects the variable x:

$$\begin{array}{rl} \mathsf{isGuard}\,x\,(\mathsf{Pos}\,(\mathsf{Eq}\,t_1\,t_2)) & \longleftrightarrow \mathsf{False} \\ \mathsf{isGuard}\,x\,(\mathsf{Neg}\,(\mathsf{Eq}\,t_1\,t_2)) & \longleftrightarrow \bigvee_{i=1}^{2} t_i = \mathsf{Var}\,x \wedge \exists f\,ts.\ t_{3-i} = \mathsf{Fn}\,f\,ts \\ \mathsf{isGuard}\,x\,(\mathsf{Pos}\,(\mathsf{Pr}\,p\,ts)) & \longleftrightarrow x \in \bigcup\,\mathsf{set}\,(\mathsf{map}\,\mathsf{nv}\,ts) \wedge \mathsf{policy}\,(\mathsf{sort}\,x)\,p = \mathsf{T} \\ \mathsf{isGuard}\,x\,(\mathsf{Neg}\,(\mathsf{Pr}\,p\,ts)) & \longleftrightarrow x \in \bigcup\,\mathsf{set}\,(\mathsf{map}\,\mathsf{nv}\,ts) \wedge \mathsf{policy}\,(\mathsf{sort}\,x)\,p = \mathsf{F} \end{array}$$

The notion of naked variables is generalized to account for ill-polarized predicates:

$$\begin{array}{rl} \mathsf{nv}\,(\mathsf{Pos}\,(\mathsf{Pr}\,p\,ts)) & = \{x \in \bigcup\,\mathsf{set}\,(\mathsf{map}\,\mathsf{nv}\,ts) \mid \mathsf{policy}\,(\mathsf{sort}\,x)\,p = \mathsf{F}\} \\ \mathsf{nv}\,(\mathsf{Neg}\,(\mathsf{Pr}\,p\,ts)) & = \{x \in \bigcup\,\mathsf{set}\,(\mathsf{map}\,\mathsf{nv}\,ts) \mid \mathsf{policy}\,(\mathsf{sort}\,x)\,p = \mathsf{T}\} \end{array}$$

Theorem 1. *Let Φ be a Σ-problem and σ be a Σ-sort.*

(1) *If $\sigma \rhd \Phi$, then $\sigma \blacktriangleright \Phi$ for a copy-extended policy.*

(2) *Given some extension policies, if $\sigma \blacktriangleright \Phi$ for all Σ-sorts σ, then the set of all Σ-sorts is monotonic in Φ.*

This theorem is expressed in Isabelle as a pair of sublocale inclusions. The `where` clause below instantiates Problem_Policy_Crit2's policy parameter with $\lambda\sigma\,p.\ \mathsf{C}$ to enforce the copy policy for all sorts and predicate symbols:

```
sublocale Problem_Crit1 < Problem_Policy_Crit2 where policy = (λσ p. C)
sublocale Problem_Policy_Crit2 < Mono_Problem
```

5 Sort Encodings

A naive, unsound way to translate a many-sorted FOL problem to unsorted FOL is to erase all the sorts and otherwise leave the problem unchanged. There are two main sound alternatives that encode the sort information. Sort *tags* are functions $\mathsf{t}_\sigma(X)$ that directly associate a term X with its sort σ. Sort *guards* are predicates $\mathsf{g}_\sigma(X)$ that check whether X has sort σ in the original problem. The formalized versions of these encodings follow the four steps sketched in Section 2.

Full Erasure. Full sort erasure is unsound but complete. What makes it interesting is that it is sound for the class of monotonic problems. By way of composition, it lies at the heart of the tag- and guard-based encodings. The theory prefix U distinguishes unsorted entities from their many-sorted counterparts.

SIG: The signature of the target unsorted problem has the same function and predicate symbols as the original signature but collapses the sorts into a single, implicit sort.

TRANS: The translation function e is the identity except that it forgets the sorts.

SOUND: For the soundness proof, a model of a monotonic problem is extended into a model that interprets all sorts infinitely, which in turn is transformed into an isomorphic "full" model that interprets all the sorts uniformly as $\lambda d.\,\mathsf{True}$ (i.e., $\forall\sigma.\ \forall d.\ \mathsf{int}_\mathsf{S}\ \sigma\ d$), from which it is easy to build an unsorted model for the e-translated problem:

$$\mathsf{Mono\_Model} < \mathsf{Infinite\_Model} < \mathsf{Full\_Model} < \mathsf{U.Model}$$

The last step corresponds to Theorem 1 in Bouillaguet et al. [9] and, more approximately, to Lemma 1 in Claessen et al. [10]. Incidentally, the formalization revealed a flaw in Claessen et al.: Their main result holds, but not their Lemma 3.[1]

COMPLETE: The locale Problem_UModel combines a many-sorted problem and an unsorted model with domain D of the problem's e translation. The unsorted model can be regarded as a many-sorted model in which every sort is interpreted as D.

Protector-Based Encodings. Claessen et al. observe that protectors, whether tags or guards, are not needed for terms with monotonic sorts. The sequel [3] advocates protecting only those variables that cause the monotonicity check to fail, to reduce clutter. Thus, for both tags and guards, three schemes are available: the traditional encoding, the lightweight version due to Claessen et al., and the "featherweight" version from the sequel. These are called $\widetilde{\mathsf{t}}$, $\widetilde{\mathsf{t}}?$, and $\widetilde{\mathsf{t}}??$ for tags and $\widetilde{\mathsf{g}}$, $\widetilde{\mathsf{g}}?$, and $\widetilde{\mathsf{g}}??$ for guards.

Consider the following fragment of a many-sorted problem, where S has sort st:

$$S \approx \mathsf{on} \lor S \approx \mathsf{off} \qquad\qquad \mathsf{flip}(S) \not\approx S$$

[1] The flawed lemma states that whenever there exists a model $\mathcal{M}$ where a monotonic sort σ is interpreted with a given cardinality, there exists for any larger cardinality k a model where σ has cardinality k and the other sorts have the same cardinalities as in $\mathcal{M}$. This proposition is invalid for $k > \aleph_0$ because FOL problems can encode the constraint that there exists a bijection between two infinite, and hence monotonic, sorts σ and τ, making it impossible to increase σ's cardinality without also increasing τ's. This issue is independent of which of the two definitions of monotonicity is used. We discovered it at an early stage of the formalization as we were looking for a correct formulation of Löwenheim–Skolem for many-sorted FOL.

The traditional $\widetilde{\mathsf{t}}$ encoding inserts tags around every subterm:

$$\mathsf{t}_{\mathsf{st}}(S) \approx \mathsf{t}_{\mathsf{st}}(\mathsf{on}) \lor \mathsf{t}_{\mathsf{st}}(S) \approx \mathsf{t}_{\mathsf{st}}(\mathsf{off}) \qquad \mathsf{t}_{\mathsf{st}}(\mathsf{flip}(\mathsf{t}_{\mathsf{st}}(S))) \not\approx \mathsf{t}_{\mathsf{st}}(S)$$

Since the sort st is not monotonic (its only models have cardinality 2), the $\widetilde{\mathsf{t}}?$ encoding coincides with $\widetilde{\mathsf{t}}$. In contrast, the featherweight $\widetilde{\mathsf{t}}??$ encoding tags only naked variables:

$$\mathsf{t}_{\mathsf{st}}(S) \approx \mathsf{on} \lor \mathsf{t}_{\mathsf{st}}(S) \approx \mathsf{off} \qquad \mathsf{flip}(S) \not\approx S$$

The $\widetilde{\mathsf{t}}??$-encoded problem is complemented by typing axioms that repair mismatches between tagged and untagged occurrences of well-sorted terms:

$$\mathsf{t}_{\mathsf{st}}(\mathsf{on}) \approx \mathsf{on} \qquad \mathsf{t}_{\mathsf{st}}(\mathsf{off}) \approx \mathsf{off} \qquad \mathsf{t}_{\mathsf{st}}(\mathsf{flip}(S)) \approx \mathsf{flip}(S)$$

For guards, the traditional and lightweight encodings $\widetilde{\mathsf{g}}$ and $\widetilde{\mathsf{g}}?$ protect each variable:

$$\neg\, \mathsf{g}_{\mathsf{st}}(S) \lor S \approx \mathsf{on} \lor S \approx \mathsf{off} \qquad \neg\, \mathsf{g}_{\mathsf{st}}(S) \lor \mathsf{flip}(S) \not\approx S$$

The featherweight encoding $\widetilde{\mathsf{g}}??$ guards only naked variables:

$$\neg\, \mathsf{g}_{\mathsf{st}}(S) \lor S \approx \mathsf{on} \lor S \approx \mathsf{off} \qquad \mathsf{flip}(S) \not\approx S$$

The guard encodings are completed by the axioms $\mathsf{g}_{\mathsf{st}}(\mathsf{on})$, $\mathsf{g}_{\mathsf{st}}(\mathsf{off})$, and $\mathsf{g}_{\mathsf{st}}(\mathsf{flip}(S))$.

General Encoding Procedure. The full sort erasure encoding e is part of a two-stage procedure to encode any many-sorted FOL problem into unsorted FOL. The first stage makes the problem monotonic by introducing protectors (tags or guards). This corresponds to a sound and complete encoding of many-sorted FOL into itself; the soundness proofs rely on the monotonicity criteria. The second stage merges all the sorts using e, which is sound and complete for monotonic problems.

Tags and guards are formalized separately, but for a protector kind, the traditional, lightweight, and featherweight encodings are treated as instances of a single generalized encoding. Both generalized encodings are parameterized by a partition of sorts by level of protection, via disjoint predicates $\mathsf{prot}, \mathsf{protFw}, \mathsf{unprot} : {}'s \to \mathsf{bool}$ indicating whether terms of a sort should be fully protected, protected in a featherweight fashion, or left unprotected. The last option is available only for sorts inferred monotonic by $\rhd$.

Tags. The tag encoding builds on a datatype of extended function symbols containing the old symbols as well as a tag for each sort:

$\texttt{datatype}\ ({}'f, {}'s)\ \mathsf{efsym} = \mathsf{Old}\ {}'f \mid \mathsf{Tag}\ {}'s$

SIG: Signatures over the extended symbols treat the old function symbols as before. The new symbols $\mathsf{Tag}\ \sigma$ are unary operations of sort arity $[\sigma]$ and result sort σ.

TRANS: The encoding function is specified as follows:

$$\begin{aligned}
\mathsf{t}\ (\mathsf{Var}\ x) &= \begin{cases} \mathsf{Var}\ x & \text{if } \mathsf{unprot}\ (\mathsf{sort}\ x) \\ \mathsf{Fn}\ (\mathsf{Tag}\ (\mathsf{sort}\ x))\ [\mathsf{Var}\ x] & \text{otherwise} \end{cases} \\
\mathsf{t}\ (\mathsf{Fn}\ f\ ts) &= \mathsf{t}'\ (\mathsf{Fn}\ f\ ts) \\
\mathsf{t}\ (\mathsf{Pos}\ (\mathsf{Eq}\ t_1\ t_2)) &= \mathsf{Pos}\ (\mathsf{Eq}\ (\mathsf{t}\ t_1)\ (\mathsf{t}\ t_2)) \\
\mathsf{t}\ (\mathsf{Neg}\ (\mathsf{Eq}\ t_1\ t_2)) &= \mathsf{Neg}\ (\mathsf{Eq}\ (\mathsf{t}'\ t_1)\ (\mathsf{t}'\ t_2)) \\
\mathsf{t}\ (\mathsf{Pos}\ (\mathsf{Pr}\ p\ ts)) &= \mathsf{Pos}\ (\mathsf{Pr}\ p\ (\mathsf{map}\ \mathsf{t}'\ ts)) \\
\mathsf{t}\ (\mathsf{Neg}\ (\mathsf{Pr}\ p\ ts)) &= \mathsf{Neg}\ (\mathsf{Pr}\ p\ (\mathsf{map}\ \mathsf{t}'\ ts))
\end{aligned}$$

$$\mathsf{t}'\ (\mathsf{Var}\ x) = \begin{cases} \mathsf{Fn}\ (\mathsf{Tag}\ (\mathsf{sort}\ x))\ [\mathsf{Var}\ x] & \text{if } \mathsf{prot}\ (\mathsf{sort}\ x) \\ \mathsf{Var}\ x & \text{otherwise} \end{cases}$$
$$\mathsf{t}'\ (\mathsf{Fn}\ f\ ts) = \begin{cases} \mathsf{Fn}\ (\mathsf{Tag}\ (\mathsf{res}\ f))\ [\mathsf{Fn}\ (\mathsf{Old}\ f)\ (\mathsf{map}\ \mathsf{t}'\ ts)] & \text{if } \mathsf{prot}\ (\mathsf{res}\ f) \\ \mathsf{Fn}\ (\mathsf{Old}\ f)\ (\mathsf{map}\ \mathsf{t}'\ ts) & \text{otherwise} \end{cases}$$

The t function tags naked variables unless they are of an unprotected sort. The auxiliary function t' adds tags only for fully protected sorts; it is invoked on all subterms except naked variables.

The tag axioms $\mathcal{A}x_\Phi$—needed to repair mismatches between tagged and untagged terms in the featherweight encoding $\tilde{\mathsf{t}}??$—have the form $\mathsf{Pos}\,(\mathsf{Eq}\,(\mathsf{Fn}\,(\mathsf{Tag}\,(\mathsf{res}\,f)\,[t]))\,t)$, where $t = \mathsf{Fn}\ (\mathsf{Old}\ f)\ (\mathsf{map}\ \mathsf{Var}\ xs)$ and xs is a list of distinct variables of sorts $\mathsf{arity}_\mathsf{F}\ f$, for all function symbols f such that $\mathsf{protFw}\ (\mathsf{res}\ f)$. The encoding of a problem is $\mathsf{t}\ \Phi = \{\mathsf{map}\ \mathsf{t}\ c \mid c \in \Phi\} \cup \mathcal{A}x_\Phi$.

SOUND: Given a model of the fixed problem Φ, a model of $\mathsf{t}\ \Phi$ is obtained by extending it with interpreting tags as the identity functions.

COMPLETE: Completeness is more difficult. To convey a sense of the complexity, let us quote the informal proof, in which x stands for $\tilde{\mathsf{t}}?$ or $\tilde{\mathsf{t}}??$ ($\tilde{\mathsf{t}}$ is analogous to $\tilde{\mathsf{t}}?$) and $[\![\Phi]\!]_x$ denotes the x-encoding of the NNF problem Φ [4, §4.4]:

> A model of $[\![\Phi]\!]_x$ is *canonical* if all tag functions t_σ are interpreted as the identity. From a canonical model, we obtain a model of Φ by leaving out t_σ. It then suffices to prove that whenever there exists a model $\mathcal{M}$ of $[\![\Phi]\!]_x$, there exists a canonical model $\mathcal{M}'$.
>
> For $\tilde{\mathsf{t}}?$, values of a tagged type σ are systematically accessed through t_σ. Hence, we can safely permute the entries of the function table of each t_σ so that it is the identity for the values in its range. We then construct $\mathcal{M}'$ by removing the domain elements for which t_σ is not the identity. It is a model by Lemma 4.13 [which states that substructures of NNF models are models if they preserve existential witnesses].
>
> For $\tilde{\mathsf{t}}??$, the construction must take possibly nonmonotonic types into account. No permutation is necessary for these thanks to the typing axioms, which ensure that the tag functions are the identity for well-typed terms. For each $\sigma \not\triangleright \Phi$, we remove the model elements for which t_σ is not the identity. The typing axioms ensure that the substructure is well-defined: each tag function is the identity for at least one element and also for each element within the range of a non-tag function. The equations $\mathsf{t}_\sigma(X) \approx X$ generated for existential variables ensure that witnesses are preserved, as required by Lemma 4.13.

Relying on permutations is intuitive on paper, but in the proof assistant it is simpler to combine the permutation and the reduction to a canonical model:

$$\mathsf{int}_\mathsf{F}\ f\ as = \begin{cases} \mathsf{eint}_\mathsf{F}\,(\mathsf{Tag}\,(\mathsf{res}\,f))\ [\mathsf{eint}_\mathsf{F}\,(\mathsf{Old}\ f)\,(\mathsf{map}_2\,\mathsf{q}\,(\mathsf{arity}_\mathsf{F}\ f)\ as)] & \text{if } \mathsf{prot}\,(\mathsf{res}\,f) \\ \mathsf{eint}_\mathsf{F}\,(\mathsf{Old}\ f)\,(\mathsf{map}_2\,\mathsf{q}\,(\mathsf{arity}_\mathsf{F}\ f)\ as) & \text{otherwise} \end{cases}$$

Here, eint_F denotes the int_F component of the fixed model of $\mathsf{t}\ \Phi$, and map_2 applies a binary function elementwise on parallel lists. The auxiliary function q maps a sort σ

and an element d to d if either $\mathsf{unprot}\ \sigma$ or d is in the range of $\mathsf{eint_F}\ (\mathsf{Tag}\ \sigma)$; otherwise, it maps σ, d to $\mathsf{eint_F}\ (\mathsf{Tag}\ \sigma)\ d$. The proof that the resulting structure is a model of the original problem Φ involves defining suitable back-and-forth functions between the two structures. Finally, proving monotonicity of $\mathsf{t}\ \Phi$ is reduced to showing that the first criterion always succeeds on the translated problem: $\mathsf{Problem}_\Phi < \mathsf{Problem\_Crit1}_{\mathsf{t}\,\Phi}$.

Guards. The guard encoding requires extending the signature with guard predicates:

$\texttt{datatype}\ ('p,'s)\ \mathsf{epsym} = \mathsf{Old}\ 'p \mid \mathsf{Guard}\ 's$

Each symbol $\mathsf{Guard}\ \sigma$ has arity $[\sigma]$ and contributes axioms to the translated problem.

The soundness proof extends models of Φ into models of $\mathsf{g}\ \Phi$ by interpreting the guard predicates as true everywhere. The completeness part is easier for guards than for tags. A canonical model is one where all guard predicates are interpreted as true everywhere. The proof handles the three levels of protection uniformly, reflecting the more uniform nature of $\widetilde{\mathsf{g}}$??—there are no counterparts to the "typing axioms that repair mismatches between tagged and untagged occurrences of well-sorted terms" of $\widetilde{\mathsf{t}}$??.

Monotonicity is proved using the second criterion, with the extension policy C for the predicates $\mathsf{Old}\ p$ and F for the distinguished symbols $\mathsf{Guard}\ \sigma$. This is a departure from the informal proof, which inlines the model extension argument without appealing to the monotonicity criterion.

6 First-Order Logic with Quantifiers

This and the next two sections are concerned with lifting the results presented in the previous sections to negation normal form and structures with arbitrarily large domains.

The locales for quantified FOL formulas in NNF are either the same or similar to those for CNF; the theory prefix Q is used for disambiguation (e.g., Q.Model). No cardinality assumption is made about the universe. Terms and atoms are as for CNF, but formulas can nest positive connectives and quantifiers arbitrarily.

The following declaration gives an approximation of the syntactic category of formulas. The actual type identifies them modulo α-equivalence (variable renaming):

$\texttt{datatype}\ ('s,'f,'p)\ \mathsf{fm} =$
$\quad \mathsf{Pos}\ (('f,'p)\ \mathsf{atm}) \mid \mathsf{Conj}\ (('s,'f,'p)\ \mathsf{fm})\ (('s,'f,'p)\ \mathsf{fm}) \mid \mathsf{All}\ 's\ \mathsf{var}\ (('s,'f,'p)\ \mathsf{fm}) \mid$
$\quad \mathsf{Neg}\ (('f,'p)\ \mathsf{atm}) \mid \mathsf{Disj}\ (('s,'f,'p)\ \mathsf{fm})\ (('s,'f,'p)\ \mathsf{fm}) \mid \mathsf{Ex}\ 's\ \mathsf{var}\ (('s,'f,'p)\ \mathsf{fm})$

The proper formal management of binding syntax modulo α-equivalence is a topic of extensive research in λ-calculus and programming languages. FOL syntax poses similar challenges. In particular, substitution and its interplay with the semantics is difficult to handle rigorously; for example, a standard textbook [16] dedicates dozens of lemmas to these preliminaries, with rough proof sketches. Many of these refer to properties of any syntax with static bindings, falling under the scope of a general metatheory of syntax formalized by Popescu et al. [23, 24]. A prominent feature of this framework—distinguishing it from the more established Nominal Isabelle [14], based on nominal logic [21]—is that it is centered around the notion of substitution:

- The framework defines substitution, including parallel and unary variants, and provides a large collection of basic facts about the interaction of substitution with free variables and the other operators.

- It provides a recursor for defining operators that are directly compositional with substitution. (In contrast, the nominal logic recursor targets compositionality with permutations, a less useful concept.)

This unconventional focus is appropriate: Substitution is without doubt the central syntactic operator in logics and type systems.

Another main feature is the facilitation of semantic interpretation of syntax, which is problematic in frameworks optimized for manipulating finitary syntax. For example, Pitts encounters “a really nontrivial freshness condition on binders” [22, § 6.3] he needs to discharge in the context of applying the nominal recursor to interpret the λ-calculus in a semantic domain. This feature is illustrated below for interpreting FOL syntax.

The framework requires the user to provide semantic domains—for FOL, types $\mathcal{T}$, $\mathcal{A}$, and $\mathcal{F}$ for interpreting terms, atoms, and formulas—as well as first-order operations corresponding to the non-binding constructors other than for variables (e.g., $\mathsf{FN} : {'f} \to \mathcal{T}\ \mathsf{list} \to \mathcal{T}$) and second-order operations corresponding to the binders: $\mathsf{ALL} : {'s} \to (\mathcal{T} \to \mathcal{F}) \to \mathcal{F}$ and $\mathsf{EX} : {'s} \to (\mathcal{T} \to \mathcal{F}) \to \mathcal{F}$.

In exchange, the framework produces the functions $\mathsf{int}_{\mathsf{Tm}} : \mathsf{tm} \to (\mathsf{var} \to \mathcal{T}) \to \mathcal{T}$, $\mathsf{int}_{\mathsf{At}} : \mathsf{atm} \to (\mathsf{var} \to \mathcal{T}) \to \mathcal{A}$, and $\mathsf{int}_{\mathsf{Fm}} : \mathsf{fm} \to (\mathsf{var} \to \mathcal{T}) \to \mathcal{F}$ that interpret syntax in the semantic domains. They map variables according to a valuation ξ. They map the action of non-binding constructors to that of the corresponding semantic operators, and similarly for binding constructors but in a valuation-sensitive way. For example:

$$\begin{aligned}
\mathsf{int}_{\mathsf{Tm}}\ (\mathsf{Var}\ x)\ \xi &= \xi\ x \\
\mathsf{int}_{\mathsf{Tm}}\ (\mathsf{Fn}\ f\ ts)\ \xi &= \mathsf{FN}\ f\ (\mathsf{map}\ (\lambda t.\ \mathsf{int}_{\mathsf{Tm}}\ t\ \xi)\ ts) \\
\mathsf{int}_{\mathsf{At}}\ (\mathsf{Eq}\ t_1\ t_2)\ \xi &= \mathsf{EQ}\ (\mathsf{int}_{\mathsf{Tm}}\ t_1\ \xi)\ (\mathsf{int}_{\mathsf{Tm}}\ t_2\ \xi) \\
\mathsf{int}_{\mathsf{Fm}}\ (\mathsf{Conj}\ \varphi_1\ \varphi_2)\ \xi &= \mathsf{CONJ}\ (\mathsf{int}_{\mathsf{Fm}}\ \varphi_1\ \xi)\ (\mathsf{int}_{\mathsf{Fm}}\ \varphi_2\ \xi) \\
\mathsf{int}_{\mathsf{Fm}}\ (\mathsf{All}\ \sigma\ x\ \varphi)\ \xi &= \mathsf{ALL}\ \sigma\ (\lambda d.\ \mathsf{int}_{\mathsf{Fm}}\ \varphi\ \xi[x \mapsto d])
\end{aligned}$$

where $\xi[x \mapsto d]$ denotes the function that maps x to d and otherwise coincides with ξ. So far, this looks like the standard interpretation of binding syntax in a semantic domain, except that here the recursive definition is modulo α-equivalence (which is a priori difficult to achieve in a proof assistant). The framework also derives compositionality of substitution w.r.t. valuation update and obliviousness of the interpretation w.r.t. fresh variables in a systematic, FOL-agnostic way:

$$\begin{aligned}
\mathsf{int}_{\mathsf{Fm}}\ \varphi[t/x]\ \xi &= \mathsf{int}_{\mathsf{Fm}}\ \varphi\ \xi[x \mapsto \mathsf{int}_{\mathsf{Tm}}\ t\ \xi] \\
\mathsf{int}_{\mathsf{Fm}}\ \varphi\ \xi &= \mathsf{int}_{\mathsf{Fm}}\ \varphi\ \xi' \qquad \text{if } \xi \text{ and } \xi' \text{ differ only on variables fresh for } \varphi
\end{aligned}$$

In the first equation, $\varphi[t/x]$ denotes capture-free substitution of t for x in φ.

A many-sorted structure $(\mathsf{int}_{\mathsf{S}}, \mathsf{int}_{\mathsf{F}}, \mathsf{int}_{\mathsf{P}})$ can be organized as a semantic domain by taking $\mathcal{T} = \omega$, $\mathcal{A} = \mathcal{F} = \mathsf{bool}$, $\mathsf{FN} = \mathsf{int}_{\mathsf{F}}$, $\mathsf{EQ} = (=)$, $\mathsf{CONJ} = (\wedge)$, $\mathsf{ALL}\ \sigma\ P = (\forall d.\ \mathsf{int}_{\mathsf{S}}\ \sigma\ d \longrightarrow P\ d)$, and so on. This yields the recursive equations

$$\begin{aligned}
[\![\mathsf{Var}\ x]\!]_\xi &= \xi\ x \\
[\![\mathsf{Fn}\ f\ ts]\!]_\xi &= \mathsf{int}_{\mathsf{F}}\ f\ (\mathsf{map}\ (\lambda t.\ [\![t]\!]_\xi)\ ts) \\
\vDash_\xi \mathsf{Eq}\ t_1\ t_2 &\longleftrightarrow [\![t_1]\!]_\xi = [\![t_2]\!]_\xi \\
\vDash_\xi \mathsf{Conj}\ \varphi_1\ \varphi_2 &\longleftrightarrow \vDash_\xi \varphi_1 \wedge \vDash_\xi \varphi_2 \\
\vDash_\xi \mathsf{All}\ \sigma\ x\ \varphi &\longleftrightarrow \forall d.\ \mathsf{int}_{\mathsf{S}}\ \sigma\ d \longrightarrow \vDash_{\xi[x \mapsto d]} \varphi
\end{aligned}$$

which characterize term interpretation (with $[\![t]\!]_\xi = \mathsf{int}_{\mathsf{Tm}}\ t\ \xi$), atom satisfaction ($\models_\xi a = \mathsf{int}_{\mathsf{At}}\ a\ \xi$), and formula satisfaction ($\models_\xi \varphi = \mathsf{int}_{\mathsf{Fm}}\ \varphi\ \xi$). These functions are defined in the Q.Structure locale. The framework also produces the substitution lemma $\models_\xi \varphi[t/x] \longleftrightarrow \models_{\xi[x \mapsto [\![t]\!]_\xi]} \varphi$. In the next section, the notations $\models \varphi$ and $\models \Phi$ abbreviate $\forall\xi.\ \models_\xi \varphi$ and $\forall\varphi \in \Phi.\ \models \varphi$. The structure can also be made explicit—e.g., $(\mathsf{int}_{\mathsf{S}}, \mathsf{int}_{\mathsf{F}}, \mathsf{int}_{\mathsf{P}}) \models_\xi \varphi$.

If the orientation toward substitution is the main strength of the framework, its main weakness is the lack of automation. For each desired binding syntax type, users must currently instantiate the general theorems manually, much like mathematicians do routinely when applying universal algebra to groups or rings. The instantiation is tedious due to the large number of theorems. Despite the availability of "template files," this process can take days and thousands of lines of proof text. Automation in the form of a definitional package, which would provide the basic convenience expected by users of Nominal Isabelle (while supporting substitution natively), remains for future work.

7 Classical Metatheorems

The lifting argument from countable CNF structures to unbounded NNF structures (Section 8) relies on clausification and Löwenheim–Skolem for many-sorted FOL with equality. Earlier formalizations focus on unsorted FOL without equality [2,12,25]. Sorts and equality are tedious to formalize, and they often fail to reward the formalizer with deep logical insight, but they are central to monotonicity and sort encodings.

Clausification. The translation of a finite quantified problem into clausal form involves skolemizing all the existentially quantified variables into function symbols that take the universally quantified variables in scope as arguments. Skolemization is surprisingly difficult to treat formally; for example, Harrison [12] claims that it poses greater challenges than completeness. On the positive side, clausification can be seen as an instance of the general semantic interpretation principle introduced in Section 6.

The definition of clausification and its soundness and completeness proof follow the four-step institutional approach.

SIG: Skolemization introduces new function symbols Sko $\sigma s\ x$, built from a list of sorts σs (specifying the arity) and a variable name x, while preserving the sorts of Σ-symbols:

```
datatype 'f efsym = Old 'f | Sko ('s list) var
```

TRANS: The clausification function cls takes a Σ-formula φ, an environment $\rho : \mathsf{var} \to \mathsf{tm}$, a list of universal variables vs, and a set of fresh variables V as arguments. In addition to massaging the connectives, it replaces existential variables by new symbols that depend on vs, replaces bound universal variables by fresh variables from V, and substitutes free variables according to ρ to produce a Σ'-clause.

The characteristic equations for cls are obtained by instantiating the semantic interpretation principle with $\mathcal{T} = \mathsf{tm}$, $\mathcal{A} = \mathsf{atm}$, and $\mathcal{F} = \mathsf{var\ list} \to \mathsf{var\ set} \to \mathsf{fm}$, taking suitable operators on these domains, and letting cls be $\mathsf{int}_{\mathsf{Fm}}$. The interesting cases are

$$\begin{aligned}\mathsf{cls}\ (\mathsf{All}\ \sigma\ x\ \varphi)\ \rho\ vs\ V &= \mathsf{cls}\ \varphi\ \rho[x \mapsto \mathsf{Var}\ v]\ (v \# vs)\ (V \setminus \{v\})\\ \mathsf{cls}\ (\mathsf{Ex}\ \sigma\ x\ \varphi)\ \rho\ vs\ V &= \mathsf{cls}\ \varphi\ \rho[x \mapsto \mathsf{Fn}\ f\ (\mathsf{map\ Var}\ vs)]\ vs\ (V \setminus \{v\})\end{aligned}$$

where $v \in V$ is some variable of sort σ and $f = \mathsf{Sko}\ (\mathsf{map\ sort}\ vs)\ v$ is the Skolem function symbol, which is applied to the universal variables vs. For closed formulas, clausification is defined as $\mathsf{clausify}\ \varphi = \mathsf{cls}\ \varphi\ \rho\ []\ \mathsf{UNIV}$ for some irrelevant choice of ρ.

As a simple example, let $\varphi = \mathsf{All}\ \sigma\ x\ (\mathsf{Ex}\ \tau\ y\ (\mathsf{Eq}\ (\mathsf{Var}\ x)\ (\mathsf{Var}\ y)))$, let v_1, v_2 be the variables picked from UNIV and $\mathsf{UNIV} \setminus \{v_1\}$, and let $f = \mathsf{Sko}\ [\sigma]\ v_2$. Then

$$\begin{aligned}
&\mathsf{clausify}\ \varphi \\
=\ &\mathsf{cls}\ \varphi\ \rho\ []\ \mathsf{UNIV} \\
=\ &\mathsf{cls}\ (\mathsf{Ex}\ \tau\ y\ (\mathsf{Eq}\ (\mathsf{Var}\ x)\ (\mathsf{Var}\ y)))\ \rho[x \mapsto \mathsf{Var}\ v_1]\ [v_1]\ (\mathsf{UNIV} \setminus \{v_1\}) \\
=\ &\mathsf{cls}\ (\mathsf{Eq}\ (\mathsf{Var}\ x)\ (\mathsf{Var}\ y))\ \rho[x \mapsto \mathsf{Var}\ v_1,\ y \mapsto \mathsf{Fn}\ f\ [\mathsf{Var}\ v_1]]\ [v_1]\ (\mathsf{UNIV} \setminus \{v_1, v_2\}) \\
=\ &\mathsf{Eq}\ (\mathsf{Var}\ v_1)\ (\mathsf{Fn}\ f\ [\mathsf{Var}\ v_1])
\end{aligned}$$

SOUND: Soundness is proved in the Structure locale, which fixes a Σ-structure ($\mathsf{int_S}$, $\mathsf{int_F}$, $\mathsf{int_P}$). The "Skolem model" predicate $\mathsf{skmod}\ \varphi\ \rho\ vs\ V\ eint_{\mathsf{F}}\ eint'_{\mathsf{F}}$ transforms, for each valuation $\xi : \mathsf{var} \to {'u}$, an extended structure $eint_{\mathsf{F}}$ such that $\models_\xi \mathsf{cls}\ \varphi\ \rho\ vs\ V$ into an extended structure $eint'_{\mathsf{F}}$ such that $\models_{\xi \diamond \rho} \varphi$, where $\diamond$ composes valuations with environments. The introduction rules of skmod emulate cls's equations; for example,

$$\frac{\mathsf{skmod}\ \varphi\ \rho[x \mapsto \mathsf{Fn}\ f\ (\mathsf{map\ Var}\ vs)]\ vs\ (V \setminus \{v\})\ eint_{\mathsf{F}}[f \mapsto F]\ eint'_{\mathsf{F}}}{\mathsf{skmod}\ (\mathsf{Ex}\ \sigma\ x\ \varphi)\ \rho\ vs\ V\ eint_{\mathsf{F}}\ eint'_{\mathsf{F}}}$$

where $v \in V$ and $F : {'u}\ \mathsf{list} \to {'u}$ is a suitable interpretation for the Skolem symbol f, defined so that $F\ us$ gives an arbitrary u such that $(\mathsf{int_S}, \mathsf{int_F}, \mathsf{int_P}) \models_{\xi \diamond \rho[x \mapsto u]} \varphi$, where ξ maps vs to us elementwise. The skmod relation is total on the last argument. For closed formulas φ such that $(\mathsf{int_S}, \mathsf{int_F}, \mathsf{int_P}) \models \varphi$, starting with an extension $\mathsf{eint_F}$ of $\mathsf{int_F}$, skmod yields $\mathsf{eint'_F}$ such that $(\mathsf{int_S}, \mathsf{eint'_F}, \mathsf{int_P}) \models \mathsf{clausify}\ \varphi$. Thus, if φ has a model, then $\mathsf{clausify}\ \varphi$ also has a model.

For problems, we define $\mathsf{clausify}\ \Phi = \mathsf{clausify}\ (\bigwedge \Phi)$, where $\bigwedge \Phi$ is the conjunction of all formulas in Φ, which must be finite. The locale Q.Model fixes Φ and a model, which is also a model of the formula $\bigwedge \Phi$. By soundness of clausify on closed formulas, this yields a model of $\mathsf{clausify}\ \Phi$.

COMPLETE: For completeness, it suffices to show that the backward structure translation of a model of $\mathsf{clausify}\ \Phi$ is a model of Φ. This is straightforward.

Löwenheim–Skolem. The proof of the downward Löwenheim–Skolem theorem is based on a formalization of a complete inference system, described in a separate paper [8]. In the Q.Model locale, which fixes a problem and model, it constructs a syntactic Henkin model. Since this model has a countable universe, there exists an isomorphic copy on ω (the countably infinite universe fixed throughout Sections 4 and 5). This yields $\mathsf{Q.Model}_{'u} < \mathsf{Q.Model}_{\omega}$.

Using the obvious sound and complete embedding embed of CNF problems into NNF problems, it is possible to transfer the Löwenheim–Skolem theorem to CNF:

$$\mathsf{Model}_{'u,\Phi} < \mathsf{Q.Model}_{'u,\mathsf{embed}\ \Phi} < \mathsf{Q.Model}_{\omega,\mathsf{embed}\ \Phi} < \mathsf{Model}_{\omega,\Phi}$$

To summarize the results of this section:

Theorem 2. *An NNF problem Φ has a model iff* clausify Φ *has a model.*

Theorem 3. *An NNF problem has a model iff it has a countable model.*

8 Lifting to Arbitrary Structures and Formulas with Binders

The focus on clausal form and countable structures is a useful simplification, but it is not faithful to the NNF-based paper proof [3] (or to the implementation in Sledgehammer). Thanks to a lifting argument that relies on clausification and Löwenheim–Skolem, the final results are free of such restrictions.

Figure 1 shows how the results are connected. Starting at the top with a satisfiable quantified problem Φ, the problem is first clausified, then by Löwenheim–Skolem it is countably satisfiable (by taking $'u = \omega$). On the left-hand side, the clausified problem is further encoded using tags or guards ($x \in \{\mathsf{t}, \mathsf{g}\}$) and shown to pass one of the monotonicity criteria ($i = 1$ for t and 2 for g), meaning it is monotonic. On the right-hand side, the encoded problem is satisfiable. Merging the two branches yields a monotonic satisfiable problem, whose erasure is a satisfiable unsorted problem. Since every translation step is also shown complete, the right-hand side can also be traversed bottom-up, producing a model of the original problem from a model of the translated unsorted problem. The overall translation is thus sound and complete.

Theorem 4. *Given $x \in \{\mathsf{t}, \mathsf{g}\}$ and a finite many-sorted NNF problem Φ, let Φ' be the unsorted CNF problem $\mathsf{e}\,(x\,(\mathsf{clausify}\,\Phi))$, i.e., the sort-erased x-translated clausified Φ.*

(1) *For each model $\mathcal{M}$ of Φ (forming together with Φ an instance of $\mathsf{Q.Model}_{\Phi}$), there exists a model $\mathcal{M}'$ of Φ' (forming together with Φ' an instance of $\mathsf{U.Model}_{\Phi'}$).*

(2) *Conversely, for every model of Φ', there exists a model of Φ.*

The formal proof puts together many constructions and results of independent interest, notably soundness of the monotonicity criteria (Theorem 1), soundness and completeness of clausification (Theorem 2), and downward Löwenheim–Skolem (Theorem 3).

$$\begin{tikzcd}
 & \mathsf{Q.Model}_{'u,\,\Phi} \arrow[d, dash] & \\
 & \mathsf{Model}_{'u,\,\mathsf{clausify}\,\Phi} \arrow[d, dash] & \\
 & \mathsf{Model}_{\omega,\,\mathsf{clausify}\,\Phi} \arrow[dl, dash] \arrow[ddr, dash] & \\
\mathsf{Problem}_{\omega,\,\mathsf{clausify}\,\Phi} \arrow[d, dash] & & \\
\mathsf{Problem\_Crit}i_{\omega,\,x\,(\mathsf{clausify}\,\Phi)} \arrow[d, dash] & & \mathsf{Model}_{\omega,\,x\,(\mathsf{clausify}\,\Phi)} \arrow[ddl, dash] \\
\mathsf{Mono\_Problem}_{\omega,\,x\,(\mathsf{clausify}\,\Phi)} \arrow[dr, dash] & & \\
 & \mathsf{Mono\_Model}_{\omega,\,x\,(\mathsf{clausify}\,\Phi)} \arrow[d, dash] & \\
 & \mathsf{U.Model}_{\omega,\,\mathsf{e}\,(x\,(\mathsf{clausify}\,\Phi))} \arrow[d, dash] & \\
 & \mathsf{U.Model}_{'u,\,\mathsf{e}\,(x\,(\mathsf{clausify}\,\Phi))} &
\end{tikzcd}$$

Figure 1. The verified translation pipeline

9 Conclusion

This paper describes a framework and a methodology for formalizing applications of many-sorted first-order logic while acting as a companion to recent papers on sort encodings [3, 9, 10]. To readers from the proof assistant community, it also provides a contribution to the ongoing binder representation debate. And to readers rooted in algebraic methods, it shows a practical application of the theory of institutions in a context where the translation functions cannot be assumed to be uniform.

The formalization widely reaffirmed already proved results. On one occasion, it revealed a flaw in a published lemma (Lemma 3 of Claessen et al. [10]). It also helped detect mistakes in a subsequent paper proof [4] before it reached any readers. The work provided the opportunity to rethink the proof; for example, the generalized monotonicity concept, in terms of sets of sorts, arose during the formalization.

A potential practical benefit of this work is connected to step-by-step proof reconstruction. Although the encodings are sound, the inferences in a machine-generated proof may violate the sort discipline, resulting in failures in Sledgehammer's proof replay. In future work, we want to investigate the feasibility of connecting the soundness proofs of the encodings with a verified checker for unsorted FOL proofs.

The advantages of machine-checked metatheory are well known from programming language research, where papers are often accompanied by formal developments and proof assistants have made it into the classroom. Paradoxically, in the automated reasoning community, we have not been very enthusiastic about formalizing our own results. This paper reported on some steps we have taken to address this.

Acknowledgement. We thank Tobias Nipkow for making this work possible. Jesper Bengtson, Nicholas Smallbone, Mark Summerfield, Dmitriy Traytel, and several anonymous reviewers suggested improvements to earlier versions of this paper. The research was supported by the Deutsche Forschungsgemeinschaft (DFG) projects Security Type Systems and Deduction (grant Ni 491/13-1), part of the program Reliably Secure Software Systems (RS^3, Priority Program 1496), and Hardening the Hammer (grant Ni 491/14-1). The authors are listed in alphabetical order.

References

[1] Ballarin, C.: Locales: A module system for mathematical theories. J. Autom. Reasoning, to appear

[2] Berghofer, S.: First-order logic according to Fitting. In: Klein, G., Nipkow, T., Paulson, L. (eds.) Archive of Formal Proofs. `http://afp.sf.net/entries/FOL-Fitting.shtml` (2007)

[3] Blanchette, J.C., Böhme, S., Popescu, A., Smallbone, N.: Encoding monomorphic and polymorphic types. In: Piterman, N., Smolka, S. (eds.) TACAS 2013. LNCS, vol. 7795, pp. 493–507. Springer (2013)

[4] Blanchette, J.C., Böhme, S., Popescu, A., Smallbone, N.: Encoding monomorphic and polymorphic types. Tech. report associated with TACAS 2013 paper [3], `http://www21.in.tum.de/~blanchet/enc_types_report.pdf` (2013)

[5] Blanchette, J.C., Krauss, A.: Monotonicity inference for higher-order formulas. J. Autom. Reasoning 47(4), 369–398 (2011)

[6] Blanchette, J.C., Popescu, A.: Formal development associated with this paper. `http://www21.in.tum.de/~popescua/fol_devel.zip` (2013)
[7] Blanchette, J.C., Popescu, A.: Sound and complete sort encodings for first-order logic. In: Klein, G., Nipkow, T., Paulson, L. (eds.) Archive of Formal Proofs. `http://afp.sourceforge.net/entries/Sort_Encodings.shtml` (2013)
[8] Blanchette, J.C., Popescu, A., Traytel, D.: Coinductive pearl: Modular first-order logic completeness, submitted, `http://www21.in.tum.de/~blanchet/compl.pdf`
[9] Bouillaguet, C., Kuncak, V., Wies, T., Zee, K., Rinard, M.: Using first-order theorem provers in the Jahob data structure verification system. In: Cook, B., Podelski, A. (eds.) VMCAI 2007. LNCS, vol. 4349, pp. 74–88. Springer (2007)
[10] Claessen, K., Lillieström, A., Smallbone, N.: Sort it out with monotonicity—Translating between many-sorted and unsorted first-order logic. In: Bjørner, N., Sofronie-Stokkermans, V. (eds.) CADE-23. LNAI, vol. 6803, pp. 207–221. Springer (2011)
[11] Goguen, J.A., Burstall, R.M.: Institutions: Abstract model theory for specification and programming. J. ACM 39(1), 95–146 (1992)
[12] Harrison, J.: Formalizing basic first order model theory. In: Grundy, J., Newey, M. (eds.) TPHOLs '98. LNCS, vol. 1479, pp. 153–170. Springer (1998)
[13] Harrison, J.: Towards self-verification of HOL Light. In: Furbach, U., Shankar, N. (eds.) IJCAR 2006. LNCS, vol. 4130, pp. 177–191. Springer (2006)
[14] Huffman, B., Urban, C.: Proof pearl: A new foundation for Nominal Isabelle. In: Kaufmann, M., Paulson, L.C. (eds.) ITP 2010. LNCS, vol. 6172, pp. 35–50. Springer (2010)
[15] Kammüller, F., Wenzel, M., Paulson, L.C.: Locales—A sectioning concept for Isabelle. In: Bertot, Y., Dowek, G., Hirschowitz, A., Paulin, C., Théry, L. (eds.) TPHOLs '99. LNCS, vol. 1690, pp. 149–166. Springer (1999)
[16] Monk, J.D.: Mathematical Logic. Springer (1976)
[17] Myreen, M.O., Davis, J.: A verified runtime for a verified theorem prover. In: van Eekelen, M., Geuvers, H., Schmaltz, J., Wiedijk, F. (eds.) ITP 2011. LNCS, vol. 6898, pp. 265–280. Springer (2011)
[18] Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL: A Proof Assistant for Higher-Order Logic, LNCS, vol. 2283. Springer (2002)
[19] Paulson, L.C., Blanchette, J.C.: Three years of experience with Sledgehammer, a practical link between automatic and interactive theorem provers. In: Sutcliffe, G., Ternovska, E., Schulz, S. (eds.) IWIL-2010 (2010)
[20] Pfenning, F., Elliott, C.: Higher-order abstract syntax. In: Wexelblat, R.L. (ed.) PLDI '88. pp. 199–208. ACM (1988)
[21] Pitts, A.M.: Nominal logic, a first order theory of names and binding. Inf. Comput. 186(2), 165–193 (2003)
[22] Pitts, A.M.: Alpha-structural recursion and induction. J. ACM 53(3), 459–506 (2006)
[23] Popescu, A., Gunter, E.L.: Recursion principles for syntax with bindings and substitution. In: Chakravarty, M.M.T., Hu, Z., Danvy, O. (eds.) ICFP 2011. pp. 346–358. ACM (2011)
[24] Popescu, A., Gunter, E.L., Osborn, C.J.: Strong normalization of System F by HOAS on top of FOAS. In: LICS 2010. pp. 31–40. IEEE (2010)
[25] Ridge, T., Margetson, J.: A mechanically verified, sound and complete theorem prover for first order logic. In: Hurd, J., Melham, T.F. (eds.) TPHOLs 2005. LNCS, vol. 3603, pp. 294–309. Springer (2005)
[26] Shankar, N.: Metamathematics, Machines, and Gödel's Proof, Cambridge Tracts in Theoretical Computer Science, vol. 38. Cambridge University Press (1994)
[27] Sutcliffe, G.: The 6th IJCAR automated theorem proving system competition—CASC-J6. AI Comm. 26(2), 211–223 (2013)
[28] Tinelli, C., Zarba, C.G.: Combining decision procedures for sorted theories. In: Alferes, J., Leite, J. (eds.) JELIA 2004. LNCS, vol. 3229, pp. 641–653. Springer (2004)