


# Certified Grounding of Extensions of First-Order Logic

Daimy Van Caudenberg<sup>1</sup> ✉ 

KU Leuven, Belgium

Carlos Cantero ✉ 

KU Leuven, Belgium

Markus Anders ✉ 

RPTU Kaiserslautern-Landau, Germany

KU Leuven, Belgium

Bart Bogaerts ✉ 

KU Leuven, Belgium

Vrije Universiteit Brussel, Belgium

---

## Abstract

This is a work-in-progress paper that presents the foundations of a novel methodology for the certified grounding of first-order logic (FOL). This methodology will be key to developing trustworthy and auditable grounders, and can be instrumental for debugging and testing. However, there are major challenges to overcome. At the level of grounding algorithms, these include (not necessarily equivalence-preserving) transformations; symmetry detection and elimination; and techniques that simplify the grounding, potentially by arguments that reason globally over the input. All of these techniques serve to simplify the problem that will be solved later. At the level of the language, for practical modelling purposes, we aim to support rich extensions of FOL with aggregates, types, and inductive definitions, which all come with their own challenges for support in the proof system at hand. In this paper, we describe the foundations of this new methodology, introducing the normal form that lies at its core, as well as the first steps towards a certifying grounder that supports rich extensions of FOL.

**2012 ACM Subject Classification** Theory of computation → Constraint and logic programming; Theory of computation → Logic and verification

**Keywords and phrases** First-Order Logic, Grounding, Proof logging

**Funding** This work was funded by the European Union (ERC, CertiFOX, 101122653). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.

## 1 Introduction

The adoption of *certified* solvers has significantly increased trust in solver outputs by allowing the external verification of the solver's results. However, these assurances are only meaningful if the solver's input accurately reflects the original problem. Because many reasoning systems rely on grounding, it is important to increase trust in this grounding process. In this paper, we describe our current work, which forms the basis for a novel methodology for certified grounding of first-order logic (FOL) over finite domains.

Essentially, the role of a grounder for plain FOL in a given finite domain is to replace each universal quantifier by a (potentially large) conjunction and each existential quantifier

---

<sup>1</sup> Corresponding author

by a disjunction. However, state-of-the-art grounding systems rely on complex techniques to reduce the size of the grounded model. Hence, there are still major challenges to overcome when developing a trustworthy grounder. Of course, the user's trust in the grounder can be increased using extensive software testing, but this does not prove the absence of bugs. One way to truly show the absence of bugs is by formally verifying the tool at hand. However, given the complex nature of the grounding software, this is infeasible. Our goal is to introduce a methodology for *certified* grounding, in the sense that the grounder produces not only a ground formula but also a (machine-verifiable) proof that the obtained grounding is correct.

In this paper, we lay the foundations for this novel methodology. We introduce a normal form for first-order logic, which has certain interesting properties when it comes to grounding. Next, we describe how to ground this normal form and briefly discuss the requirements for a suitable proof system for this grounding process. Note that the formal description of a suitable proof format is future work. Last, we show how this seemingly restrictive normal form can be extended to support extensions of FOL, including types, functions, (inductive) definitions, and aggregates. Ultimately, the goal is to allow users to express their problems using a rich language consisting of FOL sentences. By introducing a certified grounder for such a language, users can express their problems in a human-readable way, while increasing trust that the correct problem is being solved at a lower level.

## 2 Preliminaries

These preliminaries are based on [9].

**First-Order Logic** A *vocabulary*  $\mathcal{V}$  is a set of predicate and function symbols. Each predicate and function symbol has an *arity*, which is the number of arguments it takes. We denote predicate (function) symbols with arity  $n$  using  $P/n$  ( $f/n$ ). A predicate (function) with arity 0 is called a *proposition* (*object*) symbol. Note that from now on, we assume that every vocabulary contains the proposition  $\top$ , representing truth.

A *term* is a variable or an  $n$ -ary function applied to  $n$  terms. An *atom* is an  $n$ -ary predicate applied to  $n$  terms. If  $p$  and  $q$  are terms,  $p = q$  is also an atom. Next, we define a *formula*:

- an atom is a formula,
- if  $\phi$  and  $\psi$  are formulas, then so are  $\neg\phi, \phi \wedge \psi, \phi \vee \psi, \phi \Rightarrow \psi, \phi \Leftarrow \psi$  and  $\phi \Leftrightarrow \psi$ ,
- if  $\phi$  is a formula and  $x$  is a variable then  $\forall x : \phi$  and  $\exists x : \phi$  are formulas.

A *sentence* is a formula without unquantified variables. A *theory*  $\mathcal{T}$  over vocabulary  $\mathcal{V}$  is a set of sentences over the symbols of  $\mathcal{V}$ . A *structure*  $\mathcal{S}$  over vocabulary  $\mathcal{V}$  consists of a domain  $\mathcal{D}$  and an interpretation for all symbols in  $\mathcal{V}$ . A structure is *partial* if it does not interpret all symbols in  $\mathcal{V}$ . From now on, we will assume that the domain  $\mathcal{D}$  is finite.

Given a vocabulary  $\mathcal{V}$ , a partial structure  $\mathcal{S}$  over  $\mathcal{V}$  and a theory  $\mathcal{T}$  over  $\mathcal{V}$ , the *model expansion problem* consists of finding a structure  $\mathcal{S}'$  such that:

- $\mathcal{S}'$  expands  $\mathcal{S}$  (meaning it has the same domain, and agrees with  $\mathcal{S}$ ),
- and  $\mathcal{S}'$  satisfies  $\mathcal{T}$  (i.e., it is a model of  $\mathcal{T}$ ).

From now on, we assume that the vocabulary  $\mathcal{V}$  is extended with an object symbol  $v/0$  for every value  $v \in \mathcal{D}$ .

► **Example 1.** We now define the pigeonhole problem as a model expansion problem. First, we define the vocabulary  $\mathcal{V} = \{\text{Pigeon}/2, \text{Hole}/2, \text{SitsIn}/1\}$ , which contains predicates that allow us to distinguish pigeons from holes, and to assign pigeons to holes. Next, we define

the theory;

$$\mathcal{T} = \left\{ \begin{array}{l} \forall h : \text{Hole}(h) \Leftrightarrow \neg \text{Pigeon}(h). \\ \forall p, h : (\neg \text{Hole}(h) \vee \neg \text{Pigeon}(p)) \Rightarrow \neg \text{SitsIn}(p, h). \\ \forall p : \text{Pigeon}(p) \Rightarrow \exists h : \text{Hole}(h) \wedge \text{SitsIn}(p, h). \\ \forall p_1, p_2, h : (\text{Hole}(h) \wedge \text{Pigeon}(p_1) \wedge \text{Pigeon}(p_2) \wedge p_1 \neq p_2) \Rightarrow \neg \text{SitsIn}(p_1, h) \vee \neg \text{SitsIn}(p_2, h). \end{array} \right\}$$

The first sentence of the theory ensures that pigeons are not holes, and vice versa. Next, the second sentence ensures correct typing for the  $\text{SitsIn}/2$  predicate. The last two sentences ensure that each pigeon sits in a hole and that each hole can contain at most one pigeon. Given the structure  $\mathcal{S} = \{\mathcal{D} = \{P_1, P_2, H\}, \text{Pigeon} = \{P_1, P_2\}, \text{Hole} = \{H\}\}$  with fewer pigeons than holes, model expansion of the structure  $\mathcal{S}$  given theory  $\mathcal{T}$  fails.

In what follows, let  $\bar{x}$  denote  $x_1 \dots x_n$  for some  $n \geq 0$ , intended to represent a tuple of variables; and let  $\phi(\bar{x})$  mean some of the free variables of  $\phi$  are among  $\bar{x}$ . Hence,  $\phi(x)$  means that  $x$  is a free variable in  $\phi$ . We use  $\phi(x : v)$  to denote the formula obtained by replacing all occurrences of  $x$  by  $v \in \mathcal{D}$ , and by extension  $\phi(\bar{x} : \bar{v})$  denotes the formula obtained by replacing  $x_i$  in  $\bar{x}$  by the corresponding  $v_i$  in  $\bar{v}$ . Given an  $n$ -ary predicate  $P/n$  (function  $f/n$ ),  $P(\bar{v}_n)$  ( $f(\bar{v}_n)$ ) denotes the application of predicate  $P$  (function  $f$ ) to arguments  $(v_1, \dots, v_n) \in \mathcal{D}^n$ . As a result, for  $P/0$ , we have that  $P(\bar{v}_n) = P()$ , and for  $f/0$  we obtain  $f(\bar{v}_n) = f()$ .

### 3 Grounding first-order logic

Before introducing the new normal form, which will form the foundation for our proof system, we give a brief introduction to grounding FOL. Given a theory  $\mathcal{T}$  over vocabulary  $\mathcal{V}$  and a (partial) structure  $\mathcal{S}$ , the task of grounding consists of reducing  $\mathcal{T}$  to an equivalent *propositional theory*  $\mathcal{T}'$  over  $\mathcal{V}_{prop}^{\mathcal{S}}$ , i.e., an equivalent quantifier- and variable-free theory over a vocabulary that contains only propositional symbols. First, given a vocabulary  $\mathcal{V}$  and a (partial) structure  $\mathcal{S}$ , we construct the propositional vocabulary  $\mathcal{V}_{prop}^{\mathcal{S}}$  such that it contains:

- a propositional symbol  $P_{\bar{v}_n}$  for each predicate  $P/n \in \mathcal{V}$  and tuple  $(\bar{v}_n) \in \mathcal{D}^n$
- a propositional symbol  $f_{\bar{v}_n, v}$  for each function  $f/n \in \mathcal{D}$  and tuple  $(\bar{v}_n, v) \in \mathcal{D}$
- a propositional symbol  $eq_{v, w}$  for each tuple  $(v, w) \in \mathcal{D}$ .

The theory should also be extended such that the semantics of functions are correct, which is done by adding the following propositional formulas for each function  $f/n \in \mathcal{D}$ :

$$\bigwedge_{v \in \mathcal{D}} \left( \bigwedge_{v' \in \mathcal{D}: v' \neq v} \neg f_{\bar{v}_n, v} \vee \neg f_{\bar{v}_n, v'} \right) \quad (1)$$

$$\bigvee_{v \in \mathcal{D}} f_{\bar{v}_n, v} \quad (2)$$

Next, we define the instantiation rules for quantifiers:

► **Definition 2** (Instantiation Rules for quantifiers). *Given a variable  $x$  and a formula  $\psi$ , a universal quantifier can be instantiated as follows;*

$$\forall_{\text{inst}} \frac{\forall x : \psi(x)}{\bigwedge_{v \in \mathcal{D}} \psi(x : v)}$$

and similarly, the existential quantifier can be instantiated using the following rule;

$$\exists_{\text{inst}} \frac{\exists x : \psi(x)}{\bigvee_{v \in D} \psi(x : v)}$$

These instantiation rules allow for the transformation of *sentences* into equivalent quantifier- and variable-free expressions. However, to obtain an equivalent *propositional* theory, all non-propositional atoms (i.e., equations and predicates with arity  $n > 0$ ) need to be instantiated as well. This can be done using the vocabulary  $\mathcal{V}_{\text{prop}}^{\mathcal{S}}$ .

We illustrate this using a toy example.

► **Example 3.** Given a vocabulary  $\mathcal{V} = \{A/1, B/1, C/1, D/2\}$  and structure  $\mathcal{S}$  containing a finite domain  $\mathcal{D} = \{v_1, \dots, v_4\}$ , the goal is to obtain a propositional grounding for the following formula;

$$\forall x : A(x) \vee (B(x) \Rightarrow \exists y : C(y) \wedge D(x, y)). \quad (3)$$

First, we apply the instantiation rule  $\forall_{\text{inst}}$ , obtaining:

$$\bigwedge_{v \in \mathcal{D}} A(v) \vee (B(v) \Rightarrow \exists y : C(y) \wedge D(v, y)).$$

Next, applying  $\exists_{\text{inst}}$  grounds the formula to an equivalent quantifier-free formula:

$$\bigwedge_{v \in \mathcal{D}} A(v) \vee \left( B(v) \Rightarrow \bigvee_{w \in \mathcal{D}} C(w) \wedge D(v, w) \right).$$

Then we replace all atoms by their counterparts in  $\mathcal{V}_{\text{prop}}^{\mathcal{S}}$ :

$$\bigwedge_{v \in \mathcal{D}} A_v \vee \neg B_v \vee \bigvee_{w \in \mathcal{D}} C_w \wedge D_{v,w}. \quad (4)$$

Note that this yields a propositional conjunction of  $|\mathcal{D}|$  propositional formulas containing  $2|\mathcal{D}| + 2$  propositional symbols (or their negation). If the goal is to solve this formula using a SAT solver, this formula needs to be transformed into conjunctive normal form (i.e., a conjunction of a disjunction of propositions or their negations). However, this causes the formula to grow exponentially, caused by the disjunction of conjunctions in the formula.

This example illustrates that this grounding method risks producing formulas of impractical size. Hence, it is not surprising that in practice, state-of-the-art grounding tools apply many optimizations to reduce the size of the grounded model. These techniques can largely be split into two categories. The first technique consists of rewriting the original theory  $\mathcal{T}$  to reduce the size of the final model [7, 6].

► **Example 4.** This example is based on [7, 6]. We are given the formula  $\forall x : \forall y : P(x, A) \vee P(y, A)$  over vocabulary  $\mathcal{V} = \{P/2, A/0\}$  as well as a structure  $\mathcal{S}$  containing the finite domain  $\mathcal{D}$ . Grounding this formula would yield a conjunction of  $|\mathcal{D}|^2$  formulas containing a disjunction of two propositional symbols each. However, the equivalent formula

$$(\forall x : P(x, A) \vee T) \wedge (\forall y : P(y, A) \vee \neg T)$$

over  $\mathcal{V} \cup \{T/0\}$ , yields only  $2|\mathcal{D}|$  formulas of the same size after grounding.

The second technique aims to efficiently determine small groundings by taking the available knowledge (described by a structure  $\mathcal{S}$ ) into account. In practice, there is often a partial structure available describing the specific instance of the problem (for example, a graph and a list of colors in a graph coloring problem, or a list of pigeon and holes in the pigeonhole problem).

► **Example 5.** The grounding in Equation (4) from Example 3 can be significantly reduced, if a structure  $\mathcal{S}$  is given which interprets predicates  $B/1$  and  $C/1$ . This can be done by replacing all applications of those predicates with their truth values according to  $\mathcal{S}$ . Replacing  $B/1$  and simplifying yields the following formula:

$$\bigwedge_{v \in \mathcal{D} : \mathcal{S} \models B(v)} A(v) \vee \bigvee_{w \in \mathcal{D}} C(w) \wedge D(v, w).$$

Finally, replacing  $C/1$  and simplifying yields the following:

$$\bigwedge_{v \in \mathcal{D} : \mathcal{S} \models B(v)} A(v) \vee \bigvee_{w \in \mathcal{D} : \mathcal{S} \models C(w)} D(v, w),$$

which consists of a conjunction of  $|\{v \in \mathcal{D} : \mathcal{S} \models B(v)\}| \leq |\mathcal{D}|$  formulas consisting of a disjunction of  $d'' + 1$  atoms or their negation, where  $d'' = |\{w \in \mathcal{D} : \mathcal{S} \models C(w)\}| \leq |\mathcal{D}|$ .

In this paper, we focus on this second technique. We do this by introducing a normal form which allows users to describe so-called *conditions* for the quantifiers. These conditions indicate which parts of the ground formula are actually relevant, as indicated by the available input structure. In the next section, we formalize this new normal form.

## 4 Ground Normal Form

The *Ground Normal Form* (GNF) supports compact grounding by attaching conditions to parts of formulas, allowing us to skip irrelevant ground instances when their satisfaction can be determined directly from the structure. Thus, it generalizes the structural simplifications of Example 5 and systematizes them into the grounding process itself. Initially, these conditions are described by the users themselves, however, the goal is to allow users to ground arbitrary sentences in FOL (given a finite domain), and to automatically detect suitable conditions, as done by Wittrockx et al. [8].

The two main advantages of the Ground Normal Form are that:

- formulas in this normal form immediately ground to conjunctive normal form (CNF, a conjunction of disjunctions of propositions or their negation),
- and the use of *binary* quantifiers allows for a smaller grounding by limiting the domain over which a quantifier is instantiated, using the available information.

We deliberately chose to define a normal form that grounds to Conjunctive Normal Form (CNF), the standard format used by most SAT solvers. These solvers are among the most efficient reasoning tools available, and since proof logging is a requirement for participation in the annual SAT competition, most state-of-the-art solvers are certifying. By grounding to this normal form, users gain access to a large portfolio of certifying solving tools in order to reason about the problem at hand. By also providing a machine-verifiable proof that this grounding process correct, the users trust in the tool chain is increased, knowing that the SAT solver is solving the correct problem.

To do this, we will also develop a proof logging system, allowing grounders to log which instantiation and evaluation rules are applied. Note that the goal of a proof is to show that the obtained grounding is correct (in the sense that it is equivalent to the given FOL formula), the goal is *not* to show that the problem has been solved correctly. The effectiveness of this proof logging system hinges on balancing expressivity and efficiency: the format must be expressive enough to integrate into the grounder with minimal overhead, while also being efficient enough to allow straightforward verification of each rule application. Equally

important is the trustworthiness of the proof checker itself—if the checker is unreliable, the value of the proof is lost. Therefore, the verification of proof steps should be simple enough to allow the checker to be formally verified.

Before introducing these so-called binary quantifiers, we adapt our definition of the model expansion problem. The vocabulary  $\mathcal{V}$  is partitioned into two vocabularies:  $\mathcal{V}_{in}$  and  $\mathcal{V}_{out}$ , respectively representing the “input” and “output” vocabularies. Intuitively,  $\mathcal{V}_{in}$  contains those symbols whose interpretation is already provided by the given structure, and  $\mathcal{V}_{out}$  contains those symbols whose interpretation remains to be determined. Hence, given two vocabularies  $\mathcal{V}_{in}$  and  $\mathcal{V}_{out}$ , a structure  $\mathcal{S}_{in}$  over  $\mathcal{V}_{in}$  and a theory  $\mathcal{T}$  over  $\mathcal{V} = \mathcal{V}_{in} \cup \mathcal{V}_{out}$ , the model expansion task is to find a  $\mathcal{V}$ -structure  $\mathcal{S}$  that:

- *Expands*  $\mathcal{S}_{in}$  (meaning it has the same domain and agrees with  $\mathcal{S}_{in}$  on all symbols in  $\mathcal{V}_{in}$ ),
- and *Satisfies*  $\mathcal{T}$  (i.e., it is a model of  $\mathcal{T}$ ).

To take full advantage of the information available in  $\mathcal{S}_{in}$ , GNF uses binary quantifiers (inspired by [4]).

► **Definition 6.** *Given formulas  $\varphi$  and  $\psi(x)$  in FOL, the binary quantifiers used in GNF are defined as follows:*

$$\begin{aligned}\forall x[\varphi(x)] : \psi(x) &\stackrel{\text{def}}{=} \forall x : \varphi(x) \Rightarrow \psi(x) \\ \exists x[\varphi(x)] : \psi(x) &\stackrel{\text{def}}{=} \exists x : \varphi(x) \wedge \psi(x).\end{aligned}$$

We call  $\varphi$  a condition on  $\psi$  and write  $Qx : \psi(x)$  as an abbreviation for  $Qx[\top] : \psi(x)$ , where  $Q \in \{\forall, \exists\}$ . Given  $n \geq 0$  and  $Q \in \{\forall, \exists\}$ , we write  $Q\bar{x}[\varphi]$  as an abbreviation for

$$Qx_1[\top] : Qx_2[\top] : \dots : Qx_n[\varphi(\bar{x})],$$

and  $Q\bar{x}[\bar{\varphi}]$  is an abbreviation for

$$Qx_1[\varphi_1(x_1)] : Qx_2[\varphi_2(x_1, x_2)] : \dots : Qx_n[\varphi_n(\bar{x})].$$

The ground normal form is defined as follows:

► **Definition 7.** *A FOL formula is in Ground Normal Form if it is a universally quantified disjunction of existentially quantified atoms or their negation, such that every quantification has a condition and every condition consists only of symbols belonging to  $\mathcal{V}_{in}$ .*

Given a formula in GNF, we get that for each value  $v \in \mathcal{D}$  and each formula  $Qx : [\varphi(x)] : \psi(x)$  with binary quantifier  $Q \in \{\forall, \exists\}$ , only  $\mathcal{S}_{in}$  is required to determine the truth value of  $\varphi(v)$ . As such, it is guaranteed that these conditions can be evaluated during the grounding phase.

► **Example 8.** The FOL formula in Equation (3) can be expressed in GNF as follows:

$$\begin{aligned}\forall x : A(x) \vee B(x) &\Rightarrow \exists y : C(y) \wedge D(x, y) \\ \Leftrightarrow \forall x : A(x) \vee \neg B(x) \vee \exists y : C(y) \wedge D(x, y) &\quad \text{implication to conjunction} \\ \Leftrightarrow \forall x : B(x) \Rightarrow A(x) \vee \exists y : C(y) \wedge D(x, y) &\quad \text{reordering, conjunction to implication} \\ \Leftrightarrow \forall x[B(x)] : A(x) \vee \exists y[C(y)] : D(x, y) &\quad \text{Definition 6}\end{aligned}$$

For now, we have chosen to limit this normal form to only include atoms that do not contain function symbols. In the future, we will expand this normal form to include function and object symbols as well, and eventually, the goal is to ground *arbitrary* FO sentences over finite domains. We strive to go even further than that and to also support a rich system of extensions for FOL, such as types, arithmetic, aggregates, and even (inductive) definitions.

## 5 Grounding GNF

One of the main advantages of the *Ground Normal Form* is that sentences in GNF immediately ground to CNF (as opposed to an arbitrary propositional formula). Furthermore, the conditions of the binary quantifiers limit the size of the obtained grounding. We first introduce instantiation rules for the binary quantifiers.

► **Definition 9** (Instantiation Rules for binary quantifiers).

$$\forall_{\text{inst}}^{\text{GNF}} \frac{\forall x[\varphi(x)] : \psi(x)}{\bigwedge_{v \in D: \mathcal{S}_{in} \models \varphi(x:v)} \psi(x:v)}$$

where  $\forall x[\varphi(x)] : \psi(x)$  is in GNF, and its dual

$$\exists_{\text{inst}}^{\text{GNF}} \frac{\forall \bar{x}[\bar{\varphi}] : \exists y[\xi(\bar{x}, y)] : \psi_1(\bar{x}, y) \vee \dots \vee \psi_m(\bar{x})}{\forall \bar{x}[\bar{\varphi}] : \bigvee_{v \in D: \mathcal{S}_{in} \models \exists \bar{x}: \xi(\bar{x}, v)} \psi_1(\bar{x}, v) \vee \dots \vee \psi_m(\bar{x})}$$

where  $m > 0$  and  $\forall \bar{x}[\bar{\varphi}] : \exists y[\xi(\bar{x}, y)] : \psi_1(\bar{x}, y) \vee \dots \vee \psi_m$  is in GNF.

Given a sentence  $\phi$  in GNF, an equivalent quantifier-free formula can be obtained by applying the instantiation rules  $\forall_{\text{inst}}^{\text{GNF}}$  and  $\exists_{\text{inst}}^{\text{GNF}}$ . Note that the conditions of these instantiation rules limit the scope of the instantiation: instead of instantiating  $x$  by every value in the domain, only values that fulfill the condition are taken into account. Because it is required that these conditions are defined over the input vocabulary  $\mathcal{V}_{in}$  (and thus interpreted by  $\mathcal{S}_{in}$ ), the truth value of a condition  $\varphi(x : v)$  can be fully determined. This is done by grounding  $\varphi(x : v)$  to an equivalent, quantifier-free formula before evaluating the remaining atoms using their intuitive interpretation and the evaluations defined in  $\mathcal{S}_{in}$ . For GNF to support functions and object symbols, an approach similar to the one described Section 3 can be used.

► **Example 10.** Grounding the formula from Example 8 immediately yields the result obtained in Example 5;

$$\begin{aligned} & \forall x[B(x)] : A(x) \vee \exists y[C(y)] : D(x, y) \\ \Leftrightarrow & \bigwedge_{v \in D: \mathcal{S} \models B(v)} A(v) \vee \exists y[C(y)] : D(x, y) & \forall_{\text{inst}}^{\text{GNF}} \\ \Leftrightarrow & \bigwedge_{v \in D: \mathcal{S} \models B(v)} A(v) \bigvee_{w \in D: \mathcal{S} \models C(w)} D(v, w) & \exists_{\text{inst}}^{\text{GNF}} \end{aligned}$$

## 6 Architecture of the grounder

We have implemented a prototype for the GNF grounder using C++. It consists of a parser, which parses the theory, input structure, and in- and output vocabularies before checking them for syntactical and semantical errors. This parser was created using the parser generator tool ANTLR [5]. This tool generates a parser that creates a parse tree for a given grammar. Each ANTLR parser also comes with several classes that allow users to work with the obtained parse tree. Each obtained parse tree is then transformed into an abstract syntax tree representing the formula at hand. This abstract syntax tree can then be grounded and simplified, which returns a new abstract syntax tree representing a formula in conjunctive normal form. The prototype has not been extended with proof logging yet, but this is the very next step in the development process. Developing and describing a suitable proof format is work in progress.

## 7 Grounding Extensions of FOL

Once the prototype has reached a state where it is possible to ground GNF *with* proof logs, we will extend our scope to include some very useful extensions of first-order logic. The reason to work with FOL and its extensions has been inspired by the expressivity of the FO( $\cdot$ ) [3] language used by the IDP [2] and IDP-Z3 [1] reasoning systems. We illustrate the idea of extending the present methodology by introducing a type system.

In the case of *multi-sorted* (or typed) logic, the vocabulary consists of a finite set  $T = \{T_1, \dots, T_n\}$  of types, as well as signatures for each predicate and function symbol (i.e., respectively of the form  $T_1 \times \dots \times T_n \rightarrow \mathbb{B}$  and  $T_1 \times \dots \times T_n \rightarrow T_{n+1}$  for  $n$ -ary symbols), and a domain  $\mathcal{D}_{T_i}$  for each type  $T_i \in T$ . In order to model multi-sorted logic using FOL, add the following sentences for each  $n$ -ary function  $f(T_1, \dots, T_n) : T_{n+1}$  in  $\mathcal{V}$  to ensure that they are type safe:

$$\bigwedge_{i=1, \dots, n} \forall x_1, \dots, x_n, d : f(x_1, \dots, x_n) = d \Rightarrow T_i(x_i) \\ \forall x_1, \dots, x_n, d : f(x_1, \dots, x_n) = d \Rightarrow T_{n+1}(d).$$

Predicates are treated similarly.

Next, for each type  $T_i \in T$ , add the following formula to ensure that a value's type is unique:

$$\bigwedge_{T_j \in T \setminus T_i} \forall x [T_i(x)] : \neg T_j(x).$$

Conveniently, these formulas already are (a conjunction of) formulas in GNF.

Similarly, support for theories such as arithmetic, sets, and aggregates can be added, by extending the original theory accordingly and by instantiating where needed. Adding support for inductive definitions is a non-trivial task that needs a different approach since inductive definitions cannot be expressed using FOL. As such, we will reconsider inductive definitions at a later stage in the research.

## 8 Conclusions and Future Work

In this paper, we introduce the normal form that lies at the core of our novel methodology for certified grounding. We describe how a grounder built around this seemingly restrictive normal form can be expanded to support a rich input language based on first-order logic and its many extensions. As such, this novel methodology has the potential to support many high-level reasoning languages. We describe the need for a certifying grounding methodology, and introduce the requirements for a proof system for certified grounding. Given that the goal is to devise a methodology which is *certifying*, this method could form an important step towards fully trustworthy reasoning systems. To make this methodology as applicable as possible, we invite insights on how a certifying grounder could be integrated into existing solving pipelines, including any requirements that must be met to easily incorporate this methodology into existing systems.

---

References

---

- 1 Pierre Carbonnelle, Simon Vandeveld, Joost Vennekens, and Marc Denecker. Interactive configurator with fo(.) and idp-z3, 2023. URL: <https://arxiv.org/abs/2202.00343>, arXiv: 2202.00343.
- 2 Broes De Cat, Bart Bogaerts, Maurice Bruynooghe, Gerda Janssens, and Marc Denecker. Predicate logic as a modeling language: the IDP system. In Michael Kifer and Yanhong Annie Liu, editors, *Declarative Logic Programming: Theory, Systems, and Applications*, pages 279–323. ACM / Morgan & Claypool, 2018. doi:10.1145/3191315.3191321.
- 3 Marc Denecker and Eugenia Ternovska. A logic of nonmonotone inductive definitions. *ACM Trans. Comput. Log.*, 9(2):14:1–14:52, 2008. doi:10.1145/1342991.1342998.
- 4 Michaelis Michael and A. V. Townsend. Binary quantification systems. *Notre Dame Journal of Formal Logic*, 36(3):382–395, 1995. doi:10.1305/ndjfl/1040149354.
- 5 Terence Parr. *The Definitive ANTLR 4 Reference*. Pragmatic Bookshelf, 2nd edition, 2013.
- 6 Deepak Ramachandran and Eyal Amir. Compact propositional encoding of first-order theories. In *Proceedings of the 20th National Conference on Artificial Intelligence - Volume 1, AAAI'05*, page 340–345. AAAI Press, 2005.
- 7 Stephan Schulz. A comparison of different techniques for grounding near-propositional cnf formulae. In *Proceedings of the Fifteenth International Florida Artificial Intelligence Research Society Conference*, page 72–76. AAAI Press, 2002.
- 8 Johan Wittocx, Maarten Mariën, and Marc Denecker. Grounding with bounds. In Dieter Fox and Carla P. Gomes, editors, *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, July 13-17, 2008*, pages 572–577. AAAI Press, 2008. URL: <http://www.aaai.org/Library/AAAI/2008/aaai08-091.php>.
- 9 Johan Wittocx, Maarten Mariën, and Marc Denecker. Grounding FO and FO(ID) with bounds. *J. Artif. Intell. Res.*, 38:223–269, 2010. doi:10.1613/jair.2980.