Non-cyclic sorts for first-order satisfiability

Konstantin Korovin*

School of Computer Science
The University of Manchester
United Kingdom
korovin@cs.man.ac.uk

Abstract. In this paper we investigate the finite satisfiability problem for first-order logic. We show that the finite satisfiability problem can be represented as a sequence of satisfiability problems in a fragment of many-sorted logic, which we call the non-cyclic fragment. The non-cyclic fragment can be seen as a generalisation of the effectively propositional fragment (EPR) in the many-sorted setting. We show that the non-cyclic fragment is decidable by instantiation-based methods and present a linear time algorithm for checking whether a given clause set is in this fragment. One of the distinctive features of our finite satisfiability translation is that it avoids unnecessary flattening of terms, which can be crucial for efficiency. We implemented our finite model finding translation in iProver and evaluated it over the TPTP library. Using our translation it was possible solve a large class of problems which could not be solved by other systems.

1 Introduction

Currently, the most successful methods for finite model finding in first-order logic are based on exhaustive flattening of function terms [3, 10]. We argue that flattening can have a detrimental effect on efficiency and present a new translation of the finite model finding problem into a fragment of many-sorted logic, which we call *the non-cyclic fragment*.

One of the main properties of the non-cyclic fragment is that it has a finite Herbrand universe and therefore can be seen as a natural generalisation of the EPR fragment (or Bernays-Shönfinkel-Ramsey fragment) in the many-sorted setting. The non-cyclic fragment is defined by imposing a condition on the signature, which prohibits cyclic dependencies between functions. When this condition is satisfied we call the signature non-cyclic. Variants of this fragment have been investigated in a different context in [1, 12, 14]. In this paper we take an algorithmic point of view. First, we argue that instantiation-based reasoning methods such as Inst-Gen [15, 20] and Model Evolution [6] are decision procedures for the non-cyclic fragment. Second, we present a linear time algorithm for checking whether a many-sorted signature is non-cyclic and more generally for classifying sorts into cyclic and non-cyclic sorts. Let us note that in a number of applications such as hardware verification and knowledge representation, signatures can contain many thousands of symbols and hence we need efficient algorithms for checking whether a signature is non-cyclic.

* Supported by a Royal Society University Fellowship.

We observe that in many cases problems do not fall completely into the EPR or more generally non-cyclic fragment, but rather contain combinations of EPR/non-cyclic sorts with cyclic sorts. We propose to take advantage of this sort separation in the setting of finite model finding. For this we extended a framework of the EPR-based finite model finding developed in [3]. The translation in [3] is based on exhaustive flattening of terms which allows one to replace functions with predicates. Exhaustive flattening of terms is necessary when searching for minimal models with respect to the number of elements. However, flattening can also be detrimental for performance of reasoning methods due to weakening the role of unification. In this paper we propose to reduce the amount of flattening without compromising completeness with respect to finite satisfiability. Our main idea is to use cyclic/non-cyclic sort separation to restrict flattening to a subset of sorts, which we call the sort-restricted flattening. In this way we avoid flattening of certain terms and translate the problem of finite model finding into a sequence of satisfiability problems in the non-cyclic fragment. Our sort-restricted transformation is complete for finite satisfiability but the minimality requirement on the obtained models is relaxed: only flattened sorts will be minimal. Since the non-cyclic fragment can be decided by instantiation-based methods it is natural to to use instantiation based reasoning systems such as iProver [19], Darwin [4] or Equinox [8].

We implemented our sort classification algorithm in iProver and evaluated it over the TPTP library [28] which is the largest available collection of first-order problems. Since most of the problems in TPTP are unsorted we use a sort inference algorithm to automatically annotate first-order problems with sorts, which is similar to one used by Paradox [10]. Our experiments show interesting results. We observe that many problems in domains ranging from verification to knowledge representation contain combinations of EPR, non-cyclic and cyclic sorts. Therefore it seems promising to advance reasoning methods which can benefit from a such combination. Second, we show that using our sort-restricted transformation it was possible to solve a large class of problems with the rating 1 from TPTP v5.3 (54 in total), which could not be solved by any other known system. The sort-restricted transformation helped iProver to win the FNT (First-order Non-Theorems) division at CASC@Turing 2012.

Related work. In [3], an EPR-based translation of finite satisfiability was developed based on exhaustive flattening of terms. In this paper we provide a translation into the non-cyclic fragment of many-sorted logic which allows one to avoid unnecessary flattening. Many-sorted logic has been intensively used in the SMT community and its advantage for first-order modelling has been advocated in e.g., [1, 12, 14]. The non-cyclic fragment can be shown to be equivalent to the St_0 fragment in [1]. In this paper we present a linear-time algorithm for checking whether the formula is in the non-cyclic fragment, propose a separation of sorts into cyclic and non-cyclic with applications to finite satisfiability, and argue that instantiation-based methods are decision procedures for this fragment. An interesting method for satisfiability of quantified formulas in the SMT setting was presented in [17], which is based on representing relevant domains using set constraints. This approach can be also applied to first-order logic, but unlike our method, is not necessarily complete with respect to finite satisfiability. It would be interesting to investigate how our method can be combined with [17], and a very recent approach to finite model finding in the SMT setting presented in [24, 25].

2 Preliminaries

In this paper we consider many-sorted first-order logic with equality. A signature is a tuple $\Sigma = \langle \mathcal{S}, \mathcal{F}, \mathcal{P}, arity_{\mathcal{F}}, arity_{\mathcal{P}} \rangle$ consisting of a non-empty set of sorts $\mathcal{S}$, a set of function symbols $\mathcal{F}$, a set of predicate symbols $\mathcal{P}$, arity functions $arity_{\mathcal{F}} : \mathcal{F} \to \mathcal{S}^* \times \mathcal{S}$ and $arity_{\mathcal{P}} : \mathcal{P} \to \mathcal{S}^*$, where $\mathcal{S}^*$ denotes the set of finite sequences of sorts. For a function symbol f with arity $arity_{\mathcal{F}}(f) = \langle \langle s_0, \ldots, s_{n-1} \rangle, s_n \rangle$, we call $s_0, \ldots, s_{n-1}$ *argument sorts* and s_n the *value sort* of f. A constant is a function with the empty sequence of arguments. We assume that there is at least one constant of each sort. In this paper we consider only finite signatures. For each sort s we consider a countable set of variables of this sort denoted as V_s. Equality over a sort s will be denoted by $\simeq_s$ and is assumed to be a logical symbol (not included in Σ), defining equality over elements of the same sort. We will omit index s in $\simeq_s$ when the sort is clear from the context. Well-sorted terms and atoms are built from variables and sort-respecting applications of function and predicate symbols in the usual way. We say that a term t is of sort s, denoted as $sort(t) = s$, if either t is a variable of sort s or the top function symbol of t has the value sort s. A literal is an atom or its negation and a clause is a multi-set of literals. We will not distinguish clauses equivalent up to renaming of variables.

A *many-sorted interpretation (structure)* A (over Σ) consists of 1) a domain $\mathrm{dom}(A)$ which is a disjoint union of non-empty sets $\cup_{s \in \mathcal{S}} A_s$ indexed by sorts, 2) a collection of functions $f^A : A_{s_0} \times \cdots \times A_{s_{n-1}} \mapsto A_{s_n}$ where $arity_{\mathcal{F}}(f) = \langle \langle s_0, \ldots, s_{n-1} \rangle, s_n \rangle$, for each $f \in \mathcal{F}$, and 3) a collection of relations $P^A : A_{s_0} \times \cdots \times A_{s_{n-1}}$ where $arity_{\mathcal{P}}(P) = \langle s_0, \ldots, s_{n-1} \rangle$ for each $P \in \mathcal{P}$.

Unsorted first-order logic can be seen as an instance of sorted logic with a single sort.

An expression is ground if it does not contain variables. A *Herbrand universe* over a signature Σ is the set of all ground terms. It is folklore knowledge that automated reasoning methods such as resolution, superposition and instantiation can be straightforwardly adapted from unsorted logic to many-sorted by changing the unification algorithm to sort-aware unification.

3 Non-cyclic sorts and finite Herbrand universe

First we consider the EPR fragment of first-order logic in clausal form, which can be defined as follows. An *EPR signature* is a finite signature which does not contain function symbols other than constants. The *EPR fragment in clausal form* consists of sets of clauses over an EPR signature. One of the main properties of EPR signatures is that the set of all ground terms (the *Herbrand universe*) over such a signature is finite. This implies that the set of all (not necessarily ground) instances of any finite set of EPR clauses is also finite. A direct consequence of this is that reasoning methods such as Inst-Gen [15, 20], Model Evolution [6], Equinox [8], BUMG [5] and DPLL(SX) [23] are decision procedures for the EPR fragment.

In unsorted first-order logic EPR signatures are exactly the signatures with a finite Herbrand universe. As noted in [1, 14], in the presence of sorts, signatures different from EPR can also have a finite Herbrand universe. We characterise them below.

Definition 1. *Consider a signature $\Sigma = \langle \mathcal{S}, \mathcal{F}, \mathcal{P}, arity_{\mathcal{F}}, arity_{\mathcal{P}} \rangle$. A* sort dependency graph *of Σ is a directed graph $SD(\Sigma) = \langle \mathcal{S}, SR \rangle$ with the set of vertices $\mathcal{S}$ and the edge relation SR such that $SR(s_1, s_2)$ if and only if there is a function symbol $f \in \mathcal{F}$ with an argument sort s_1 and the value sort s_2.*

A *path* (of *length* n) in a graph G is a sequence of vertices $v_0, \ldots, v_n$ such that each pair (v_i, v_{i+1}) is in the edge relation of G, where $0 \leq n$ and $0 \leq i < n$. Note that we allow a path to be of the zero length, i.e., consist of a single vertex. A path is *non-trivial* if its length is strictly greater than 0.

Definition 2. *A sort s is called* cyclic *in Σ if there exists a non-trivial path in the sort dependency graph from s to s, otherwise it is called non-cyclic. A signature Σ is called* cyclic *if there is a cyclic sort in Σ and otherwise it is called non-cyclic.*

Non-cyclic signatures can be seen as a natural generalisation of the EPR signatures preserving the property of having a finite Herbrand universe.

Proposition 1. *The Herbrand universe over any non-cyclic signature is finite. Conversely, if a Herbrand universe over a signature is finite then the signature is non-cyclic.*

Proof. It is easy to see that the depth of any term is bounded from above by the longest path in the sort dependency graph which in the case of non-cyclic signatures is bounded by the number of function symbols. In the case when a signature is cyclic we can construct terms of unbounded depth. From this proposition follows.

We define *the non-cyclic clausal fragment of first-order logic* to consist of sets of clauses over a non-cyclic signature. In a similar way as for the EPR fragment it is easy to see that instance based methods are also decision procedures for the non-cyclic fragment. We formulate this as a theorem for Inst-Gen [15, 20] and Inst-Gen-Eq [16, 22] but it also holds for other instantiation based methods such as Model Evolution. Inst-Gen is an instantiation-based method, complete for first-order logic and Inst-Gen-Eq is its extension with superposition-based equational reasoning. In a nutshell, Inst-Gen and Inst-Gen-Eq combine efficient ground reasoning with gradual instantiations of clauses, based on first-order reasoning.

Theorem 1. *Inst-Gen and Inst-Gen-Eq are decisions procedures for the non-cyclic fragment with equality.*

Proof. (Sketch) Consider a set of clauses over a non-cyclic signature Σ. From Proposition 1 it follows that the Herbrand universe over Σ is finite. Therefore the maximal depth of terms (including non-ground terms) is also bounded. The Inst-Gen calculus only generates instances of the original clauses and since the depth of terms is bounded there are only finitely many such instances. The Inst-Gen calculus treats equality axiomatically. Let us note that adding axioms of equality does not change the non-cyclicity of a clause set.

Inst-Gen-Eq replaces axiomatic equality with superposition-based equational reasoning. Inst-Gen-Eq uses substitutions extracted from unit superposition proofs for instantiating the original clauses. Since the term depth is bounded, there are only finitely many such superposition proofs and only finitely many instances of clauses can be generated.

After analysing problems from the TPTP library we observed that in many cases problems do not fall completely into the EPR or more generally non-cyclic fragment but rather contain combinations of EPR/non-cyclic sorts with cyclic sorts. We refer to Section 7 for details. Our next goal is to partition the signature into cyclic and non-cyclic parts and extend finite model finding methods to gain from this partition.

A *strongly connected component (SCC)* of a directed graph G is a maximal induced subgraph G' of G such that for each pair of vertices in G' there is a path connecting them in G'. A vertex v in a graph G is called *looping* if there is an edge from v to v. An SCC of a graph is called *trivial* if it consists of a non-looping vertex, otherwise it is called *non-trivial*.

Proposition 2. *Consider a signature Σ. A sort s is cyclic in Σ if and only if s belongs to a non-trivial SCC of the sort dependency graph of Σ.*

Proof. If a sort s belongs to a non-trivial SCC then either: 1) s is looping and hence s is cyclic, or 2) there is another sort s' in the same SCC. In the latter case there is a path from s to s' and a path from s' to s and therefore s is also cyclic. For the other direction it is easy to see that if there is a non-trivial path from s to s then this path belongs to the same SCC and therefore this SCC is non-trivial.

The set of all SCCs of the sort dependency graph of a signature Σ will be denoted by $SCC(\Sigma)$. Define the set of cyclic sorts over Σ as $\mathcal{CS}(\Sigma)$ and the set of non-cyclic sorts as $\mathcal{NCS}(\Sigma)$. For any signature Σ, the set of sorts of this signature $\mathcal{S}$ can be partitioned into cyclic and non-cyclic sorts, i.e., $\mathcal{S} = \mathcal{CS}(\Sigma) \cup \mathcal{NCS}(\Sigma)$.

Next we use Tarjan's linear-time algorithm for finding strongly connected components of directed graphs [29] for partitioning a signature into cyclic and non-cyclic sorts.

Theorem 2. *There is a linear-time algorithm that given a signature Σ partitions sorts of Σ into cyclic and non-cyclic sorts.*

Proof. The required algorithm can proceed as follows.

1. Construct the sort dependency graph $SD(\Sigma)$ from Σ.
2. Apply Tarjan's linear time algorithm [29] to obtain the set of all strongly connected components $SCC(\Sigma)$ of $SD(\Sigma)$.
3. The set of all sorts that occur in trivial components of $SCC(\Sigma)$ will be the set of all non-cyclic sorts $\mathcal{NCS}(\Sigma)$ and the set of all sorts that occur in non-trivial components is the set of all cyclic sorts $\mathcal{CS}(\Sigma)$.

It is easy to see that all three steps can be done in time linear in the size of the signature.

Theorem 2 also provides us with a linear time algorithm for checking whether a given signature is cyclic: partition the sort dependency graph into SCC and check whether there is a non-trivial component.

Example 1. Consider a signature $\Sigma = \langle \mathcal{S}, \mathcal{F}, \mathcal{P}, arity_{\mathcal{F}}, arity_{\mathcal{P}} \rangle$ where $\mathcal{S} = \{s_0, s_1, s_2\}$, $\mathcal{F} = \{f, g, h, c_0, c_1, c_2\}$ and $arity(f) = \langle \langle s_0, s_1 \rangle, s_2 \rangle$, $arity(g) = \langle \langle s_0 \rangle, s_0 \rangle$, $arity(h) = \langle \langle s_0 \rangle, s_1 \rangle$ and $arity(c_i) = \langle \langle \rangle, s_i \rangle$ for $0 \leq i \leq 2$. An example of a well-formed term in

this signature can be $t = f(g(x), h(g(g(x))))$, where x is a variable of sort s_0. The sort dependency graph $SD(\Sigma)$ and its strongly connected components $SCC(\Sigma)$ are shown on Figure 1. From this we can see that the set of cyclic sorts is $\mathcal{CS}(\Sigma) = \{s_0\}$ and the set of non-cyclic sorts is $\mathcal{NCS}(\Sigma) = \{s_1, s_2\}$. We can see that the term t is of a non-cyclic sort but contains subterms of cyclic and non-cyclic sorts.

Fig. 1. The sort dependency graph $SD(\Sigma)$ and its strongly connected components $SCC(\Sigma)$.

4 EPR-based finite model finding

In this section we overview a translation of the finite satisfiability problem into the EPR fragment on which we will base our translation into the non-cyclic fragment. Our presentation follows [3] with a slight adaptation to the sorted setting. Let us recapture several notions from [3]. An atom is called *(completely) flat* if it has one of the following forms: 1) $p(x_0, \ldots, x_{n-1})$ where p is a predicate, 2) $x \not\simeq f(x_0, \ldots, x_{n-1})$, where f is a function symbol, or 3) $x \simeq y$. Let us note that function symbols can occur only in flat atoms of the from $x \not\simeq f(x_0, \ldots, x_{n-1})$. A literal is flat if its atom is flat. A clause is flat if all its literals are flat. Consider a clause $C[t]$. A *flattening transformation* applied to $C[t]$ wrt. t produces a clause $x \not\simeq t \vee C[x]$ where x is a fresh variable. By applying the flattening transformation we can transform any set of clauses into an equivalent set of flat clauses [2, 3, 7]. Let $\mathcal{FT}(S)$ denote a set of flat clauses obtained from S by applying the flattening transformation.

Consider a set of flat clauses S over a signature Σ. For each function symbol in Σ with arity $arity(f) = \langle\langle s_0, \ldots, s_{n-1}\rangle, s_n\rangle$ we introduce a new predicate symbol P_f with arity $arity(P_f) = \langle s_0, \ldots, s_{n-1}, s_n\rangle$. Informally, the predicate P_f will be used to represent the function f with the last argument representing the value of f. We call these introduced predicates as *function predicates*. Now we can eliminate functions from our clause set by applying the following *function elimination transformation*:

$$y \not\simeq f(x_0, \ldots, x_{n-1}) \vee C \Rightarrow \neg P_f(x_0, \ldots, x_{n-1}, y) \vee C.$$

Let $\mathcal{FE}(S)$ denote the set of clauses obtained by exhaustive applications of function elimination to S. In order to ensure that a function predicate P_f represents a graph of a function we need to require P_f to be left-total and right-unique, as defined below. Consider an interpretation I with a relation P of $arity(P) = \langle s_0, \ldots, s_{n-1}, s_n\rangle$, where $0 \leq n$. The relation P is called *left-total* in I if for every sequence of elements

$a_0, \ldots, a_{n-1}$ in I, of the respective sorts $s_0, \ldots, s_{n-1}$, there exists an element $a_n \in I$ of sort s_n such that $P(a_0, \ldots, a_n)$ holds in I. The relation P is called *right-unique* in I if whenever $I \vDash P(a_0, \ldots, a_{n-1}, a_n)$ and $I \vDash P(a_0, \ldots, a_{n-1}, a'_n)$ then $I \vDash a_n \simeq a'_n$. It is shown in [3] that we can drop the right-uniqueness requirement when we consider flat clauses, preserving finite satisfiability. When we consider finite domains we can express left-totality in the EPR fragment as shown below.

Let us consider finite satisfiability of flat clauses. For each sort s we fix a finite set of constants $d^s_1, \ldots, d^s_k$, called *domain constants*, representing all elements of this sort. Collection of all domain constants will be called a *constant domain* and denoted by $\mathcal{D}$.

Then left-totality of a function predicate P_f over a constant domain can be expressed using the following *totality axiom*:

$$P_f(x_0, \ldots, x_{n-1}, d^s_1) \vee \cdots \vee P_f(x_0, \ldots, x_{n-1}, d^s_k),$$

where $d^s_1, \ldots, d^s_k$ are all domain constants of the sort s.

For a constant domain $\mathcal{D}$, let $TAx(\mathcal{D})$ denote the set of all totality axioms of function predicates in the signature. For a set of clauses S and a constant domain $\mathcal{D}$ we call the set of clauses $BFM(S, \mathcal{D}) = \mathcal{FE}(\mathcal{FT}(S)) \cup TAx(\mathcal{D})$ the *basic finite model finding translation* of S with respect to $\mathcal{D}$.

Theorem 3. *[3] A set of clauses S is satisfiable over a finite constant domain $\mathcal{D}$ if and only if $BFM(S, \mathcal{D})$ is satisfiable.*

As shown in [3], due to flattening it is also possible to eliminate the equality predicate altogether by introducing axioms stating disequality of the domain constants: $\mathcal{E}(\mathcal{D}) = \{d^s_i \not\simeq_s d^s_j \mid i \neq j, d^s_i, d^s_j \in \mathcal{D}\}$. After adding the disequality axioms one can replace the equality predicate $\simeq_s$ by a fresh binary predicate $\mathcal{E}_s$ for each sort s, preserving satisfiability. We denote this translation as $BFME(S, \mathcal{D})$.

Let us note that the result of applying any of the translations BFM, or $BFME$ is always an EPR set of clauses, and therefore instantiation-based methods can be used for checking satisfiability of $BFM(S, \mathcal{D})$ and $BFME(S, \mathcal{D})$.

Without loss of generality we can restrict ourselves to the Herbrand interpretations which in this case are built over the domain constants. The search for finite satisfiability then starts with a constant domain consisting of a single constant in each sort and then proceeds by iteratively adding new constants until $BFME(S, \mathcal{D})$ becomes satisfiable. One of the properties of this approach is that if a set of clauses is finitely satisfiable then we obtain a minimal model with respect to the number of domain elements.

5 Flattening and finite model finding

As we have seen, flattening is essential for the basic finite model finding translation. Unfortunately, it also can have a detrimental effect on reasoning methods.

Example 2. Consider an unsorted signature consisting of n unary predicates $P_1, \ldots P_n$ and n constants $c_1, \ldots, c_n$. Consider the following set of ground unit clauses:

$$S = \bigcup_{1 \leq i \leq n} \{P_i(c_i)\} \bigcup \bigcup_{1 \leq i < j \leq n} \{\neg P_i(c_j)\}.$$

Satisfiability of S can be trivially shown by propositional reasoning, (considering $P_i(c_j)$ as propositional atoms). Let us consider the basic finite model translation applied to S. After flattening and introduction of function predicates P_{c_i} for each constant c_i, $1 \leq i \leq n$ we obtain the following set of clauses:

$$\mathcal{FE}(\mathcal{FT}(S)) = \bigcup_{1\leq i\leq n} \{\neg P_{c_i}(x) \vee P_i(x)\} \bigcup \bigcup_{1\leq i<j\leq n} \{\neg P_{c_j}(x) \vee \neg P_i(x)\}.$$

It is easy to see that any satisfying interpretation for this set of clauses will have the domain size at least n. Therefore the finite model finding procedure will iteratively increase the domain size, by adding domain axioms until n is reached. Moreover reasoning with intermediate domain sizes can be nontrivial due to introduced symmetries.

Example 2 is deliberately simple. Let us slightly modify this example by adding a "dummy" argument to each predicate P_i and replace $P_i(c_j)$ with $P_i(c_j, f(x))$. Then, although this change obviously does not affect satisfiability, the resulting set is challenging for instantiation-based systems.

We can see that even for simple problems flattening can introduce variables into the problem and increase the search space. Our approach aims at reducing the amount of unnecessary flattening. First we can observe that if our problem is EPR as in Example 2, then we can apply instantiation-based methods directly to such a problem without applying the flattening transformation. In the next section we restrict flattening further to terms of cyclic sorts.

6 Sort-restricted flattening

Our approach is to restrict flattening to specified sorts and at the same time keep the resulting translation in the non-cyclic fragment.

Consider a signature Σ with the set of sorts $\mathcal{S}$. Consider a subset of $\mathcal{S}' \subseteq \mathcal{S}$. We say that an interpretation I is *$\mathcal{S}'$-finite* if each sort in $\mathcal{S}'$ has a finite domain in I. A formula is *$\mathcal{S}'$-finitely satisfiable* if it is satisfied in an $\mathcal{S}'$-finite interpretation. $\mathcal{S}'$-finite satisfiability generalises finite satisfiability, that is if a formula is finitely satisfiable then it is also $\mathcal{S}'$-finitely satisfiable for any $\mathcal{S}' \subseteq \mathcal{S}$, but the converse in general does not hold.

The *sort-restricted flattening transformation* with respect to $\mathcal{S}'$ is defined as follows:

$$L[t] \vee C \Rightarrow x \not\simeq t \vee L[x] \vee C,$$

where:

1. t is not a variable,
2. $sort(t) \in \mathcal{S}'$,
3. $L[t]$ has one of the forms: $t \simeq s$, $s \simeq t$, $t \not\simeq s$, or $(\neg)P[t]$ for a predicate P, and
4. x does not occur in $L[t] \vee C$.

The result of exhaustive application of the sort-restricted flattening transformation to a set of clauses S, is denoted as $\mathcal{FTR}(\mathcal{S}', S)$.

We sort-restrict other ingredients of the finite satisfiability transformation:

1. function elimination is restricted to functions with value sorts in $\mathcal{S}'$, denoted by $\mathcal{FER}(\mathcal{S}', S)$;
2. a sort-restricted finite constant domain (or just sort-restricted constant domain) is a collection of constants $\bar{d}^{s_1}, \ldots, \bar{d}^{s_m}$, denoted by $\mathcal{DR}(\mathcal{S}')$, where $\mathcal{S}' = \{s_1, \ldots, s_m\}$ and each $\bar{d}^{s_i}$ is a non-empty sequence of constants of sort s_i. We assume that domain constants are fresh for the signature Σ;
3. totality axioms for function predicates over a sort-restricted constant domain $\mathcal{DR}(\mathcal{S}')$ are defined as in Section 4 and will be denoted as $TAx(\mathcal{DR})$.

Consider a set of clauses S over signature Σ. We say that S is satisfiable in a sort-restricted constant domain $\mathcal{DR}(\mathcal{S}')$ (or $\mathcal{DR}(\mathcal{S}')$ *satisfiable*) if there is a model I of S which can be expanded with constants from $\mathcal{DR}(\mathcal{S}')$ so that each element in I of a sort $s \in \mathcal{S}'$ is named by a constant in $\mathcal{DR}(\mathcal{S}')$. It is easy to see that S is $\mathcal{S}'$*-finitely* satisfiable if and only if there is a sort-restricted constant domain $\mathcal{DR}(\mathcal{S}')$ such that S is $\mathcal{DR}(\mathcal{S}')$ satisfiable.

For a set of clauses S, a subset of sorts $\mathcal{S}' \subseteq \mathcal{S}$ and a sort-restricted constant domain $\mathcal{DR}(\mathcal{S}')$ we call the set of clauses $BFMR(\mathcal{S}', S, \mathcal{DR}) = \mathcal{FER}(\mathcal{S}', \mathcal{FTR}(\mathcal{S}', S)) \cup TAx(\mathcal{DR})$ *the sort-restricted finite model finding translation* (or just the sort-restricted translation) of S with respect to $\mathcal{S}'$ and $\mathcal{DR}(\mathcal{S}')$.

Theorem 4. *Consider a signature Σ with a set of sorts $\mathcal{S}$, a subset of sorts $\mathcal{S}' \subseteq \mathcal{S}$ and a sort-restricted constant domain $\mathcal{DR}(\mathcal{S}')$. A set of clauses S over Σ is $\mathcal{DR}(\mathcal{S}')$ satisfiable if and only if the sort-restricted translation $BFMR(\mathcal{S}', S, \mathcal{DR})$ is satisfiable.*

Proof. Adaptation of results from [3].

Similar to the basic case we can eliminate equality predicate over sorts in $\mathcal{S}'$ by first adding axioms stating disequality of the domain constants:

$$\mathcal{E}(\mathcal{DR}) = \{d_i^s \not\simeq_s d_j^s \mid i \neq j, d_i^s, d_j^s \in \mathcal{DR}\}$$

and then replacing $\simeq_s$ by a fresh binary predicate $\mathcal{E}_s$ for each sort $s \in \mathcal{S}'$. We denote this translation as $BFMER(\mathcal{S}', S, \mathcal{DR})$. Let us note that after applying $BFMER(\mathcal{S}', S, \mathcal{DR})$, equality can still remain between terms of sorts which are not in $\mathcal{S}'$.

In order to keep $BFMER(\mathcal{S}', S, \mathcal{DR})$ in a decidable fragment we propose to restrict flattening to a superset of cyclic sorts. For this, we define a set of sorts to which we do not apply flattening to be any subset of non-cyclic sorts $\mathcal{NCS}(\Sigma)$, which we call *non-flattening sorts* and denote by $\mathcal{NFS}(\Sigma)$. Then, the set of sorts to which we apply flattening will be $\mathcal{S} \setminus \mathcal{NFS}(\Sigma)$, which we call *flattening sorts* and denote by $\mathcal{FS}(\Sigma)$. Examples of non-flattening sorts include the empty set of sorts, the set of EPR sorts and the set of all non-cyclic sorts.

Proposition 3. *Consider a signature Σ, a set of flattening sorts $\mathcal{FS}(\Sigma)$ and a sort-restricted constant domain $\mathcal{DR}(\mathcal{FS})$. Then, for any set of clauses S the sort-restricted translation $BFMER(\mathcal{FS}, S, \mathcal{DR})$ is in the non-cyclic fragment.*

The search for finite satisfiability then starts with a sort-restricted constant domain consisting of a single constant in each flattening sort and then proceeds by iteratively

adding new constants into the sort-restricted domain until $BFMER(\mathcal{FS}, S, \mathcal{D})$ becomes satisfiable. From Theorem 4 and our remarks above it follows that this method is complete with respect to finite model finding and more generally with respect to $\mathcal{FS}(\Sigma)$-finite model finding. In particular, if a set of clauses has a finite model then the procedure will find a sort-restricted constant domain which satisfies this set of clauses.

Let us note an essential difference between basic and sort-restricted translations: in the basic case the finite model finding is restricted to minimal models (with respect to the number of elements), whereas in the sort-restricted case this requirement is relaxed so that only domains of flattening sorts will be minimal but domains of non-flattening sorts can be arbitrary interpretations.

Example 3. Consider the problem from Example 2. Since all sorts in this example are EPR we can take them as the set of non-flattening sorts $\mathcal{NFS}(\Sigma)$. In this case the sort-restricted finite model finding transformation will not change the set of clauses.

Example 4. Let us consider the signature Σ from Example 1 and a clause C:

$$f(x, h(g(x))) \simeq f(x, c_1) \vee h(x) \simeq c_1.$$

Let us apply sort-restricted finite model finding transformation to C with the set of non-flattening sorts to be the set of all non-cyclic sorts: $\mathcal{NFS} = \{s_1, s_2\}$. The result of exhaustive application of sort-restricted flattening to C is:

$$y \not\simeq g(x) \vee f(x, h(y)) \simeq f(x, c_1) \vee h(x) \simeq c_1.$$

The result of sort-restricted function elimination is:

$$\neg P_g(x, y) \vee f(x, h(y)) \simeq f(x, c_1) \vee h(x) \simeq c_1.$$

And the domain axioms are of the form:

$$P_g(x, d_1^{s_0}) \vee \ldots \vee P_g(x, d_k^{s_0}),$$

where $d_1^{s_0}, \ldots, d_k^{s_0}$ are domain constants of sort s_0.

Let us compare this to the case when $\mathcal{NFS}$ is the empty set, which reduces the sort-restricted transformation to the basic transformation.

After applying flattening to C we obtain a much longer clause:

$$\begin{array}{c} y_1 \not\simeq g(x) \vee y_2 \not\simeq h(y_1) \vee y_3 \not\simeq c_1 \vee y_4 \not\simeq f(x, y_2) \vee y_5 \not\simeq f(x, y_3) \vee y_6 \not\simeq h(x) \\ \vee \\ y_4 \simeq y_5 \vee y_6 \simeq y_3. \end{array}$$

We can also note that in this example basic flattening introduces positive equations between variables like $y_4 \simeq y_5$ and $y_6 \simeq y_3$ which can be problematic for reasoning methods.

After function elimination we obtain:

$$\begin{array}{c} \neg P_g(x, y_1) \vee \neg P_h(y_1, y_2) \vee \neg P_{c_1}(y_3) \vee \neg P_f(x, y_2, y_4) \vee \neg P_f(x, y_3, y_5) \vee \neg P_h(x, y_6) \\ \vee \\ y_4 \simeq y_5 \vee y_6 \simeq y_3. \end{array}$$

And domain axioms are of the form:

$$P_g(x_0, d_1^{s_0}) \vee \ldots \vee P_g(x_0, d_{k_0}^{s_0})$$
$$P_h(x_0, d_1^{s_1}) \vee \ldots \vee P_h(x_0, d_{k_1}^{s_1})$$
$$P_{c_1}(d_1^{s_1}) \vee \ldots \vee P_{c_1}(d_{k_1}^{s_1})$$
$$P_f(x_0, x_1, d_1^{s_2}) \vee \ldots \vee P_f(x_0, x_1, d_{k_2}^{s_2}).$$

As we can see from this example, basic transformation can result in a considerably larger set of clauses. Moreover positive equations between variables can be introduced which can be avoided when the sort-restriction transformation is applied.

Iterative flattening. Let us briefly discuss an extension of our method which further refines flattening applications. This is based on the following observation. Consider a signature Σ with the set of sorts $\mathcal{S}$. If we apply the sort-restricted flattening to a single sort $s \in \mathcal{S}$, then all function symbols with the value sort s will be replaced by predicate symbols, resulting in a new signature Σ'. It is easy to see that the sort dependency graph for Σ' can be obtained from the sort dependency graph of Σ by removing edges adjacent to s. In particular, if we pick s from a non-trivial strongly connected component, after eliminating all edges adjacent to s we may be able to decompose this strongly connected component further into smaller components. The process of flattening sorts one by one and decomposing the corresponding strongly connected components can be repeated until only trivial components remain. The advantage of this approach is that we can reduce the number of sorts that require flattening even further. As an example let as consider a signature Σ with the set of sorts $\mathcal{S} = \{s_0, \ldots, s_n\}$, forming a single cycle $s_0, \ldots, s_n, s_0$ in the sort dependency graph. There is only one strongly connected component in this sort dependency graph, which is this cycle itself. After flattening only one sort, say s_0, the sort dependency graph can be completely decomposed into trivial strongly connected components. This allows us to avoid flattening of the remaining sorts $\{s_1, \ldots, s_n\}$.

7 Implementation and evaluation

Our implementation is based on the iProver system [19]. iProver is based on the Inst-Gen calculus which is complete for first-order logic. As we argued in Section 3, Inst-Gen is also a decision procedure for the non-cyclic fragment. iProver accepts first-order problems in CNF form. For problems in full first-order syntax (FOF) we used Vampire [18, 26] as an external clausifier, optionally E prover [27] can also be used as a clausifier.

For our evaluation we used the TPTP library v5.3 [28], which contains 15,550 FOF and CNF problems. Our experiments were run on a cluster of Linux machines with memory limit 2GB and 2.33GHz CPU. We separate our experimental results into two classes. The first class is related to sort inference and the second class is related to experiments with sort-restricted finite model finding.

Sort inference. iProver implements a sort inference algorithm similar to one implemented in Paradox [10] which transforms unsorted first-order clauses into a set of sorted

clauses. This algorithm first assigns different sorts to all predicate arguments, function arguments and values, then it applies a union-find algorithm to merge sorts forced to be equal due to variable dependencies, or occurrences of the equality predicate. Extracted sorts are monotone in the sense of [9], which means that for any model satisfying a set of clauses we can extend domain of any sort with new elements without affecting satisfiability. The overall sort inference resulted in 4,090 problems with non-trivial sorts. We implemented an algorithm for classifying sorts into cyclic, EPR and non-cyclic sorts as described in Section 3. Our implementation uses an OCaml library ocamlgraph [11] for computing strongly connected components of directed graphs.

There are 1,383 problems which were recognised by iProver as being pure EPR problems (after clausification). For clarity we exclude pure EPR problems from experiments below. Of course, all discussed methods are trivially applicable to them. We found that after removing pure EPR problems, 1,195 remaining problems have at least one non-cyclic sort, which is around 1/3 of all problems with non-trivial sorts; 1,077 problems have at least one EPR sort. Collectively over all problems there are 56,679 sorts, 18,502 non-cyclic sorts and among them 9,569 EPR sorts . From this we can see that the number of EPR sorts is approximately the same as non-EPR non-cyclic sorts. Also, we can see that most problems with non-cyclic sorts combine EPR and non-EPR non-cyclic sorts.

Problems with non-cyclic sorts are spreading over many domains of TPTP (even after excluding pure EPR problems), most notably software and hardware verification: SWV, SWW, HWV; knowledge representation SWB, KRS, CSR; algebra: TOP, GEO, SET, SEU, RNG; natural language processing: NLP; planning: PLA; and other domains: PUZ, MGT, MSC, SYN. This indicates that non-cyclic sorts occur naturally in many applications and we believe reasoning methods can be tuned to benefit from this. We also believe that if problems were sorted by domain experts rather than by using automatic sort inference, considerably more problems would be identified to have sorts and non-cyclic sorts in particular.

Sort-restricted finite model finding. We implemented our sort-restricted finite model finding translation $BFMER$, as described in Section 6. Our implementation also features symmetry breaking based on sorts similar as it is done in Paradox. Using sort-restricted finite model finding we were able to solve 54 problems in TPTP v5.3 with the rating 1, all from the KRS domain. We also found a bug in TPTP v5.3 where a problem with the rating 1 (KRS264+1), was stated to be a "Theorem" but our experiments showed it to be satisfiable. We thank Geoff Sutcliffe for helping to debug this problem which resulted in fixing an axiomatisation in the most recent version of TPTP v5.4. iProver with the sort-restricted finite model finding participated in the latest CASC competition, and these enhancements helped iProver to win the FNT (First-order Non-Theorems) division at CASC@Turing 2012. iProver also participated in the evaluation of TPTP v5.4. As the result, some satisfiable problems with the rating 1 in TPTP v5.3 are of a lower rating in TPTP v5.4. In total, iProver solved 72% of problems which are classified as Satisfiable or CounterSatisfiable in TPTP v5.3. We experimented with two cases of non-flattening sorts: 1) all EPR sorts, and 2) all non-cyclic sorts. We observed that most problems are solved in the first case (72%). In the second case, there are a number of problems which could not be solved by the first case, but overall per-

formance is a bit worse: only 69% of problems were solved. One possible explanation can be that flattening can still be beneficial in some cases due to axiomatic treatment of equality in iProver. We expect that this can be amended by using iProver-Eq [21] which integrates equality using superposition-based reasoning. Another explanation can be that in some cases searching for minimal models can still be quicker. Our method gives flexibility on which sorts to flatten, we can choose any superset of cyclic sorts or apply even more fine-grained flattening based on iterative flattening as discussed in Section 6. We leave it for the future work to find best strategies for selecting sorts for flattening.

8 Conclusion and future work

In this paper we investigated the non-cyclic fragment of many-sorted first-order logic with equality. We showed that the non-cyclic fragment is decidable by instantiation-based methods. We presented a linear time algorithm for checking whether a given signature is non-cyclic and more generally for classifying sorts into non-cyclic, EPR and cyclic. We presented a translation of finite model finding into a sequence of satisfiability problems in the non-cyclic fragment, which avoids flattening terms of non-cyclic sorts. We implemented our sort classification and finite model finding translation in iProver. Experimental results are encouraging and we were able to solve a large class of problems which could not be solved by other systems.

For the future work we are planning to integrate sort-restricted finite model finding into iProver-Eq. We will also investigate how reasoning methods can benefit from our sort classification in the refutation setting. It is interesting to investigate combinations of the non-cyclic fragment with other theories in the spirit of [13]. We are planning to investigated combinations of our approach to finite satisfiability with resent SMT-based approaches [17, 24, 25].

iProver with implemented features for sort classification and sort-restricted finite model finding is available at: http://www.cs.man.ac.uk/~korovink/iprover/

Acknowledgments

The author is grateful to anonymous reviewers for providing detailed comments which helped to improve this paper.

References

1. A. Abadi, A. Rabinovich, and M. Sagiv. Decidable fragments of many-sorted logic. *J. Symb. Comput.*, 45(2):153–172, 2010.
2. L. Bachmair, H. Ganzinger, and A. Voronkov. Elimination of equality via transformation with ordering constraints. In C. Kirchner and H. Kirchner, editors, *CADE*, volume 1421 of *LNCS*, pages 175–190. Springer, 1998.
3. P. Baumgartner, A. Fuchs, H. de Nivelle, and C. Tinelli. Computing finite models by reduction to function-free clause logic. *J. Applied Logic*, 7(1):58–74, 2009.
4. P. Baumgartner, A. Fuchs, and C. Tinelli. Implementing the model evolution calculus. *International Journal on Artificial Intelligence Tools*, 15(1):21–52, 2006.

5. P. Baumgartner and R. Schmidt. Blocking and other enhancements for bottom-up model generation methods. In *Third International Joint Conference on Automated Reasoning (IJCAR'06)*, volume 4130 of *LNCS*, pages 125–139. Springer, 2006.
6. P. Baumgartner and C. Tinelli. The Model Evolution calculus. In *Proc. CADE-19*, number 2741 in LNCS, pages 350–364. Springer, 2003.
7. D. Brand. Proving theorems with the modification method. *SIAM J. Comput.*, 4(4):412–430, 1975.
8. K. Claessen. The anatomy of Equinox - an extensible automated reasoning tool for first-order logic and beyond - (talk abstract). In N. Bjørner and V. Sofronie-Stokkermans, editors, *23rd International Conference on Automated Deduction, CADE-23*, volume 6803 of *LNCS*, pages 1–3. Springer, 2011.
9. K. Claessen, A. Lilliestr̈om, and N. Smallbone. Sort it out with monotonicity - translating between many-sorted and unsorted first-order logic. In N. Bjørner and V. Sofronie-Stokkermans, editors, *23rd International Conference on Automated Deduction, CADE-23*, volume 6803 of *LNCS*, pages 207–221. Springer, 2011.
10. K. Claessen and N. Sörensson. New techniques that improve MACE-style model finding. In P Baumgartner and C. Fermüller, editors, *CADE- 19 Workshop: Model Computation Principles, Algorithms, Applications*, pages 11–27, 2003.
11. S. Conchon, J.-C. Filliâtre, and J. Signoles. ocamlgraph. available at http://ocamlgraph.lri.fr.
12. P. Fontaine. *Techniques for verification of concurrent systems with invariants*. PhD thesis, Institut Montefiore, Université de Liège, Belgium, 2004.
13. P. Fontaine. Combinations of theories and the Bernays-Schönfinkel-Ramsey class. In *VERIFY'07*, CEUR Workshop Proceedings. CEUR-WS.org, 2007.
14. P. Fontaine and E. P. Gribomont. Decidability of invariant validation for paramaterized systems. In *the 9th International Conference on Tools and Algorithms for the Construction and Analysis of Systems(TACAS'03)*, volume 2619 of *LNCS*, pages 97–112. Springer, 2003.
15. H. Ganzinger and K. Korovin. New directions in instantiation-based theorem proving. In *Proc. 18th IEEE Symposium on LICS*, pages 55–64. IEEE, 2003.
16. H. Ganzinger and K. Korovin. Integrating equational reasoning into instantiation-based theorem proving. In *CSL'04*, volume 3210 of *LNCS*, pages 71–84, 2004.
17. Y. Ge and L. M. de Moura. Complete instantiation for quantified formulas in satisfiability modulo theories. In *the 21st International Conference on Computer Aided Verification (CAV'09)*, volume 5643 of *LNCS*, pages 306–320. Springer, 2009.
18. K. Hoder, Z. Khasidashvili, K. Korovin, and A. Voronkov. Preprocessing techniques for first-order clausification. In G. Cabodi and S. Singh, editors, *Formal Methods in Computer-Aided Design (FMCAD'12)*, pages 44–51. IEEE, 2012.
19. K. Korovin. iProver - an instantiation-based theorem prover for first-order logic (system description). In *the 4th International Joint Conference on Automated Reasoning*, volume 5195 of *LNCS*, pages 292–298. Springer, 2008.
20. K. Korovin. Inst-Gen - a modular approach to instantiation-based automated reasoning. In A. Voronkov and C. Weidenbach, editors, *Programming Logics, Essays in Memory of Harald Ganzinger*, volume 7797 of *LNCS*, pages 239–270. Springer, 2013.
21. K. Korovin and C. Sticksel. iProver-Eq: an instantiation-based theorem prover with equality. In J. Giesl and R. Hähnle, editors, *5th International Joint Conference on Automated Reasoning, IJCAR'10*, volume 6173 of *LNCS*, pages 196–202. Springer, 2010.
22. K. Korovin and C. Sticksel. Labelled unit superposition calculi for instantiation-based reasoning. In C. G. Fermüller and A. Voronkov, editors, *LPAR-17*, volume 6397 of *LNCS*, pages 459–473, 2010.
23. R. Piskac, L. de Moura, and N. Bjørner. Deciding effectively propositional logic using DPLL and substitution sets. *J. Autom. Reasoning*, 44(4):401–424, 2010.

24. A. Reynolds, C. C. Tinelli, A. Goel, and S. Krstiċ. Finite model finding in smt. In *CAV 2013*, 2013. to appear.
25. A. Reynolds, C. C. Tinelli, A. Goel, S. Krstiċ, M. Deters, and Barrett C. Quantifier instantiation techniques for finite model finding in SMT. In *CADE-24*, 2013. to appear.
26. A. Riazanov and A. Voronkov. The design and implementation of VAMPIRE. *AI Communications*, 15(2-3):91–110, 2002.
27. S. Schulz. System description: E 0.81. In *IJCAR*, volume 3097 of *LNCS*, pages 223–228. Springer, 2004.
28. G. Sutcliffe. The TPTP Problem Library and Associated Infrastructure: The FOF and CNF Parts, v3.5.0. *Journal of Automated Reasoning*, 43(4):337–362, 2009.
29. R. E. Tarjan. Depth-first search and linear graph algorithms. *SIAM J. Comput.*, 1(2):146–160, 1972.