Computing Finite Models by Reduction to Function-Free Clause Logic

Peter Baumgartner[1], Alexander Fuchs[2], Hans de Nivelle[3], and Cesare Tinelli[2]

[1] National ICT Australia (NICTA), Peter.Baumgartner@nicta.com.au
[2] The University of Iowa, USA, {fuchs,tinelli}@cs.uiowa.edu
[3] Max-Planck-Institut für Informatik, Germany, nivelle@mpi-inf.mpg.de

Abstract. Recent years have seen considerable interest in procedures for computing finite models of first-order logic specifications. One of the major paradigms, MACE-style model building, is based on reducing model search to a sequence of propositional satisfiability problems and applying (efficient) SAT solvers to them. A problem with this method is that it does not scale well, as the propositional formulas to be considered may become very large.

We propose instead to reduce model search to a sequence of satisfiability problems made of function-free first-order clause sets, and to apply (efficient) theorem provers capable of deciding such problems. The main appeal of this method is that first-order clause sets grow more slowly than their propositional counterparts, thus allowing for more space efficient reasoning.

In the paper we describe the method in detail and show how it is integrated into one such prover, Darwin, our implementation of the Model Evolution calculus. The results are general, however, as our approach can be used in principle with any system that decides the satisfiability of function-free first-order clause sets.

To demonstrate its practical feasibility, we tested our approach on all satisfiable problems from the TPTP library. Our methods can solve a significant subset of these problems, which overlaps but is not included in the subset of problems solvable by state-of-the-art finite model builders such as Paradox and Mace4.

1 Introduction

Methods for model computation can be classified as those that directly search for a finite model, like the extended PUHR tableau method [BT98], the methods in [Bez05,dNM06] and the methods in the SEM-family [Sla92,Zha95,McC03], and those based on transformations into certain fragments of logic and relying on corresponding readily available systems (see [BS06] for a recent approach).

The latter approach includes the family of MACE-style model builders [McC03]. These systems search for finite models, essentially, by searching the space of interpretations with domain sizes $1, 2, \ldots$, in increasing order, until a model is found. The MACE-style model builder with the best performance today is perhaps the Paradox system [CS03]. We present in this paper a new approach in the MACE/Paradox tradition which however capitalizes on new advances in instantiation-based first-order theorem proving, as opposed to advances in propositional satisfiability as in the case of MACE and Paradox. The general idea in our approach is the same: to find a model with n elements for a given a clause set possibly with equality, the clause set is first converted into a simpler form by means of the following transformations.

1. Each clause is flattened.
2. Each n-ary function symbol is replaced by an $n+1$-ary predicate symbol and equality is eliminated.
3. Clauses are added to the clause set that impose totality constraints on the new predicate symbols, but over a domain of cardinality n.

The details of our transformation differ in various aspects from the MACE/Paradox approach. In particular, we add no functionality constraints over the new predicate symbols. The main difference, however, is that we evenually reduce the original problem to a satisfiability problem over function-free clause logic (without equality), not over propositional logic. As a consequence of the different target logic, we do not use a SAT solver to look for models. Instead, we use a variant of *Darwin* [BFT06a], our implementation of the Model Evolution calculus [BT03], which can decide satisfiability in that logic.

While we do take advantage of some of the distinguishing features of *Darwin* and the Model Evolution calculus, especially in the way models are constructed, our method is general enough that it could use without much additional effort any other decision procedure for function-free clause logic, for instance, any implementation of one of the several instance-based methods for first-order reasoning that are currently enjoying a growing popularity.

In this paper we illustrate our method in some detail, presenting the main translation and its implementation within *Darwin*, and discussing our initial experimental results in comparison with Paradox itself and with Mace4 [McC03], a competitive, non-MACE-like (despite the name) model builder. The results indicate that our method is rather promising as it can solve 1074 of the 1251 satisfiable problems in the TPTP library [SS98]. These problems are neither a subset nor a superset of the sets of 1083 and 802 problems respectively solved (under the same experimental settings) by Paradox and Mace4.

2 Preliminaries

We use standard terminology from automated reasoning. We assume as given a signature $\Sigma = \Sigma_{\mathrm{F}} \cup \Sigma_{\mathrm{P}}$ of function symbols Σ_{F} (including constants) and predicate symbols Σ_{P}. As we are working (also) with equality, we assume Σ_{P} contains a distinguished binary predicate symbol $\approx$, used in infix form, with $\not\approx$ denoting its negation. Terms, atoms, literals and formulas over Σ and a given (denumerable) set of variables V are defined as usual. A clause is a (finite) implicitly universally quantified disjunction of literals. A *clause set* is a finite set of clauses. We use the letter C to denote clauses and the letter L to denote literals.

For a given atom $P(t_1,\ldots,t_n)$ (possibly an equation) the terms $t_1,\ldots,t_n$ are also called the *top-level* terms (of $P(t_1,\ldots,t_n)$).

With regards to semantics, we use the notions of (first-order) *satisfiability* and E-*satisfiability* in a completely standard way. If $\mathfrak{I}$ is an (E-)interpretation then $|\mathfrak{I}|$ denotes the domain (or universe) of $\mathfrak{I}$. Recall that in E-interpretations the equality relation is interpreted as the *identity relation*, i.e. for every E-interpretation $\mathfrak{I}$ it holds $\approx^{\mathfrak{I}} = \{(d,d) \mid d \in |\mathfrak{I}|\}$. We are primarily interested in computing *finite* models, which are models (of the given clause set) with a finite domain.

In the remainder of the paper, we assume that M is a given (finite) clause set over signature $\Sigma = \Sigma_{\mathrm{F}} \cup \Sigma_{\mathrm{P}}$, where Σ_{F} (resp. Σ_{P}) are the function symbols (resp. predicate symbols) occuring in M.

3 Finite Model Transformation

In this section we give a general description of the tranformations we apply to the input problem to reduce it to an equisatisfiable problem in function-free clause logic without equality. We do that by defining various transformation rules on clauses.

In the rules, we write $L \vee C \rightsquigarrow C' \vee C$ to indicate that the clause $C' \vee C$ is obtained from the clause $L \vee C$ by (single) application of one of these rules.

3.1 Basic Transformation

(1) Abstraction of positive equations.

$$\begin{array}{lcrl}
s \approx y \vee C & \rightsquigarrow & s \not\approx x \vee x \approx y \vee C & \text{if } s \text{ is not a variable and} \\
 & & & x \text{ is a fresh variable} \\
x \approx t \vee C & \rightsquigarrow & t \not\approx y \vee x \approx y \vee C & \text{if } t \text{ is not a variable and} \\
 & & & y \text{ is a fresh variable} \\
s \approx t \vee C & \rightsquigarrow & s \not\approx x \vee t \not\approx y \vee x \approx y \vee C & \text{if } s \text{ and } t \text{ are not variables and} \\
 & & & x \text{ and } y \text{ are fresh variables}
\end{array}$$

These rules make sure that all (positive) equations are between variables.

(2) Flattening of non-equations.

$$\begin{array}{lcll}
(\neg)P(\ldots,s,\ldots) \vee C & \rightsquigarrow & (\neg)P(\ldots,x,\ldots) \vee s \not\approx x \vee C & \text{if } P \neq \approx, s \text{ is not a variable, and} \\
 & & & x \text{ is a fresh variable}
\end{array}$$

(3) Flattening of negative equations.

$$\begin{array}{lcll}
f(\ldots,s,\ldots) \not\approx t \vee C & \rightsquigarrow & f(\ldots,x,\ldots) \not\approx t \vee s \not\approx x \vee C & \text{if } s \text{ is not a variable} \\
 & & & \text{and } x \text{ is a fresh variable}
\end{array}$$

(4) Separation of negative equations.

$$\begin{array}{lcll}
s \not\approx t \vee C & \rightsquigarrow & s \not\approx x \vee t \not\approx x \vee C & \text{if neither } s \text{ nor } t \text{ is a variable,} \\
 & & & \text{and } x \text{ and } y \text{ are fresh variables}
\end{array}$$

This rule makes sure that at least one side of a (negative) equation is a variable. Notice that this property is also satisfied by the transformations (2) and (3).

(5) Removal of trivial negative equations.

$$x \not\approx y \vee C \;\leadsto\; C\sigma \quad \text{where } \sigma = \{x \mapsto y\}$$

(6) Orientation of negative equations.

$$x \not\approx t \vee C \;\leadsto\; t \not\approx x \vee C \quad \text{if } t \text{ is not a variable}$$

For a clause C, let the *basic transformation of* C, denoted as $\mathcal{B}(C)$, be the clause obtained from C by applying the transformations (1)-(6), in this order, each as long as possible.[4] We extend this notation to clause sets in the obvious way, i.e., $\mathcal{B}(M)$ is the clause set consisting of the basic transformation of all clauses in M.

The two flattening transformations alone, when applied exhaustively, turn a clause into a *flat* one, where a clause is *flat* if:

1. each top-level term of each of its negative equations is a variable or has the form $f(x_1,\ldots,x_n)$, where f is a function symbol, $n \geq 0$, and $x_1,\ldots,x_n$ are variables;
2. each top-level term of each of its non-equations is a variable.

Similar flattening transformations have been considered before as a means to deal more efficiently with equality within calculi for first-order logic without equality [Bra75,BGV98].

The basic transformation above is correct in the following sense.

Lemma 1 (Correctness of $\mathcal{B}$). *The clause set M is E-satisfiable if and only if $\mathcal{B}(M)$ is E-satisfiable.*

Proof. That flattening preserves E-satisfiability (both ways) is well-know (cf. [Bra75]). Regarding transformations (1), (4), (5) and (6), the proof is straightforward or trivial. □

3.2 Conversion to Relational Form

It is not hard to see that, for any clause C, the following holds for the clause set $\mathcal{B}(C)$:

1. each of its positive equations is between two variables,
2. each of its negative equations is flat and of the form $f(x_1,\ldots,x_n) \not\approx y$, and
3. each of its non-equations is flat.

After the basic transformation, we apply the following one, turning each n-ary function symbol f into a (new) $n+1$-ary predicate symbol R_f.

(7) Elimination of function symbols.

$$f(x_1,\ldots,x_n) \not\approx y \vee C \;\leadsto\; \neg R_f(x_1,\ldots,x_n,y) \vee C$$

[4] It is easy to see that this process always terminates.

Let $\mathcal{B}_R(M)$ be the clause set obtained from an exhaustive application of this transformation to $\mathcal{B}(M)$.

Recall that an $n+1$-ary relation R over a set A is *left-total* if for every $a_1,\ldots,a_n \in A$ there is an $b \in A$ such that $(a_1,\ldots,a_n,b) \in R$. The relation R is *right-unique* if whenever $(a_1,\ldots,a_n,b) \in R$ there is no other tuple of the form $(a_1,\ldots,a_n,b')$ in R.

Because of the above properties (1)–(3) of $\mathcal{B}(M)$, the transformation $\mathcal{B}_R(M)$ is well-defined, and will produce a clause set with no function symbols. This transformation however is not unsatisfiability preserving unless one considers only left-total interpretations for the predicate symbols R_f. More formally:

Lemma 2 (Correctness of $\mathcal{B}_R$). *The clause set M is E-satisfiable if and only if there is an E-model $\mathfrak{I}$ of $\mathcal{B}_R(M)$ such that $(R_f)^{\mathfrak{I}}$ is left-total, for every function symbol $f \in \Sigma_{\mathrm{F}}$.*

Proof. The direction from left to right is easy. For the other direction, let $\mathfrak{I}$ be an E-model of $\mathcal{B}_R(M)$ such that $(R_f)^{\mathfrak{I}}$ is left-total for every function symbol $f \in \Sigma_{\mathrm{F}}$.

Recall that functions are nothing but left-total and right-unique relations. We will show how to obtain from $\mathfrak{I}$ an E-model $\mathfrak{I}'$ of $\mathcal{B}_R(M)$, that preserves left-totality and adds right-uniqueness, i.e., such that $(R_f)^{\mathfrak{I}'}$ is both left-total and right-unique for all $f \in \Sigma_{\mathrm{F}}$. Since such an interpretation is clearly a model of $\mathcal{B}(M)$, it will the follow immediately by Lemma 1 that M is E-satisfiable.

We obtain $\mathfrak{I}'$ as the interpretation that is like $\mathfrak{I}$, except that $(R_f)^{\mathfrak{I}'}$ contains exactly one element $(d_1,\ldots,d_n,d)$, for every $d_1,\ldots,d_n \in |\mathfrak{I}|$, chosen arbitrarily from $(R_f)^{\mathfrak{I}}$ (this choice exists because $(R_f)^{\mathfrak{I}}$ is left-total). It is clear from the construction that $(R_f)^{\mathfrak{I}'}$ is right-unique and left-total. Trivially, $\mathfrak{I}'$ interpretes $\approx$ as the identity relation, because $\mathfrak{I}$ does, as $\mathfrak{I}$ is an E-interpretation. Thus, $\mathfrak{I}'$ is an E-interpretation, too.

It remains to prove that with $\mathfrak{I}$ being a model of $\mathcal{B}_R(M)$ then so is $\mathfrak{I}'$. This follows from the fact that every occurrence of a predicate symbol R_f, with $f \in \Sigma_{\mathrm{F}}$, in the clause set $\mathcal{B}_R(M)$ is in a negative literal. But then, since $(R_f)^{\mathfrak{I}'} \subseteq (R_f)^{\mathfrak{I}}$ by construction, it follows immediately that any clause of $\mathcal{B}_R(M)$ satisfied by $\mathfrak{I}$ is also satisfied by $\mathfrak{I}'$. □

The significance of this lemma is that it requires us to interpret the predicate symbols R_f as left-total relations, *but not necessarily as right-unique ones*. Consequently, right-uniqueness will not be axiomatized below.

3.3 Addition of Finite Domain Constraints

To force left-totality, one could add the Skolemized version of axioms of the form

$$\forall x_1,\ldots,x_n \exists y\, R_f(x_1,\ldots,x_n,y)$$

to $\mathcal{B}_r(M)$. The resulting set would be E-satisfiable exactly when M is E-satisfiable.[5] However, since we are interested in finite satisfiability, we use finite approximations of these axioms. To this end, let d be a positive integer, the *domain size*. We consider

[5] Altogether, this proves the (well-known) result that function symbols are “syntactic sugar”. They can always be eliminated in an equisatisfiability preserving way, at the cost of introducing existential quantifiers.

$$\begin{array}{l} R_{c_1}(1) \vee \cdots \vee R_{c_1}(d) \\ R_{c_2}(1) \vee \cdots \vee R_{c_2}(d) \\ \vdots \\ R_{c_m}(1) \vee \cdots \vee R_{c_m}(d) \end{array}$$

(a)

$$\begin{array}{l} R_{c_1}(1) \\ R_{c_2}(1) \vee R_{c_2}(2) \\ \vdots \\ R_{c_d}(1) \vee \cdots \vee R_{c_d}(d) \\ R_{c_{d+1}}(1) \vee \cdots \vee R_{c_{d+1}}(d) \\ \vdots \\ R_{c_m}(1) \vee \cdots \vee R_{c_m}(d) \end{array}$$

(b)

Fig. 1. Totality axioms for constants and their triangular form

the expansion of the signature of $\mathcal{B}_r(M)$ by d fresh constant symbols, which we name $1,\ldots,d$. Intuitively, instead of the totality axiom above we can now use the axiom

$$\forall x_1,\ldots,x_n \exists y \in \{1,\ldots,d\}\ R_f(x_1,\ldots,x_n,y) \ .$$

Concretely, if f is an n-ary function symbol let the clause

$$R_f(x_1,\ldots,x_n,1) \vee \cdots \vee R_f(x_1,\ldots,x_n,d)$$

be the *d-totality axiom for f*, and let $\mathcal{D}(d)$ be the set of all d-totality axioms for all function symbols $f \in \Sigma_{\mathrm{F}}$. The set $\mathcal{D}(d)$ axiomatizes the left-totality of $(R_f)^{\mathfrak{I}}$, for every function symbol $f \in \Sigma_{\mathrm{F}}$ and interpretation $\mathfrak{I}$ with $|\mathfrak{I}| = \{1,\ldots,d\}$.

3.4 Symmetry Breaking

Symmetries have been identified as a major source for inefficiencies in constrain solving systems. *Value symmetry* applies to a problem when a permutation of the values (or better, value vector) assigned to variables constitutes a solution to the problem, too. A dual symmetry property may apply to the (decision) variables of a problem, giving rise to *variable symmetry*. Breaking such symmetries has been recognized as a source for considerable efficiency gains.

It is easy to break some value symmetries introduced by assigning domain values to constants. Suppose Σ_{F} contains m constants $c_1,\ldots,c_m$. Recall that $\mathcal{D}(d)$ contains, in particular, the axioms shown in Figure 1(a). Similarly to what is done with Paradox, these axioms can be replaced by the more “triangular” form shown in Figure 1(b). This form reflects symmetry breaking of assigning values for the first d constants. In fact, one could further strengthen the symmetry breaking axioms by adding (unit) clauses like $\neg R_{c_1}(2),\ldots\neg R_{c_1}(d)$. We do not add them, as they do not constrain the search for a model further (they are all pure).

In the sequel we will refer to the clause set as described here as $\mathcal{D}(d)$.

3.5 Putting all Together

Since we want to use clause logic *without* equality as the target logic of our overall transformation, the only remaining step is the explicit axiomatization of the equality symbol $\approx$ over domains of size d—so that we can exploit Lemma 2 in the (interesting) right-to-left direction. This is easily achieved with the clause set

$$\mathcal{E}(d) = \{i \not\approx j \mid 1 \leq i, j \leq d \text{ and } i \neq j\} \ .$$

Finally then, we define the *finite-domain transformation of M* as the clause set

$$\mathcal{F}(M,d) := \mathcal{B}_R(M) \cup \mathcal{D}(d) \cup \mathcal{E}(d) \ .$$

Putting all together we arrive at the following first main result:

Theorem 3 (Correctness of the Finite-Domain Translation). *Let d be a positive integer. Then, M is E-satisfiable by some finite interpretation with domain size d if and only if $\mathcal{F}(M,d)$ is satisfiable.*

Proof. Follows from Lemma 2 and the comments above on $\mathcal{D}(d)$ and $\mathcal{E}(d)$, together with the observation that if $\mathcal{F}(M,d)$ is satisfiable it is satisfiable in a Herbrand interpretation with universe $\{1,\ldots,d\}$.

More precisely, for the only-if direction assume as given a Herbrand model $\mathfrak{I}$ of $\mathcal{F}(M,d)$ with universe $\{1,\ldots,d\}$. It is clear from the axioms $\mathcal{E}(d)$ that $\mathfrak{I}$ assigns false to the equation $(d' \approx d'')$, for any two different elements $d', d'' \in \{1,\ldots,d\}$. Now, the model $\mathfrak{I}$ can be modified to assign true to all equations $d' \approx d'$, for all $d' \in \{1,\ldots,d\}$ and the resulting E-interpretation will still be a model for $\mathcal{F}(M,d)$. This is, because the only occurences of negative equations in $\mathcal{F}(M,d)$ are those contributed by $\mathcal{E}(d)$, which are still satisfied after the change.[6] It is this modified model that can be turned into an E-model of M. □

This theorem suggests immediately a (practical) procedure to search for finite models, by testing $\mathcal{F}(M,d)$ for satisfiability, with $d = 1,2,\ldots$, and stopping as soon as the first satisfiable set has been found. Moreover, any reasonable such procedure will return in the satisfiable case a Herbrand representation (of some finite model).

Indeed, the idea of searching for a finite model by testing satisfiability over finite domains of size $1,2,\ldots$ is implemented in our approach and many others (Paradox [CS03], Finder [Sla92], Mace [McC94], Mace4 [McC03] , SEM [Zha95] to name a few).

4 Implementation

We implemented the transformation described so far within our theorem prover *Darwin*. In addition to being a full-blown theorem prover for first-order logic without equality, *Darwin* is a decision procedure for the satisfiability of function-free clause sets, and thus is a suitable back-end for our transformation. We call the combined system FM-*Darwin* (for Finite Models *Darwin*).

[6] Notice, in particular, that $\mathcal{B}_R(M)$ contains only positive occurences of equations, if any.

Conceptually, FM-*Darwin* builds on *Darwin* by adding to it as a front-end an implementation of the transformation $\mathcal{F}$ (Section 3.5), and invoking Darwin on $\mathcal{F}(M,d)$, for $d = 1,2,...$, until a model is found. In reality, FM-*Darwin* is built *within Darwin* and differs from the conceptual procedure described so far in the following ways:

1. The search for models of increasing size is built in *Darwin*'s own restarting mechanism. For refutational completeness *Darwin* explores its search space in an iterative-deepening fashion with respect of certain *depth* measures. The same mechanism is used in FM-*Darwin* to restart the search with an increased domain size $d+1$ if the input problem has no models of size d.

2. FM-*Darwin* implements some obvious optimizations over the transformation rules described in Section 3. For instance, the tranformations (1)–(4) are done in parallel, depending on the structure of the current literal. Transformation (6) is done implicitly as part of tranformation (7), when turning equations into relations. Also, when flattening a clause, the same variable is used to abstract different occurrences of a subterm.

3. Because the clause sets $\mathcal{F}(M,d)$ and $\mathcal{F}(M,d+1)$, for any d, differ only in the their subsets $\mathcal{D}(d) \cup \mathcal{E}(d)$ and $\mathcal{D}(d+1) \cup \mathcal{E}(d+1)$, respectively, there is no need to regenerate the constant part, and this is not done.

4. Similarly to SAT solvers based on the DPLL procedure, *Darwin* has the ability to learn new (entailed) clauses—or *lemmas*—in failed branches of a derivation, which is helpful to prune search space in later branches [BFT06b]. Some of the learned lemmas are independent from the current domain size and so can be carried over to later iterations with larger domain sizes. To do that, each clause in $\mathcal{D}(d+1)$ is actually *guarded* by an additional literal M_d standing for the current domain size. In FM-*Darwin*, lemmas depending on the current domain size d, and only those, retain the guard M_d when they are built, making it easy to eliminate them when moving to the next size $d+1$.

5. Recall from step (7) in the transformation (Section 3.2) that every function symbol is turned into a predicate symbol. In our actual implementation, we go one step further and use a meta modeling approach that can make the final clause set produced by our translation more compact, and possibly speed up the search as well, thanks to the way models are built in the Model Evolution calculus. The idea is the following.

For every $n > 0$, instead of generating an $n+1$-ary relation symbol R_f for each n-ary function symbol $f \in \Sigma_\mathrm{F}$ we use an $n+2$-ary relation symbol R_n, for all n-ary function symbols. Then, instead of translating a literal of the form $f(x_1,...,x_n) \not\approx y$ into the literal $\neg R_f(x_1,...,x_n,y)$, we translate it into the literal $R_n(f,x_1,...,x_n,y)$, treating f as a zero-arity symbol. The advantage of this translation is that instead of needing one totality axiom per relation symbol R_f with $f \in \Sigma_\mathrm{F}$ we only need one per function symbol *arity* (among those found in Σ_F). [7] The d-totality axioms then take the more general form

$$R_n(y,x_1,\ldots,x_n,1) \vee \cdots \vee R_n(y,x_1,\ldots,x_n,d)$$

[7] Consequently, this translation is actually applied for a given arity only if there are at least two symbols of that arity.

where the variable y is meant to be quantified over the (original) function symbols in Σ_F. Note that the zero-arity symbols representing the original function symbols in the input are in addition to the domain constants, and of course never interact with them.[8]

6. Like Paradox, FM-*Darwin* performs a kind of sort inference in order to improve the effectiveness of symmetry breaking. Each function and predicate symbol of arity n in Σ is assigned a type respectively of the form $S_1 \times \ldots \times S_n \rightarrow S_{n+1}$ and $S_1 \times \ldots \times S_n$, where all sorts S_i are initially distinct. Each term in the input clause set is assigned the result sort of its top symbol. Two sorts S_i and S_j are then identified based on the input clause set by applying a union-find algorithm with the following rules. First, all sorts of different occurrences of the same variable in a clause are identified; second, the result sorts of two terms s and t in an equality $s \approx t$ are identified; third, for each term or atom of the form $f(\ldots,t,\ldots)$ the argument sort of f at t's position is identified with the sort of t.

All sorts left at the end are assumed to have the same size. This way, when a sorted model is found (with all sorts having some size d), it can be translated into an unsorted model by an isomorphic translation of each sort into a single domain of size d. This implies that one can conceptually search for a sorted model and apply the symmetry breaking rules independently for each sort, and otherwise do everything else as described in the previous section. In addition to generally improving performance, this makes the whole procedure less fragile, as the order in which the constants are chosen for the symmetry breaking rules can have a dramatic impact on the search space.

7. Splitting clauses. Paradox and Mace2 use transformations that, by introducing new predicate symbols, can split a flat clause with many variables into several flat clauses *with fewer variables*. For instance, a clause of the form

$$P(x,y) \vee Q(y,z)$$

whose two subclauses share only the variable y can be transformed into the two clauses

$$P(x,y) \vee S(y) \qquad \neg S(y) \vee Q(y,z)$$

where the predicate symbol in the *connecting* literal $S(y)$ is fresh. This sort of transformation preserves (un-)satisfiability. Thus, in this example, where the number of variables in a clause is reduced by from 3 to 2, procedures based on a full ground instantiation of the input clause set may benefit from of having to deal with the $O(2n^2)$ ground instances of the new clauses instead of $O(n^3)$ ground instances of the original clause.[9]

As it happens, reducing of the number of variables per clause is not necessary helpful in our case. Since (FM-)*Darwin* does not perform an exhaustive ground instantiation of its input clause set, splitting clauses can actually be counter-productive because it forces the system to populate contexts with instances of connecting literals like $S(y)$

[8] They are intuitively of a different sort S. Moreover, by the Herbrand theorem, we can consider with no loss of generality only interpretations that populate the sort S precisely with these constants, and no more.

[9] A similar observation was made in [**?**] and exploited beneficially to solve planning problems by reduction to SAT.

above. Our experiments indicate that this is generally expensive unless the connecting literals do not contain any variables. Still, in contrast to Darwin, where in general clause splitting is only an improvement for ground connecting literals, for FM-*Darwin* splitting in all cases gives a slight improvement.[10]

8. Naming subterms. Clauses with deep terms lead to long flat clauses. To avoid that, deep subterms can be extracted and named by an equation. For instance, the clause set

$$P(h(g(f(x)),y)) \qquad Q(f(g(z)))$$

can be replaced by the clause set

$$P(h_2(x,y)) \qquad Q(h_1(x)) \qquad h_2(x,y) = h(h_1(x),y) \qquad h_1(x) = g(f(x))$$

where h_1 and h_2 are fresh function symbols. When carried out repeatedly, reusing definitions across the whole clause set, this transformation yields to shorter flattened clauses.

We tried some heuristics for when to apply the transformation, based on how often a term occurs in the clause set, and how big the flattened definition is (i.e., how much it is possible to save by using the definition). The only consistent improvement on TPTP problems was achieved when introducing definitions only for ground terms. This solves 16 more problems, 14 of which are Horn. Thus, currently only ground terms are flattened by default with this transformation in FM-*Darwin*.

5 Experimental Evaluation

5.1 Space Efficiency

Our reduction to clause sets encoding finite E-satisfiability is similar to, and indeed inspired by, the one in Paradox [CS03]. The most significant difference is, as we mentioned, that in Paradox the whole counterpart of our clause set $\mathcal{F}(M,d)$ is grounded out, simplified and fed into a SAT solver (Minisat). In our case, $\mathcal{F}(M,d)$ is fed directly to a theorem prover capable of deciding the satisfiability of function-free clause sets. This has the advantage of often being more space-efficient: in Paradox, as the domain size d is increased, the number of ground instances of a clause grows exponentially in the number of variables in the clause [CS03]. In contrast, in our transformation no ground instances of the clause set $\mathcal{F}$ are produced. The subsets $\mathcal{D}$ and $\mathcal{E}$ do grow with the domain size d; however, the number of clauses in $\mathcal{D}(d)$ remains constant in d while their length grows only linearly in d. The number of clauses in $\mathcal{E}(d)$, which are all unit, grows instead quadratically.

As far as preprocessing the input clause set is concerned then, our approach already has a significant space advantage over Paradox's. This is crucial for problems that have models of a relatively large size (more than 6 elements, say, for functions arities of 10), where Paradox's eager conversion to a propositional problem is simply unfeasible because of the huge size of the resulting formula. A more accurate comparison, however,

[10] In our experiments on the TPTP (Section 5.2) it helped to solve eight additional satisfiable problems.

	FM-*Darwin*			**Mace4**	**Paradox**		
n	**|Max. ctxt|**	**Mem**	**Time**	**Time**	**# Vars**	**# Clauses**	**Time**
3	14	1	< 1	< 1	14	0	< 1
4	24	1	< 1	< 1	301	123	< 1
5	37	1	< 1	< 1	3192	534	< 1
6	53	1	< 1	< 1	46749	7919	< 1
7	72	1	< 1	178	823666	46749	12
8	94	1	5.1	Fail at size 7	Inconclusive, size ≥ 7		36
9	119	1	50	Fail at size 6	Inconclusive, size ≥ 5		9.6
10	147	1	566	Fail at size 4	Inconclusive, size ≥ 4		3.6

Table 1. Comparison of Darwin and Paradox on Example 4, for $n = 3, \ldots, 9$. All **Time** results are CPU time in seconds. Specific column entries for **FM-*Darwin*** : $|$**Max. ctxt**$|$ – maximum context size needed in derivation; **Mem** – required memory size in megabytes. Column entry for **Mace4**: "Fail at size d" – Memory limit of 400 MB exhausted during search for a model with size d. Specific column entries for **Paradox**: **# Vars** – the number of propositional variables of the translation into propositional logic for domain size n; **# Clauses** – likewise, the number of propositional clauses; "Inconclusive, size $\geq d$": Paradox gave up after the time stated.

needs to take the dynamics of model search into account. By using *Darwin* as the backend for our transformation, we are able to keep space consumption down also during search. Being a DPLL-like system, *Darwin* never derives new clauses.[11] The only thing that grows unbounded in size in *Darwin* is the *context*, the data structure representing the current candidate model for the problem. With function-free clause sets the size of the context depends on the number of possible ground instances of input *literals*, a much smaller number than the number of possible ground instances of input *clauses*. In addition, our experiments show that the context basically never grows to its worst-case size.

The different asymptotic behaviours between FM-*Darwin* and Paradox can be verified experimentally with the following simple problem.

Example 4 (Too big to ground). Let p be an n-ary predicate symbol, $c_1, \ldots, c_n$ (distinct) constants, and $x, x_1, \ldots, x_n$ (distinct) variables. Then consider the clause set consisting of the following $n \cdot (n-1)/2 + 1$ unit clauses, for $n \geq 0$:

$$p(c_1, \ldots, c_n)$$

$$\neg p(x_1, \ldots, x_{i-1}, x, x_{i+1}, \ldots, x_{j-1}, x, x_{j+1}, \ldots, x_n) \qquad \text{for all } 1 \leq i < j \leq n$$

The first clause just introduces n constants. Any (domain-minimal) model has to map them to at most n domain elements. The remaining clauses force the constants to be mapped to pairwise distinct domain elements. Thus, the smallest model has exactly n elements. This clause set is perhaps the simplest clause set to specify a domain with n elements. □

We ran the example for $n = 3, \ldots, 10$ on FM-*Darwin*, Mace4 and Paradox and obtained the results in Table 1. These results confirm our expectations on FM-*Darwin*'s

[11] Except for lemmas of which, however, it keeps only a fixed number during a derivation.

Problem Type		**Problems**	**FM-*Darwin***		**Mace4**		**Paradox 1.3**	
Horn	**Equality**		**Solved**	**Time**	**Solved**	**Time**	**Solved**	**Time**
no	no	607	575	3.9	394	3.0	578	0.9
no	yes	383	312	4.3	190	7.8	264	0.4
yes	no	65	51	17.5	37	0.2	59	2.1
yes	yes	196	136	7.0	181	3.6	182	5.3
all		1251	1074	5.1	802	4.1	1083	1.6

Table 2. Comparison of FM-*Darwin*, Mace4, and Paradox 1.3 over all satisfiable TPTP problems, also grouped based on being Horn and/or containing equality. **Solved Problems** gives the number of problems solved by a configuration, **Time** the average time used to solve these problems.

greater scalability with respect to space consumption. The growth of the (propositional) variables and clauses within Paradox clearly shows exponential behaviour. In contrast, *Darwin*'s contexts grow much more slowly.

5.2 Comparative Evaluation on TPTP

We evaluated the effectiveness of our approach on all the satisfiable problems of the TPTP 3.1.1 in comparison to Paradox 1.3 and Mace4.[12] All tests were run on Xeon 2.4Ghz machines with 1GB of RAM, with the imposed limits of 300s of CPU time and 512MB of RAM. FM-*Darwin* was run with the *grounded* learning option and with an upper limit of 500 lemmas (see [BFT06b] for more details on these options), Paradox and Mace4 in the default configuration.

The results given in Figure 2 show that in terms of solved problems FM-*Darwin* significantly outperforms Mace4. Overall, our system is almost as good as Paradox, outperforming it over the non-Horn problems in the set. More precisely, FM-*Darwin* solves 328 problems that Mace4 cannot solve—Mace4 runs out of time for 169 problems and out of memory for the remaining ones—and solves 82 problems that Paradox can not solve—on all these problems Paradox runs out of memory or gives up. We sampled some of these problems and re-ran Paradox without memory and time limits, but to no avail. For problem `NLP049-1`, for instance, about 10 million (ground) clauses were generated for a domain size of 8, consuming about 1 GB of memory, and the underlying SAT solver could not complete its run within 15 minutes.

In contrast, on all problems FM-*Darwin* never uses more than 200 MB of memory, and in most cases less than 50 MB. In conclusion then, both the artificial problem in Example 4 and the more realistic problems in the TPTP library support our thesis that FM-*Darwin* scales better on bigger problems, that is, problems with a larger set of ground instances for non-trivial domain sizes.

On the other hand, Paradox and to a lesser extent Mace4 tend to solve problems faster than FM-*Darwin*. We expect, however, that the difference in speed will decrease in later implementations of our system as we refine and improve our approach further.

[12] Since *Darwin* native input language is clausal, we used the eprover 0.91 to convert non-clausal TPTP problems into clause form.

6 Conclusions

Recent years have seen considerable interest in procedures for computing finite models of first-order logic specifications. In this paper we overcome a major problem with established, leading methods—embodied by systems like Paradox and Mace4—which do not scale well with the required domain size of the (smallest) models. These methods are essentially based on propositional reasoning. In contrast, we proposed instead to reduce model search to a sequence of satisfiability problems made of function-free first-order clause sets, and to apply (efficient) theorem provers capable of deciding such problems.

In this paper we presented our approach in some detail and argued for its correctness. We then provided results from a comparative evalution of our prover, Mace4 and Paradox, demonstrating that the expected space advantages do indeed occur. The evaluation also shows that FM-*Darwin*, our initial implementation of our approach built on top of the *Darwin* theorem prover, is already competitive with state-of-the-art model builders.

We believe that the performance of FM-*Darwin* has still considerable room for improvement. One main opportunity of improvement is that currently there is no explicit symmetry breaking mechanism for function symbols of arity greater than zero. Another is that the disequality of domain elements is still explicitly axiomatized by ground axioms over the domain constants. In future work, we intend to explore the possibility of adapting existing first-order level symmetry breaking techniques to our method, and building-in equality over domain constants into FM-*Darwin*.

While FM-*Darwin* scales better memory-wise than the other systems considered, it generally struggles like all other finite model-finders with problems (such as the TPTP problem `LAT053-1`) whose smallest model is relatively large (20 or more elements). Increasing the scalability towards larger domain sizes is then certainly a main area of further research.

Acknowledgements. We thank the reviewers for their helpful comments.

References

[Bez05] M. Bezem. Disproving distributivity in lattices using geometry logic. In *Proc. CADE-20 Workshop on Disproving*, 2005.

[BFT06a] Peter Baumgartner, Alexander Fuchs, and Cesare Tinelli. Implementing the model evolution calculus. *International Journal of Artificial Intelligence Tools*, 15(1):21–52, 2006.

[BFT06b] Peter Baumgartner, Alexander Fuchs, and Cesare Tinelli. Lemma learning in the model evolution calculus. Submitted, Mai 2006.

[BGV98] Leo Bachmair, Harald Ganzinger, and Andrei Voronkov. Elimination of equality via transformation with ordering constraints. In Claude Kirchner and Hlne Kirchner, editors, *Automated Deduction — CADE 15*, LNAI 1421, Lindau, Germany, July 1998. Springer-Verlag.

[Bra75] D. Brand. Proving theorems with the modification method. *SIAM Journal on Computing*, 4:412–430, 1975.

[BS06] Peter Baumgartner and Renate Schmidt. Blocking and other enhancements for bottom-up model generation methods. In *Proc. International Joint Conference on Automated Reasoning (IJCAR)*, 2006. To appear. Long version.

[BT98] Franois Bry and Sunna Torge. A Deduction Method Complete for Refutation and Finite Satisfiability. In *Proc. JELIA*, LNAI. Springer, 1998.

[BT03] Peter Baumgartner and Cesare Tinelli. The Model Evolution Calculus. In Franz Baader, editor, *CADE-19 – The 19th International Conference on Automated Deduction*, volume 2741 of *Lecture Notes in Artificial Intelligence*, pages 350–364. Springer, 2003.

[CS03] Koen Claessen and Niklas Sörensson. New techniques that improve mace-style finite model building. In Peter Baumgartner and Christian G. Fermüller, editors, *CADE-19 Workshop: Model Computation – Principles, Algorithms, Applications*, 2003.

[dNM06] Hans de Nivelle and Jia Meng. Geometric resolution: A proof procedure based on finite model search. In *Proc. International Joint Conference on Automated Reasoning (IJCAR)*, 2006. To appear.

[McC94] W. McCune. A davis-putnam program and its application to finite first-order model search: Qusigroup existence problems. Technical report, Argonne National Laboratory, 1994.

[McC03] W. McCune. Mace4 reference manual and guide. Technical Report ANL/MCS-TM-264, Argonne National Laboratory, 2003.

[Sla92] John Slaney. Finder (finite domain enumerator): Notes and guide. Technical Report TR-ARP-1/92, Australian National University, Automated Reasoning Project, Canberra, 1992.

[SS98] G. Sutcliffe and C.B. Suttner. The TPTP Problem Library: CNF Release v1.2.1. *Journal of Automated Reasoning*, 21(2):177–203, 1998.

[Zha95] Hantao Zhang. Sem: a system for enumerating models. In *IJCAI-95 — Proceedings of the 14th International Joint Conference on Artificial Intelligence, Montreal*, pages 298–303, 1995.