Computing Finite Models by Reduction to Function-Free Clause Logic

Peter Baumgartner

NICTA, Australia

Alexander Fuchs

The University of Iowa, USA

Hans de Nivelle

University of Wroclaw, Poland

Cesare Tinelli

The University of Iowa, USA

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 because the propositional formulas to be considered may become very large.

We propose instead to reduce model search to a sequence of satisfiability problems consisting 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 this paper we describe our proposed reduction in detail and discuss how it is integrated into the Darwin prover, 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.

Key words: Automated Theorem Proving, Model Building

1 Introduction

Methods for model computation can be classified as those that directly search for a finite model, like the extended PUHR tableau method (Bry and Torge, 1998), the methods in (Bezem, 2005; de Nivelle and Meng, 2006) and the methods in the SEM-family (Slaney, 1992; Zhang and Zhang, 1995; McCune, 2003), and those that are based on transformations into certain fragments of logic and which rely on readily available systems for these fragments (see (Baumgartner and Schmidt, 2006) for a recent approach).

The latter approach includes the family of MACE-style model builders (McCune, 2003). These systems search for finite models essentially by constructing a sequence of translations corresponding to interpretations with domain sizes $1, 2, \ldots$, in increasing order, until a model has been found. The target logic used by MACE-style model builders is propositional logic. The model builder from this class with the best performance today is probably Paradox (Claessen and Sörensson, 2003).

We present in this paper a new approach in the MACE/Paradox tradition which however exploits new advances in instantiation-based first-order theorem proving. Instead of using propositional logic as a target logic, we use function-free clause logic, a decidable fragment of first-order logic whose language consists of clauses over a signature containing no function symbols of arity greater than zero. Theorem provers for instantiation-based calculi like the Model Evolution (Baumgartner and Tinelli, 2003), the Disconnection (Letz and Stenz, 2001) and the Inst-Gen (Ganzinger and Korovin, 2003) calculus are natural and efficient decision procedures for this fragment. This is in contrast with provers for saturation-based calculi (such as, for instance, Resolution), where all known approaches for deciding this fragment have in the end to resort to ground instantiation of variables.

The general idea of our approach is the same as that of MACE-style model finders. To find a model with n elements for a given a clause set (possibly with equality), the clause set is first converted into the target logic by means of the following transformations:

(1) Each clause is flattened (nested function symbols are removed).

Email addresses: Peter.Baumgartner@nicta.com.au (Peter Baumgartner), fuchs@cs.uiowa.edu (Alexander Fuchs), nivelle@ii.uni.wroc.pl (Hans de Nivelle), tinelli@cs.uiowa.edu (Cesare Tinelli).

(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 crucial difference, however, is the choice of a more expressive target logic that is much closer to the logic than propositional logic. To find models in this logic we use a variant of *Darwin* (Baumgartner et al., 2006a), our implementation of the Model Evolution calculus. (Baumgartner and Tinelli, 2003), 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 does not depend on *Darwin* or the Model Evolution calculus. Without much additional effort, we could use any other decision procedure for function-free clause logic, such as, for example, current implementations of the other instantiation-based calculi mentioned above.

In this paper we illustrate our method in some detail, presenting the main translation and its implementation within *Darwin*, and discuss our initial experimental results in comparison with Paradox itself and with Mace4 (McCune, 2003), 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 (Sutcliffe and Suttner, 1998). 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 (Robinson and Voronkov, 2001, e.g.). We work with clauses over a signature Σ of function and predicate symbols, possibly with equality. As usual, we call 0-arity function symbols *constants*. We use the distinguished infix symbol $\approx$ for the equality predicate, and the notation $s \not\approx t$ as an abbreviation of $\neg(s \approx t)$.

We define terms, atoms, literals and formulas over Σ and a given (enumerable) set of variables V 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 $P(t_1, \ldots, t_n)$).

We also use the usual notion of substitution. We denote substitutions by the letter σ or more concretely by finite maps of the form $\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$ from variables to terms.

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 $\mathfrak{I}$. If P is an n-ary predicate symbol, $P^{\mathfrak{I}}$ denotes the relation over $|\mathfrak{I}|^n$ that $\mathfrak{I}$ associates to P (similarly for function symbols). Recall that in E-interpretations the equality relation is interpreted as the *identity relation*, i.e. for every E-interpretation $\mathfrak{I}$, $\approx^{\mathfrak{I}} = \{(d, d) \mid d \in |\mathfrak{I}|\}$.

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 consider with no loss of generality only input problems expressed as a (finite) set of clauses. We fix one such set M and let $\Sigma = \Sigma_{\mathrm{F}} \cup \Sigma_{\mathrm{P}}$ be its signature, where Σ_{F} (resp. Σ_{P}) are the function symbols (resp. predicate symbols) occurring in M.

3 Finite Model Transformation

In this section we describe a set of transformations that we apply to the input problem to reduce it to an equisatisfiable problem in function-free clause logic without equality, for a given domain size.

In the rules, the letters s and t denote terms, while the letters x and y denote variables. 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 a (single) application of one of the rules.

3.1 Basic Transformation

(1) Abstraction of positive equations.

$$s \approx y \vee C \rightsquigarrow \quad s \not\approx x \vee x \approx y \vee C \quad \text{if } s \text{ is not a variable and } x \text{ is a fresh variable}$$

$$x \approx t \vee C \rightsquigarrow \quad t \not\approx y \vee x \approx y \vee C \quad \text{if } t \text{ is not a variable and } y \text{ is a fresh variable}$$

$$s \approx t \vee C \rightsquigarrow s \not\approx x \vee t \not\approx y \vee x \approx y \vee C \quad \text{if } s \text{ and } t \text{ are not variables and } x \text{ and } y \text{ are fresh variables}$$

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

(2) Flattening of non-equations.

$$(\neg)P(\ldots, s, \ldots) \vee C \rightsquigarrow (\neg)P(\ldots, x, \ldots) \vee s \not\approx x \vee C \quad \text{if } P \text{ is not } \approx, \; s \text{ is not a variable, and } x \text{ is a fresh variable}$$

(3) Flattening of negative equations.

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

(4) Separation of negative equations.

$$s \not\approx t \vee C \rightsquigarrow s \not\approx x \vee t \not\approx x \vee C \quad \text{if } s \text{ and } t \text{ are not variables, and } x \text{ is a fresh variable}$$

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.

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

(6) Orientation of negative equations.

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

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. [1]

[1] It is easy to see that this process is guaranteed to terminate.

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.

This transformation follows closely the one applied by the Paradox MACE-style model finder. The only significant difference is in (1), where, in contrast to Paradox, we abstract also equations of the form $s \approx x$ and $x \approx s$. This relieves us from the need to add functionality axioms to the transformed clause set, as explained below (Section 3.2). Note that our possibly larger number of fresh variables does not have the negative impact it would have in Paradox's case, as the final clause set is not grounded in our case (Section 5).

The two flattening transformations (2) and (3) alone, when applied exhaustively, turn a clause into a *flat* one, i.e., a clause consisting of *flat literals*, where a literal is flat if its atom, modulo the orientation of $\approx$, has the form $x \approx y$, $x \approx f(x_1, \ldots, x_n)$, $f(x_1, \ldots, x_n) \approx g(y_1, \ldots, y_m)$, or $P(x_1, \ldots, x_n)$ where the x's and the y's are variables, f, g are function symbols (possibly of 0-arity), and P is a predicate symbol.

Similar flattening transformations have been considered before as a means to deal more efficiently with equality within calculi for first-order logic without equality (Brand, 1975; Bachmair et al., 1998).

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. (Brand, 1975)). 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.

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

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

For example, the application of (1) – (6) transforms the unit clause $a \approx f(z)$ into $a \not\approx x \vee f(z) \not\approx y \vee x \approx y$. Applying (7) as well yields $\neg R_a(x) \vee \neg R_f(z, y) \vee x \approx y$.

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 then 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 tuple $(d_1, \ldots, d_n, d)$, 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.

What is left to prove is that when $\mathfrak{I}$ is a model of $\mathcal{B}_R(M)$ 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 easily that any clause of $\mathcal{B}_R(M)$ satisfied by $\mathfrak{I}$ is also satisfied by $\mathfrak{I}'$. $\square$

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 Adding Finite-Domain Constraints

In order to enforce 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.[2] 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 the expansion of the signature of $\mathcal{B}_R(M)$ with d *domain values*, that is, d fresh constant symbols, which we name $1, \ldots, d$. Intuitively, for each E-interpretation of cardinality d, 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 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

[2] Altogether, this proves the (well-known) result that function symbols are "syntactic sugar". They can always be eliminated in a satisfiability preserving way, at the cost of introducing existential quantifiers.

Lemma 2 in the (interesting) right-to-left direction. This is easily achieved with the clause set [3]

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

Finally then, we define the *finite-domain transformation of M for size d* 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, for being a set of universal formulas with no function symbols other than the constants $1, \ldots, d$, the set $\mathcal{F}(M, d)$ is satisfiable if and only if 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 occurrences of negative equations in $\mathcal{F}(M, d)$ are those contributed by $\mathcal{E}(d)$, which are still satisfied after the change. [4] It is this modified model that can be turned into an E-model of M. $\square$

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

[3] Notice that as equality is now completely axiomatized, we could have chosen to replace $\approx$ by a fresh predicate symbol, say E.

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

(Paradox (Claessen and Sörensson, 2003), Finder (Slaney, 1992), Mace (McCune, 1994), Mace4 (McCune, 2003), SEM (Zhang and Zhang, 1995) 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*).

Conceptually, FM-*Darwin* builds on *Darwin* by adding to it as a front-end an implementation of the transformation $\mathcal{F}$ (Section 3.4), 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 as detailed below.

The difference consists in a number of technical improvements that help the performance of our method while preserving its correctness. These improvements are not difficult to prove correct, so for most of them we will leave the correctness proofs to the reader.

4.1 Preprocessing Improvements

Initial Transformation

FM-*Darwin* implements some obvious optimizations over the transformation rules described in Section 3. For instance, the transformations (1)–(4) are done in parallel, depending on the structure of the current literal. Transformation (6) is done implicitly as part of transformation (7), when turning equations into relations. Also, when flattening a clause, the same variable is used to abstract different occurrences of identical subterms. The latter improvement, which is trivially satisfiability preserving, is justified by its very small cost in *Darwin* [5] and the fact that it leads to a significant performance improvement in a number of cases.

[5] Because common subexpressions are always shared.

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

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

can be replaced by the clause set

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

where h_1 and h_2 are fresh function symbols. When carried out repeatedly, reusing definitions across the whole clause set, this transformation yields shorter flattened clauses. It is not hard to show, especially when the extracted terms are ground, that this sort of transformation is satisfiability preserving. We do not delve in its formal definition and proof of correctness here because of the following.

We tried some heuristics for when to apply the transformation, based on how often a term occurs in the clause set, and how much larger the flattened clause would be without extracting some subterms first. The only consistent improvement on TPTP problems was achieved when definitions were introduced for ground terms only. This solved 16 more problems, 14 of which were Horn. Thus, currently only ground terms are extracted by default with this transformation in FM-*Darwin*.

Functionality Axioms

While our transformation does not require us to axiomatize functionality, experiments showed that doing so is more often beneficial than not, if only marginally overall. Therefore, by default we add the following functionality axiom for each predicate symbol R_f

$$\neg R_f(x_1, \ldots, x_n, d) \vee \neg R_f(x_1, \ldots, x_n, d')$$

for all domain elements d, d' with $d < d'$. For additional flexibility, FM-*Darwin* leaves the user the option to omit these axioms.

Splitting Clauses

Systems like 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. It is well known that this sort of transformation preserves satisfiability. 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. [6]

Now, reducing the number of variables per clause is not necessarily helpful in our case. Since (FM-)*Darwin* does not perform an exhaustive ground instantiation of its input clause set, splitting clauses can actually be counterproductive because it forces the system to populate its model representation with instances of connecting literals like $S(y)$ above. Our experiments indicate that this is generally expensive unless the connecting literals contain no 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. In particular, in our experiments on the TPTP library (Section 5.2) it helped to solve eight additional satisfiable problems.

Symmetry Breaking

Symmetries, in particular *value symmetries*, have been identified as a major source of inefficiencies in constraint solving. A constraint satisfaction problem exhibits value symmetry if permuting the values of a partial solution for the problem (i.e., an assignment of values to a subset of the problem's variables that satisfies a subset of the constraints) gives another partial solution. Breaking such symmetries often produces considerable efficiency gains—with no loss of generality if one is not interested in symmetric solutions.

In our context, it is easy to break some of the value symmetries introduced by assigning domain values to constant symbols. 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). It is easy to see

[6] A similar observation is made in (Kautz and Selman, 1996) and is used successfully to solve planning problems by reduction to SAT.

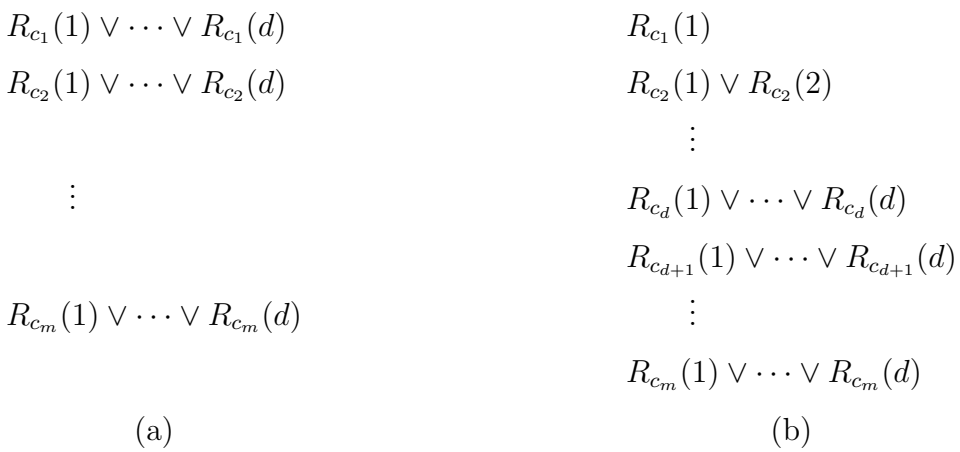

Fig. 1. Totality axioms for constants and their triangular form

that the triangular form has less satisfying interpretations over the domain $\{1, \ldots, d\}$ than the first form, and that, nevertheless, any interpretation satisfying the first form is isomorphic to an interpretation satisfying the second. 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, however, as they do not constrain the search for a model further (they are all *pure*).

This improvement is quite effective, especially when combined with others. In concrete, it yields speed ups on all problems, allowing the system to solve between 40 and 70 additional satisfiable TPTP problems, depending on whether it is used alone or with sort inference (described next).

We point out that a similar symmetry breaking trick exists for unary function symbols. This one however is not effective for FM-*Darwin*; hence we do not describe it here and instead refer the reader to (Claessen and Sörensson, 2003) where it is illustrated for Paradox.

Sort Inference

Like Paradox, FM-*Darwin* performs a kind of sort inference to improve the effectiveness of symmetry breaking. Our sort inference algorithm is essentially the same as the type reconstruction algorithm used in programming languages with parametric types (but no subtypes) such as ML. [7]

At the beginning, 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

[7] An earlier use of it in automated deduction in (unsorted) first-order logic can be found in (Baumgartner et al., 1997).

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, the sorts assigned to 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 taken to be disjoint and of the same size. In essence, this is achieved by using annotated versions $\{1^S, \ldots, d^S\}$ of the domain values for each sort S. This way, when a sorted model is found it can be translated into an unsorted model by an isomorphic translation of each sort into a single domain of size d.

Searching for a sorted model instead of an unsorted one allows the system to apply the symmetry breaking axioms independently for each sort. For example, if the constant symbols in Σ were a, b, c and d, and we used them in alphabetical order in the totality axioms, for the unsorted problem we would get the triangular form

$$R_a(1),\ R_b(1) \vee R_b(2),\ R_c(1) \vee R_c(2) \vee R_c(3),\ R_d(1) \vee R_d(2) \vee R_d(3) \vee R_d(4).$$

In contrast, if sort inference determined a, b, and c to be of one sort and d of a different sort, say, the axioms would be

$$R_a(1),\ R_b(1) \vee R_b(2),\ R_c(1) \vee R_c(2) \vee R_c(3),\ R_d(1).$$

The latter axiomatization amounts to recognizing, and breaking, more symmetries than in the unsorted case, and leads to substantial reductions in the search space as well as improvements in performance.

In practice, sort inference produces more than one sort for more than 40% of all TPTP problems, and in particular for more than 60% of all satisfiable ones. Applying symmetry-breaking by sort then leads to about 30 additional solved problems and a general speed up of a factor of two, compared to symmetry breaking in the unsorted case.

It is worth noting that, by also reducing the number of possible ways to order the input constants for the triangular form totality axioms, sorting also makes the whole system more robust, since different orders can have a dramatic impact on the shape of the search space.

Meta Modeling

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 generally 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}). For example, if the Σ_{F} contains the function symbols $f_1, \ldots, f_n$ of arity n, then instead of one totality axiom per function symbol

$$R_{f_1}(x_1, \ldots, x_n, 1) \vee \cdots \vee R_{f_1}(x_1, \ldots, x_n, d)$$
$$\cdots$$
$$R_{f_n}(x_1, \ldots, x_n, 1) \vee \cdots \vee R_{f_n}(x_1, \ldots, x_n, d)$$

it suffices to have the following single totality axiom for all function symbols of arity n

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

where the variable y is meant to be quantified over the (original) function symbols in Σ_{F}. Furthermore, in all reasonable proof procedures based on the Model Evolution calculus y will be instantiated in a totality axiom only if there is a complementary literal of the form $\neg R_n(f, x'_1, \ldots, x'_n, d)$, thus ensuring that y will be instantiated only with zero-arity symbols representing function symbols of arity n.[8] 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.[9]

Meta modeling is not a new idea. Similar forms of it were already extensively used in early applications of automated reasoning in Artificial Intelligence (see, e.g., (Genesereth and Nilsson, 1987)). While its correctness is not as immediate as in the case of the other transformations presented so far, we do not discuss it here because meta modeling turns out to provide only a very modest improvement in FM-*Darwin*, in terms of time as well as memory. We think there are several reasons for this. First, the generalization can only pay

[8] While this is not required for correctness, it ensures that the transformation does not increase the search space.

[9] 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.

off (and consequently is only applied) if for a given arity there are at least two symbols of that arity, otherwise it merely introduces unification overhead. Second, the symmetry breaking axioms prevent its application to constant symbols. Third, when sort inference is applied it is not enough to generalize function symbols by arity alone, instead their sorts have to be taken into account as well. Altogether this makes it questionable whether the increase in complexity introduced by this transformation is justified.

Initial Domain Size

Following again the example of Paradox, FM-*Darwin* performs some static analysis of the input clause set to quickly determine a (possibly suboptimal) lower bound k on the cardinality of any model of the clause set. Roughly, this is done by identifying cliques of disequations entailed by the clause set. Then, the search starts with k as the initial domain size instead of 1.

The computation of a lower bound is done by default because of its very small overhead. However, we must add that in our experiments it did not lead to any substantial performance improvement overall.

4.2 Run-time Improvements

Restarts

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.

By modifying the treatment of equality we could allow for increasing the domain size in steps greater than 1. That is, when going from domain size d to domain size $d+m$, we would add the axioms $\mathcal{E}(d+1)$ instead of $\mathcal{E}(d+m)$. This would enforce a lower domain size of $d + 1$ instead of $d + m$. Furthermore, as *Darwin* has no native support for equality we would need to add the standard axioms of equality, that is reflexivity, symmetry, transitivity, and predicate substitution axioms. This ensures that the domain elements are in an equivalence relation, if a model of domain size smaller than $d + m$ is found. Since it turned out in our experiments that this approach is significantly less efficient, we consider in the following only a fixed increment of 1.

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 re-generate the constant part, and this is not done.

Conflict-based Learning

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. Using learned lemmas is helpful in pruning later branches of the search space (Baumgartner et al., 2006b). 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 D_d standing for the current domain size. In FM-*Darwin*, lemmas depending on the current domain size d, and only those, retain the guard D_d when they are built, making it easy to eliminate them when moving to the next size $d+1$.

Inspired by the mechanism used in (Jia and Zhang, 2006) for a constraint-based model finder, lemmas produced by *Darwin* can in some cases be generalized in FM-*Darwin*. The basic idea of (Jia and Zhang, 2006) is to store explored branches of the search tree so that later in the search branches that are redundant due to some isomorphism can be detected and pruned. However, in contrast to their approach, ours does not need an additional data structure to store this information. That information can be incorporated into the existing lemma learning mechanism, and more compactly, too, since lemmas capture the reason for a failure in a branch, instead of the whole branch.

The idea is the following. Assume for now, just for simplicity, that we do not perform any sort inference on the input clause set M (or that the sort inference procedure generates a single sort S). When the number m of input constants (of sort S) is smaller than the current domain size d, the symmetry breaking triangular form for the totality axioms forces the first m domain values to be the interpretation of the input constants, but imposes no constraints on the remaining $d-m$ domain values.[10] As a consequence, every Herbrand model I of the clause set $\mathcal{F}(M,d)$ is invariant under any permutation p of $(m+1,\ldots,d)$. In other words, if the model satisfies a ground literal L, it will also satisfy the literal obtained by applying p to L. This means that whenever $\mathcal{F}(M,d)$ entails a formula $\varphi(v_1,\ldots,v_k)$ containing the domain values $v_1,\ldots,v_k$

[10] Note that naming of subterms and splitting of clauses might introduce Skolem constants, to which symmetry breaking is applied just as to an input constant, thus potentially increasing the number of constrained domain elements.

from $\{m+1,\ldots,d\}$ it will also entail the formula

$$\forall x_1,\ldots,x_k.\ \bigwedge_{1\le i\le k} x_i \in \{m+1,\ldots,d\}\ \wedge \bigwedge_{1\le i<j\le k} x_i \not\approx x_j\ \Rightarrow\ \varphi(x_1,\ldots,x_k)$$

where $\varphi(x_1,\ldots,x_k)$ is obtained from $\varphi(v_1,\ldots,v_k)$ by replacing, for each i, every occurrence of the value v_i with the fresh variable x_i.

In the general case of more than one sort, this kind of generalization is applied to lemmas containing unconstrained domain constants as follows. During preprocessing, the system adds to $\mathcal{F}(M,d)$ a unit clause of the form $Per^S(v)$ (for "v is permutable in S") for each inferred sort S and domain value v of sort S that is unconstrained by the symmetry breaking axioms for S. Then, during search, every computed lemma $C(v_1,\ldots,v_k)$ containing unconstrained values $v_1,\ldots,v_k$ of sort S, say, is generalized to the lemma

$$\bigvee_{1\le i\le k} \neg Per^S(x_i)\ \vee \bigvee_{1\le i<j\le k} x_i \approx x_j\ \vee\ C(x_1,\ldots,x_k)$$

where $x_1,\ldots,x_k$ are fresh variables (of sort S). This lemma is then further generalized by applying to it the same process but for another sort, until all unconstrained domain values have been eliminated.

With the resulting generalized lemma the system can break more symmetries at run time than with the original lemma. In fact, the search process will avoid not just any (candidate) model I that falsifies the original lemma but also any model obtained from I by a well-sorted permutation of the unconstrained domain values.

Combined with the improvement described next, which reduces the overhead of using longer clauses, generalized lemmas lead to shorter derivations, a smaller search space *and* smaller run-times overall. Nevertheless, while using the original lemmas leads in general to a significant speed up of a factor of 2 to 4 (see (Baumgartner et al., 2006b) for details), the magnitude of the further lemma generalization in our experimental evaluation was so far minimal. Specifically, the additional speed up factor is only 1.11 over the whole TPTP library. More important, the number of solved problems is essentially unchanged.

Constraint-based approach

FM-*Darwin* has a facility for treating equality and permutability predicates as built-in constraints. In this approach, every clause of the form

$$C\ \vee \bigvee_{i,\iota} \neg Per^{S_\iota}(x_i)\ \vee \bigvee_{i,j} x_i \approx x_j$$

where C contains no disequations and no permutability literals, is rewritten as a *constrained clause* of the form $C \mid \Gamma$ where Γ is the constraint set

$$\bigcup_{i,\iota}\{Per^{S_\iota}(x_i)\} \;\cup\; \bigcup_{i,j}\{x_i \not\approx x_j\} \;.$$

Darwin's inference process is based on generating instances of its input clause set M'—in this case of the set $\mathcal{F}(M,d)$ plus any lemmas added along the way—and choosing literals from these instances to build a candidate model of the clause set. These instances are generated by computing certain unifiers, called *context unifiers*, for each clause C in M', and applying them to C—see (Baumgartner et al., 2006a) for more details.

In the regular approach, if the clause contains an equation $x \approx y$ with x and y of some sort S, the computation of the context unifiers will attempt to instantiate x and y to all domain values for S. Similarly, if the clause contains a permutability literal $Per^S(x)$, it will attempt to instantiate x to all unconstrained domain values for S.

In the constraint-based approach, context unifiers are computed as usual but using only the clause part C of a constrained clause $C \mid \Gamma$. Then, each context unifier σ for C is further refined into the unifier $\sigma\theta$ for each solution θ of the constraint $\Gamma\sigma$ over the sort domains. These solutions are computed using constraint satisfaction techniques that treat sort assignments to variables as well as permutability constraints as domain constraints, and disequations as disequality constraints.

The main advantages of this approach are that (i) it is not necessary to include in $\mathcal{F}(M,d)$ the quadratically many ground disequations $v \not\approx v'$ for all distinct domain values nor the linearly many ground permutability predicates $Per^S(v)$; (ii) *Darwin*'s inference rules operate on shorter clauses, especially in case of generalized lemmas, and (iii) computing the context unifiers $\sigma\theta$ using the specialized constraint solving algorithm for the θ part is more efficient than computing $\sigma\theta$ directly with *Darwin*'s context unification procedure.

We finally remark that meta modeling and generalized lemma learning are the only improvements specific to the targeted function-free clause logic, and potentially to the Model Evolution calculus. All other optimizations are applicable in the original propositional MACE-style setting as well.

5 Experimental Evaluation

5.1 Space Efficiency

As we have seen, our reduction to a clause set $\mathcal{F}(M, d)$ encoding finite E-satisfiability is heavily influenced by the one done in Paradox, but with the difference that in Paradox the whole counterpart of our clause set $\mathcal{F}(M, d)$ is grounded out, simplified and fed into a SAT solver.

Feeding the set $\mathcal{F}(M, d)$ directly instead to a theorem prover capable of deciding the satisfiability of function-free clause sets 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 (Claessen and Sörensson, 2003). 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, 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 function 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, needs to take the dynamics of model search into account.

By using *Darwin* as the back-end 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, let

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

	FM-*Darwin*			Mace4	Paradox		
n	**|Cont|**	**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	1.1	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, Mace4 and Paradox on Example 4, for $n = 3, \ldots, 10$. All **Time** results are CPU time in seconds. **FM-*Darwin*** columns: **|Cont|**, maximum context size needed in derivation; **Mem**, required memory size in megabytes. **Mace4** columns: "Fail at size d", memory limit of 400 MB exhausted during search for a model with size d. **Paradox** columns: **Vars**, number of propositional variables of the translation to propositional logic for domain size n; **Clauses**, likewise, number of propositional clauses; "Inconclusive, size $\geq d$", Paradox gave up after the time stated.

$c_1, \ldots, c_n$ be (distinct) constants, and let $x, x_1, \ldots, x_n$ be (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 in first-order logic without equality. □

We ran the example for $n = 3, \ldots, 10$ on FM-*Darwin*, Mace4 1.3, and Paradox, and obtained the results in Table 1. These results confirm our expectations on FM-*Darwin*'s 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 context grows much more slowly.

Problem Type		Problems	FM-*Darwin*		Mace4		Paradox 1.3	
Horn	**≈**		**Sol**	**Time**	**Sol**	**Time**	**Sol**	**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, grouped based on being Horn and/or containing equality. **Sol** gives the number of problems solved by a configuration, **Time** the average time used to solve these problems.

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. Since *Darwin*'s native input language is clausal, we used the E prover (Schulz, 2004) version 0.91 to convert non-clausal TPTP problems into clause form. 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. [12] Paradox and Mace4 were run in their respective default configuration, under the assumption that it would be the most effective.

We report here only the results of FM-*Darwin*'s default configuration, which uses sort inference and symmetry breaking but neither generalized lemmas nor constraint solving, because the results for the alternative configurations are only marginally different.

The results given in Table 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. We speculate that a factor in Paradox's superior performance for Horn problems might be the very efficient unit propagation algorithm of its underlying SAT solver, based on the two-watched literals scheme (Moskewicz et al., 2001). Since the only non-Horn clauses introduced by the transformation are the totality ax-

[12] We refer the reader to (Baumgartner et al., 2006b) for more details on this option. In brief, it makes *Darwin* generate less general lemmas, but much more efficiently, making it *Darwin*'s most effective learning option in most cases.

ioms, and since lemmas learned from Horn clauses are still Horn, solving Horn problems in Paradox probably requires only a minimal amount of actual search and reduces to a large degree to unit propagation.

Looking at the experimental results in more detail, 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. While both approaches have the same complexity for a satisfiable problem, that is exponential for each domain size, this cost is paid eagerly in the propositional approach.

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.

*5.3 (FM-)*Darwin *Variants on TPTP*

We mentioned that *Darwin* is refutation complete prover for clause logic and a decision procedure for function-free clause logic. [13] In addition, and contrary to other provers, *Darwin* is often able to find (possibly infinite) models of satisfiable clause sets containing function symbols. On the other hand, FM-*Darwin* is a "finite-model complete" model finder for clause logic—that is, with enough resources it is guaranteed to find a finite model for any input clause set that has one. FM-*Darwin* is also able to decide function-free clause logic by stopping the search when it fails to find a model of size up to the number of input constants. [14]

[13] *Darwin* does accept clause sets with equality as well but it processes them rather inefficiently by simply adding relevant equality axioms to the input clauses.
[14] It is a well-known property that a clause set in this logic is satisfiable iff it has a model with size no greater than the number of its constant symbols.

It is interesting then to see how FM-*Darwin* compares to plain *Darwin*, to assess the advantage of extending the latter to the former. We measured the performance of the two systems on the relevant subsets of the TPTP library, namely all the satisfiable and all the function-free problems.

Since *Darwin*, contrary to FM-*Darwin*, is not complete for computing finite models it is instructive to single out the results for function-free clause logic problems. To this end, we created the following three disjoint problem classes—where we classify a non-clausal TPTP problem based on its clause version as obtained by using the E prover for clausification:

- NFF/S, consisting of all satisfiable problems containing (non-constant) function symbols,
- FF/S, consisting of all satisfiable function-free problems, and
- FF/U, consisting of all unsatisfiable function-free problems.

The tests in this section (Table 3 and Table 4) where done under the same conditions as for the previous tests, with the only difference that we used faster 3Ghz Pentium machines—which explains why FM-*Darwin* solves more than the 1074 satisfiable problems reported earlier.

	NFF/S			**FF/S**			**FF/U**		
Solver	**Pro**	**Sol**	**Time**	**Pro**	**Sol**	**Time**	**Pro**	**Sol**	**Time**
Darwin	910	404	3.9	341	338	0.6	642	641	0.2
FM-*Darwin*	910	758	3.8	341	322	6.4	642	519	0.6

Table 3
Comparison of *Darwin* and FM-*Darwin* over all satisfiable TPTP problems containing function symbols (NFF/S), all satisfiable function-free problems (FF/S), and all unsatisfiable function-free problems (FF/U). **Pro** gives the number of problems for a class, **Sol** gives the number of problems solved by a configuration, **Time** the average time used to solve these problems.

As shown in Table 3, the experiments confirm our expectation that FM-*Darwin* is superior to *Darwin* for problems with function symbols (NFF/S), and by considerable a margin. For function-free problems the results are reversed, also as expected. In particular, the difference in favor of *Darwin* is quite dramatic for the unsatisfiable problems (FF/U). FM-*Darwin*'s worse performance in this case is easily explained by observing that the system recognizes that the input problem is unsatisfiable only after it has tried, and failed, to find a model for each cardinality up to the number of input constants. *Darwin*, on the other hand, directly builds a refutation of the problem. For the satisfiable function-free problems (FF/S) *Darwin* tends to significally outperform FM-*Darwin* only on problems that do not have models of small size. The reason is again that *Darwin* does not perform an exhaustive model search by increasing

Symmetry reduction	Sort inference	Sol	Time
no	no	1012	10.3
yes	no	1053	8.1
yes	yes	1080	4.6

Table 4
Influence of symmetry reduction and sort inference, obtained by running FM-*Darwin* over all 1251 satisfiable TPTP problems. **Sol** gives the number of problems solved by a configuration, **Time** the average time used to solve these problems.

size.

The important point is that FM-*Darwin* clearly outperforms *Darwin* overall on the satisfiable problems (NFF/S + FF/S) by solving 1080 problems over a total of 1251, versus the 742 solved by *Darwin*. *Darwin* can solve only 21 satisfiable problems that FM-*Darwin* cannot solve. Of those, 16 are function-free clause problems, and only for one of the other problems does *Darwin* return an infinite model. So, at least on the TPTP library, *Darwin*'s capability to find infinite Herbrand models does not seem to be an advantage.

We also ran FM-*Darwin* in different configurations to test the impact of symmetry reduction and sort inference (Section 4.1).

Table 4 contains results for three configurations: a basic one with no symmetry reduction, and so no sort inference either; one with symmetry reduction, via the triangular form totality axioms, but no sort inference; one with sort inference and symmetry reduction by sort. As can be easily seen from the table, symmetry reduction is quite effective, especially when done sort-wise. It increases the number of solved problems while also decreasing the average solving time.

5.4 CASC J3

We conclude this section by reporting the results of the last CASC competition, CASC-J3, held as part of the 2006 Federated Logic Conference, and including FM-*Darwin*, *Darwin*, ad Paradox among its participants. CASC (the CADE ATP System Competition) is an annually competition for first-order provers, based on the TPTP library (Sutcliffe and Suttner, 2006).

Table 5 shows some comparative results for the SAT and EPR divisions of CASC-J3. SAT contains only satisfiable problems with function symbols, while EPR contains only function-free (satisfiable and unsatisfiable) clause prob-

Division	**FM-*Darwin***		***Darwin* 1.3**		**Paradox 1.3**	
	Sol	**Time**	**Sol**	**Time**	**Sol**	**Time**
SAT	70	13.6	18	31.6	90	5.7
	FM-*Darwin*		***Darwin* 1.3**		**Vampire 8.0**	
	Sol	**Time**	**Sol**	**Time**	**Sol**	**Time**
EPR	92	10.33	100	4.7	78	4.19

Table 5
Results of selected systems for the SAT and EPR divison of CASC-J3 (100 problems per division). **Sol** gives the number of problems solved by a configuration, **Time** the average time used to solve these problems.

lems. FM-*Darwin* finished third in the SAT division, after two versions of Paradox (1.3 and 2.0, in this order), and third in the EPR divison, after *Darwin* and DCTP, a Disconnection Calculus prover that, like *Darwin*, is a decision procedure for function-free clause logic. Paradox's results for the EPR division are not reported in the table because only Paradox 2.0 participated there, performing worse than Paradox 1.3 would have.

Consistently with our previous evaluation, FM-*Darwin* performed reasonably well on the satisfiable problems in the SAT division. An explanation for Paradox 1.3 looking significantly superior to FM-*Darwin* is that satisfiable function-free clause problems are included in the EPR division, but not in the SAT division, and that among these there is a large number of problems for which FM-*Darwin* succeeds, but Paradox fails.

The results show again that *Darwin* is very efficient on EPR problems, providing the basis for FM-*Darwin*'s efficiency. In this division, *Darwin* and FM-*Darwin* compare very favorably to saturation-based provers, such as for instance, Vampire, a frequent winner of the other divisions of CASC. In detail, Vampire solves 48 unsatisfiable EPR problems, but only 30 satisfiable ones. This result highlights the fact that while systems such as Vampire are highly efficient in a general refutation setting, they are not well-suited for finite model finding, especially if they are to rely on the transformations presented in this paper. Another shortcoming of such provers versus instantiation-based provers like *Darwin* or DCTP is that, even when they succeed in determining that an input problem is satisfiable, it is usually not easy for them to output (a finite representation of) an actual model.

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 evaluation 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 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 one. Possible mechanisms are again similar to those implemented in Paradox (Claessen and Sörensson, 2003). We plan to investigate additional, more powerful symmetry breaking techniques that detect and break symmetries dynamically during the search for a model.

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 anonymous reviewers for their useful comments on improving the paper's presentation. We also thank Koen Claessen for his feedback on this work and his kind explanations of the inner workings of Paradox.

The second and fourth author were partially supported by grant 0237422 from the National Science Foundation.

References

Bachmair, L., Ganzinger, H., Voronkov, A., Jul. 1998. Elimination of equality via transformation with ordering constraints. In: Kirchner, C., Kirchner, H. (Eds.), Automated Deduction — CADE 15. LNAI 1421. Springer-Verlag, Lindau, Germany, pp. 175–190.

Baumgartner, P., Fuchs, A., Tinelli, C., 2006a. Implementing the model evolution calculus. International Journal of Artificial Intelligence Tools 15 (1), 21–52.

Baumgartner, P., Fuchs, A., Tinelli, C., 2006b. Lemma learning in the model evolution calculus. In: Hermann, M., Voronkov, A. (Eds.), Logic for Programming, Artificial Intelligence and Reasoning (LPAR). Vol. 4246 of LNAI. Springer, pp. 572–586.

Baumgartner, P., Furbach, U., Stolzenburg, F., 1997. Computing Answers with Model Elimination. Artificial Intelligence 90 (1–2), 135–176.

Baumgartner, P., Schmidt, R., 2006. Blocking and other enhancements for bottom-up model generation methods. In: Furbach, U., Shankar, N. (Eds.), Automated Reasoning – Third International Joint Conference on Automated Reasoning (IJCAR). Vol. 4130 of LNAI. Springer, pp. 125–139.

Baumgartner, P., Tinelli, C., 2003. The Model Evolution Calculus. In: Baader, F. (Ed.), CADE-19 – The 19th International Conference on Automated Deduction. Vol. 2741 of Lecture Notes in Artificial Intelligence. Springer, pp. 350–364.

Bezem, M., 2005. Disproving distributivity in lattices using geometry logic. In: Proc. CADE-20 Workshop on Disproving. pp. 2–9.

Brand, D., 1975. Proving theorems with the modification method. SIAM Journal on Computing 4, 412–430.

Bry, F., Torge, S., 1998. A Deduction Method Complete for Refutation and Finite Satisfiability. In: Proc. 6th European Workshop on Logics in AI (JELIA). Vol. 1489 of LNAI. Springer, pp. 122–138.

Claessen, K., Sörensson, N., 2003. New techniques that improve mace-style finite model building. In: Baumgartner, P., Fermüller, C. G. (Eds.), CADE-19 Workshop: Model Computation – Principles, Algorithms, Applications. pp. 11–27.
URL `http://www.cs.miami.edu/~geoff/Conferences/CADE-19/WS4/`

de Nivelle, H., Meng, J., 2006. Geometric resolution: A proof procedure based on finite model search. In: Furbach, U., Shankar, N. (Eds.), Proc. International Joint Conference on Automated Reasoning (IJCAR). Vol. 4130 of LNAI. Springer, pp. 303–317.

Ganzinger, H., Korovin, K., 2003. New directions in instantiation-based theorem proving. In: LICS. IEEE Computer Society, pp. 55–64.

Genesereth, M. M., Nilsson, N. J., 1987. Logical Foundations of Artificial Intelligence. Morgan Kaufmann.

Jia, X., Zhang, J., 2006. A powerful technique to eliminate isomorphism in finite model search. In: Furbach, U., Shankar, N. (Eds.), Proc. International

Joint Conference on Automated Reasoning (IJCAR). Vol. 4130 of LNAI. Springer, pp. 318–331.

Kautz, H., Selman, B., 1996. Pushing the envelope: Planning, propositional logic, and stochastic search. In: Proceedings of the 13th National Conference on Artificial Intelligence and the Eighth Innovative Applications of Artificial Intelligence Conference. Portland, OR, USA, pp. 1194–1201.

Letz, R., Stenz, G., 2001. Proof and Model Generation with Disconnection Tableaux. In: Nieuwenhuis, R., Voronkov, A. (Eds.), Logic for Programming, Artificial Intelligence, and Reasoning, 8th International Conference, LPAR 2001, Havana, Cuba. Vol. 2250 of Lecture Notes in Artificial Intelligence. Springer, pp. 142–156.

McCune, W., 1994. A davis-putnam program and its application to finite first-order model search: Qusigroup existence problems. Tech. rep., Argonne National Laboratory.

McCune, W., 2003. Mace4 reference manual and guide. Tech. Rep. ANL/MCS-TM-264, Argonne National Laboratory.

Moskewicz, M. W., Madigan, C. F., Zhao, Y., Zhang, L., Malik, S., Jun. 2001. Chaff: Engineering an efficient SAT solver. In: Proceedings of the 38th Design Automation Conference. Las Vegas, Nevada, pp. 530–5.

Robinson, A., Voronkov, A. (Eds.), 2001. Handbook of Automated Reasoning. Elsevier.

Schulz, S., 2004. System Description: E 0.81. In: Basin, D., Rusinowitch, M. (Eds.), Proc. of the 2nd IJCAR, Cork, Ireland. Vol. 3097 of LNAI. Springer, pp. 223–228.

Slaney, J., 1992. Finder (finite domain enumerator): Notes and guide. Tech. Rep. TR-ARP-1/92, Australian National University, Automated Reasoning Project, Canberra.

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

Sutcliffe, G., Suttner, C., 2006. The State of CASC. AI Communications 19 (1), 35–48.

Zhang, J., Zhang, H., 1995. Sem: a system for enumerating models. In: IJCAI-95 — Proceedings of the 14^{th} International Joint Conference on Artificial Intelligence, Montreal. pp. 298–303.