Available online at www.sciencedirect.com

ScienceDirect

Journal of Applied Logic 7 (2009) 58–74

JOURNAL OF APPLIED LOGIC

www.elsevier.com/locate/jal

Computing finite models by reduction to function-free clause logic

Peter Baumgartner[a,*], Alexander Fuchs[b], Hans de Nivelle[c], Cesare Tinelli[b]

[a] *NICTA, Australia*
[b] *The University of Iowa, USA*
[c] *University of Wroclaw, Poland*

Available online 2 August 2007

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.

Keywords: 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 [9], the methods in [7,11] and the methods in the SEM-family [18,22,25], and those that are based on transformations into certain fragments of logic and which rely on readily available systems for these fragments (see [5] for a recent approach).

* Corresponding author.
E-mail addresses: Peter.Baumgartner@nicta.com.au (P. Baumgartner), fuchs@cs.uiowa.edu (A. Fuchs), nivelle@ii.uni.wroc.pl (H. de Nivelle), tinelli@cs.uiowa.edu (C. Tinelli).

1570-8683/$ – see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.jal.2007.07.005

The latter approach includes the family of MACE-style model builders [18]. 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 [10].

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 [6], the Disconnection [16] and the Inst-Gen [12] 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).
(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* [2], our implementation of the Model Evolution calculus. [6], 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 [18], 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 [23]. 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 ([20], 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