System Description: IVY*

William McCune[1] and Olga Shumsky[2]

[1] Mathematics and Computer Science Division
Argonne National Laboratory, U.S.A.
mccune@mcs.anl.gov
http://www.mcs.anl.gov/~mccune

[2] Department of Electrical and Computer Engineering
Northwestern University, U.S.A.
shumsky@ece.nwu.edu

Abstract. IVY is a verified theorem prover for first-order logic with equality. It is coded in ACL2, and it makes calls to the theorem prover Otter to search for proofs and to the program MACE to search for countermodels. Verifications of Otter and MACE are not practical because they are coded in C. Instead, Otter and MACE give detailed proofs and models that are checked by verified ACL2 programs. In addition, the initial conversion to clause form is done by verified ACL2 code. The verification is done with respect to finite interpretations.

1 Introduction

Our theorem provers Otter [6,7,10] and EQP [4,8] and our model searcher MACE [3,5] are being used for practical work in several areas. Therefore, we wish to have very high confidence that the proofs and models they produce are correct. However, these are high-performance programs, coded in C, with many tricks, hacks, and optimizations, so formal verification of the programs is not practical.

Instead, our approach is to have the C programs give their results explicitly as detailed proof objects or models, and to have separate *checker programs* check the results. The checker algorithms are relatively simple and straightforward, so it is practical to apply program verification techniques to them. In particular, we use the ACL2 program verification system to prove that if the checker program accepts a proof, then the proof is correct.

Otter can convert first order formulas into clauses (by normal form translation and Skolemization), but it is not able to include these preprocessing steps as part of the proof objects. Therefore, we have recoded the clause form translator in ACL2 and proved its soundness directly. The result is a hybrid system, named IVY, that (1) is driven by ACL2 code, (2) calls ACL2 functions for the

* This work was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, U.S. Department of Energy, under Contract W-31-109-Eng-38.

preprocessing, (3) calls an external program to search for a proof or a model, and (4) calls ACL2 checker functions to check the results. The top-level soundness theorems have the form: *If IVY claims a proof, then the input formula is a theorem.* A weakness of the verification method is that the soundness proofs are with respect to finite interpretations. In Section 6 we discuss an approach for all interpretations.

ACL2 (A Computational Logic for Applicative Common Lisp) [2, 1], is the successor to the Boyer-Moore theorem prover. ACL2 is a specification/programming language, based on Common Lisp, together with an environment for proving theorems about the programs. Its strength is automated support for proving inductive theorems about recursively defined programs.

2 Specification of the Logic

We use ACL2 to define a first-order logic, and this becomes the specification for our verification. The definitions of *well-formed term* and *well-formed formula* are straightforward. We next define the semantics of our logic by defining *interpretation* of a first-order language. This part is nonstandard, because we restrict ourselves to finite interpretations; see Section 6. Finally, we define *evaluation* of a formula in an interpretation. The evaluation function is a pair of mutually recursive functions, in which one recurses through the structure of formulas, and the other (called for quantified formulas) recurses through the elements of the domain of the interpretation. In particular, the function `(FEVAL F I)` evaluates formula `F` in interpretation `I`.

3 The Proof Procedure

The proof search procedure is standard for first-order resolution/paramodulation theorem provers. Starting with the negation of a conjecture, we (1) convert to negation-normal form, (2) rename bound variables, (3) Skolemize, (4) move universal quantifiers to the top, (5) convert to conjunctive normal form, (6) search for a refutation (or a model), and (7) check the refutation (or model).

Steps 1, 2, 4, and 5 produce an equivalent formula, and Skolemization produces an equiconsistent formula.

In IVY, the preprocessing steps (1–5) are coded in ACL2, the search step (6) is accomplished by calling Otter or MACE, and the checker step (7) is coded in ACL2.

4 Soundness Theorems

The function to convert formulas to negation-normal form is `(NNF F)`, and the soundness theorem states that `NNF` produces an equivalent formula:

```
(EQUAL (FEVAL (NNF F) I)
       (FEVAL F I)).
```

The soundness theorems for steps 2, 4, and 5 of the proof procedure are similar. The soundness theorem for Skolemization is more complicated, because we have to extend the interpretation with the new Skolem symbols:

```
(EQUAL (FEVAL (SKOLEMIZE F) (SKOLEMIZE-EXTEND F I))
       (FEVAL F I)).
```

Steps 6 and 7 of the proof procedure are combined in an ACL2 function `(REFUTE-N-CHECK F)` which calls Otter (see Sec. 5) and the checker function. If Otter finds a refutation, and if the checker accepts the refutation, `REFUTE-N-CHECK` returns `FALSE` (the contradictory formula of our logic); otherwise `REFUTE-N-CHECK` returns the input formula `F`. Hence, it always produces an equivalent formula, and the soundness theorem is

```
(EQUAL (FEVAL (REFUTE-N-CHECK F) I)
       (FEVAL F I)).
```

All of the preprocessing functions, `REFUTE-N-CHECK`, and a few other functions are composed into a top-level function `(PROVED F)`, which takes the positive form of a conjecture, checks that it is well-formed and closed, negates it, and applies the proof procedure. The top-level soundness theorem is

```
(IMPLIES (PROVED F)
         (AND (WFF F)
              (NOT (FREE-VARS F))
              (FEVAL F I))).
```

In other words, if IVY claims a proof of a conjecture `F`, then `F` is a closed well-formed formula that is true in all (finite) interpretations. Of course, to accept this theorem, a user must accept our ACL2 definition of first-order logic and the soundness of the ACL2 system. But the point is that the user doesn't have to trust Otter, which does the hard part of the work.

The other side of the problem, searching for countermodels, is easier because checking a claimed model produced by the C program MACE is done by simply evaluating the negation of the conjecture in the claimed model. The top-level function `(COUNTERMODEL F)` is analogous to `(PROVED F)`: it checks that the conjecture `F` is closed and well formed, negates it, preprocesses it, calls MACE to search for a finite model, and checks that the negation of `F` is true in any model found by MACE. The soundness theorem for `(COUNTERMODEL F)` is nearly trivial, because the evaluation property we need to prove is checked by `COUNTERMODEL`:

```
(IMPLIES (COUNTERMODEL F)
         (AND (WFF F)
              (NOT (FREE-VARS F))
              (NOT (FEVAL F (COUNTERMODEL F))))).
```

In other words, if IVY claims a countermodel to a conjecture `F`, then `F` is a closed well-formed formula that is false in some interpretation.

5 Interface to the C Code

The function `REFUTE-N-CHECK` takes the universal closure of a conjunction of clauses and returns an equivalent formula. First it transforms the input formula into an initial proof object. Next it calls the function `EXTERNAL-PROVER` which augments the initial proof object with additional steps that represent some derivation (a derivation of the empty clause if we are lucky). Then it checks that each step of the proof object follows from preceding steps.

In the ACL2 environment, `EXTERNAL-PROVER` is a *defstub*, that is, we tell ACL2 that it exists but that we don't know any other properties of it. We use ACL2 to prove properties of `REFUTE-N-CHECK` (e.g., soundness), but these properties are necessarily independent of `EXTERNAL-PROVER`.

At run time, a Common Lisp function `EXTERNAL-PROVER` is loaded along with the ACL2 code, and the Common Lisp version of `EXTERNAL-PROVER` overrides the ACL2 defstub.[1] The Common Lisp version of `EXTERNAL-PROVER` contains operating system calls to build an input file for Otter, run Otter, and read and process Otter's output. If the Common Lisp version of `EXTERNAL-PROVER` returns a proof object that is not well formed or is unsound, the check fails, and `REFUTE-N-CHECK` returns its input.

A similar situation holds when searching for a countermodel with MACE. A defstub `EXTERNAL-MODELER` is used in the ACL2 environment when defining functions and proving properties, and a Common Lisp version of `EXTERNAL-MODELER`, which calls MACE, is loaded at run time.

It is possible to use the preprocessing and proof checking functions of IVY with other first-order resolution/paramodulation provers and model searchers, provided they produce appropriate proof objects or models. (The format for proof object can be found in [9].) This can be accomplished by simply rewriting the Common Lisp version of the `EXTERNAL-PROVER` or `EXTERNAL-MODELER` to call the desired program.

6 The Finite Domain Assumption

Our approach of proving soundness with respect to finite interpretations is certainly questionable. Consider, for example, the sentence

```
(IMP (ALL X (ALL Y (IMP (= (F X) (F Y))
                        (= X Y))))
     (ALL X (EXISTS Y (= (F Y) X)))),
```

that is, one-to-one functions are onto. It is *not* valid, but it is true for finite domains. Could IVY claim to have a proof of such a nontheorem?

We strongly believe that it could not—that the weakness is in the metaproof method rather than the first-order proof procedure. Nonetheless, we are pursuing a general approach that covers infinite interpretations.

[1] According to the ACL2 designers, having an ACL2 function call a Common Lisp function in this way is not officially endorsed, but it is acceptable in this situation.

ACL2 has an encapsulation feature that allows it to reason safely about incompletely specified functions. We believe we can use encapsulation to abstract the finiteness.[2] In our current specification, the important way in which finiteness enters the picture is by the definition of `FEVAL-D`, which recurses through the domain. This function, in effect, expands universally quantified formulas into conjunctions and existentially quantified formulas into disjunctions. Instead of `FEVAL-D`, we can consider a constrained function that chooses an element of the domain, if possible, that makes a formula true. When evaluating an existentially quantified formula, we substitute the chosen element for the existentially quantified variable and continue evaluating. (Evaluation of universally quantified variables requires some fiddling with negation.) However, proving the soundness of Skolemization may present complications in this approach.

7 Performance and Availability

Aside from the overhead of starting up ACL2, the performance of IVY is essentially the same as the performance of Otter's autonomous mode or MACE with its default settings. IVY cannot yet accept parameters to be passed to Otter or MACE.

The latest version of IVY is available from

`http://www.mcs.anl.gov/~mccune/ivy.`

A more complete paper on IVY can be found in [9].

References

1. M. Kaufmann, P. Manolios, and J Moore, editors. *Using the ACL2 Theorem Prover: A Tutorial Introduction and Case Studies.* Kluwer Academic, 2000. To appear.
2. M. Kaufmann and J Moore. An industrial strength theorem prover for a logic based on Common Lisp. *IEEE Transactions on Software Engineering*, 23(4):203–213, 1997.
3. W. McCune. A Davis-Putnam program and its application to finite first-order model search: Quasigroup existence problems. Tech. Report ANL/MCS-TM-194, Argonne National Laboratory, Argonne, IL, May 1994.
4. W. McCune. EQP. `http://www.mcs.anl.gov/~AR/eqp/`, 1994.
5. W. McCune. MACE: Models and Counterexamples. `http://www.mcs.anl.gov/AR/mace/`, 1994.
6. W. McCune. Otter. `http://www.mcs.anl.gov/AR/otter/`, 1994.
7. W. McCune. Otter 3.0 Reference Manual and Guide. Tech. Report ANL-94/6, Argonne National Laboratory, Argonne, IL, 1994.
8. W. McCune. 33 basic test problems: A practical evaluation of some paramodulation strategies. In Robert Veroff, editor, *Automated Reasoning and its Applications: Essays in Honor of Larry Wos*, chapter 5, pages 71–114. MIT Press, 1997.

[2] This approach was suggested by Matt Kaufmann.

9. W. McCune and O. Shumsky. IVY: A preprocessor and proof checker for first-order logic. Preprint ANL/MCS-P775-0899, Argonne National Laboratory, Argonne, IL, 1999. To appear in [1].
10. W. McCune and L. Wos. Otter: The CADE-13 competition incarnations. *J. Automated Reasoning*, 18(2):211–220, 1997.