

System Description: Kimba, A Model Generator for Many-Valued First-Order Logics^{*}

Karsten Konrad¹ and D. A. Wolfram²

¹ Fachbereich Informatik, Universität des Saarlandes

² Department of Computer Science, The Australian National University

1 Overview

KIMBA is the first model generation program which implements a semi-decision procedure for finite satisfiability of first-order logics with finitely many truth values. The procedure enumerates the finite models of its input and can be used to compute efficiently domain minimal models whose positive part is minimal in size. KIMBA has been implemented in the constraint logic programming language Oz [6] and is based on a tableaux calculus that translates deduction problems into Constraint Satisfaction Problems (CSPs). The constraint propagators needed to solve these problems are realized as concurrent procedures that can make use of Oz's built-in capabilities for solving CSPs.

We describe the current prototype focusing on the method for generating and solving CSPs.

2 Theoretical Background

Model generators are automated deduction systems that prove the satisfiability of logical theories by generating models [3]. KIMBA is a model generator for many-valued logics which semi-decides the finite satisfiability of first-order theories and decides the satisfiability of propositional ones. Its proof procedure is a generalisation of Hähnle's translation of deduction problems into mixed-integer programming [2]. Hähnle's method constructs *constraint tableaux* for arbitrary propositional theories and produces sets of mixed-integer inequations which can be solved by an external constraint solver. The set of inequations generated is linear in size to that of the input theory and the solutions produced by the constraint solver are models of this input. The method does not require clause normalisation, an advantage in practice where suitable normalisation routines may not be available for the given logic. It also avoids the combinatorial explosion of tableau branches in certain many-valued logics where the branching factor of propositional rules can equal the number of truth values. However, Hähnle's method cannot be used for first-order model generation, and the mixed-integer inequations generated quickly lead to intractable CSPs.

^{*} In: Harald Ganzinger, editor, *Automated Deduction — CADE-16: Sixteenth International Conference on Automated Deduction*, Lecture Notes in Artificial Intelligence (Springer, Berlin, 1999) (To appear).

The model generator KIMBA generalises Hähnle’s idea to a constraint tableau system whose propagation mechanism can be tailored to the underlying logic. KIMBA accepts higher-order specifications and makes use of Oz’s built-in propagators and constraint solving system for integer variables. It translates quantified formulas into constraints over finite nonempty domains of constants. In the case of first-order quantification, the domain used is a finite set of individual constants including those that occur in the problem specification. Using iterative deepening over the size of this domain, the constraint solving process in Oz enumerates all finite first-order Herbrand models of the input up to a renaming of individual constants [5]. Hence, KIMBA provides a semi-decision procedure for finite satisfiability for conventional multi-valued theorem proving systems such as $3TAP$ [1] which do not halt in general on a satisfiable input theory, even when the input has a finite model.

Quantifications of the form $\forall X_\alpha P(X)$ or $\exists X_\alpha P(X)$, with X_α being a higher-order variable of type α , are translated to constraints over a given signature Σ_α of higher-order constants of type α . This actually implements a restricted form of higher-order model generation [5] and is a compact way to formalise properties of finite sets of predicates and functions.

3 Building Constraint Tableaux

Tableaux expansion in KIMBA starts with a set of pairs of the form $V_F: F$ where F is an initial closed formula and V_F is a finite domain integer variable whose value is associated with the truth value of F . Functions are translated into relations in a preprocessing step. A formula F can be signed by the user with a truth value v indicating that F must be interpreted as v in all models. The value v is an integer that can range from 0 to n where n is the highest possible truth value in the many-valued logic considered. Initial formulas F are signed by default with n , and their sign immediately determines their truth variables V_F .

$$\begin{array}{ccc}
\frac{V_F: F_1 \vee F_2}{V_{F_1}: F_1 \quad V_{F_2}: F_2} \vee & \frac{V_{\neg F}: \neg F}{V_F: F} \neg & \frac{V: \forall x_\alpha F}{V_{F,c_1}: [c_1/x]F \quad \vdots \quad V_{F,c_k}: [c_k/x]F} \forall \\
\mathbf{D}(V_{F_1}, V_{F_2}, V_F) & \mathbf{N}(V_F, V_{\neg F}) & \mathbf{A}(\{V_{F,c_1}, \dots, V_{F,c_k}\}, V)
\end{array}$$

Fig. 1. Three of KIMBA’s tableaux rules for generating constraints

Figure 1 shows three of the rules that decompose the logical structure of the input. The rules for \wedge , \Rightarrow , \Leftrightarrow and \exists are analogous. The \vee -rule introduces two new pairs $V_{F_1}: F_1$ and $V_{F_2}: F_2$ for the components of a disjunction and adds a

constraint $\mathbf{D}(V_{F_1}, V_{F_2}, V_F)$ which constrains the values of the integer variables of the disjunctive formula and its components. The \neg -rule is very similar to \vee , but introduces only one new pair for the formula in the scope of the negation, and a constraint $\mathbf{N}(V_F, V_{\neg F})$. The constraint predicates \mathbf{D} and \mathbf{N} are implemented as concurrent procedures in Oz. They must be specified separately for each logic as we shall below for a 3-valued version of \mathbf{N} .

The \forall -rule adds pairs $\{V_{F,c_1}: [c_1/x]F, \dots, V_{F,c_k}: [c_k/x]F\}$ for all instantiations $[c_i/x_\alpha]F$ of $\forall x_\alpha F$ that can be made with the current domain $\Sigma_\alpha = \{c_1, \dots, c_k\}$ of constants of type α . The truth variable V associated with $\forall x_\alpha F$ is then constrained by $\mathbf{A}(\{V_{F,c_1}, \dots, V_{F,c_k}\}, V)$. Again, the operational semantics of \mathbf{A} depends on the logic considered. For instance, in classical logic where 1 denotes truth and 0 denotes falsity, the constraint \mathbf{A} can be defined as follows:

$$\mathbf{A}(\{V_{F,c_1}, \dots, V_{F,c_k}\}, V) ::= (V_{F,c_1} + \dots + V_{F,c_k} = k) \Leftrightarrow V = 1$$

This means that the truth value V of a universally quantified formula is 1 iff the sum $V_{F,c_1} + \dots + V_{F,c_k}$ of the truth values of the formula's instantiations is equal to k . This projection of the validity of arithmetic constraints into integer variables that denote truth values is called *constraint reification*. Oz provides a constraint language for finite-domain integer variables that allows us to translate quantified formulas directly into reified constraints like the one above.

Unlike conventional tableaux systems, KIMBA's calculus does not split the current tableau branch during tableau construction. The multiple decomposition of identical formulas in different branches, a well-known weak point in many tableaux systems, is avoided.

4 Solving Constraints in Oz

Defining a logic in KIMBA means implementing propagator procedures for the logical connectives and the quantifiers. Each propagator defines the semantics of its associated connective or quantifier. KIMBA's generic translation of deduction into Oz constraints allows us to design dedicated, optimised propagators for each logic. Oz itself already provides an efficient set of propagators for the standard connectives in classical logic. Additionally, the prototype implements a 3-valued logic for partiality [4]. Below are two operationally equivalent propagator procedures for negation (\mathbf{N}) in our 3-valued logic. While the second procedure appears to be more elegant, it introduces an arithmetic equation, as in Hähnle's approach, which can be harder to process than the simple symbolic propagation in the first procedure.

```

proc {N1 V1 V2}
  [V1 V2] ::: 0#2
  thread
    if V1 = 0 then V2 = 2
    [] V1 = 1 then V2 = V1
    [] V1 = 2 then V2 = 0
    [] V2 = 0 then V1 = 2
    [] V2 = 1 then V1 = V2
    [] V2 = 2 then V1 = 0
  end
end
end

proc {N2 V1 V2}
  [V1 V2] ::: 0#2
  {FD.minus 2 V1 V2}
end

```

Procedure N1 first restricts its two parameters to values between 0 (“false”) and 2 (“true”), where 1 is used for the truth value “undefined”. Then it starts a concurrent process which waits for one of these parameters to be determined and determines the other one accordingly. Procedure N2 does the same, simply by constraining the values of V_1 and V_2 by the equation $V_2 = 2 - V_1$.

Propagation is concurrent, so the process of determining integer variables V_F can partly take place during tableau construction. An unsatisfiable tableau branch can often be closed immediately when a variable is constrained inconsistently. After a branch is saturated and the initial propagation has been completed, undetermined variables are provisionally restricted further, and the next iteration of propagation begins. As an heuristic, KIMBA restricts first those variables whose value affects as many other variables and constraints as possible. It also minimises the number of positive literals that are validated in the model at the same time. The process of variable distribution and propagation is repeated until all integer variables have a unique value. If it is not possible to assign each formula a unique integer, then the tableau is unsatisfiable and KIMBA restarts model search by extending the current domain of individuals with a newly generated constant and by building up a new constraint tableau.

The iterative deepening search for models will always find those models first whose domain of individuals is minimal. Additionally, search can be bounded by the number of literals that are validated. This implements another weak form of minimal model reasoning. By combining this restriction with domain minimality, KIMBA efficiently computes those domain minimal models of the input whose presentation as a finite set of positive literals is as short as possible.

5 System Performance

The table below shows KIMBA’s performance on some selected problems from the the TPTP library [8]. Times were taken on a Sparc Ultra 1.

Problem	PUZ001-1	PUZ005-1	PUZ017-1	PUZ031-1	MSC007-2 (5)	MSC007-2 (8)
Time	0.3s	4.6s	0.7s	1.6s	0.3s	107.3s

The original TPTP formalisations are clausal theories that have been re-formalised for KIMBA using first- and higher-order specifications in classical 2-valued logic. The TPTP formalisations of the logical puzzles PUZ001-1 to PUZ031-1 are unsatisfiable. In contrast to this, the KIMBA formalisation for each puzzle is satisfiable, and each model produced corresponds to a solution.

Cook's pigeon-hole problem, MSC007-2, is a classical benchmark proof problem for propositional theorem provers. The table shows the performance of KIMBA on problem instantiations with five and eight holes.

PUZ017-1 is very hard for most theorem provers because its first-order clause representations leads to a large search space. KIMBA's exceptionally good performance can be explained by an elegant higher-order formalisation. For instance, the specification *every job is held by at least one person* in PUZ017-1 has a natural formalisation in KIMBA as follows:

$$\forall J_{t \rightarrow o} \text{Job}_{(t \rightarrow o) \rightarrow o}(K) \Rightarrow \exists x_i \text{Person}_{t \rightarrow o}(x) \wedge J(x)$$

With this form of quantification, KIMBA keeps entities of different type separately. In a first-order logic, all quantified entities must be individuals. This usually leads to a larger than necessary domain for which model generation and theorem proving becomes exponentially more complex.

The KIMBA prototype does not employ any of the sophisticated data structures and low-level optimisations that can be found in other comparable model generation programs such as FINDER [7]. Hence, KIMBA is not yet efficient enough for instance for quasi-group existence problems where our translation of deduction into constraint solving does not effectively restrict large search spaces.

References

1. B. Beckert, R. Hähnle, P. Oel, and M. Sulzmann. The tableau-based theorem prover $\mathcal{I}^A\mathcal{P}$, version 4.0. In M. McRobbie and J. Slaney, editors, *Proc., 13th International Conference on Automated Deduction (CADE), New Brunswick, NJ, USA*, LNCS 1104. Springer, 1996.
2. R. Hähnle. Many-valued logic and mixed integer programming. *Annals of Mathematics and Artificial Intelligence*, 12(3,4):231–264, Dec. 1994.
3. R. Hasegawa. Model generation theorem provers and their applications. In L. Sterling, editor, *Proceedings of the 12th International Conference on Logic Programming*, pages 7–8, Cambridge, MA, USA, June13–18 1995. MIT Press.
4. M. Kerber and M. Kohlhase. A mechanization of strong Kleene logic for partial functions. SEKI-Report SR-93-20 (SFB), Universität des Saarlandes, Saarbrücken, 1993.
5. K. Konrad and D. A. Wolfram. Finite model generation for many-valued higher-order logics. Forthcoming, 1999.
6. Programming Systems Lab Saarbrücken, 1998. Oz Webpage: <http://www.ps.uni-sb.de/oz/>.
7. J. Slaney. FINDER (Finite Domain Enumerator): Notes and guide. Technical Report TR-ARP-1/92, Australian National University Automated Reasoning Project, Canberra, 1992.
8. C. Suttner and G. Sutcliffe. The TPTP problem library (TPTP v2.2.0). Technical Report 97-04, Department of Computer Science, James Cook University, Townsville, Australia, 1998.