**SWORD v1.0**

Jean Christoph Jung  Andre Sülflow  Robert Wille  Rolf Drechsler

Institute of Computer Science
University of Bremen
28359 Bremen, Germany
{jeanjung,suelflow,rwille,drechsle}@informatik.uni-bremen.de

**Introduction**

In this paper, we describe SWORD – a decision procedure for bit-vector logic that uses SAT techniques and exploits word-level information [5]. The main idea of SWORD is based on the following observation: While current SAT solvers perform very well on instances with a large number of logic operations, their performance on arithmetic operations degrades with increasing data-path width. In contrast, pure word-level approaches are able to handle arithmetic operations very fast, but suffer from irregularities in the word-level structure (e.g. bit slicing).

SWORD tries to combine the best of both worlds: On the one hand, it includes fast propagation, sophisticated data structures, as well as advanced techniques like non-chronological backtracking and learning from modern SAT solvers. On the other hand word-level information is exploited in the decision heuristic and during propagation. Additionally, rewriting on word-level and bit-level is performed before starting the search process.

**Implementation**

The overall architecture of SWORD is shown in Figure 1. At first, the instance is rewritten on word-level, i.e., rules for distributivity and commutativity are applied. Afterwards, the resulting (already simplified) instance is translated to an AIG-like data structure [3]. More precisely, a data structure not only supporting and-nodes but also iff-nodes is used for rewriting. The rules for and-nodes are adapted from [4] and have been extended for iff-nodes.

After rewriting, the bit-level data structure is converted into Conjunctive Normal Form (CNF) and given to the solve engine (see right side of Figure 1). Basically, the solve engine of SWORD is a DPLL style decision procedure as deployed in many state-of-the-art SAT solvers [2]: While free variables remain, a free variable is assigned, and its implications are propagated. If a conflict occurs, it is analyzed and a conflict clause is learnt.

SWORD extends the basic algorithm by so called “modules” (for more details see [5]). Modules can be instantiated for, principally, any sub-unit of the formula under consideration. However, currently supported modules are multiplication and addition. The motivation of modules is twofold: First, they are used for
propagation where a translation to CNF would be too expensive. Second, high level information is exploited inside modules, e.g., for making decisions.

Applying modules yields several new options. As an example, there is a choice between using the SAT solver’s decision heuristic and the decision heuristic of a module (as depicted on the right side of Figure 1). The decision heuristic of a module depends on the type of the module and uses module specific high level information. For example, the heuristic of the addition module differs from that of the multiplication module.

SWORD supports the QF_BV logic defined in [6] and is implemented in C++ on top of the SAT solver MiniSat [2]. The parse routine of the solver is based on the grammar of Smt2Sf [1]. SWORD will participate in the QF_BV division at the SMT competition 2009 using random seed 10659.

References