# SPEAR Theorem Prover

Domagoj Babić *
(Theorem prover architect)
University of British Columbia

Frank Hutter
(Search parameter optimization)
University of British Columbia

## Abstract

SPEAR is a bit-vector arithmetic theorem prover designed for proving software verification conditions. The core of the theorem prover is a fast and simple SAT solver, which is described in this paper.

***Keywords*** Theorem proving, boolean satisfiability, parameter optimization, modular arithmetic, bit-vector arithmetic, machine arithmetic

## 1. Introduction

SPEAR is a theorem prover for bit-vector arithmetic, designed for software verification, but is also fast on other industrial problems, like bounded hardware modelchecking. When given bit-vector arithmetic constraints, SPEAR performs elaborate encoding and optimization of constraints. Together with structural information, the encoded formula is passed to the core SAT solver. Given CNF input, SPEAR acts like an ordinary SAT solver, and does not attempt to reconstruct structural information, which is typically lost when the industrial instances are encoded into CNF.

The following sections describe the main features of SPEAR architecture, and parameter optimization.

## 2. Architecture

The core of SPEAR is a custom-made DPLL SAT solver. Problems coming from the software verification domain frequently require fast path enumeration and fast refutation of infeasible paths. As SAT solvers are very effective tools for both enumeration and refutation, the decision to base the bit-vector arithmetic decision procedure on a SAT solver came naturally. Custom-made SAT solver offers significantly more opportunities for application-specific optimization than off-the-shelf SAT solvers.

SPEAR features highly optimized boolean constraint propagation (BCP), very similar to the BCP routine in HYPERSAT [1]. Several other features were borrowed from HYPERSAT: phase selection heuristic and algorithm for finding the next watched literal. Clause representation is similar as well. A number of other features was modelled after Minisat [3]: frequent restarts and learned clause minimization. The implementation of the clause minimization was improved in several ways. For instance, Minisat uses stack-based work queue for clause minimization, while SPEAR uses a FIFO, which has a predictable memory access pattern, and is easier to optimize.

A number of various phase and variable decision heuristics has been implemented in SPEAR. A simple phase selection heuristic that always picks `false` phase first for each decision literal tends to perform well on instances generated from circuits. However, we found that the HYPERSAT phase selection heuristic performs much better in general. Depending on the average length of implication chains, HYPERSAT picks either the phase with more or less enqueued clauses on watched lists. If implication chains are long, implying the phase that results in more unsatisfied clauses increases the likelihood of running into a conflict, effectively decreasing the average length of implication chains. If the chains are short (more frequent case for industrial benchmarks), picking the phase that satisfies more watched clauses tends to reduce the total amount of computation. Since the second case is more frequent, that is the default phase selection heuristic in SPEAR.

SPEAR is very configurable. Almost all search parameters (roughly 25 parameters) are modifiable from the command line. Besides setting individual parameters, SPEAR also supports predefined parameter sets for specific problems. This is an important feature, because various combinations of parameters can have drastic effects on the runtimes. For instance, even with a very lightweight application-specific parameter optimization, we observed 500 X performance improvement on software verification instances over the best manually optimized parameter configuration (manually optimized for HW BMC), and over 100 X over manually optimized Minisat [4]. The next section presents parameter optimization in more detail.

## 3. Parameter Optimization

Determining appropriate values for an algorithm's free parameters is a challenging and cumbersome task in the design of effective algorithms for hard problems. It is, however, well worth the effort since good parameter settings often make the difference between solving a problem in seconds and solving it in hours (or not at all).

We believe that for complex parameter tuning tasks automatic (or semi-automatic) approaches can outperform manual approaches while at the same time considerably reducing the time algorithm designers need to spend for tuning their algorithms. In [4], we demonstrated that this was indeed the case when tuning SPEAR´s parameters for solving certain classes of industrial bounded model checking instances as well as software verification instances: automatic tuning resulted in speedup factors of 4.5 for bounded model checking and 500 for software verification.

We attribute the potential of automatic algorithm configuration to the fact that during development algorithm designers typically only track performance on a few instances, limiting expensive batch experiments to infrequent intervals. This bears the risk of "over-tuning" performance to the used instances with poor generalization to other instances, even ones with very similar characteristics [2, 5]. Further, humans tend to focus on single algorithm components instead of grasping the complex interplay of all components taken together.

Automatic tools for parameter optimization also pave the way to an automatic algorithm design, viewed as the combination of alternative building blocks. For example, two tree search algorithms that only differ in their preprocessing and variable heuristics can be

* His research is supported by a Microsoft Graduate Fellowship.

seen as a single algorithm with two nominal parameters. Thus, constructing the best algorithm for a domain can be seen as a parameter optimization problem.

SPEAR is an excellent testbed for automatic parameter optimization for the following reasons:

- It has a large number of parameters of various types. Its 25 parameters include categorical choices between heuristics, nominal parameters, as well as integer and continuous parameters.
- It shows state-of-the-art performance for a practically relevant class of problem instances, and tuning it will thus be of high practical relevance. In particular, in our experimental analysis SPEAR consistently showed the best results for solving software verification instances.

The method we used for parameter optimization is the same we used in [4]. It is called ParamILS and views parameter tuning as an optimisation problem [5]. In a nutshell, it performs an iterated local search in parameter configuration space, computing the objective function to be maximized as the geometric mean speedup over the default parameters. Since the optimization objective was good performance in the SAT Race, and instances were announced to be similar to the competitions from previous years, we used the union of the SAT Race 2006 instances and the instances from the industrial track of the SAT Competition 2007 as a training benchmark set.

From these instances, we omitted those that were not solved by any of the following solvers within 900 seconds on our machines:[1] Eureka, MXC, Minisat07, Picosat, Rsat2.0, Tinisat, SPEAR with parameters `fh1.0` from the SAT Competition 2007, and SPEAR with parameters setting for bounded model checking and software verification from [4]. This process left 333 instances for tuning. As the cutoff time for a single run during tuning we used 900 CPU seconds.

We used a combination of BasicILS(333) and FocusedILS for tuning: initially we found a good parameter configuration with 20 separate runs of FocusedILS, and then improved it with BasicILS (which is easier to parallelize on a computer cluster). This lead to parameter configuration `fh1.3`, which we used for the qualification round. After the qualification we performed another round of tuning with 20 separate runs of FocusedILS, which yielded the improved parameter configuration `fh1.4`.

Figure 1 compares the performance of parameter configuration `fh1.0` (used in our submission to the SAT Competition 2007) and `fh1.4` (used in this year's submission). Notice that `fh1.4` reduced the number of time-outs from 101 to 61 instances. This is very significant as can be seen by a comparison with other state-of-the-art industrial SAT solvers: in our experiments the solvers Spear (`fh1.4`), PicoSAT, Spear (`fh1.3`), Eureka, Rsat2.0, MiniSAT07, Spear (`fh1.0`), TiniSAT, and MXC timed out on 66, 73, 81, 88, 91, 100, 101, 114, and 120 instances. Although Spear(`fh1.4`) performed best in these experiments, this by no means implies that it should win the SAT Race: our experiments only used last year's solvers, and were run on a different architecture than the competition. Furthermore, the set of instances will likely be somewhat different than the one we used, which will compromise the optimality of our chosen parameter setting.

## 4. Future Work

SPEAR is a bit-vector arithmetic decision procedure, and the custom-made SAT solver is only one of its components. Although

[1] Implementations of these solvers were obtained from the SAT Competition 2007 website. Experiments were performed on a cluster of 55 dual 3.2GHz Intel Xeon PCs with 2MB cache and 2GB RAM, running OpenSuSE Linux 10.1.

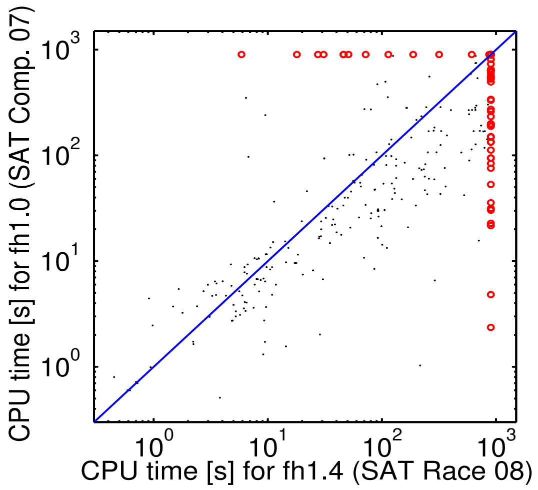


**Figure 1.** Performance of old SPEAR parameters tuned for the SAT Competition 2007 (`fh1.0`) vs. new SPEAR parameters tuned for the SAT Race 2008 (`fh1.4`); every dot represents one instance, every red circle a time-out at 900 CPU second. Setting `fh1.0` timed out on 101/333 instances, while setting `fh1.4` only timed out on 66/333 instances; on instances solved with both parameter settings, the average runtimes are 158s (`fh1.0`) and 91s (`fh1.4`).

SPEAR's expression simplifier could be significantly improved, it has become very hard to improve SPEAR's overall performance without the application-specific tuning. The Satisfiability Modulo Theories (SMT) approach offers much more flexibility and research opportunities than a bit-vector arithmetic decision procedure based on a SAT solver. Hence, future research, if there will be any, will be focused on transforming SPEAR into a full-blown SMT prover and researching practical approaches to solving bit-vector arithmetic constraints within the SMT framework.

On the parameter optimization side, we are evaluating the use of model-based approaches that would not only yield a well-performing parameter configuration, but also provide information about the importance of each parameter, the interaction of parameters, and interactions between search parameters and instance features.

## References

[1] Domagoj Babic, Jesse Bingham, and Alan J. Hu. B-cubing: New possibilities for efficient sat-solving. *IEEE Trans. Comput.*, 55(11):1315–1324, 2006.

[2] M. Birattari. *The Problem of Tuning Metaheuristics as seen from a Machine Learning perspective*. PhD thesis, Universite Libre de Bruxelles, Facult'e des Sciences Appliqu'ees, IRIDIA, Institut de Recherches Interdisciplinaires et de D'eveloppements en Intelligence Artificielle, 2005.

[3] Niklas Eén and Armin Biere. Effective preprocessing in sat through variable and clause elimination. In *SAT*, volume 3569 of *Lecture Notes in Computer Science*, pages 61–75. Springer, 2005.

[4] F. Hutter, D. Babić, H. H. Hoos, and A. J.Hu. Boosting verification by automatic tuning of decision procedures. In *Formal Methods in Computer Aided Design (FMCAD'07)*, 2007.

[5] F. Hutter, H. H. Hoos, and T. Stützle. Automatic algorithm configuration based on local search. In *Proc. of the Twenty-Second Conference on Artifical Intelligence (AAAI '07)*, pages 1152–1157, 2007.