Short User Manual for Bound Propagation in Vampire

Ioan Dragan Laura Kovács Andrei Voronkov
Konstantin Korovin

1 Installation

In order to use vampire with bound propagation one simply needs to download the binary and run it from command line. The binary found on the webpage `http://www.complang.tuwien.ac.at/ioan/boundPropagation` is compiled to run on 64bit linux machines. If you need the binaries compiled for a different operating system or for a different architecture, please let us know by sending an email to ioan.dragan@tuwien.ac.at requesting it.

Prerequisites. One has to make sure that it also has the GNU Multiple Precision Arithmetic Library correctly installed and configured. Instructions on how to download, compile and install the GNU MP library can be found at `http://gmplib.org/`. Uppon request it is possible to statically link the library into the binaries.

2 Usage

In the current version of the tool we have implemented a variety of strategies for picking the next variable to be assigned. All this strategies can be specified using the flag `--bp_variable_selector`. Details about the values this option takes can be found in 2.

Concerning the assignment selection, we implemented some strategies for picking a good value. In order to specify a different assignment selector than the default one, you have to use the `--bp_assignment_selector` flag with the options presented in 2.

Besides the choice of assignment and variable selectors, we have implemented more options. Due to the implementation of these strategies, we can try to figure out good values for the default parameters. Up to now we have tested around 3000 strategies.

In the following we present a summary of available options and possible values for them.

`--mode` ***bpa*** - this specifies the mode to be bound propagation

`--input_syntax` with either

smtlib problems must be in smtlib 1.2 format

smtlib2 problems must be in smtlib 2.0 format

mps problems from MIPLIB

`-t --time_limit` integer value, specifies the timeout for solving the current problem. If no value is specified, the default timeout is 60 seconds.

`--bp_start_with_precise` : values ***on, off***. Default value is off.

`--bp_assignment_selector` The option specifies which assignment selector to be used. We have implemented some strategies and the values for each of them are:

alternating : This alternates in picking upper bound with picking lower bound

bmp : This tries to find a nicely represented value in the interval formed by floor(lowerBound) and ceil(upperBound) which is still in the original interval.

lower_bound : Always pick the lower bound

middle : Pick a value in the middle of the interval. (lower+upper/2)

random : Pick a random value in the interval.

rational : This assignment selector is based on the continue fraction decomposition algorithm.

smallest : Pick the smallest value possible in the interval

tight : Try to pick a number close to either upper or lower bound. This is also set as default value for this option.

tightish : Similar to the tight one, but the "close" value is bigger

upper_bound : Always pick the upper bound.

`--bp_conflict_selection` Can take the following values:

least_recent Pick the least recent conflict.

most_recent Pick the most recent conflict.

shortest Set as default value for the conflict selection.

`--bp_conservative_assignment_selection` : This option enables the conservative behavior of the assignment selector. Meaning that we keep track of the previous value we picked and if it is still fits the bounds, then we keep it. The option is used in combination with `-as` option. Values for this options are ***on, off***. By default this option is turned *off*.

`--bp_add_collapsing_inequalities` : This option enables the usage of collapsing constraints in bound propagation. Values for this option are **on, off**. By default this option is set to off.

`--bp_variable_selector` : This option takes care of the variable selector of choice. Values for variable selector are:

> ***conflicting*** counts how many times a variable appears in a conflict and picks the one which appears more often
>
> ***conflicting_and_collapsing*** - counts both the number of conflicts in which a variable appears and also in how many collapsing inequalities it appears. Again pick the best one.
>
> ***first*** - pick the first variable which is eligible for being bounded
>
> ***look_ahead***
>
> ***random*** picks a random eligible variable from the variables which are to be bounded. Default value for this option.
>
> ***recent_collapsing*** - pick a variable from the recent collapsing ineq
>
> ***recent_conflicting*** - pick a variable which proved to generate conflict
>
> ***tightest_bound*** - pick the variable with the tightest bounds

`--bp_almost_half_bounding_removal` With values: ***bounds_on, off, on***

`--bp_fm_elimination` - Fourier-Motzkin elimination, with values ***on, off***

`--bp_fm_balance` - integer, specifing when to stop eliminating variables

`--bp_max_prop_length` - integer value, maximal propagated equality length

`--bp_bound_imporvement_limit` - integer value, specifies the max number of bounds which have been infered from the current constraint

3 Experiments

We conducted a series of experiments using this implementation of bound propagation. We focused on three main categories of problems: (i) problems generated using Gorrila [4] (ii) hard problems extracted from SMTLIB benchmarks [1] using Hard Reality [4] and (iii) problems taken from MIPLIB benchmarks [3].

For the SMTLIB and MIPLIB benchmarks we tried out aprox. 3000 of strategies for solving the problems. Taking into account all the strategies, we managed to solve 49 out of 128 problems from SMTLIB. Concerning the MIPLIB, we managed to solve 107 problems out of a total of 224. Compared for example with yices [2] , which solved 86 problems from the 128 SMTLIB problems, and 127 out of the 224 problems taken from MIPLIB.

4 Experiments

If you are interested in more details about the experiments or you are in need of examples for combination of strategies for solving problems, we can provide them to you.

Solver	Sat	Avg. Time	Unsat	Avg. Time	Unknown
Vampire	2797	0.3624	17100	0.2362	1576
Z3	3193	0.0727	18205	0.1446	75
Yices	3206	0.001	18267	0.0018	0

Table 1: Some experiments on problems generated with GoRRiLa.

Solver	Sat	Avg. Time	Unsat	Avg. Time	Unknown
Vampire	79	6.43	28	4.54	117
Z3	96	6.20	25	2.13	103
Yices	103	3.44	26	0.31	95

Table 2: Experiments on MIPLIB problems.

References

[1] C. Barrett, A. Stump, and C. Tinelli. The Satisfiability Modulo Theories Library (SMT-LIB). `www.SMT-LIB.org`, 2010.

[2] B. Dutertre and L. de Moura. A Fast Linear-Arithmetic Solver for DPLL(T). In *Proc. of CAV*, pages 81–94, 2006.

[3] T. Koch, T. Achterberg, E. Andersen, O. Bastert, T. Berthold, R. E. Bixby, E. Danna, G. Gamrath, A. M. Gleixner, S. Heinz, A. Lodi, H. Mittelmann, T. Ralphs, D. Salvagnin, D. E. Steffy, and K. Wolter. MIPLIB 2010. *Mathematical Programming Computation*, 3(2):103–163, 2011.

[4] K. Korovin and A. Voronkov. GoRRiLA and Hard Reality. In *Proc. of Ershov Memorial Conference (PSI)*, pages 243–250, 2011.

Solver	Sat	Avg. Time	Unsat	Avg. Time	Unknown
Vampire	33	5.23	18	3.78	77
Z3	76	2.89	22	3.40	30
Yices	85	9.90	26	6.02	17

Table 3: Experiments on the SMTLIB (Hard Reality) examples.