A branching heuristic in CVC4 SMT Solver

Kshitij Bansal

December 11, 2012

Abstract

This article summarizes work done with Clark Barrett on branching heuristics in CVC4. In particular, it explores an approach to use a non-clausal solver in conjunction with the clausal solver in an SMT solver. It is intended to document and explain an algorithm implemented in CVC4[1].

Introduction. At the heart of an SMT solver is a SAT solver which works with the Boolean structure of the formula. Modern SAT solvers typically work with clausal (CNF) formulas using an approach termed *conflict driven clause learning* [2]. In essence, the solvers follow a DPLL-style approach which involves guessing truth value of literals till a *conflict* is found. From the conflict, they extract and *learn* the reason of the conflict thus cutting down search space. We will present a strategy to pick the literals to branch upon, and related experimental results.

SMT solvers work with arbitrary formulas (not necessarily clauses) and even though non-clausal SAT solvers exist and could in principle be employed, because of the better performance of modern clausal solvers the formulas are translated into a set of clauses (see, for instance, Tseitin-Transformation [3]). Nevertheless, there are instances where a clausal solver will make non-optimal choices because it doesn't see the original propositional structure. We demonstrate this with the help of an example.

Say, the original formula, ϕ, has the propositional structure represented as a tree as in Figure 1. Each letter denotes the Boolean variable corresponding to the node in the clausal SAT solver. Further assume that d corresponds to an atomic proposition for which the SAT solver has guessed the value true. From this information it can be inferred that b must be true, but even further the assignments to nodes in the subtree rooted at e will not affect the satisfiability of the formula. The latter detail cannot be inferred by the clausal SAT solver, and it might very well decide to branch on g next. With this observation in mind, we propose a way of running a non-clausal solver in conjunction with a clausal solver aiming to eliminate poor decisions such as these and cutting down the search space.

[1] The implementation as of writing this document may be found in `https://github.com/CVC4/CVC4/blob/f056522a587d1b080224992355be070b73d97a3b/src/decision/justification_heuristic.h` and the corresponding cpp file.

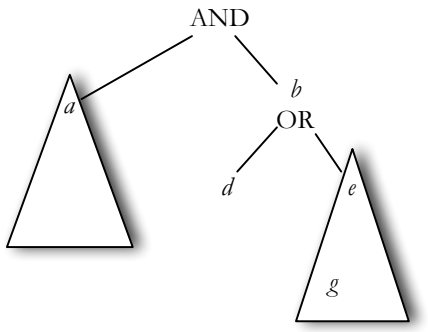

Figure 1: Propositional structure on a candidate formula ϕ, input to an SMT solver.

Justification Heuristic. Let N denote the nodes of the formula tree, with the leaf nodes corresponding to atomic formulas. We abuse notation and also use N to denote the corresponding variables in the SAT solver. A (partial) *valuation* of these variables is a (partial) map assigning truth values, $\gamma : N \rightharpoonup \{\mathsf{true}, \mathsf{false}\}$. We refer to the *domain* of a valuation γ by $\mathrm{dom}(\gamma)$. If $v \in \mathrm{dom}(\gamma)$ we say it *has a valuation*. By *valuation* of node v we refer to $\gamma(v)$.

Definition 1. *Given a formula tree with nodes N and a valuation γ, we define a node v to be* justified *inductively:*

- *For a leaf node v, we say a node is justified if it has a valuation, that is $v \in \mathrm{dom}(N)$.*
- *For a non-leaf node, we require that v has a valuation, and further:*
  - *if v is an AND node and $\gamma(v)$ is* true*, each child has a valuation* true *and is justified,*
  - *if v is an AND node and $\gamma(v)$ is* false*, at least one child has a valuation* false *and is justified,*
  - *if v is an OR node and $\gamma(v)$ is* true*, at least one child has a valuation* true *and is justified,*
  - *if v is an OR node and $\gamma(v)$ is* false*, each child has a valuation* false *and is justified,*
  - *if v is a NOT node, the child has an opposite valuation and is justified,*
  - *if v is an IF-THEN-ELSE node, the IF node is justified and also:*
    - *if the valuation of the IF node is* true*, the THEN node is justified,*
    - *if the valuation of the IF node is* false*, the ELSE node is justified,*

The branching heuristic takes as input the non-clausal formula, and the partial valuation of the nodes in the SAT solver and tries to *justify* the root node to be true. It does this by doing a depth-first search, and inductively trying to justify a node. If it succeeds in justifying the root, we can conclude that the problem is satisfiable. If not, the algorithm returns the first leaf node that cannot be justified as the next literal to branch on.

We note that though we presented the above algorithm to work with formula as a tree, in practice, solvers work with DAGs. The ideas remain the same and the algorithm can be easily adapted.

Evaluation. We implemented the above algorithm in the CVC4 [1] SMT Solver. At the time of writing this document, this specific heuristic could be forced in the solver by specifying `--decision=justification` command line argument. We also tested a variant of this heuristic wherein the clausal SAT solver makes use of the non-clausal solver only to determine if root node can be justified to true and not to pick branching literals. This mode may be enabled using `--decision=justification-stoponly` in the CVC4 solver.

We evaluated our branching strategies on various subsets of the SMTLIB benchmark suite. In case of theory of bit-vectors, which tend to be heavy in propositional structure, we found the algorithm to be very helpful[2]. See Figure 2. The tests were done on machines with Intel Core 2 processors and 2 GB memory. The variant *justification-stoponly*, led to slight improvements in performance on some quantifier free linear real arithmetic benchmarks[3].

Future work. Our work motivates further exploration of branching heuristics in the context of SMT solving, especially determining which strategies work well with what type of formulas and why.

References

[1] Clark Barrett, Christopher L. Conway, Morgan Deters, Liana Hadarean, Dejan Jovanović, Tim King, Andrew Reynolds, and Cesare Tinelli. Cvc4. In *Proceedings of the 23rd international conference on Computer aided verification*, CAV'11, pages 171–177, Berlin, Heidelberg, 2011. Springer-Verlag.

[2] Joao P. Marques-Silva, Ines Lynce, and Sharad Malik. *Conflict-Driven Clause Learning SAT Solvers*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, chapter 4, pages 131–153. IOS Press, February 2009.

[3] G. S. Tseitin. On the complexity of derivations in the propositional calculus. *Studies in Mathematics and Mathematical Logic*, Part II:115–125, 1968.

[2] `http://cvc4.cs.nyu.edu/regress-results/compare_jobs.php?job_id=4498&reference_id=4499`

[3] `http://cvc4.cs.nyu.edu/regress-results/compare_jobs.php?job_id=4587&reference_id=4588`

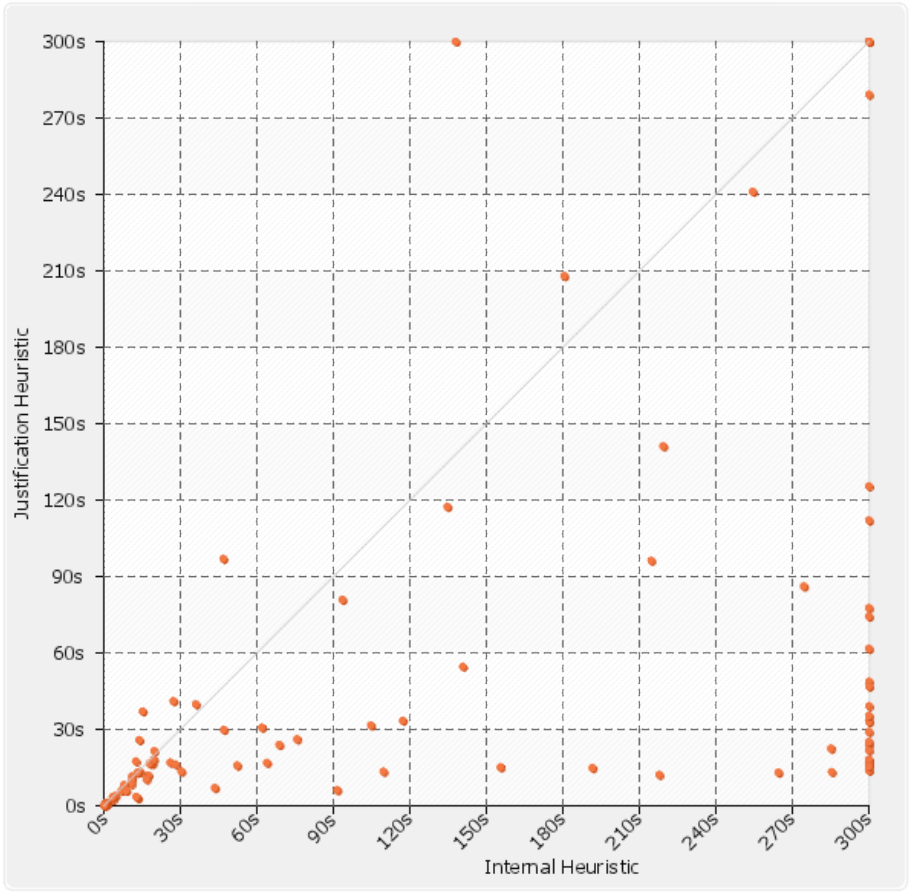

Figure 2: Comparison of two kinds of branching heuristics implemented in CVC4 on SMTLIB2 bit-vector benchmarks as an XY-scatter plot. The X-axis values are running times when the clausal SAT solver uses internal heuristics to branch and the Y-axis when it relies on an external non-clausal solver aware of the full propositional structure.