# On Variable Orderings in MCSAT for Non-linear Real Arithmetic (extended abstract)

Jasper Nalbach
RWTH Aachen University
jasper.nalbach@rwth-aachen.de

Gereon Kremer
RWTH Aachen University
gereon.kremer@cs.rwth-aachen.de

Erika Ábrahám
RWTH Aachen University
abraham@cs.rwth-aachen.de

**Abstract**

*Satisfiability-modulo-theories* (*SMT*) solving is a technique for checking the satisfiability of logical formulas. In this context, recently a framework called *model-constructing satisfiability calculus* (*MCSAT*) has been introduced which relaxes some design restrictions of the classical SMT setting and allows more freedom to construct an efficient interplay between the search for models in the Boolean and the theory domains. In this paper we discuss issues of *dynamic variable orderings* to drive the MCSAT model search for the theory of *non-linear real arithmetic*.

## 1 Problem Statement

*Satisfiability-modulo-theories (SMT)* solving is a technique for checking the satisfiability of (usually quantifier-free) first-order logic formulas. In classical lazy SMT solving, the search is decomposed in two interacting modules: at the Boolean level, truth values are searched for the theory constraints of an input formula such that the Boolean structure (the *Boolean abstraction*) of the formula is satisfied; this Boolean search is assisted by theory consistency checks for the corresponding constraints as selected by the current Boolean truth assignment.

Accordingly, a classical SMT solver consists of two components: a Boolean solver and a theory solver. The Boolean solver, which is typically a SAT solver based on techniques called DPLL [DLL62] and CDCL [MsS99], searches for a solution for the Boolean abstraction by *deciding* which truth values to try for the theory constraints, *propagating* already assigned Boolean values to derive implications, and *resolving* Boolean conflicts occurring during propagation. During this search, the SAT solver[1] passes theory constraints corresponding to the current (possibly partial) Boolean model to the theory solver. The theory solver checks the consistency of the given constraints in the theory, usually by constructing a model for the theory variables, or returns an explanation for the unsatisfiability in cases where no such assignment exists. The latter explanations are added to the original formula to exclude theory-inconsistent Boolean models from the further search. The search continues

In: J. Abbott, A. Griggio (eds.): Proceedings of the 4th SC-square Workshop, Bern, Switzerland, 10th July 2019, published at http://ceur-ws.org

[1]In the following, when we talk about a SAT solver then we mean one based on DPLL and CDCL.

until either both the SAT and theory solver have constructed a complete assignment for both the Boolean and theory variables or all Boolean assignments have been excluded.

Recently, a framework called *model-constructing satisfiability calculus* (*MCSAT*) was introduced [JdM12, dMJ13], which allows the simultaneous construction of Boolean and theory models, enabling more freedom for the design of the interplay between the search in the Boolean and in the theory domains. In this approach, the Boolean and theory modules from the classical framework are merged into a single solver that works on the Boolean structure and the theory simultaneously. This solver performs decisions not only on the Boolean (abstraction) variables, but also on the theory variables by making *theory decisions*. Throughout the procedure, the two models are kept *consistent*, meaning that the truth values of constraints at the Boolean level never contradict to their evaluation under the theory model. When the current partial theory model cannot be further extended to a complete and consistent theory model, *theory conflict resolution* is applied to generate an *explanation* in form of a lemma that can be learned to exclude this and "similar" inconsistent guesses from the further search.

In *quantifier-free non-linear real arithmetic*, the constraints are polynomial equalities or inequalities over the real domain. For this theory, we have implemented in our `SMT-RAT` solver [CKJ+15] different theory solver modules for classical SMT solving. Additionally, we have also adapted for MCSAT the *cylindrical algebraic decomposition method* (*CAD*), the *virtual substitution method* (*VS*) and the *Fourier-Motzkin variable elimination method* (*FM*) for generating explanations. For CAD, we employ the approach outlined in [JdM12] (commonly known as `nlsat`), where we support different projection operators, and alternatively implement an adaptation of the OneCell-CAD approach presented in [Bro13, BK15] for explanation generation as described in [Neu18]. The adaptation of VS is based on our presentation from [ÁNK17], where elimination is restricted to the unassigned variables. The Fourier-Motzkin variable elimination is adapted in the spirit of [JBdM13], enhanced with explicit support for equalities and weak inequalities.

It is well-known that the variable ordering is a crucial ingredient both in theory and practice for many solving techniques. For all of the above methods – SAT solving, FM, VS and CAD – the variable ordering can move the performance from the worst-case to "almost trivial" or vice-versa. At the same time finding the best (or at least a "good") variable ordering in a reasonable amount of time is essentially impossible in all cases. Thus heuristics are heavily used in practice to find reasonably good variables orderings quickly.

The original MCSAT approach for *non-linear real arithmetic* [JdM12] assumes a static ordering of theory variables. This static variable ordering guides also the Boolean search: the solver identifies those clauses that are *univariate* in the first unassigned variable in the static theory variable ordering under the current theory model (i.e., the first unassigned theory variable is the only unassigned one occurring in them) and performs Boolean propagations and Boolean decisions only with respect to those clauses. If all such clauses are satisfied at the Boolean level then the next theory decision is made, extending the theory model in agreement with the Boolean model (if possible). While we can use a form of *conflict-driven activity-based* heuristics (like VSIDS) for the Boolean variables, the static theory variable ordering heavily restricts the flexibility here.

The restriction to consider univariate clauses only was already lifted in [dMJ13] and the possibility to even include the theory variables in such an activity-based heuristic – essentially treat Boolean and theory variables equally in one combined variable ordering – was proposed [JBdM13]. Note however that [dMJ13] described the general MCSAT framework independently of a specific theory and [JBdM13] only discusses *linear real arithmetic* (combined with uninterpreted functions). Unfortunately, this does not directly transfer to *non-linear real arithmetic*: while *regular polynomial constraints* have a clear semantic meaning independently of the variable ordering, we need to consider *extended polynomial constraints* (as defined in [JdM12]) that pose additional challenges under changing variable orders.

In the remainder of this paper we discuss how to design MCSAT for non-linear real arithmetic with *dynamic* theory variable ordering, meaning that the ordering of any variables may change throughout the computation. This includes necessary algorithmic modifications, some considerations for the implementation and completeness issues. Finally, we report on some preliminary experiments. We assume the reader to be reasonably familiar with the MCSAT framework as presented in [JdM12, dMJ13] as well as SAT solving and classical SMT solving.

## 2 Variable orderings in MCSAT

The fundamental idea of MCSAT is to construct the Boolean model and theory model simultaneously. This allows the Boolean reasoning and the theory reasoning to complement (and possibly learn from) each other more flexibly than in traditional SMT solving. The *variable scheduler* is the crucial component driving this process, mainly determining how we interleave Boolean and theory reasoning. Given the importance of the variable ordering on practical performance, it is possible (even probable) that more elaborate computations in the scheduling heuristics pay off in terms of overall solving time.

When devising a variable scheduling heuristic we need to answer two questions: Firstly, how should we interleave (or prioritize) the Boolean and theory reasoning? Secondly, how should the variables (both Boolean and theory) be ordered? For Boolean variables solvers usually rely on conflict-driven activity-based heuristics like VSIDS while theory variables are mostly static (based on some heuristic) throughout the whole solving process.

Experience shows that constructing a *good* variable ordering upfront is hard. Even for a pure CAD (or VS) computation that is not embedded in some other framework (like traditional SMT or MCSAT) the results are mixed. The existing heuristics almost exclusively focus on syntactic properties (like variable degree or number of occurrences), and adversarial inputs seem to be neither hard to construct nor particularly contrived or artificial.

Similar to CDCL-style SAT solving – and following [JBdM13] – we therefore use a dynamic variable ordering for theory variables as well in the hope to learn an efficient variable ordering during solving.

## 2.1 MCSAT Extensions in `SMT-RAT`

Our implementation contains several changes compared to the descriptions from [JdM12, dMJ13, JBdM13] that influence which heuristics can be used and how we can implement them. Some are clearly beyond the previously published descriptions while others are possibly intended by the authors but have not been made explicit.

- Our core solver is based on a CDCL-style SAT solver (`MiniSAT`) and thus incorporates all common optimizations and heuristics that go beyond what is presented in [JdM12, dMJ13] for the Boolean reasoning.
- A set of *active literals* is maintained that contains those literals occurring in not-yet satisfied clauses. It is sufficient to decide only literals from this set to obtain a complete procedure.
- Before a Boolean decision is made, we check for feasibility together with the current theory model. That is, after deciding the constraint we do not run into an immediate theory conflict. If the constraint is not feasible, we insert its negation as a *lazy theory propagation.* If this propagation leads to a conflict later on, the explanation is generated on request. This is essentially one concrete way how to apply the T-Propagate rule from [dMJ13].
- Whenever a theory assignment is computed, we optionally apply substitution of the current partial model in the assigned literals, collect those that are linear after the substitution and use a Simplex-based solver to check their consistency. This may yield a satisfying model for the whole set of assigned literals, or allow to determine infeasibility earlier.
- We combine multiple *explanation backends* (similar to [JBdM13]) based on FM, VS, CAD as in [JdM12] and OneCell-CAD. Besides allowing to fall back to "more complete" explanation backends, the variable scheduler could also exploit a particular combination of backends by favouring constraints that can be handled by more efficient methods (for example linear constraints).

## 2.2 Heuristics

To evaluate the implementation and get a first feeling on the impact in practice, we consider several different variable orderings here:

**Boolean first.** All Boolean variables are decided first with an activity-based dynamic ordering before any theory decision is made. We use a static theory variable ordering based on features like maximum degrees or coefficients roughly following the "triangular ordering" from [EBDW14]. Many possibilities here are yet unexplored, for example other variable orderings proposed in [EBDW14] or [DSS04].

**Theory first.** Same as *Boolean first*, but theory decisions are made before any Boolean decision is made. Note that after deciding all theory variables, only Boolean variables not representing a theory constraint need to be decided.

**Univariate constraints.** Like before we use a static ordering for theory variables and a dynamic ordering for Boolean variables. For Boolean decisions we only consider variables that are univariate under the current theory model and perform a theory decision if none is left. This interleaves the Boolean and theory reasoning and, combined with *active literals tracking*, is very similar to `nlsat` from [JdM12].

**Uniform activies.** As in [JBdM13], the activity is tracked for both Boolean and theory variables in a conflict-driven manner (like VSIDS). For Boolean variables, those are the resolution variables, and for theory variables, those are the ones occurring in the corresponding theory constraints, counted only once per conflict. The unassigned variable (Boolean or theory) with highest activity is decided first.

**Random.** A random ordering over all variables (Boolean and theory) is fixed and decided in this order. This strategy is only used as a reference.

## 3 Issues with dynamic variable orderings

As already noted using a dynamic ordering for theory variables has some consequences that go well beyond what we know from regular SMT solving.

### 3.1 Handling extended polynomial constraints

Explanations from a CAD-based explanation backend may contain *extended polynomial constraints* of the form

$$y \sim \mathrm{root}_i(p(z, x_1, \ldots, x_n))$$

where $\sim \in \{<, >, =, \neq, \leq, \geq\}$, $y, z, x_1, \ldots, x_n$ are real-valued variables, $p$ is a polynomial with rational coefficients and variables $z, x_1, \ldots, x_n$. If $x_1, \ldots, x_n$ are assigned to values $\alpha(x_1), \ldots, \alpha(x_n)$, then $\mathrm{root}_i(p(z, x_1, \ldots, x_n))$ represents the $i$th zero of the univariate polynomial $p(z, \alpha(x_1), \ldots, \alpha(x_n))$ in $z$. Such a constraint evaluates to true if the $i$th root *exists* and $\alpha(y)$ compares to this root as indicated by $\sim$, and to false otherwise.

Assuming a fixed theory variable ordering, the way how such extended polynomial constraints are constructed within the explanation guarantees that for each of these constraints $x_1, \ldots, x_n$ are assigned before $y$ and thus the above semantics are well-defined. Changing the theory variable ordering may lead to a situation where $y$ is assigned while some $x_i \in \{x_1, \ldots, x_n\}$ is still unassigned if the respective constraint was generated under a different variable ordering. In some sense, we no longer have a *constraint on $y$* but a *constraint on $x_i$*. Since the semantics for such an expression is not well-defined, a consistent value for $x_i$ cannot be calculated making theory decisions impossible. Note that the above semantics could be extended, but actually using constraints under these extended semantics has proven to be extremely difficult – if not impossible – in practice.

We currently employ a rather simple solution: we disable (i.e. exclude from theory consistency checks) all constraints of the form $y \sim \mathrm{root}_i(p(z, x_1, \ldots, x_n))$ where some $x_1, \ldots, x_n$ is not yet assigned but $y$ has been chosen for a decision. We observe that constraints are not necessarily disabled if the ordering changes but only if the ordering becomes *incompatible* ($y$ moves before one of $x_1, \ldots, x_n$). Note that constraints can safely be re-enabled, whenever its ordering is compatible with the current one again: Then, they do not immediately evaluate to a value as at least $y$ is not assigned.

Note that after disabling a constraint, the regions excluded by it might be considered again. However, this does endanger termination: For any region and for each particular ordering, at most finitely many explanations are generated excluding this region. Additionally, any two explanations excluding a common region but generated under different incompatible orderings either are unequal or one of them is not generated (as then, no constraint would have been disabled); thus, no explanation is learned twice.

### 3.2 Completeness

While a SAT solving process advances with any Boolean conflict resolution, for MCSAT the explanation backends need to fulfill additional requirements for completeness. The completeness proof of MCSAT from [JdM12] requires that all constraints occurring in an explanation clause must come from some finite set of constraints that depends on the input formula; we then say that an explanation backend fulfils the *finite basis property.* This property holds for all the described explanation backends individually under a static theory variable ordering based on the following (inductive) argument: given a finite set of $k$-dimensional constraints, these explanation backends can construct only finitely many new constraints of dimension $k - 1$ or less and we can (conceptually easily) enumerate all of them. Based on this enhanced set, we can construct the constraints of dimension $k - 2$ or less, and so on, ultimately obtaining a finite basis.

This argument still holds if we combine multiple explanation backends using the same static variable ordering. Note that the variable ordering of these explanation backends is not identical by construction: while the ordering on the assigned variables is fixed by the MCSAT trail we can have multiple unassigned variables whose ordering is unspecified. Though explanations never contain unassigned variables, it is not immediately clear whether explanation backends that use different orderings for unassigned variables can be combined safely.

If the variable ordering is dynamic, we lose the finite basis property as we can see from the counterexample for Fourier-Motzkin and VS shown in Example 1 and we assume similar examples to exist for CAD. Though this technically does not prove incompleteness, we doubt that some weaker property exists so that the procedure still terminates in all cases with an infinite basis for explanations.

**Example 1** *For the set of constraints*

$$\{\underbrace{x_1 = 2}_{c_1},\ \underbrace{x_1 = 2x_2}_{c_2},\ \underbrace{x_2 = 2x_1}_{c_3}\}$$

*an infinite sequence of new constraints can be created by either FM or VS steps:*

- *eliminating* $x_1$ *in* $\{c_1, c_2\}$ *gives* $c_4 : x_2 = 1$,
- *eliminating* $x_2$ *in* $\{c_3, c_4\}$ *gives* $c_5 : 2x_1 = 1$,
- *eliminating* $x_1$ *in* $\{c_2, c_5\}$ *gives* $c_6 : 4x_2 = 1$,
- *eliminating* $x_2$ *in* $\{c_3, c_6\}$ *gives* $c_7 : 8x_1 = 1, \ldots$

We can assure completeness with a stronger version of what we described as *disabled constraints* in Section 3.1: we store the variable ordering that was used when some clause was constructed and disable the whole clause for decisions and propagations when the current variable ordering is incompatible. With the same argument as given in Section 3.1, the procedure remains complete as lemmas are not generated twice and we only have finitely many variable orderings.

However, our main goal in making the theory variable ordering dynamic is not to work on the core problem from different perspectives – as one could understand CDCL-style SAT solving – but rather find an advantageous ordering dynamically and eventually settle on this one. The (syntactic) finite basis property ensures that every conflict makes at least a certain amount of progress (by excluding one of finitely many regions). Ever-changing theory variable orderings can easily lead to non-termination in MCSAT as we have seen in Example 1 by (potentially) excluding infinitesimally small regions. Note however that we consider this a theoretical issue: if we ensure that the theory variable ordering becomes stable at some point, similar to how restarts are handled in CDCL-style SAT solving, we can assume this to be safe in practice.

## 4 Experimental Results

The relative performance of the heuristics we defined above on the SMT-LIB benchmark set for QF_NRA are shown in Table 1 and Figure 1. It should be noted that a large part of the SMT-LIB [BFT16] benchmark set for QF_NRA has no complex Boolean structure. Throughout the experiments, we used a combination of the FM, VS and Onecell-CAD explanation backends by calling them in this order and falling back to the next backend if one failed.

Table 1: Solved instances by variable ordering heuristics

| **Solver** | **SAT** | **UNSAT** | **overall** | |
|---|---|---|---|---|
| Random | 4533 | 4694 | 9227 | 80.3 % |
| Boolean first | 4532 | 4716 | 9248 | 80.5 % |
| Univariate | 4565 | 4721 | 9286 | 80.8 % |
| Univariate + active literals | 4644 | 4775 | 9419 | 82.0 % |
| Uniform activities | 4701 | 4774 | 9475 | 82.5 % |
| Theory first | 4698 | 4779 | 9477 | 82.5 % |

We observe that the *Boolean first* strategy works only slightly better than a random ordering and significantly worse than most other heuristics. *Univariate constraints* improves upon this, in particular if combined with active literals tracking which is close to the `nlsat` heuristic from [JdM12]. However, active literals tracking

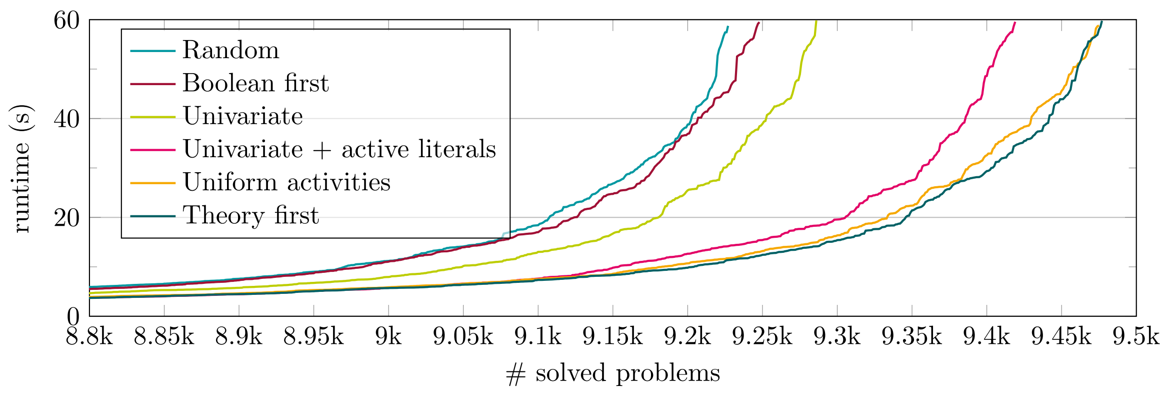

Figure 1: Performance of variable ordering heuristics

brings no benefit when combined with *Theory first* or *Uniform activities*. The *Uniform activities* and *Theory first* strategies perform best with *Theory first* having a slight advantage in terms of the average running time.

The success of both the *Theory first* and *Univariate + active literals* strategies may indicate that the `nlsat` strategy from [JdM12] is effective because Boolean decisions are delayed, and it actually delays them even more than the *Univariate + active literals* strategy: while we only consider univariate literals from not yet satisfied clauses, `nlsat` only considers literals from univariate clauses. We did not yet implement the `nlsat` strategy in our solver.

One may assume that the *Uniform activities* strategy performs well because it essentially converges towards the *Theory first* ordering. This is however not the (only) reason for its effectiveness. We experimented with ways to further increase the activities of theory variables compared to Boolean variables: increasing activities of theory variables multiple times per conflict; strictly favouring theory variables in the case of equal activities; using the *Theory first* strategy where the theory variables are ordered according to activities. All of those performed worse than just using uniform activities, suggesting that *Uniform activities* and *Theory first* are effective for different reasons.

## 5 Future work

We have seen that the variable ordering has a significant impact on the overall performance and might be the key to better running times in practice. We only had a first glimpse of the possibilities that result from dynamic variable orderings. It is not clear yet that lifting all restrictions are strictly beneficial, but the preliminary experiments we presented could be evidence for this.

We aim at investigating other variable ordering schemes that combine established dynamic heuristics from SAT solving with variable ordering heuristics borrowed from the computer algebra community. It should be noted however that experimenting with variable orderings should not be overly specialized by “overfitting” to the SMT-LIB benchmark set.

Dynamic variable orderings also give rise to theoretical questions about the completeness of MCSAT. While we presented evidence for non-termination (under extremely pessimistic assumptions about MCSAT), we conjecture that termination can be assured for reasonable (dynamic) variable orderings with arguments resembling the use of restarts in SAT solving.

## References

[ÁNK17] Erika Ábrahám, Jasper Nalbach, and Gereon Kremer. Embedding the virtual substitution method in the model constructing satisfiability calculus framework. In *Proc. of SCSC 2017*, volume 1974 of *CEUR Workshop Proceedings*, 2017.

[BFT16] Clark Barrett, Pascal Fontaine, and Cesare Tinelli. The Satisfiability Modulo Theories Library (SMT-LIB). www.SMT-LIB.org, 2016.

[BK15] Christopher W. Brown and Marek Košta. Constructing a single cell in cylindrical algebraic decomposition. *Journal of Symbolic Computation*, 70:14–48, 2015.

[Bro13] Christopher W. Brown. Constructing a single open cell in a cylindrical algebraic decomposition. In *Proc. of ISSAC 2013*, pages 133–140, 2013.

[CKJ$^+$15] Florian Corzilius, Gereon Kremer, Sebastian Junges, Stefan Schupp, and Erika Ábrahám. SMT-RAT: An open source C++ toolbox for strategic and parallel SMT solving. In *Proc. of SAT 2015*, volume 9340 of *LNCS*, pages 360–368, 2015.

[DLL62] Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397, 1962.

[dMJ13] Leonardo de Moura and Dejan Jovanović. A model-constructing satisfiability calculus. In *Proc. of VMCAI 2013*, pages 1–12, 2013.

[DSS04] Andreas Dolzmann, Andreas Seidl, and Thomas Sturm. Efficient projection orders for CAD. In *Proc. of ISSAC 2004*, pages 111–118, 2004.

[EBDW14] Matthew England, Russell Bradford, James H. Davenport, and David Wilson. Choosing a variable ordering for truth-table invariant cylindrical algebraic decomposition by incremental triangular decomposition. In *Proc. of ICMS 2014*. 2014.

[JBdM13] Dejan Jovanović, Clark Barrett, and Leonardo de Moura. The design and implementation of the model constructing satisfiability calculus. In *Proc. of FMCAD 2013*, pages 173–180, 2013.

[JdM12] Dejan Jovanović and Leonardo de Moura. Solving non-linear arithmetic. In *Proc. of IJCAR 2012*, pages 339–354, 2012.

[MsS99] Joaäo P. Marques-silva and Karem A. Sakallah. Grasp: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48:506–521, 1999.

[Neu18] Malte Neuß. Implementation of OneCell CAD for nonlinear explanations. Master's thesis, RWTH Aachen University, 2018.