NRA-LS at the SMT Competition 2023

Minghao Liu*[1,3], Kunhang Lv*[4], Fuqi Jia[1,3], Rui Han[1,3], Yu Zhang[2,3], Pei Huang[5], Feifei Ma[1,2,3], and Jian Zhang[1,3]

[1] State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China
[2] Laboratory of Parallel Software and Computational Science, Institute of Software, Chinese Academy of Sciences, Beijing, China
[3] University of Chinese Academy of Sciences, Beijing, China
[4] School of EECS, Peking University, Beijing, China
[5] Stanford University, Stanford, USA
{liumh,maff}@ios.ac.cn

1 Introduction

Satisfiability modulo theory (SMT) solving for quantifier-free formulas in nonlinear real arithmetic (QF_NRA) is important in many applications. State-of-the-art SMT solvers have made great progress to solve this problem. However, the time and memory usage of them on some hard instances may be unacceptable, especially when high-order polynomials appear in the formula. NRA-LS is an SMT solver for QF_NRA theory, which can improve the performance on some high-order satisfiable instances through a local search (LS) algorithm. NRA-LS wraps cvc5-1.0.5[1] as the back-end solver.

2 Architecture of NRA-LS

The framework of NRA-LS is shown in Algorithm 1. At the beginning, the maximum order of polynomials in the formula is computed, and those formulas will be handled specially if they contain high-order polynomials, which means the order is larger than 10 in the implementation.

Initial model generation. NRA-LS tries to assign values to the variables, evaluates the level to which the assertions are satisfied, and adjusts the values. Then the top-k assignments are output as initial models. However, these 'models' cannot satisfy all the assertions in most cases, so NRA-LS makes fewer variables fixed and tests the satisfiability of a set of sub-formulas.

* The first two authors contributed equally to this work.
[1] https://github.com/cvc5/cvc5.

Algorithm 1 Framework of NRA-LS

Input: an SMT(QF_NRA) formula ϕ
Output: `sat/unsat/unknown`

```
1: if φ contains high-order polynomial then
2:    S_1, S_2, ..., S_k ← generate_init_model(φ);
3:    for i from 1 to k do
4:       while |S_i| ≠ 0 do
5:          S_i ← generate_partial_assignment(φ, S_i);
6:          res ← run_back_end_solver(φ ∧ S_i);
7:          if res = sat then
8:             return res;
9:          else if res = unsat then
10:            continue;
11:         else
12:            break;
13:         end if
14:      end while
15:   end for
16: end if
17: return run_back_end_solver(φ);
```

Sub-formulas Testing. Given an initial model, NRA-LS calls back-end solver to test if the model is valid by appending additional assertions to the original formula. If `unsat` is returned, NRA-LS will reduce the number of fixed variables, and test the new sub-formula iteratively until getting `sat` or the time limit is exceeded. If `sat` is returned, the original formula is also satisfiable.

Time slots assignment. NRA-LS assigns the time slots into three parts. Suppose the time limit to solve a single formula is T. First, it takes $5\%T$ to run back-end solver on the original formula, which aims to exclude easy benchmarks. Next, the time limit for each attempt that tests a sub-formula is set to $2.5\%T$. Finally, if the result cannot be determined, the rest of the time is assigned to run back-end solver on the original formula.

3 What's New in 2023

In this year, we have focused on strengthening the algorithm for generating the initial models and improving the construction strategy of sub-formulas. We are still striving to add more detailed descriptions, which would be available soon.

4 Project Website

For more information and resources of NRA-LS, please refer to our website:

https://github.com/minghao-liu/NRA-LS.