Research Work Presentation

Zhonghan Wang

February 2024

Institute of Software Chinese Academy

- SMT Solving based on Z3 (October 2021 - August 2023)
  - Implement strategy portfolio for nonlinear arithmetic (z3++, smt-comp 2022&2023 gold medal)
  - Design a new local search algorithm (z3-nra-ls, accepted in VMCAI'2024)
  - Clause level dynamic MCSat algorithm (currently working)
- Hybrid System Verification (February 2021 - June 2021)

NRA in Z3-Plus-Plus

Introduction

- polynomial (nonlinear)

$$p := x|c|p + p|p * p$$

- atom

$$a := b|p > 0|p < 0$$

- formula

$$f := a|\neg f|f \wedge f|f \vee f$$

SMT: Given a formula, find a complete assignment to satisfy.

Implementation of Z3 Plus Plus (z3pp)

z3-plus-plus.github.io

View My GitHub Profile

Z3++

Overview

Z3++ is a derived SMT solver based on Z3. It participates in the SMT-COMP 2022, and significantly improves Z3 on the following logics:

QF_IDL, QF_LIA, QF_BV, QF_NIA and QF_NRA

It is a project mainly developed in State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China.

Detailed description and source code are available at the github repository.

Contact

z3_plus_plus@outlook.com

Awards

At the FLoC Olympic Games, Z3++ won 2 gold medals (6 in total) for Biggest Lead Model Validation and Largest Contribution Model Validation.

People

Leader:

Shaowei Cai.

Hosted on GitHub Pages — Theme by orderedlist

https://z3-plus-plus.github.io/

z3pp file tree

```
yogurt-shadow@LAPTOP-PVNS2OMS MINGW64 /c/code/z3pp/src/nlsat
$ tree .
.
|-- CMakeLists.txt
|-- nlsat_assignment.h
|-- nlsat_clause.cpp
|-- nlsat_clause.h
|-- nlsat_evaluator.cpp
|-- nlsat_evaluator.h
|-- nlsat_explain.cpp
|-- nlsat_explain.h
|-- nlsat_interval_set.cpp
|-- nlsat_interval_set.h
|-- nlsat_justification.h
|-- nlsat_params.pyg
|-- nlsat_scoped_literal_vector.h
|-- nlsat_simple_checker.cpp
|-- nlsat_simple_checker.h
|-- nlsat_solver.cpp
|-- nlsat_solver.h
|-- nlsat_symmetry_checker.cpp
|-- nlsat_symmetry_checker.h
|-- nlsat_types.cpp
|-- nlsat_types.h
|-- nlsat_variable_ordering_strategy.cpp
|-- nlsat_variable_ordering_strategy.h
`-- tactic
    |-- CMakeLists.txt
    |-- goal2nlsat.cpp
    |-- goal2nlsat.h
    |-- nlsat_tactic.cpp
    |-- nlsat_tactic.h
    |-- qfnra_nlsat_tactic.cpp
    `-- qfnra_nlsat_tactic.h

1 directory, 30 files
```

File tree of z3 nlsat

Portfolio of Z3pp: variable ordering

- variable ordering of nlsat (nlsat_variable_ordering_strategy.cpp)
  - number of univariate polynomials
  - max degree of variable
  - BROWN: max degree, max degree of total terms, number of terms containing the variable
  - TRIANGULAR: max degree, max leading coefficient degree, sum of degree

Portfolio of Z3pp: Interval Constraint Propagation (nlsat_simple_checker.cpp)

- Target Instances: MBO - Methylene Blue Oscillator System
- Whether certain polynomial has a zero where all variables are positive.
- Example:

$$f := h1 > 0 \land h2 > 0 \land h3 > 0 \land h1^3 + 2h1h2 + h3^4 = 0$$

- Implementation:

$$\begin{aligned} 2h1 > 0 \rightarrow h1^3 > 0 \\ h1 > 0 \land h2 > 0 \rightarrow h1h2 > 0 \\ h3 > 0 \rightarrow h3^4 > 0 \end{aligned}$$

Portfolio of Z3pp: symmetry (nlsat_symmetry_checker.cpp)

Instance: Hong (fully symmetry)

Example 5. [12]

Hong_n

$$\exists x_1, \ldots, \exists x_n \sum_{i=1}^{n} x_i^2 < 1 \wedge \prod_{i=1}^{n} x_i > 1$$

Hong2_n

$$\exists x_1, \ldots, \exists x_n \sum_{i=1}^{n} x_i^2 < 2n \wedge \prod_{i=1}^{n} x_i > 1$$

Example 6. (**C**_n_r) Whether the distance between the ball $B_r(\bar{x})$ and the complement of $B_8(\bar{x})$ is less than $\frac{1}{1000}$?

$$\exists_{i=1}^{n} x_i, \exists_{i=1}^{n} y_i \sum_{i=1}^{n} x_i^2 < r \wedge \sum_{i=1}^{n} y_i^2 > 8^2 \wedge \sum_{i=1}^{n} (x_i - y_i)^2 < \frac{1}{1000^2}$$

Our solver LiMbs solves all the 21 examples shown in Table 1. LiMbs is faster than the other solvers on 15 examples. Only LiMbs can solve 9 of the examples within a reasonable time while other solvers either run time out or return unknown state. From this we can see that our algorithm has great potential in solving satisfiability of polynomial formulas, especially considering that our prototype solver is a small program with less than 1000 lines of codes. For Hong_n and Hong2_n, though our solver is much faster than Z3, CVC4 is the one that performs best. We note that the examples of Hong_n and Hong2_n are all symmetric. This reminds us it is worth exploiting symmetry to optimize our solver's performance.

Insert ordering clauses for variables: If x, y, z are symmetry, insert

$$x \leq y \leq z$$

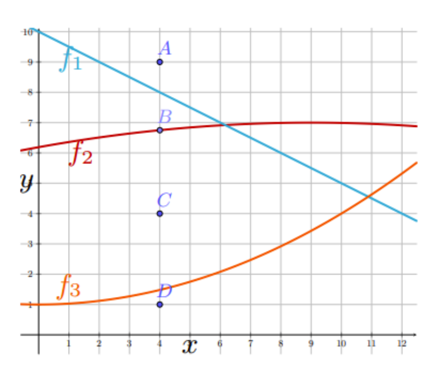
Portfolio of Z3pp: sample cell projection (nlsat_explain.cpp)

Definition 1. *Suppose $\bar{a}$ is a sample of $\bar{x}$ in $\mathbb{R}^n$ and $F = \{f_1, \ldots, f_r\}$ is a polynomial set in $\mathbb{Z}[\bar{x}]$ where $\bar{x} = (x_1, \ldots, x_n)$. The* sample-cell projection *of F on x_n at $\bar{a}$ is*

$$\mathtt{Proj}_{sc}(F, x_n, \bar{a}) = \bigcup_{f \in F} \mathtt{s\_coeff}(f, x_n, \bar{a}) \cup \bigcup_{f \in F} \{\mathtt{disc}(f, x_n)\} \cup \bigcup_{\substack{f \in F, g \in \\ \mathtt{s\_poly}(F, x_n, \bar{a}), \\ f \neq g}} \{\mathtt{res}(f, g, x_n)\}$$

- difference from McCallum's projection: calculate resultant only between sample polynomials
- sample polynomials: one or two polynomials whose root is the closest to the assignment point

Portfolio of Z3pp: sample polynomials



Demo for sample polynomial

Z3pp: competition result on QF_NRA (single query)

Sequential Performance

Solver	Error Score	Correct Score	CPU Time Score	Wall Time Score	Solved	Solved SAT	Solved UNSAT	Unsolved	Abstained	Timeout	Memout
Z3++-fixed[n]	0	2641	379531.82	379433.129	2641	1340	1301	267	0	265	0
2019-Par4[n]	0	2629	394912.029	356695.171	2629	1292	1337	279	0	221	58
cvc5	0	2545	525901.735	526314.738	2545	1244	1301	363	0	363	0
NRA-LS	0	2488	550489.833	551413.565	2488	1198	1290	420	0	5	0
Yices2	0	2341	702255.323	702324.97	2341	1150	1191	567	0	567	0
z3-4.8.17[n]	0	2275	666874.65	666955.286	2275	1229	1046	633	0	499	0
SMT-RAT-MCSAT 22.06	0	2189	895361.649	895423.466	2189	1123	1066	719	0	674	21
veriT+raSAT+Redlog	0	1879	1206512.928	1206107.221	1879	905	974	1029	0	989	0
MathSAT[n]	0	1544	1671561.013	1671677.835	1544	417	1127	1364	0	1364	0
Z3++	6	2634	379866.348	379759.488	2634	1333	1301	274	0	264	1

Parallel Performance

Solver	Error Score	Correct Score	CPU Time Score	Wall Time Score	Solved	Solved SAT	Solved UNSAT	Unsolved	Abstained	Timeout	Memout
2019-Par4[n]	0	2650	412116.989	346590.821	2650	1310	1340	258	0	200	58
Z3++-fixed[n]	0	2641	379553.38	379423.299	2641	1340	1301	267	0	265	0
cvc5	0	2545	526363.395	526298.488	2545	1244	1301	363	0	363	0
NRA-LS	0	2488	550607.043	551413.405	2488	1198	1290	420	0	5	0
Yices2	0	2341	702330.553	702302.83	2341	1150	1191	567	0	567	0
z3-4.8.17[n]	0	2275	666962.06	666934.046	2275	1229	1046	633	0	499	0
SMT-RAT-MCSAT 22.06	0	2189	895429.739	895399.226	2189	1123	1066	719	0	674	21
veriT+raSAT+Redlog	0	1879	1206582.328	1206082.811	1879	905	974	1029	0	989	0
MathSAT[n]	0	1544	1671701.033	1671625.855	1544	417	1127	1364	0	1364	0
Z3++	6	2634	379887.798	379749.928	2634	1333	1301	274	0	264	1

https://tools-comp.github.io/2022/results/qf-nonlinearrealarith-single-query

Dynamic Ordering of nlsat

- Using VSIDS and LRB branching heuristic in mcsat framework, instead of static ordering
- what means dynamic: decide branching variable using state information
- Using reverse order of assigned variables for cylindrical algebraic decomposition
- dynamic clause learning: remove useless clauses after each restart

solver	solved	unsat	sat	unsolved
z3_nlsat	10730	5546	5184	1404
dnlsat_v1	10883	5611	5272	1251
dnlsat_v2	10967	5612	5355	1167

Local Search Method

- use boundary score (cell score) data structure for local search
- incremental computation of arith variable score
- temporary relaxation of equality constraints

Category	#inst	Z3	CVC5	Yices	Ours
20161105-Sturm-MBO	120	0	0	0	**84**
20161105-Sturm-MGC	2	**2**	0	0	0
20170501-Heizmann	69	3	1	0	**6**
20180501-Economics-Mulligan	93	**93**	89	91	87
2019-ezsmt	63	**54**	51	52	18
20200911-Pine	245	**235**	201	**235**	224
20211101-Geogebra	112	**109**	91	99	100
20220314-Uncu	74	73	66	**74**	73
LassoRanker	684	155	**304**	122	284
UltimateAtomizer	48	**41**	34	39	26
hycomp	525	**311**	216	227	272
kissing	42	**33**	17	10	**33**
meti-tarski	4391	**4391**	4345	4369	4356
zankl	136	70	61	58	**99**
Total	6604	5570	5476	5376	**5662**

Future work (SAT View)

what previous work brings ?

- MCSAT brings assignment to arithmetic variables directly
- Local Search operates on arithmetic variables
- NRA solution space consists of CAD cells, like bool assignment for SAT

what future work changes ?

- hybrid solvers like SAT, cooperate local search and MCSAT in SMT
- operates on cells rather than assignment points or sample points (difficulty: heavy CAD against light ls)

Local Search for Nonlinear Arithmetic

SMT Solving

SMT-NRA helps in many areas

- Nonlinear hybrid automata
- Generating ranking function for termination analysis (LassoRanker Benchmark)
- Constraint Programming Solving
- Automatic or interactive theorem prover (Isabelle or Coq)
- Biological networks
- ......

Syntax of SMT(NRA)

- polynomial: $p ::= x \mid c \mid p + p \mid p - p \mid p \times p$
- atoms: $a ::= b \mid p = 0 \mid p > 0 \mid p < 0$
- formula: $f ::= a \mid \neg f \mid f \wedge f \mid f \vee f$

SMT: Determine whether the formula is satisfied by some assignment (local search focuses), or prove unsat

Example:
$x^2 + y^2 \leq 1 \wedge x + y < 1 \wedge x + z > 0$
assignment with $\{x \to 0, y \to 0, z \to 1\}$ satisfies all clauses.

Fragment of Local Search

Input : A set of clauses F
Output: An assignment of variables that satisfy F, or failure
Initialize assignment to variables;
while $\top$ **do**
    if all clauses satisfied **then**
        return success with assignment;
    end
    if time or step limit reached **then**
        return failure;
    end
    Critical move procedure.
end

Algorithm 1: Basic Fragment of Local Search

Fragment of Local Search

```
var, new_value, score ← best move according to make-break score;
if score > 0 then
    Perform move, assigning var to new_value;
end
else
    Update clause weight according to PAWS scheme;
    repeat
        cls ← random unsatisfied clause;
        var, new_value, score ← critical move making cls satisfied;
        if score ≠ −∞ then
            Perform move, assigning var to new_value;
        end
    until 3 times;
    if no move performed in previous loop then
        Change assignment of some variable in some unsatisfied clause;
    end
end
```

Local Search for SAT and SMT

Problem / LS	SAT	SMT
Operation (Move)	Flip	Critical Move
Score Definition	Weighted unsat clauses	
Score Computation	Cached score	No Caching, time costly

- What LS for SAT brings us:
  - Maintain scoring information after each iteration.
- Difficulty:
  - Predetermine critical move shift value.
- Our Solution
  - Introduce Scoring Boundaries

Infeasible Set

Definition

infeasible set of a clause c with respect to an assignment $asgn$ is the set of values that the variables in c can take under $asgn$ such that c is unsatisfied.

Example

Current assignment: $\{x \mapsto 1\}$
Calculate infeasible set for y:

- $x^2 + y^2 \leq 1 : (-\infty, 0) \cup (0, \infty)$.
- $x + y < 1 : [0, \infty)$.

If we choose values from infeasible set, the satisfied clause will be unsatisfied, which changes the whole score.

Make-break Intervals

Definition

make-break interval is a combination of (in)feasible intervals of arithmetic varaible x with respect to **all clauses**.

Example

Current assignment: $\{x \mapsto 1, y \mapsto 1, z \mapsto 1\}$
Calculate infeasible set for each clause.

- $x^2 + y^2 \leq 1$ (unsat): $(-\infty, 0) \cup (0, \infty)$.
- $x + y < 1$ (unsat): $[0, \infty)$.
- $x + z > 0$ (sat): $(-\infty, -1]$.

Combined information: x: $(-\infty, -1] \mapsto 0$, $(-1, 0) \mapsto 1$, $[0, 0] \mapsto 1$, $(0, \infty) \mapsto 0$.

Traditional Computation

Input : unsat clauses F
Output: Best critical move (variable, value)
foreach variable v in unsat clauses **do**
  foreach unsat clause c with v **do**
    Compute interval-score info of v in c.
  end
  Combine interval-score information.
  Update best var-value move.
end
return best critical move

Repeated computation:

- variable's (in)feasible set
- clause's sat staus

Boundary

Definition. A quadruple $\langle val, is\_open, is\_make, cid \rangle$, where val is a real number, is_open and is_make are boolean values, and cid is a clause identifier.

Meaning

- val : make-break value.
- is_open : active or not at val point.
- is_make : make or break, increase or decrease score.
- cid : causing clause.

Sorting: First ordered by val, then by is_open ($\perp < \top$).

Boundary

Current assignment: $\{x \mapsto 1, y \mapsto 1, z \mapsto 1\}$

- $x^2 + y^2 \leq 1$: starting score 0, boundary set $\{(0, \bot, \top, 1), (0, \top, \bot, 1)\}$, indicating no change for large negative values, make at boundary $[0, \cdots$, followed by break at boundary $(0, \cdots$.
- $x + y < 1$: starting score 1, boundary set $\{(0, \bot, \bot, 1)\}$, indicating make at large negative values, and break at boundary $[0, \ldots$.
- $x + z > 0$: starting score -1, boundary set $\{(-1, \top, \top, 1)\}$, indicating break at large negative values, and make at boundary $(-1, \ldots$.

sorted boundary set: $\{(-1, \top, \top, 1), (0, \bot, \top, 1), (0, \bot, \bot, 1), (0, \top, \bot, 1)\}$

Boundary Example

boundary set: $\{(-1,\top,\top,1),(0,\bot,\top,1),(0,\bot,\bot,1),(0,\top,\bot,1)\}$

Starting score: Score when x moves to -∞.
Maintain and Change: We maintain the boundary info for all arithmetic varaibles, unless the neighbour does a critical move.

Algorithm for computing boundary

Input : Variable v that is modified
Output: Make-break score for all variables

```
S ← {} ;                                          // set of updated variables
for clause cls that contains v do
    for variable v' appearing in cls do
        add v' to S;
        recompute starting score and boundary of v' with respect to cls;
    end
end
for variable v' in S do
    recompute best critical move and score in terms of boundary information;
end
```

Complexity of Values

Definition

We define a preorder $\prec_c$ on algebraic numbers as follows. $x \prec_c y$ if x is rational and y is irrational, or if both x and y are rational numbers, and the denominator of x is less than that of y. We write $x \sim_c y$ if neither $x \prec_c y$ nor $y \prec_c x$.

Previous work ignores equalities constraints, or only consider multi-linear (one-degree)examples.

Our Solution: Introducing relaxation, temporary enlarge the point irrational interval

Relaxation

Example

Given assignment $\{x \mapsto 1, y \mapsto 1\}$

$$z^3 \geq 5x^2 + y \vee z^3 \leq 3x + 3y$$

$$z^2 = x^2 + y^3$$

Both situations force z to an irrational number.

Relaxation

- If the constraint is of the form $p = 0$, it is relaxed into the pair of inequalities $p < \epsilon_p$ and $p > -\epsilon_p$.
- If the constraint is of the form $p \geq 0$, it is relaxed into $p > -\epsilon_p$. Likewise, if the constraint is of the form $p \leq 0$, it is relaxed into $p < \epsilon_p$.
- **Slacked var:** the var that is being assigned.

Restore

Input : slacked clauses
Output: succeed or not
for each slacked clause cls **do**
    $v \leftarrow$ slacked variable in cls;
    $accu\_val \leftarrow inf\_set(cls)$;
    move v to $accu\_val$;
end
for variable v' in slacked clauses **do**
    recompute best critical move and score in terms of boundary information;
end
return number of unsat clauses == 0

Local Search with Relaxation

```
Input  : A set of clauses F
Output: An assignment of variables that satisfy F, or failure
Initialize assignment to variables;
while ⊤ do
    if all clauses satisfied then
        success ← find exact solution;
        if success then
            return success with assignment;
        end
        else
            Restore relaxed constraints to original form;
            success ← find exact solution by limited local search;
            if success then
                return success with assignment;
            end
        end
    end
    if time or step limit reached then
        return failure;
    end
    Proceed traditional local search (slack).
end
```

Implementation Detail

code available at: `https://github.com/yogurt-shadow/LS_NRA`

Preprocessing

- Combine constraints $p \geq 0$ and $p \leq 0$ into equality $p = 0$.
- Eliminate variable x in an equation of the form $c \cdot x + q = 0$, where c is a constant and q is a polynomial with degree at most 1 and containing at most 2 variables.

Restart mechanism Two-level restart mechanism with two parameters $T_1 = 100$ and $T_2 = 100$.

- **Minor restart:** randomly change one of the variables in one of the unsatisfied clauses.
- **Major restart:** reset the value of all variables.

Overall Result

Category	#inst	Z3	cvc5	Yices	Ours	Unique
20161105-Sturm-MBO	120	0	0	0	**88**	88
20161105-Sturm-MGC	2	**2**	0	0	0	0
20170501-Heizmann	60	3	1	0	**8**	6
20180501-Economics-Mulligan	93	**93**	89	91	90	0
2019-ezsmt	61	**54**	51	52	19	0
20200911-Pine	237	**235**	201	**235**	224	0
20211101-Geogebra	112	**109**	91	99	101	0
20220314-Uncu	74	73	66	**74**	70	0
LassoRanker	351	155	**304**	122	272	13
UltimateAtomizer	48	**41**	34	39	27	2
hycomp	492	**311**	216	227	304	11
kissing	42	**33**	17	10	**33**	1
meti-tarski	4391	**4391**	4345	4369	4351	0
zankl	133	70	61	58	**100**	27
Total	6216	5570	5476	5376	**5687**	148

Scatter Plot

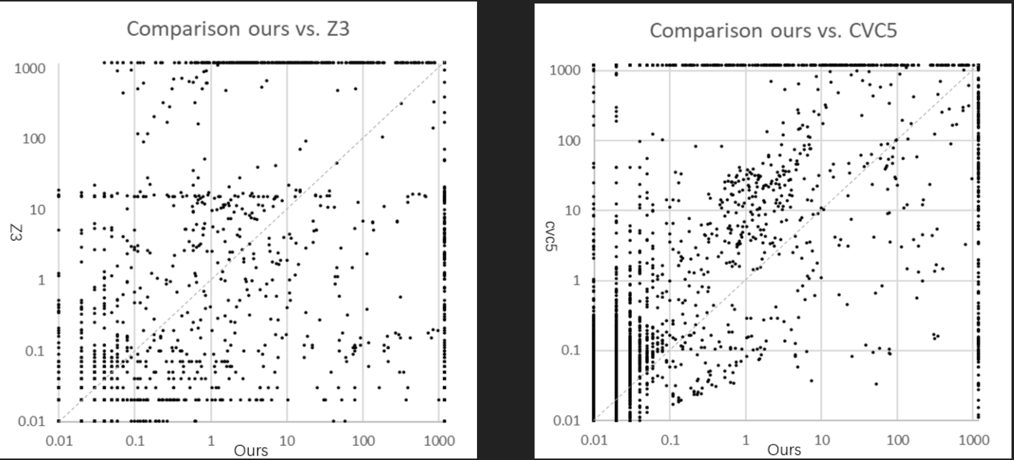


Figure 1: Scatter plots of running time vs. Z3 and cvc5.

Category	#inst	Incremental	Naive	Limit-45
20161105-Sturm-MBO	120	88	85	85
20161105-Sturm-MGC	2	0	0	0
20170501-Heizmann	60	8	5	5
20180501-Economics-Mulligan	93	90	89	89
2019-ezsmt	61	19	19	15
20200911-Pine	237	224	222	222
20211101-Geogebra	112	101	101	101
20220314-Uncu	74	70	70	70
LassoRanker	351	272	264	269
UltimateAtomizer	48	27	26	26
hycomp	492	304	298	298
kissing	42	33	32	33
meti-tarski	4391	4351	4352	4352
zankl	133	100	100	100
Total	6216	5687	5663	5665

Table 1: Comparison of incremental computation

Category	#inst	Relaxation	Threshold	NoOrder
20161105-Sturm-MBO	120	88	100	99
20161105-Sturm-MGC	2	0	0	0
20170501-Heizmann	60	8	9	3
20180501-Economics-Mulligan	93	90	89	86
2019-ezsmt	61	19	19	19
20200911-Pine	237	224	223	222
20211101-Geogebra	112	101	98	92
20220314-Uncu	74	70	70	70
LassoRanker	351	272	277	278
UltimateAtomizer	48	27	26	20
hycomp	492	304	211	164
kissing	42	33	31	27
meti-tarski	4391	4351	4353	4360
zankl	133	100	100	100
Total	6216	5687	5606	5540

Table 2: Comparison of temporary relaxation of constraints

Future Work

- Integrate into z3++ solver `https://z3-plus-plus.github.io/`
- Cacheing about cylindrical cells by CAD (we enter the same cell multiple times, how can we find that?)
- incorporate with other algorithms, like MCSAT or varaible substitution.
- used for nonlinear optimization

Application: Hybrid System Verification

Hybrid System

Hybrid systems refer to systems that have both continuous and discrete behaviors.

- Application
  - Transportation and spaceflight
  - Robots and medical devices
- Proving Method
  - Model Checking
  - Theorem Proving (KeymaeraX, HHLPY)
- Modeling Language about hybrid system
  - Dynamic differential logic (dL)
  - Hybrid Communication Sequential Process (HCSP)

Sequential fragment of HCSP

Hybrid CSP: an extension of Hoare's Communicating Sequential Processes to include continuous evolution, with modeling communicating processes running in parallel.
Commands in HCSP:

$$
\begin{aligned}
S, T \quad ::= \quad & \texttt{skip} \mid x := e \mid x := *\,(B) \mid S; T \mid \texttt{if}\ B\ \texttt{then}\ S\ \texttt{else}\ T \mid S \mathbin{++} T \mid S* \\
& \mid \langle \dot{\boldsymbol{x}} = \boldsymbol{e}\ \&\ D \rangle
\end{aligned}
$$

Proof rules based on invariants

Definition (Invariant Triple)

Let P and Q be predicates on the variables of an ODE $\dot{\boldsymbol{x}} = \boldsymbol{e}$. Let $\gamma : [0, T] \to \mathbb{R}^n$ be a solution of the ODE such that $\gamma(t)$ satisfies P for all $t \in [0, T]$ and such that $\gamma(0)$ satisfies Q. If for all such solutions γ, $\gamma(t)$ satisfies Q for all $t \in [0, T]$, then we say that Q is an invariant of ODE $\dot{\boldsymbol{x}} = \boldsymbol{e}$ under domain P, written as

$$[\![P]\!]\langle \dot{\boldsymbol{x}} = \boldsymbol{e} \rangle [\![Q]\!]$$

Sequential HCSP Program

The syntax for annotated sequential HCSP programs is:

$$
\begin{array}{rcl}
\mathcal{S}, \mathcal{T} & ::= & \mathtt{skip} \mid x := e \mid x := *(B) \mid \mathcal{S}; \mathcal{T} \mid \mathtt{if}\ B\ \mathtt{then}\ \mathcal{S}\ \mathtt{else}\ \mathcal{T} \mid \\
& & \mathcal{S} \mathbin{++} \mathcal{T} \mid \mathcal{S} *\ \text{invariant}\ [I_1] \ldots [I_n] \mid \\
& & \langle \dot{\mathbf{x}} = \boldsymbol{e} \,\&\, D \rangle\ \text{invariant gvar}_1 \ldots \text{gvar}_k, \text{ode\_inv}_1 \ldots \text{ode\_inv}_n \mid \\
& & \langle \dot{\mathbf{x}} = \boldsymbol{e} \,\&\, D \rangle\ \text{solution}
\end{array}
$$

The only addition to the syntax of HCSP is that each loop is followed by a list of invariants $I_1, \ldots, I_n$, and each ODE is either followed by a list of ghost variable declarations and a list of invariant annotations, each of which specify an invariant to be proved using one of (dI), (dbx), or (bc) rules, or followed by the annotation “solution” to indicate that the (sln) rule is to be used.

Verification Condition Generation

Definition (Verification Condition)

Given a Hoare triple $\{P_1 \wedge \cdots \wedge P_m\}\mathcal{S}\{Q_1 \wedge \cdots \wedge Q_n\}$ to verify, we define the set of all VCs to be

$$
\begin{aligned}
&\mathrm{VC}(\{P_1 \wedge \cdots \wedge P_m\}\mathcal{S}\{Q_1 \wedge \cdots \wedge Q_n\}) = \\
&\quad \{P_1 \wedge \cdots \wedge P_m \rightarrow R \mid R \in \mathrm{pre}(\mathcal{S}, \{Q_1, \ldots, Q_n\})\} \cup && \text{(pre)} \\
&\quad \{\tilde{P}_1 \wedge \cdots \wedge \tilde{P}_{\tilde{m}} \rightarrow R \mid R \in \mathrm{vc}(\mathcal{S}, \{Q_1, \ldots, Q_n\})\} && \text{(vc)}
\end{aligned}
$$

where $\tilde{P}_1, \ldots, \tilde{P}_{\tilde{m}}$ is the subset of the preconditions $P_1, \ldots, P_m$ whose variables are never reassigned in $\mathcal{S}$, and the functions pre and vc are defined below.

Verification Condition Generation

Given an annotated program $\mathcal{S}$ and a set $\{Q_1, \ldots, Q_n\}$ of postconditions, we denote the set of derived preconditions as $\mathrm{pre}(\mathcal{S}, \{Q_1, \ldots, Q_n\})$, defined as follows.

$$\mathrm{pre}(\mathcal{S}, \{Q_1, \ldots, Q_n\}) = \mathrm{pre}(\mathcal{S}, Q_1) \cup \cdots \cup \mathrm{pre}(\mathcal{S}, Q_n) \qquad \text{(pre-multi)}$$

$$\mathrm{pre}(\mathtt{skip}, Q) = Q \qquad \text{(pre-skip)}$$

$$\mathrm{pre}(x := e, Q) = Q[e/x] \qquad \text{(pre-assn)}$$

$$\mathrm{pre}(\mathcal{S}; \mathcal{T}, Q) = \mathrm{pre}(\mathcal{S}, \mathrm{pre}(\mathcal{T}, Q)) \qquad \text{(pre-seq)}$$

$$\begin{aligned}
&\mathrm{pre}(\mathtt{if}\ B_1\ \mathtt{then}\ \mathcal{S}_1\ \mathtt{else}\ \cdots\ \mathtt{if}\ B_{n-1}\ \mathtt{then}\ \mathcal{S}_{n-1}\ \mathtt{else}\ \mathcal{S}_n, Q) = \\
&\quad \{\neg(B_1 \vee \cdots \vee B_{i-1}) \wedge B_i \to P \mid P \in \mathrm{pre}(\mathcal{S}_i, Q), 1 \le i \le n-1\} \cup \qquad \text{(pre-if)} \\
&\quad \{\neg(B_1 \vee \cdots \vee B_{n-1}) \to P \mid P \in \mathrm{pre}(\mathcal{S}_n, Q)\} \qquad \text{(pre-else)}
\end{aligned}$$

$$\mathrm{pre}(\mathcal{S}_1 \mathbin{++} \cdots \mathbin{++} \mathcal{S}_n, Q) = \mathrm{pre}(\mathcal{S}_1, Q) \cup \cdots \cup \mathrm{pre}(\mathcal{S}_n, Q) \qquad \text{(pre-choice)}$$

$$\mathrm{pre}(x := *\,(B), Q) = B[y/x] \to Q[y/x] \text{ for a fresh variable } y \qquad \text{(pre-nassn)}$$

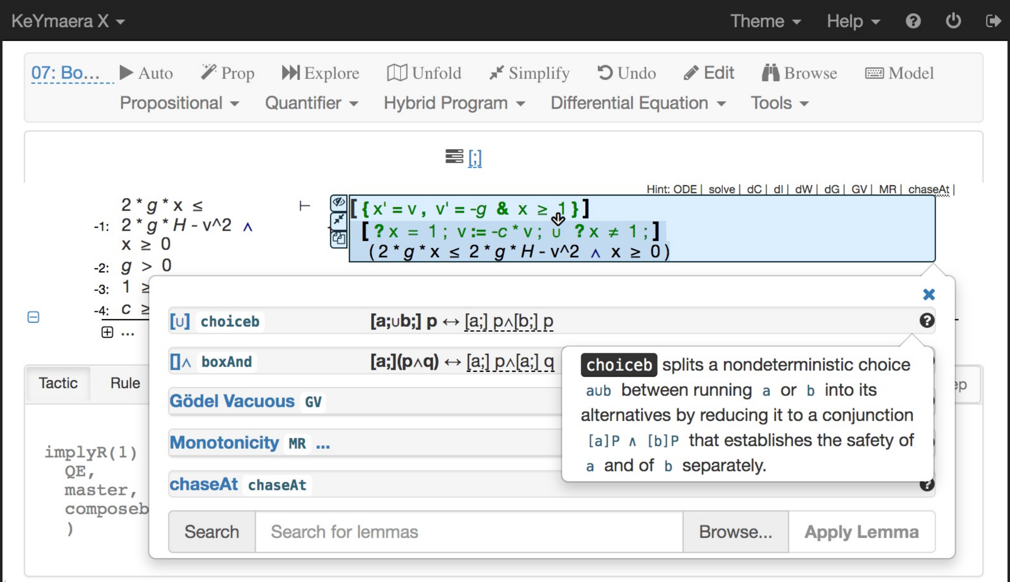
$$\mathrm{pre}(\mathcal{S} * \text{ invariant } [I_1] \ldots [I_n], Q) = \{I_j \mid 1 \le j \le n\} \qquad \text{(pre-loop)}$$

$$\begin{aligned}
&\mathrm{pre}(\langle \dot{\mathbf{x}} = \boldsymbol{e} \,\&\, D \rangle \text{ invariant gvar}_1 \ldots \text{gvar}_k, \text{ode_inv}_1 \ldots \text{ode_inv}_n, Q) = \\
&\quad P_{\mathrm{skip}} \cup P_{\mathrm{init}}
\end{aligned}$$

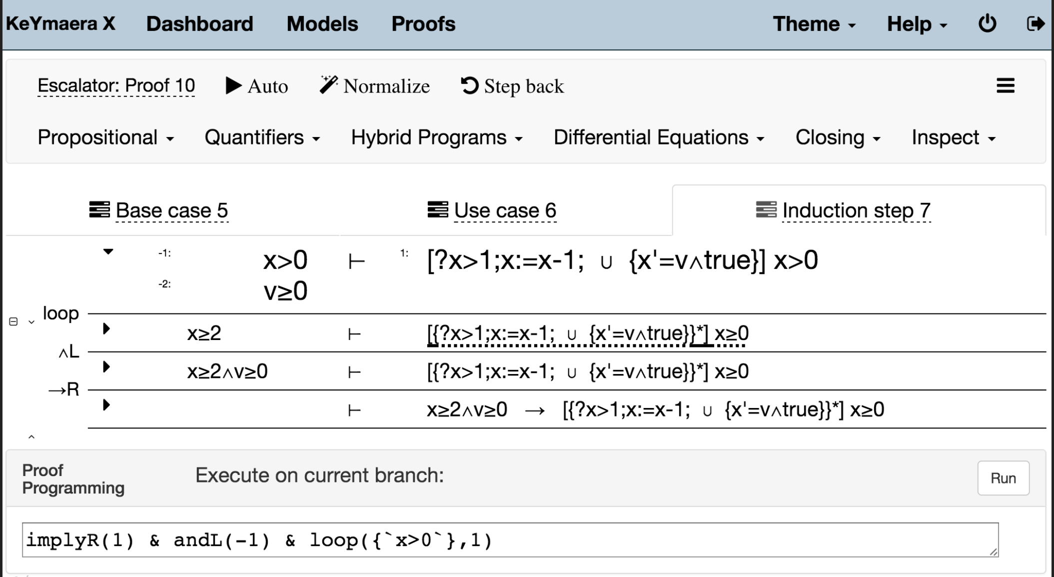
$$\mathrm{pre}(\langle \dot{\mathbf{x}} = \boldsymbol{e} \,\&\, D \rangle \text{ solution}) = P_{\mathrm{skip}} \cup P_{\mathrm{sln}}$$

where

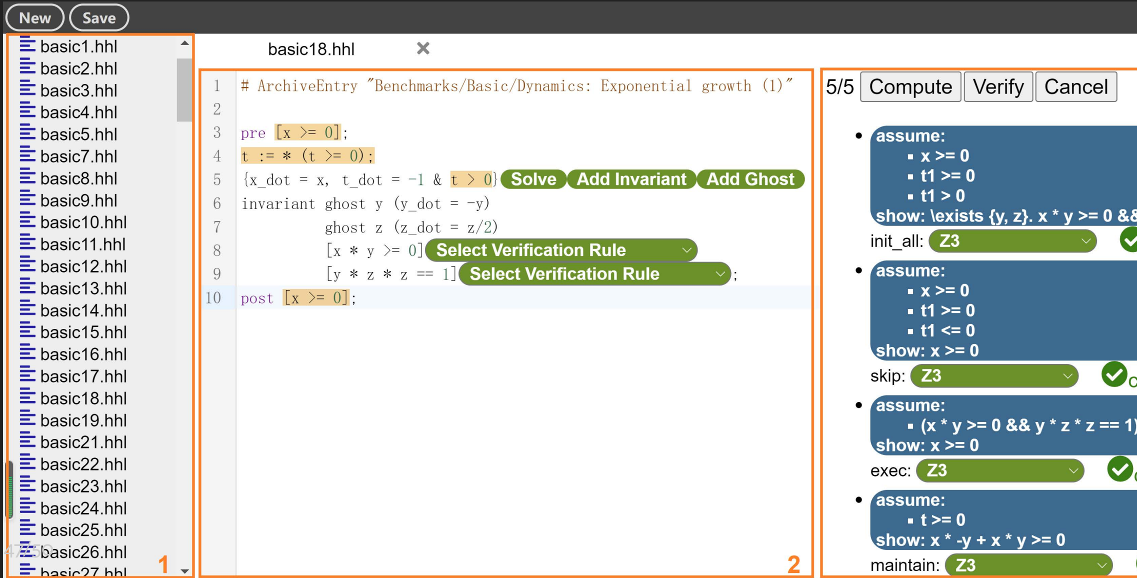
KeymaeraX: A Tool to prove hybrid program correctness



Proof based on loop invariant



HHLPy: Hybrid hoare logic based prover written in Python



Prospect

What would I contribute?

- Analysis and Verification on Program usually relies on SMT Solving (model checker tools). Incremental verification involves solving procedure in SMT tools.
- Local Search helps for bug finding, or even give a counterexample (failed test).
- Symbolic Executation Tools (KLEE)
-

Research Work Presentation

Thank you