A Tool for Formal Verification of Nonlinear Inequalities

Alexey Solovyev

August 31, 2017

Contents

References

[1] HOL Light home page
`http://www.cl.cam.ac.uk/~jrh13/hol-light`

[2] HOL Light repository
`https://github.com/jrh13/hol-light`

[3] HOL Light tutorial
`http://www.cl.cam.ac.uk/~jrh13/hol-light/tutorial_220.pdf`

[4] The Flyspeck project
`https://github.com/flyspeck/flyspeck`

[5] César Muñoz and Anthony Narkawicz, *Formalization of a Representation of Bernstein Polynomials and Applications to Global Optimization*, Journal of Automated Reasoning, DOI: 10.1007/s10817-012-9256-3
`http://shemesh.larc.nasa.gov/people/cam/Bernstein/`

1 Introduction

This document describes a tool for verification of nonlinear inequalities in HOL Light proof assistant [1, 3]. This tool was developed as a part of the Flyspeck project (a formal proof of the Kepler conjecture) [4]. The tool is capable to verify multivariate nonlinear strict inequalities on rectangular domains. More specifically, the tool can handle inequalities in the form

$$\forall \mathbf{x} \in D \implies f(\mathbf{x}) < g(\mathbf{x}),$$

where $D = \{(x_1, \ldots, x_n) \mid a_i \le x_i \le b_i\}$ and f, g are functions which may include all usual arithmetic operations, square roots, arccosines, and arctangents. The maximal number of variables is 8. Future releases of the tool will include all elementary functions and will have no restriction on the number of variables. Moreover, it will be possible to verify inequalities on non-rectangular domains.

Internally, the tool uses interval arithmetic with Taylor approximations (with second-order error terms).

The document is organized as follows. The next section describes the installation process. Then a quick introduction of tool functions is presented. After that, a more detailed description of tool functions is given and special options are described. The last two sections describe several examples and test cases.

2 Installation

First of all, if you don't have OCaml and HOL Light installed, then you need to install them. The verification tool was tested with Ocaml 3.09.3 and Ocaml 3.12.1 and with one of the latest versions of HOL Light (r149 in the HOL Light repository). HOL Light installation instructions can be found in John Harrison's HOL Light tutorial [3].

Alternatively, one can download and run the following script written by Alex Krauss: `https://bitbucket.org/akrauss/hol-light-workbench`. This script will download and install the latest version of HOL Light and other necessary programs.

The installation of the tool for verification of nonlinear inequalities is very simple. Download the distribution from
`http://code.google.com/p/flyspeck/downloads/list`
or get the latest version from the Flyspeck repository with the shell command
`svn co http://flyspeck.googlecode.com/svn/trunk/formal_ineqs`

The tool can be placed in any directory on your computer. It is important to inform HOL Light about tool's location. It can be done with the following OCaml command:

```
load_path := "path to the tool directory" :: !load_path;;
```

After the path is set, the tool can be loaded with the command

```
needs "verifier/m_verifier_main.hl";;
```

The tool loads the standard HOL Light library `Multivariate/realanalysis.ml`. The loading process of this library could take pretty long time, so it is recommended to use a checkpointed version of HOL Light with preloaded multivariate analysis libraries.

Before loading the tool, it is also possible to change some global options. These options are described in section 5.

3 Quick Start

The polynomial inequality

$$-\frac{1}{\sqrt{3}} \le x \le \sqrt{2},\ -\sqrt{\pi} \le y \le 1 \implies x^2y - xy^4 + y^6 + x^4 - 7 > -7.17995$$

can be verified with the following script

```
(* make sure that load_path contains the path to formal_ineqs *)
needs "verifier/m_verifier_main.hl";;
open M_verifier_main;;

let ineq =
  `-- &1 / sqrt(&3) <= x /\ x <= sqrt(&2) /\
   -- sqrt(pi) <= y /\ y <= &1
   ==> x pow 2 * y - x * y pow 4 + y pow 6 - &7 + x pow 4 > -- #7.17995`;;

let th, stats = verify_ineq default_params 5 ineq;;
```

The first parameter of the verification function `verify_ineq` contains verification options. We use default values given by the constant `default_params`. Available options are described in section 4.

The second parameter specifies the precision of formal floating point operations. This parameter determines the maximal number of significant digits of any result returned by a formal floating point operation. Here, digits are not decimal. Internally all natural numbers

are represented using a fixed base (see section 5 for more details). This base is relatively large (the default value is 100) to speed up arithmetic operations. Actual precision of formal floating point operations depends on the precision parameter and on the base of the internal representation of natural numbers. If the base value is 100 and the precision parameter is 5 as in the example above, then the precision of formal floating point operations is 10 decimal digits: $100^5 = 10^{10}$. Note that the verification of the example will fail if the precision parameter is 4 or less. On the other hand, if the precision parameter is 10, the verification will succeed but it will take a little more time.

The third parameter is the inequality itself given as a HOL Light term. The format of this term is simple: it is an implication with bounds of variables in the antecedent and an inequality in the consequent. The bounds of all variables should be in the form *a constant expression* <= x or x <= *a constant expression*. For each variable, upper and lower bounds must be given. The inequality must be a strict inequality ($<$ or $>$). The inequality may include `sqrt`, `atn`, and `acs` functions. The constant `pi` (π) is also allowed.

The verification function returns a HOL Light theorem and a record with some verification information which includes verification time.

4 Verification Functions

The main verification function `verify_ineq` is contained in `M_verifier_main` module defined in `verifier/m_verifier_main.hl`. The function has 3 arguments and its type is

```
verify_ineq : verification_parameters -> int -> term -> thm * verification_stats
```

The first parameter contains verification options defined in the following record

```
type verification_parameters =
{
  (* If true, then monotonicity properties can be used *)
  (* to reduce the dimension of a problem *)
  allow_derivatives : bool;
  (* If true, then convexity can be used *)
  (* to reduce the dimension of a problem *)
  convex_flag : bool;
  (* If true, then verification on internal subdomains can be skipped *)
  (* for a monotone function *)
  mono_pass_flag : bool;
  (* If true, then raw interval arithmetic can be used *)
  (* (without Taylor approximations) *)
  raw_intervals_flag : bool;
  (* If true, then an informal procedure is used to determine *)
  (* the optimal precision for the formal verification *)
  adaptive_precision : bool;
  (* This parameter might be used in cases when the certificate search *)
  (* procedure returns a wrong result due to rounding errors *)
  (* (this parameter will be eliminated when the search procedure is corrected) *)
  eps : float;
};;
```

A detailed description of these parameter is omitted in this document. In most cases, it is enough to use the constant `default_params` which turns all verification flags on and sets `eps = 0`. In rare cases, it is necessary to adjust `eps` to get a result. This can be done with the command

```
verify_ineq {default_params with eps = 1e-10} 5 ineq_tm;;
```

The second parameter of the verification function specifies the precision of formal floating point operations. This parameter determines the maximal number of significant digits of any result returned by a formal floating point operation. Here, digits are not decimal. Internally all natural numbers are represented using a fixed base (see section 5 for more details). This base is relatively large (the default value is 100) to speed up arithmetic operations. Actual precision of formal floating point operations depends on the precision parameter and on the base of the internal representation of natural numbers. In many cases, if the verification function fails, it is enough to increase the precision parameter to get a result.

The third parameter of the verification function is a HOL Light term which specifies an inequality itself. The format of this term is the following:

```
bounds of variables ==> an inequality
```

The bounds of all variables should be in the form *a constant expression* `<=` x or x `<=` *a constant expression*. For each variable, upper and lower bounds must be provided. The order in which the bounds are given is irrelevant. Bounds of variables may be connected with `/\` or with `==>`. The inequality must be a strict inequality ($<$ or $>$). The inequality may include `sqrt`, `atn`, and `acs` functions. The constant `pi` (π) is also allowed.

The verification function returns a theorem and some verification information defined in the record

```
type verification_stats =
{
  total_time : float;
  formal_verification_time : float;
  certificate : Verifier.certificate_stats;
};;
```

The field `total_time` contains total verification time. The field `formal_verification_time` contains time taken by the formal verification procedure only (this time doesn't include time for constructing a solution certificate and for other preparations). The last field `certificate` contains information about a solution certificate.

The conclusion of the returned theorem is not exactly the same as the third parameter of the verification function: the order of bounds of variables may be altered and variables which are not used in the inequality are eliminated. For example, commands

```
let th1, _ = verify_ineq default_params 3
  `&1 <= y /\ y <= &2 /\ &1 <= x /\ x <= &3 ==> x + y < &6`;;
let th2, _ = verify_ineq default_params 3
  `&1 <= y /\ y <= &2 /\ &1 <= x /\ x <= &3 ==> y < &3`;;
```

return

```
th1 = |- (&1 <= x /\ x <= &3) /\ &1 <= y /\ y <= &2 ==> x + y < &6
th2 = |- &1 <= y /\ y <= &2 ==> y < &3
```

5 Global Options

The options which affect the arithmetic operations with natural and floating point numbers must be set before the verification tool is loaded. After the verification tool is loaded, arithmetic options may not be changed. To set arithmetic options, load the file `arith_options.hl` located in the root directory of the tool. The available options are listed below.

base Determines the base for representing natural numbers. Default HOL Light representation of natural numbers is binary (i.e., its base is 2). A higher base increases speed of arithmetic operations but it also requires more memory to remember additional theorems. The default value of the base is `100`. To set a new base, use the command

```
Arith_options.base := 200;;
```

min_exp Determines the minimal exponent in the representation of floating point numbers. Each floating point number is represented as a triple (s, n, e) where s is a boolean value which determines the sign of the number, n and e are natural numbers which represent the mantissa and the exponent. The value corresponding to (s, n, e) is given by

$$f = (-1)^{\text{if } s \text{ then } 1 \text{ else } 0} \times n \times b^{e - min\_exp}$$

where b is the base of the representation of natural numbers.

cached If this value is true, then results of all natural number operations are cached. The default value is `true`.

float_cached If this value is true, then results of all floating point operations are cached. The default value is `true`.

init_cache_size Determines the initial size of the cache for results of arithmetic operations. The default value is `10000`.

max_cache_size Determines the maximal size of the cache for results of arithmetic operations. The default value is `20000`. Note: each cached operation has its own cache.

The file `verifier_options.hl` contains the option `info_print_level` which controls the amount of information printed by a verification process. This option can be changed at any time:

```
Verifier_options.info_print_level := 0;;
```

Possible values are: 0 (no information is printed); 1 (all essential information is printed); 2 (all information is printed). The default value is 1.

The next example shows how to change default options:

```
(* The arithmetic options must be set before loading the verification tool *)
needs "arith_options.hl";;
```

```
(* Increase the arithmetic base *)
Arith_options.base := 200;;

(* Increase the cache size *)
Arith_options.max_cache_size = 40000;;

(* Load the verification tool *)
needs "verifier/m_verifier_main.hl";;

(* The verification option can be changed at any time *)
Verifier_options.info_print_level := 2;;

open M_verifier_main;;
```

6 Additional Examples

The verification tool distribution contains several example files. The file `examples_poly.hl` contains polynomial inequalities from the paper [5]. The command

```
needs "examples_poly.hl";;
```

will load this file and run all polynomial inequality tests. To run all tests again, type `run_tests();;`

To run a specific test, type `run_{test_name}();;` where `{test_name}` is one of the following: `schwefel`, `rd`, `caprasse`, `lv`, `butcher`, `magnetism`, `heart`.

Here is the list of all examples.

schwefel

$$-5.8806 \times 10^{-10} < (x_1 - x_2^2)^2 + (x_2 - 1)^2 + (x_1 - x_3^2)^2 + (x_3 - 1)^2$$
$$(x_1, x_2, x_3) \in [(-10, -10, -10), (10, 10, 10)]$$

rd

$$-36.7126907 < -x_1 + 2x_2 - x_3 - 0.835634534\, x_2(1 + x_2)$$
$$(x_1, x_2, x_3) \in [(-5, -5, -5), (5, 5, 5)]$$

caprasse

$$-3.1801 < -x_1 x_3^3 + 4x_2 x_3^2 x_4 + 4x_1 x_3 x_4^2 + 2x_2 x_4^3 + 4x_1 x_3 + 4x_3^2 - 10x_2 x_4 - 10x_4^2 + 2$$
$$(x_1, x_2, x_3, x_4) \in [(-0.5, -0.5, -0.5, -0.5), (0.5, 0.5, 0.5, 0.5)]$$

lv

$$-20.801 < x_1 x_2^2 + x_1 x_3^2 + x_1 x_4^2 - 1.1 x_1 + 1$$
$$(x_1, x_2, x_3, x_4) \in [(-2, -2, -2, -2), (2, 2, 2, 2)]$$

butcher

$$-1.44 < x_6x_2^2 + x_5x_3^2 - x_1x_4^2 + x_4^2 - \tfrac{1}{3}x_1 + \tfrac{4}{3}x_4$$
$$(x_1, x_2, x_3, x_4, x_5, x_6) \in [(-1, -0.1, -0.1, -1, -0.1, -0.1), (0, 0.9, 0.5, -0.1, -0.05, -0.03)]$$

magnetism

$$-0.25001 < x_1^2 + 2x_2^2 + 2x_3^2 + 2x_4^2 + 2x_5^2 + 2x_6^2 + 2x_7^2 - x_1$$
$$(x_1, x_2, x_3, x_4, x_5, x_6, x_7) \in [(-1, -1, -1, -1, -1, -1, -1), (1, 1, 1, 1, 1, 1, 1)]$$

heart

$$-1.7435 < -x_1x_6^3 + 3x_1x_6x_7^2 - x_3x_7^3 + 3x_3x_7x_6^2 - x_2x_5^3 + 3x_2x_5x_8^2 - x_4x_8^3 + 3x_4x_8x_5^2 - 0.9563453$$
$$(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8) \in [(-0.1, 0.4, -0.7, -0.7, 0.1, -0.1, -0.3, -1.1), (0.4, 1, -0.4, 0.4, 0.2, 0.2, 1.1, -0.3)]$$

The file `examples_flyspeck.hl` contains some inequalities from the Flyspeck project [4]. The command

```
needs "examples_flyspeck.hl";;
```

will load this file and run some easy inequality tests. To rerun these tests, use the command `test_easy();;`. To run more difficult tests, type `test_medium();;` or `test_hard();;`. (Warning: medium tests require about 30 minutes, hard tests require more than 5 hours.)

Some Flyspeck inequalities are listed below.

$$\begin{aligned}
\Delta(x_1, \ldots, x_6) &= x_1x_4(-x_1 + x_2 + x_3 - x_4 + x_5 + x_6) \\
&\quad + x_2x_5(x_1 - x_2 + x_3 + x_4 - x_5 + x_6) \\
&\quad + x_3x_6(x_1 + x_2 - x_3 + x_4 + x_5 - x_6) \\
&\quad - x_2x_3x_4 - x_1x_3x_5 - x_1x_2x_6 - x_4x_5x_6, \\
\Delta_4 &= \frac{\partial \Delta}{\partial x_4}, \\
\mathrm{dih}_x\,(x_1, \ldots, x_6) &= \frac{\pi}{2} - \arctan\left(\frac{-\Delta_4(x_1, \ldots, x_6)}{\sqrt{4x_1\Delta(x_1, \ldots, x_6)}}\right), \\
\mathrm{dih}_y\,(y_1, \ldots, y_6) &= \mathrm{dih}_x\,(y_1^2, \ldots, y_6^2).
\end{aligned}$$

4717061266

$$\Delta(x_1, x_2, x_3, x_4, x_5, x_6) > 0, \quad 4 \le x_i \le 6.3504$$

7067938795

$$\mathrm{dih}_x\,(x_1, \ldots, x_6) - \pi/2 + 0.46 < 0,$$
$$4 \le x_{1,2,3} \le 6.3504,\ x_4 = 4,\ 3.01^2 \le x_{5,6} \le 3.24^2$$

3318775219

$$0 < \mathrm{dih}_y\,(y_1, \ldots, y_6) - 1.629 + 0.414(y_2 + y_3 + y_5 + y_6 - 8.0) - 0.763(y_4 - 2.52) - 0.315(y_1 - 2.0),$$
$$2 \le y_i \le 2.52$$

7 Test Results

This section contains time test results for inequalities described in the previous section. All tests were performed on Intel Core i5, 2.67GHz running Ubuntu 9.10 inside Virtual Box 4.2.0 on a Windows 7 host; the Ocaml version was 3.09.3; the base of arithmetic was 200; the caching was turned on.

Polynomial inequalities

Inequality ID	# variables	precision	total time (s)	formal verification (s)
schwefel	3	5	26.329	19.145
rd	3	5	1.593	0.017
caprasse	4	5	8.057	1.286
lv	4	5	1.875	0.030
butcher	6	5	3.609	0.035
magnetism	7	5	7.007	1.347
heart	8	5	17.298	1.277

Flyspeck inequalities

Inequality ID	precision	total time (s)	formal verification (s)
2485876245a	4	5.530	0.058
4559601669b	4	4.679	0.048
4717061266	4	27.1	0.250
5512912661	4	8.860	0.086
6096597438a	4	0.071	0.071
6843920790	4	2.824	0.076
SDCCMGA b	4	9.012	0.949
TSKAJXY-TADIAMB[1]	4	75.9	21.2
7067938795	4	431	387
5490182221	4	1726	1533
3318775219	4	17091	15226

[1]Reduced to a polynomial inequality