iSAT Quick Start Guide[*]

AVACS H1/2 iSAT Developer Team

February 12, 2010

1 Introduction

This document provides a brief introduction into the usage of iSAT, a satisfiability checker for Boolean combinations of arithmetic constraints over real- and integer-valued variables. A peculiarity of iSAT, which sets it apart from many other solvers, is that it is not limited to linear arithmetic, but can also deal with nonlinear constraints involving transcendental functions. The iSAT algorithm is a tight integration of recent SAT solving techniques with interval-based arithmetic constraint solving. For technical details, see [FHT+07].

2 Modes of Operation

The command-line tool iSAT has two different modes of operation: It can be used a) as a satisfiability checker for a single formula or b) for finding a trace of a hybrid system via bounded model checking (BMC). In the following the usage of iSAT is illustrated by means of some examples.

2.1 Single Formula Mode

Assume you want to find a pythagorean triple, i.e. a triple (a, b, c) of integer values which satisfies $a^2 + b^2 = c^2$. To use iSAT for this purpose, create a file, say `sample1.hys`, containing the following lines (without the line numbers).

```
DECL
    -- The range of each variable has to be bounded.
    int [1, 100] a, b, c;

EXPR
    -- Constraint to be solved.
    a*a + b*b = c*c;
```

In single formula mode the input file has two sections: The first section, starting with the keyword `DECL`, contains declarations of all variables occuring in the formula to be solved. The second section, starting with `EXPR`, contains the formula itself, in this case consisting of a single arithmetic constraint only. After calling '`isat sample1.hys`', iSAT generates the output displayed below, reporting $a = 63, b = 60$, and $c = 87$ as a satisfying valuation.

[*]This work has been partially funded by the German Research Council (DFG) as part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (SFB/TR 14 AVACS, `www.avacs.org`).

```
This is iSAT 1.0.

SOLVING:
    k = 0

SOLUTION:
    a (int):
        @0:  [63,63] (point interval)

    c (int):
        @0:  [87,87] (point interval)

    b (int):
        @0:  [60,60] (point interval)

RESULT:
    satisfiable

Statistics.
Number of variables                   : 8
    Boolean variables                 : 0
    Integer variables                 : 8
    Real variables                    : 0
Number of complex bounds              : 5
Number of problem clauses             : 5
Number of decisions                   : 229 / 229
Number of point splits                : 0 / 0
Number of assignments                 : 2224 / 2224
    Number of IQ additions            : 2108 / 2108
Maximum decision level                : 17 / 17
Maximum number of deductions per DL   : 49 / 49
Number of conflicts                   : 170 / 170
Average size of learnt clauses        : 7.2 / 7.2
Maximum backjump distance             : 7 / 7
Number of restarts                    : 0 / 0
Number of conflict clauses
implied by conflict clauses           : 28 / 28

time solver                           : 0.11 / 0.11 sec
```

You might have noticed, that iSAT writes the result in form of *intervals* instead of single values. This is due to the fact that calculations in iSAT are carried out in *interval arithmetic.* In contrast to the examples presented in this brief introduction, the solution intervals computed by iSAT will in general be non-point intervals.

2.2 Bounded Model Checking Mode

Bounded model checking (BMC) of a hybrid system aims at finding a run of bounded length k which

- starts in an initial state of the system,
- obeys the system's transition relation, and
- ends in a state in which a certain (desired or undesired) property holds.

The idea of BMC is to construct a formula which is satisfiable if and only if a trace with above properties exists. In case of satisfiability, any satisfying valuation of this formula corresponds to such a trace.

For specifying BMC tasks, iSAT has a second input file format which consists of four parts:

- `DECL`: As above, this part contains declarations of all variables. Furthermore, you can define symbolic constants in this section (see the definition of `f` in line 2 in the example below).
- `INIT`: This part is a formula describing the initial state(s) of the system to be investigated. In the example below, `x` is initialized to 0.6, and `jump` is set to `false`, since this is the only valuation which satisfies the constraint `!jump`, where '`!`' stands for '`not`'.

- `TRANS`: This formula describes the transition relation of the system. Variables may occur in primed (`x'`) or unprimed (`x`) form. A primed variable represents the value of that variable in the successor step, i.e. after the transition has taken place. Thus, line 14 of the example states, that if `jump` is `false` in the current state, then the value of `x` in the next state is given by its current value plus 2. The semicolon which terminates each constraint can be read as an AND-operator. Hence, `TRANS` in the example is a conjunction of three constraints.
- `TARGET`: This formula characterises the state(s) whose reachability is to be checked. In the example below, we want to find out if a state is reachable in which `x > 3.5` holds.

```
 1  DECL
 2      define f = 2.0;
 3      float [0, 1000] x;
 4      boole jump;
 5
 6  INIT
 7      x = 0.6;
 8      !jump;
 9
10  TRANS
11      jump' <-> !jump;
12
13      jump -> f * x' = x;
14      !jump -> x' = x + 2;
15
16  TARGET
17      x > 3.5;
```

When calling iSAT with the input file above, it successively unwinds the transition relation $k = 0, 1, 2, \ldots$ times, conjoins the resulting formula with the formulae describing the initial state and the target states, and thereafter solves the formula thus obtained.

From the excerpts of the tool ouput below you can see that for $k = 0, 1, 2, 3, 4$, the formulae are all unsatisfiable, for $k = 5$ however, a candidate solution is found. Note, that iSAT reports the values of `jump` and `x` for each step k of the system. After the last transition, as required, `x > 3.5` holds.

```
Settings.
Minimum splitting width    : 0.01
Absolute bound progress    : 0.001
Relative bound progress    : 0
Interval divide parameter  : 2
Decide interval borders    : random
Variable decision order    : natural
Restarts                   : disabled
Clause deletion            : enabled
Implied clause deletion    : enabled
Check SAT often            : disabled
Point splitting            : disabled
Purification               : disabled
Continue after UNKNOWN     : disabled

This is iSAT 1.0.

SOLVING:
    k = 0

RESULT:
    unsatisfiable

[...]

SOLVING:
    k = 1

RESULT:
    unsatisfiable

[...]

SOLVING:
    k = 5

CANDIDATE SOLUTION:
    jump (boolean):
        @0:  [0,0] (point interval)
        @1:  [1,1] (point interval)
        @2:  [0,0] (point interval)
        @3:  [1,1] (point interval)
        @4:  [0,0] (point interval)
        @5:  [1,1] (point interval)

    x (float):
        @0:  (0.59999999,0.60000001) (width: 1.11022302e-16)
        @1:  (2.59999999,2.60000001) (width: 4.4408921e-16)
        @2:  (1.29999999,1.30000001) (width: 2.22044605e-16)
        @3:  (3.29999999,3.30000001) (width: 4.4408921e-16)
        @4:  (1.64999999,1.65000001) (width: 2.22044605e-16)
        @5:  (3.64999999,3.65000001) (width: 4.4408921e-16)

RESULT:
    unknown

Statistics.
Number of variables                  : 41
    Boolean variables                : 27
    Integer variables                : 2
    Real variables                   : 12
Number of complex bounds             : 35
Number of problem clauses            : 86
Number of decisions                  : 0 / 0
Number of point splits               : 0 / 0
Number of assignments                : 135 / 423
    Number of IQ additions           : 132 / 415
Maximum decision level               : 0 / 0
Maximum number of deductions per DL  : 135 / 135
Number of conflicts                  : 0 / 5
Average size of learnt clauses       : 0 / 0
Maximum backjump distance            : 0 / 0
Number of restarts                   : 0 / 0
Number of conflict clauses
implied by conflict clauses          : 0 / 0

time solver                          : 0 / 0 sec
```

3 Input Language: Types, Operators, and Expressions

3.1 Types

- Types supported by iSAT are `float`, `int`, and `boole`.
- Boolean variables are identified with integer variables of range $[0, 1]$.
- When declaring a float or an integer variable you have to specify the range of this variable. Due to the internal working of iSAT these ranges have to be bounded, i.e. you have to specify a lower and an upper bound. To reduce solving time, ranges should be chosen as small as possible.
- Integer and float variables can be mixed within the same arithmetic constraint.

3.2 Operators

- Boolean operators:

Operator	Type	Num. Args.	Meaning
`and`	infix	2	conjunction
`;`	infix	2	alternative notation for '`and`', cf. precedence rules, however
`or`	infix	2	disjunction
`nand`	infix	2	negated and
`nor`	infix	2	negated or
`xor`	infix	2	exclusive or
`nxor`	infix	2	negated xor, i.e. equivalence
`<->`	infix	2	alternative notation for '`nxor`'
`impl`	infix	2	implication
`->`	infix	2	alternative notation for '`impl`'
`not`	prefix	1	negation
`!`	prefix	1	alternative notation for '`not`'

- Arithmetic operators:

Operator	Type	Num. Args.	Meaning
`+`	infix	1 or 2	unary 'plus' and addition
`-`	infix	1 or 2	unary 'minus' and subtraction
`*`	infix	2	multiplication
`abs`	prefix	1	absolute value
`min`	prefix	2	minimum
`max`	prefix	2	maximum
`exp`	prefix	1	exponential function
`sin`	prefix	1	sine (unit: radian)
`cos`	prefix	1	cosine (unit: radian)
`^`	infix	2	nth power, n (2nd argument) has to be a positive integer
`nrt`	prefix	2	nth root, n (2nd argument) has to be a positive integer

  Operators `abs`, `min`, `max`, `exp`, `sin`, `cos`, and `nrt` have to be called with their arguments being separated by commas and enclosed in brackets. Example: `nrt(x, 3)`

- Relational operators:

  `>`, `>=`, `<`, `<=`, `=`, `!=` (all infix)

- Precedence of operators: The following list shows all operators ordered by their precedence, starting with the one that binds strongest. You can use brackets in the input file to modify the order of term evaluation induced by these precedence rules.

- ▷ unary not, `^`
- ▷ unary plus, unary minus
- ▷ multiplication
- ▷ plus, minus
- ▷ `abs`, `min`, `max`, `exp`, `sin`, `cos`, `nrt`
- ▷ `>`, `>=`, `<`, `<=`, `=`, `!=`
- ▷ `and`, `nand`
- ▷ `xor`, `nxor`
- ▷ `or`, `nor`
- ▷ `impl`
- ▷ `;`

3.3 Expressions

- Let `a` and `b` be Boolean variables and `x` and `y` be float variables. Examples for expressions are:
  - ▷ `[x * y + 2 * (x - 4) >= 5 - 2 * (x - 4)]`
  - ▷ `(x > 20 and !b) xor a`
  - ▷ `b <-> {3.18 * (-5 - y * y * y) = 7}`
  - ▷ `sin(x + max(3, y)) < 0.4`
  - ▷ `abs(nrt(x^2 + y^2), 2)) <= 10.3`
- Note that the individual constraints occurring in a formula have to be conjoined using the '`;`'-operator. In addition, the last line of each formula has to be terminated with a semicolon.

4 How iSAT works

To be able to interprete iSAT's output and the tool options presented in the next section you need some basic understanding of how the tool works internally. The iSAT solver performs a backtrack search to prune the search space until it is left with a 'sufficiently small' portion of the search space for which it cannot derive any contradiction with respect to the constraints occuring in the input formula.

Initially, the search space consists of the cartesian product of the ranges of all variables occuring in the formula to be solved. Just like an ordinary (purely Boolean) SAT solver, iSAT operates by alternating between two steps:

- The *decision step* involves selecting a variable 'blindly', splitting its current interval (e.g. by using the midpoint of the interval as split point) and temporarily discarding either the lower or the upper part of the interval from the search. The solver will ignore the discarded part of the search space until the decision is undone by backtracking.
- Each decision is followed by a *deduction step* in which the solver applies a set of deduction rules that explore all the consequences of the previous decision. Informally speaking, the deduction rules carve away portions of the search space that contain non-solutions only.

Assume, for example, that the input formula consists of the single constraint $x \cdot y = 8$ and initially $x \in [2, 4]$ and $y \in [2, 4]$ holds. The solver might now decide to split the interval of x by assigning the new lower bound $x \geq 3$ to x. In the subsequent deduction phase, the solver will deduce that, due to the increased lower bound of x, the upper bound of y can be reduced to $\frac{8}{3}$ because for all other values of y the constraint $x \cdot y = 8$ is violated. After asserting $y \leq \frac{8}{3}$, thereby contracting the search space to $[3, 4] \times [2, \frac{8}{3}]$, no further deductions are possible, and the solver goes on with taking the next decision. Deduction may also yield a conflict, i.e. a variable whose interval is empty, indicating the need to backtrack.

To enforce termination of the algorithm, the solver only selects a variable x for splitting if the width $\overline{x} - \underline{x}$ of its interval $[\underline{x}, \overline{x}]$ is above a certain threshold ε, which we call *minimal splitting width*. Furthermore, the solver discards a (non-conflicting) deduced bound if it only yields a comparatively small progress with respect to the bound already in place. More precisely, a deduced lower (upper) bound b_d is ignored if $|b_c - b_d| \leq \delta_{\text{abs}}$, where b_c is the current lower (upper) bound of the respective variable and δ_{abs} is a parameter which we call *absolute progress*. The values of ε and δ can be set with the command-line options `--msw` and `--prabs`. Additionally, deductions can also be neglected if they yield too little *relative progress* δ_{rel} (to be specified by command-line option `--prrel`), i.e. a deduced lower (upper) bound b_d is ignored if $|b_c - b_d|/|b_c - b_o| \leq \delta_{\text{rel}}$, where b_c is the current lower (upper) bound of the respective variable and b_o is the corresponding upper (lower) bound, i.e. $|b_c - b_o|$ yields the width of the current interval for that variable.[1] The default values are displayed by iSAT when it is called without parameters.

These measures taken to enforce termination have some consequences which are important to understand:

- If iSAT terminates with result '`unsatisfiable`', then – assuming that there are no bugs in the implementation – you can be sure, that the formula is actually unsatisfiable.
- If the tool outputs result '`satisfiable`', then the input formula is satisfiable.

  We note however that iSAT is in general not able to compute a definite answer for all input formulae. This is due to the fact that interval arithmetic combined with splitting (floating-point) intervals yields a highly incomplete deduction calculus. We are currently working on techniques to certify satisfiability in more cases.
- As mentioned before, iSAT cannot decide all given formulae due to the methods employed. To nevertheless provide termination with at least some quantitative result, the tool may stop with result '`unknown`'. Such a result is not merely an abortion, but it is offered a so-called `CANDIDATE SOLUTION`. This means that for the given parameters ε, δ_{abs}, and δ_{rel} the solver could not detect any conflicts within the reported interval valuation. It does *not* mean, however, that the box actually contains a solution, but it can be seen as an *approximative* solution wrt. parameters ε, δ_{abs}, and δ_{rel}. Actually, in most cases there *will* be a solution within the box or at least nearby. If you think that iSAT has reported a spurious solution you should re-run the solver with smaller ε, δ_{abs}, and δ_{rel} in order to confirm (or to refute) the previous result.

5 Tool Options

The options influence the solver behavior and thereby its performance. Understanding the solving algorithm on the abstract level described in the previous section is important when using most of the options listed below.

[1] For intervals of width zero, a new non-conflicting bound cannot yield any progress as the bounds are already as tightly together as they can possibly be.

5.1 General Options

These options are the ones you will most certainly want to experiment with when solving your models.

`--heu` Select the decision heuristic which is used to determine the next variable whose interval is to be split. Possible values are

- `bmc-fwd`, which causes the variables to be sorted by their BMC depth starting with the variables in the 0-th unwinding,
- `bmc-bwd`, causing the last BMC instances to be split first,
- `b-fst`, split the Boolean variables before other integer and floating-point variables,
- `b-lst`, first split the non-Boolean variables,
- `vsids` (Variable State Independent Decaying Sum), sort (and dynamically resort during solving) the variables according to their "activity". In iSAT a decision only splits the box of a variable, and the same decision variable can be taken several times without any conflict. To avoid that the most active variable is selected for all decisions until a conflict happens we attach to each variable a counter. This counter keeps track of how often the variable acted as a decision variable since the last conflict, and we put an upper bound on the counter value of decision variables. We reset those counters when a conflict occurs or if all of them exceed the upper bound.

By combining (at most) two options, a hierarchy of splitting heuristics can be applied. For example by specifying the options `--heu=bmc-fwd --heu=b-fst`, the variables are first sorted by their BMC levels and then within each BMC depth, the Boolean variables come first.

Please note, that this order does not mean that variables that come first in the list according to the heuristic are split down to the minimum splitting width before the next variable is considered for splitting. Variables are actually split round-robin according to the list of variables which is sorted initially (in case of `bmc-fwd`, `bmc-bwd`, `b-fst`, and `b-lst`) or dynamically during solving (in case of `VSIDS`). However, as Booelean variables can only take two different values, they will become point valued after their first split. Therefore, specifying `b-fst` or `b-lst` will cause Boolean variables to be decided before any other variable is split.

`--msw` Set the minimum splitting width, i.e. iSAT will not split intervals if their current width is below this threshold.

`--prabs` Set the absolute progress. Deduced consistent bounds that refine the valuation by less than this value are neglected.

`--prrel` Set the relative progress. Deduced consistent bounds that refine the valuation by less than this percentage of the current interval width are neglected. Especially when having large domains, setting this to a non-zero value can reduce the number of deductions significantly, since even new bounds that yield absolutely large progress may in fact be negligible compared to the total search space.

`--restart` Allow the solver to undo all decisions and start again (albeit without forgetting learnt knowledge that is stored in conflict clauses).

5.2 Advanced Options

The advanced options will not necessarily be of use to you. They can, however, be useful in certain situations and have therefore been made available.

`--idv` When splitting an interval, the midpoint is only one option among many. Setting the interval divide value e.g. to 3 instead of 2, which is the default, will instead cause the split to be asymmetric, yielding a 1/3 and a 2/3 partition of the interval of which iSAT can select one for the ongoing search.

`--ch-sat-often` When set, this option will cause the solver to check whether the current interval valuation actually satisfies the formula before splitting (even if the minimum splitting width is not yet reached). Large boxes, containing satisfying valuations only, can thus be detected early. As this check is quite costly in terms of runtime, it is left best disabled unless an expectation for such large satisfying boxes seems reasonable.

`--pspl` When set, the solver may try to set variables to point values (i.e. deliberately pick one point of the current interval valuation) when splitting. For models containing real-valued variables, one can otherwise not expect the bisection approach to yield point-valued results for non-zero minimum splitting widths.

`--purification` This flag adds a so-called *purification rule* to the solver mechanisms. In case all clauses that contain complex constraints on a variable x are satisfied under the current interval valuation and all clauses in which constraints with lower[2] bounds on x are already satisfied, the solver tries to set the upper bound of the variable to its current lower bound value. The idea behind this is as follows: as the current interval valuation already satisfies all clauses with lower bound constraints on x, there is no need to set x any higher (as there would not be any other clause becoming satisfied by that). On the other hand, setting the upper bound of x to a lower value may very well satisfy some of the constraints in which upper bounds on x occur and thereby also the clauses in which these upper bound constraints are contained. The lower the upper bound for x is set, the bigger is the potential for satisfying such additional upper bound constraints and their clauses. Setting the upper bound of x to its current lower bound thus exploits this potential best and is what the purification rule does.

`--export-cldb` Export the clause database of the solver and do not solve the formula. This function allows to see which formula is actually to be solved (after normalization of the input, Tseitin transformation, BMC unwinding, etc.). However, as can be seen by reading in such an output and again using `--export-cldb`, the frontend will cause the formula that ends up in the solver to grow again due to a second Tseitin transformation (which would be expendable if the frontend would be aware that the formula is already in CNF). This option should thus only be used in case one suspects the frontend of the solver to perform changes to the formula that are wrong or unfavorable.

5.3 Print Options

Print options only affect the output of the solver, not the actual solving process.

`--nosol` Suppress the output of a solution or candidate solution, i.e. only report whether or not the formula is satisfiable but neither a box in case of a satisfying valuation nor a candidate solution box is identified.

`--pdpos` Here you can specify (by a natural number) the number of post-decimal positions of the floating-point numbers given in the resulting interval valuation. The default value is 8.

[2]Replacing upper with lower and lower with upper in this paragraph yields the analogue case that is considered as well, when purification is switched on.

5.4 BMC-related Options

The BMC-related options refer only to input formulae in the bounded model checking format. They have no influence on the solver's behavior if the formula is given in the `EXPR` format.

`--start-depth` The first unwinding depth for which a BMC formula is generated and checked for satisfiability.

`--max-depth` The last unwinding depth for which a BMC formula is generated and checked for satisfiability.

References

[FHT+07] Martin Fränzle, Christian Herde, Tino Teige, Stefan Ratschan, and Tobias Schubert. Efficient Solving of Large Non-linear Arithmetic Constraint Systems with Complex Boolean Structure. *Journal on Satisfiability, Boolean Modeling, and Computation*, 1(2007):209–236, 2007.