SiSAT
Quick Start Guide and Tutorial

Tino Teige

Carl von Ossietzky Universität Oldenburg, Germany
Research Group Hybrid Systems

Transregional Collaborative Research Center "AVACS",
German Research Council, SFB/TR 14

tino.teige@informatik.uni-oldenburg.de

February 28, 2012

1 Introduction

SiSAT is an extension of the iSAT constraint solver [1] to *existential*, *universal*, and *randomized* quantification of finite-domain variables. That is, potential formulae, called *stochastic satisfiability modulo theories* (SSMT) formulae [2,3,4,5,6], look like

$$\exists e \in \{-6, 1, 4\} : \text{Я}_{[-1\to 0.05, 0\to 0.8, 9\to 0.15]} r \in \{-1, 0, 9\} : \forall a \in \{-4, 3, 7\} :$$
$$(5e + a^2 > r/2 \vee \sin(x)^4 \leq \exp(y/2)) \vee (\neg b \vee e + r + a \geq 5xy).$$

The semantics of an SSMT formula $\Phi = \mathcal{Q} : \varphi$ with quantifier prefix $\mathcal{Q}$ and quantifier-free SMT formula φ is defined by the *probability of satisfaction* $Pr(\Phi)$ as follows, where φ is a quantifier-free SMT formula:

$$\begin{array}{lcl} Pr(\varepsilon : \varphi) & = & \begin{cases} 0 & \text{if } \varphi \text{ is unsatisfiable} \\ 1 & \text{if } \varphi \text{ is satisfiable} \end{cases} \\ Pr(\exists x \in \mathcal{D}_x\ \mathcal{Q} : \varphi) & = & \max_{v \in \mathcal{D}_x} Pr(\mathcal{Q} : \varphi[v/x]) \\ Pr(\forall x \in \mathcal{D}_x\ \mathcal{Q} : \varphi) & = & \min_{v \in \mathcal{D}_x} Pr(\mathcal{Q} : \varphi[v/x]) \\ Pr(\text{Я}_{d_x} x \in \mathcal{D}_x\ \mathcal{Q} : \varphi) & = & \sum_{v \in \mathrm{dom}(d_x)} d_x(v) \cdot Pr(\mathcal{Q} : \varphi[v/x]) \end{array}$$

where ε denotes the empty and $\mathcal{Q}$ an arbitrary quantifier prefix.

In [7], the semantics of an SSMT formula was enhanced to deal with expected values. In this case, we only allow existential and randomized quantifiers in prefix $\mathcal{Q}$. Let y be a free variable in Φ with interval domain $[l_y, u_y] \subset \mathbb{R}$. The *maximum conditional expectation of y given Φ*, denoted as $E_y(\Phi)$, is recursively defined as follows:

$$\begin{array}{lcl} E_y(\varepsilon : \varphi) & = & \max_{\sigma \models \varphi} \sigma(y) \\ E_y(\exists x \in \mathcal{D}_x\ \mathcal{Q} : \varphi) & = & \max_{v \in \mathcal{D}_x} E_y(\mathcal{Q} : \varphi[v/x]) \\ E_y(\text{Я}_d x \in \mathcal{D}_x\ \mathcal{Q} : \varphi) & = & \sum_{v \in \mathrm{dom}(d)} d(v) \cdot E_y(\mathcal{Q} : \varphi[v/x]) \end{array}$$

Both latter rules are identical to the ones for defining the (maximum) probability of satisfaction. The first rule generalizes the classical scheme by, instead of just checking for satisfiability of φ, determining the maximal value $\sigma(y) \in [l_y, u_y]$ of the random variable y over all satisfying assignments σ to φ. In case no such satisfying assignment exists, we follow the order-theoretic convention that the maximum over the empty subset of the ordered set $[l_y, u_y]$ is the minimum domain value l_y.

When restricting to existential and randomized quantification then the maximum conditional expectation is a conservative extension of the classical semantics based on (maximum) probability of satisfaction in the sense that $Pr(\mathcal{Q} : \varphi) = E_y(\mathcal{Q} : (\varphi \wedge y = 1))$ with a fresh variable y ranging over $[0, 1]$.

```
1  DECL
2     -- Declaration of non-quantified variables.
3     int   [-100, 100] a;
4     float [-100, 100] b;
5     boole             c;
6     -- Definition of symbolic constant v.
7     define v = 5.2;
8
9  PREFIX
10    -- Declaration of quantified variables.
11    E. x {1, 2, 3}:
12    R. y p = [1 -> 0.6, 2 -> 0.1, 3 -> 0.3]:
13
14 EXPR
15    -- Quantifier-free SMT formula.
16    (x * y <= 4);
17    (x = 1) -> (y <= 2 and  c);
18    (x = 2) -> (y  = 1 and !c);
19    (x = 3) -> (y  = 3 and v*(a - b^2) <= 4.5);
20    c <-> ((0.2*a + sin(b))^3 >= -0.5);
21    (y  = 1 or max(a,b)  < -5.31);
22    (y >= 2 or min(a,b)  >  6.7);
```

Fig. 1. Input file format of SiSAT: example of a single SSMT formula. Note that a line comment starts with the comment delimiter `--`. We further remark that symbols `;` and `!` stand for conjunction with lowest precedence and for negation, respectively.

2 Input formats of SiSAT

2.1 Single SSMT formula

A valid input of SiSAT is a single SSMT formula as shown in the concrete syntax in Fig. 1. The input file format consists of the following three parts:

1. `DECL`: This part contains declarations of all non-quantified variables which are interpreted as innermost existentially quantified. Supported variable types are `int`, `float`, and `boole`. Note that the range of each float and integer variable has to be bounded by an interval. Furthermore, you can define symbolic constants in this section, e.g. `define p = 5.2;`.
2. `PREFIX`: In this part, you can define the prefix of quantified variables. Note that the order of variables in this part is the same as the order in the resulting prefix. For instance, $\exists x \in \{1,2\}$ is encoded as `E. x {1,2}:`, $\forall y \in \{1,2\}$ as `A. x {1,2}:`, and $\mathbb{R}_{[1\to 0.8, 2\to 0.2]} z \in \{1,2\}$ as `R. z p = [1 -> 0.8, 2 -> 0.2]:`.
3. `EXPR`: This section contains the SMT formula which is a Boolean combination of non-linear arithmetic constraints. The syntax is the same as for the iSAT tool.[1]

2.2 Probabilistic bounded model checking

The probabilistic bounded model checking mode allows for the analysis of discrete time probabilistic hybrid automata [2,6] which are a generalization of hybrid automata in the sense that a transition can be connected to a probabilistic choice. Such probabilistic choices lead to potentially different successor modes and perform different discrete actions as shown in Fig. 2.

There is a translation scheme to encode a probabilistic hybrid automaton into an SSMT formula. To do so, the main observation is that a *non-deterministic* choice of a transition corresponds to *existential* or *universal* quantification depending on whether we want to resolve the non-determinism s.t. the reachability probability

[1] An iSAT manual can be found on `http://isat.gforge.avacs.org/documentation/quickstartguide.pdf`.

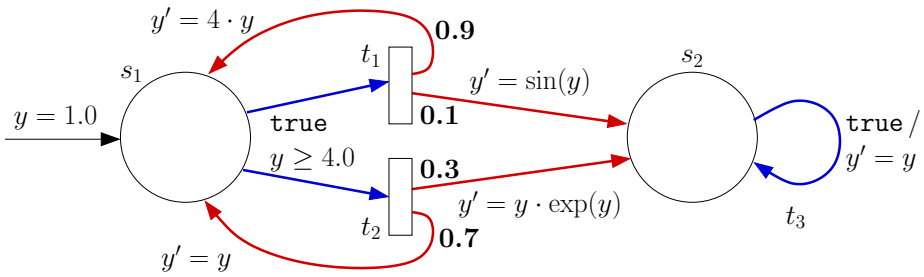

Fig. 2. A probabilistic hybrid automaton.

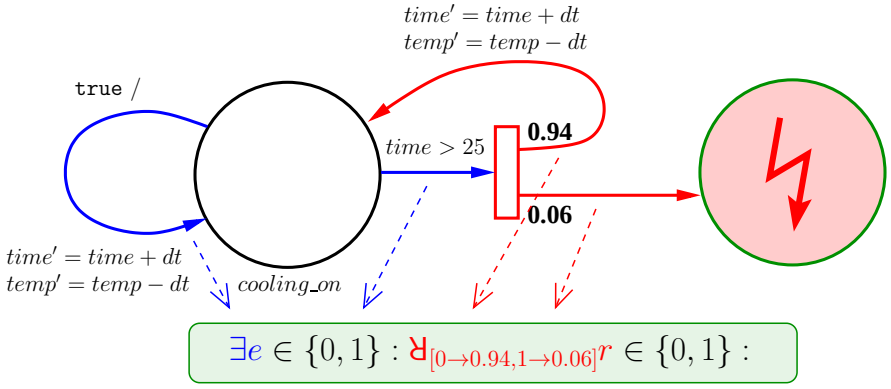

Fig. 3. Encoding non-deterministic and probabilistic choices in a PHA by existential and randomized quantification in SSMT.

should be *maximized* or *minimized*, respectively. A *probabilistic* choice can be encoded using *randomized* quantification. For an example of that idea, confer Fig. 3. A detailed description can be found in [2,6].

Probabilistic bounded model checking (PBMC) for PHAs aims at computing the reachability probability of a given system with a k-bounded behavior. That is, PBMC considers all system runs of bounded length k which

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

Given a parameter k, the idea of PBMC is to construct an SSMT formula Φ_k s.t. the probability of satisfaction of Φ_k is p if and only if the probability of reaching a specified target state in the given system within k steps is p.

For specifying PBMC tasks, SiSAT has a second input file format which consists of five parts:

1. `DECL`: As above, this part contains declarations of all non-quantified variables of the system, and definitions of constants.
2. `INIT`: This part is a formula describing the initial state of the system to be investigated.
3. `DISTR`: Here, the non-deterministic and probabilistic choices in the system are characterized by a suitable quantifier prefix.

4. `TRANS`: This formula describes the transition relation of the system. Non-quantified variables may occur in primed or unprimed form. A primed variable represents the value of that variable in the successor step, i.e. after the transition has taken place. The semicolon which terminates each constraint can be read as an AND-operator.
5. `TARGET`: This formula encodes the state(s) whose reachability is to be checked.

The SiSAT encodings of the PHA from Fig. 2 with objective of maximizing and minimizing the reachability probability are shown in Fig. 4 and Fig. 5, respectively.

Note that the following features are *not* supported at the moment:

- *Continuous-time* and *ordinary differential equations*. (The continuous behavior of PHA must be overapproximated.)
- *Continuous* (non-deterministic and stochastic) *inputs*.
- *Non-deterministic transition actions* for continuous variables.

3 Tool usage

SiSAT is a Linux command-line tool and called with `./sisat --i example.hys`.[2] By calling `./sisat`, the help menu is printed, see Fig. 6. The tool output when having called SiSAT on the SSMT formula from Fig. 1 is shown in Fig. 7.

4 Tutorial

4.1 Simple dice game

One of the simplest random experiments is *throwing a die*. When we consider a six-sided cubic die with numbers one to six on its faces then all numbers occur with the same probability $\frac{1}{6}$. To model a die, we can use randomized quantification, namely by declaring `R. z p = [1 -> 0.16666667, ..., 6 -> 0.16666667]:` in the `PREFIX` part.[3]

Let us now describe a simple dice game: the whole game comprises just one throw of one die. If the number on the die is 1 then you get no payout. For numbers 2, 3, 4, 5, and 6, your payoff is 5 €, 10 €, 15 €, 20 €, and 200 €. This is encoded as a single SSMT formula as follows:

```
1  DECL
2      float [0,10000000] payout;
3
4  PREFIX
5      R. die p =  [   1 -> 0.16666667,
6                      2 -> 0.16666667,
7                      3 -> 0.16666667,
8                      4 -> 0.16666667,
9                      5 -> 0.16666667,
10                     6 -> 0.16666667
11                 ]:
12
13 EXPR
14     die = 1 -> payout = 0;
15     die = 2 -> payout = 5;
16     die = 3 -> payout = 10;
17     die = 4 -> payout = 15;
18     die = 5 -> payout = 20;
19     die = 6 -> payout = 200;
```

[2] Please note that the filename must actually end with suffix ".hys".

[3] Please note that probability $\frac{1}{6}$ is just approximated from above by the decimal number `0.16666667`. This fact implies that also all analysis results will be approximated from above, which is considered to be a safe approximation in many applications. It is however possible to assign probability intervals to values of randomized variables such that the actual probability is safely included in the interval. For the sake of simplicity, we neglect this issue here.

```
DECL
    -- Locations:
        boole s1, s2;
    -- Continuous state components:
        float [-100,100] y;
    -- Variables encoding the guards:
        boole guard_trans1, guard_trans2, guard_trans3;
INIT
    -- Initial state: (s1, y=1.0).
        s1 and not s2;
        y = 1.0;
DISTR
    -- Variable encoding non-deterministic choices
    -- of transitions:
    -- Maximize over transitions.
        E. trans {1,2,3}:
    -- Variables encoding probabilistic choices
    -- of transitions:
        R. prob_t1 p =  [   0 -> 0.9,
                            1 -> 0.1
                        ]:
        R. prob_t2 p =  [   0 -> 0.3,
                            1 -> 0.7
                        ]:
TRANS
    -- System in exactly one state after transition.
        s1' + s2' = 1;
    -- Overapproximation of the system behavior!
    -- Transition 1
        -- Encode guard:
            guard_trans1 <-> true;
        -- Impact of probabilistic choice 0 (0.9):
            (s1 and trans = 1 and prob_t1 = 0 and guard_trans1) ->
                (y' = 4*y and s1');
        -- Impact of probabilistic choice 1 (0.1):
            (s1 and trans = 1 and prob_t1 = 1 and guard_trans1) ->
                (y' = sin(y) and s2');
    -- Transition 2
        -- Encode guard:
            guard_trans2 <-> y >= 4.0;
        -- Impact of probabilistic choice 0 (0.3):
            (s1 and trans = 2 and prob_t2 = 0 and guard_trans2) ->
                    (y' = y*exp(y) and s2');
        -- Impact of probabilistic choice 1 (0.7):
            (s1 and trans = 2 and prob_t2 = 1 and guard_trans2) ->
                    (y' = y and s1');
    -- Transition 3 (trivial self loop for target location s2)
        -- Encode guard:
            guard_trans3 <-> true;
        -- Impact of transition:
            (s2 and trans = 3 and guard_trans3) -> (y' = y and s2');
    -- For maximum reachability probability:
    -- Constrain the behavior s.t. for each location there
    -- is at least one outgoing transition which must be taken,
    -- Realized by:     One transition is taken and
    --                  corresponding guard holds.
        -- Location s1:
            s1 -> ((guard_trans1 and trans = 1) or
                   (guard_trans2 and trans = 2));
        -- Location s2:
            s2 -> (guard_trans3 and trans = 3);
TARGET
    -- Note: Target location must have a trivial self loop.
        s2;
```

Fig. 4. Input format of SiSAT: Transition relation for maximizing probability

```
DECL
    -- Locations:
        boole s1, s2;
    -- Continuous state components:
        float [-100,100] y;
    -- Variables encoding the guards:
        boole guard_trans1, guard_trans2, guard_trans3;
INIT
    -- Initial state: (s1, y=1.0).
        s1 and not s2;
        y = 1.0;
DISTR
    -- Variable encoding non-deterministic choices
    -- of transitions:
    -- Minimize over transitions.
        A. trans {1,2,3}:
    -- Variables encoding probabilistic choices
    -- of transitions:
        R. prob_t1 p =  [   0 -> 0.9,
                            1 -> 0.1
                        ]:
        R. prob_t2 p =  [   0 -> 0.3,
                            1 -> 0.7
                        ]:
TRANS
    -- System in exactly one state after transition.
        s1' + s2' = 1;
    -- Overapproximation of the system behavior!
    -- Transition 1
        -- Encode guard:
            guard_trans1 <-> true;
        -- Impact of probabilistic choice 0 (0.9):
            (s1 and trans = 1 and prob_t1 = 0 and guard_trans1) ->
                (y' = 4*y and s1');
        -- Impact of probabilistic choice 1 (0.1):
            (s1 and trans = 1 and prob_t1 = 1 and guard_trans1) ->
                (y' = sin(y) and s2');
    -- Transition 2
        -- Encode guard:
            guard_trans2 <-> y >= 4.0;
        -- Impact of probabilistic choice 0 (0.3):
            (s1 and trans = 2 and prob_t2 = 0 and guard_trans2) ->
                    (y' = y*exp(y) and s2');
        -- Impact of probabilistic choice 1 (0.7):
            (s1 and trans = 2 and prob_t2 = 1 and guard_trans2) ->
                    (y' = y and s1');
    -- Transition 3 (trivial self loop for target location s2)
        -- Encode guard:
            guard_trans3 <-> true;
        -- Impact of transition:
            (s2 and trans = 3 and guard_trans3) -> (y' = y and s2');
    -- For minimum reachability probability:
    -- Constrain the behavior s.t. for each location there
    -- is at least one outgoing transition which can be taken,
    -- Realized by:     One guard must hold.
        -- Location s1:
            s1 -> (guard_trans1 or guard_trans2);
        -- Location s2:
            s2 -> (guard_trans3);
TARGET
    -- Note: Target location must have a trivial self loop.
        s2;
```

Fig. 5. Input format of SiSAT: Transition relation for minimizing probability

```
This is SiSAT 1.0.

Usage: sisat --i <inputfile.hys> [options]

General iSAT options:
...
 --msw=[real]   : Minimum splitting width of an interval
                  (must be > 2*prabs, default: 0.01)
 --prabs=[real] : Neglect deductions with absolute progress
                  less than prabs (default: 0.001)
...
 --strongsat    : Enables check for strong satisfaction
...

BMC-related options:
  --start-depth  : BMC starting depth (default: 0)
  --max-depth    : BMC maximal number of unwindings
                   (default: 2147483647)

SiSAT options:
  --lt=[real]    : Lower threshold for satisfaction probability
                   (values: 0..1; default: 0)
  --ut=[real]    : Upper threshold for satisfaction probability
                   (values: 0..1; default: 1)
  --no-appr-sol  : Enables that approximate solutions yield
                   probability interval [0,1] (default: [1,1])
  --heu-quant=
       exceed-ut : Aims at exceeding upper threshold (default)
       miss-lt   : Aims at missing lower threshold
       natural   : Natural order
       random    : Random choice
  --no-pur-quant : Disables purification rule for quantified
                   variables (default: enabled)
  --no-sdb       : Disables solution-directed backjumping
                   (default: enabled)
  --pr-cach      : Enables caching of probability results
                   of subproblems (default: disabled)
  --sol-cach     : Enables caching solutions (PBMC)
                   (default: disabled)
  --accur=[real] : Accuracy of probability result
                   (values: 0..1; default: 0)
  --dont-stop    : Search will be continued even if a
                   counter-example was found
  --cond-exp=var : Specifies the variable for which the conditional
                   expectation should be computed. For transition
                   systems, the variable instance of the last step
                   depth is taken.
  --lt-exp=[real]: Lower threshold for conditional expectation
                   (default: minimum value of domain of 'var')
  --ut-exp=[real]: Upper threshold for conditional expectation
                   (default: maximum value of domain of 'var')
```

Fig. 6. Excerpt of the help menu of SiSAT.

Of course, you have to pay a stake for playing, say 50 €. That is, you may quadruple your stake in case of a 6, but you loose money in all other cases. To find out what the probability of winning some money is, we add the constraint `payout > 50;` to the `EXPR` part and run SiSAT on the corresponding input file "die.hys", i.e. `./sisat --i die.hys`. The output will be:

```
...
RESULT:
    Probability of satisfaction lies in [0.16666666,0.16666668]
...
```

We may also be interested in the expected value of the payout. To compute this quantitative measure with SiSAT, we first remove the latter constraint, i.e. `payout > 50;`, from the input file, and then call `./sisat --i die.hys --exp-val payout`. The result will be:

```
...
RESULT:
    Conditional expectation lies in [41.66666749,41.66666751]
...
```

```
This is SiSAT 1.0.

Settings.
Minimum splitting width    : 0.01
Absolute bound progress    : 0.001
Relative bound progress    : 0
Variable decision order    : natural
Strong satisfaction check  : enabled
Target thresholds          : 0 / 1
Quant value decision order : exceeding upper threshold
Purification (quant vars)  : enabled
Solution-dir. backjumping  : enabled
Caching probabilities      : disabled
Caching solutions (PBMC)   : disabled
Required accuracy          : 0
Probability 1 for approximate solutions is enabled.

SOLVING:
    k = 0
RESULT:
    Probability of satisfaction lies in [0.69999999,0.70000001]

Statistics.
Number of variables                    : 32
    Existential variables              : 1
    Universal variables                : 0
    Random variables                   : 1
    Boolean variables                  : 12
    Integer variables                  : 2
    Real variables                     : 16
Number of complex bounds               : 12
Number of problem clauses              : 52
Number of decisions (current / total)  : 105 / 105
(of these:) Number of decisions on
            quantified variables       : 4 / 4
(of these:) Number of decisions on
            non-singleton domains      : 3 / 3
...
SAT / UNKNOWN                          : 100% (2 / 0)
Probability of satisfaction in         : [0.69999999,0.70000001]
Accuracy of result                     : 0

Time solver                            : 0.02 / 0.02 sec
```

Fig. 7. Abridged output of SiSAT after having solved the single SSMT formula from Figure 1 using the default settings and option `--strongsat`.

From this outcome, we conclude that the stake is to high and that we loose about 8.34 € in average.

4.2 Yet another dice game

We now consider a dice game with several rounds. We start with round 1 and the payout is initially zero. Let us be in round i. If throwing one die yields number 1 then the amount of payoff is kept and the game is over if and only if the current round is odd, i.e. i is an odd number. If i is even, we go on with the next throw. For number 2, we have the same rule as for 1 except for the difference that the game is over iff i is even. If we throw number 3, nothing happens and we continue. In case of number 4, we get 50 cent per round we have played so far, i.e. $i \cdot 50$ cent, and the game continues. If we have number 5 then the payout is increased by 5 € if round is odd, otherwise there is no gain. The game goes on. Finally, if the die shows number 6 then we obtain 5 € and proceed with next round. Here is the SiSAT encoding of the dice game:

```
DECL
    float [0,10000000] payout;

    -- denotes round
    int   [0,10000000] round;
```

```
6
7       -- denotes whether odd round
8       boole is_odd_round;
9
10      -- denotes whether game is over
11      boole gameover;
12
13  INIT
14      -- payout is initially zero.
15      payout = 0;
16      -- initially game is not over
17      !gameover;
18      -- games starts with round 1
19      round = 1;
20      -- round 1 is odd
21      is_odd_round;
22
23  DISTR
24      R. die p =  [   1 -> 0.16666667,
25                      2 -> 0.16666667,
26                      3 -> 0.16666667,
27                      4 -> 0.16666667,
28                      5 -> 0.16666667,
29                      6 -> 0.16666667
30                  ]:
31
32  TRANS
33      -- number 1: no gain, game over iff odd round
34      (!gameover and die = 1) -> (
35          payout' = payout and
36          (gameover' <-> is_odd_round)
37      );
38      -- number 2: no gain, game over iff even round
39      (!gameover and die = 2) -> (
40          payout' = payout and
41          (gameover' <-> !is_odd_round)
42      );
43      -- number 3: no gain, game continues
44      (!gameover and die = 3) -> (
45          payout' = payout and
46          !gameover'
47      );
48      -- number 4: gain 50 Cent per round, game continues
49      (!gameover and die = 4) -> (
50          payout' = payout + 0.5*round and
51          !gameover'
52      );
53      -- number 5: gain 5 Euro if odd round else no gain,
54      --           game continues
55      (!gameover and die = 5) -> (
56          (is_odd_round -> payout' = payout + 5) and
57          (!is_odd_round -> payout' = payout) and
58          !gameover'
59      );
60      -- number 6: gain 5 EURO, game continues
61      (!gameover and die = 6) -> (
62          payout' = payout + 5 and
63          !gameover'
64      );
65
66      -- if game was over than it remains over
67      gameover -> (
68          payout' = payout and
69          gameover'
70      );
71
72      -- next round
73      round' = round + 1;
74
75      -- toggle flag
76      is_odd_round' <-> !is_odd_round;
77
78  TARGET
79      -- add target states to be reached here
```

This game gives a lot of questions to be analyzed. For instance, what is the probability that the game is over after k rounds. This issue can be tackled by adding the constraint `gameover;` to the `TARGET` and calling SiSAT on this file like `./sisat --i another_die.hys`. The condensed output is:

```
k = 0
Probability of satisfaction lies in [0,0]
k = 1
Probability of satisfaction lies in [0.16666666,0.16666668]
k = 2
Probability of satisfaction lies in [0.30555556,0.30555557]
k = 3
Probability of satisfaction lies in [0.42129631,0.42129632]
k = 4
Probability of satisfaction lies in [0.51774693,0.51774694]
k = 5
Probability of satisfaction lies in [0.59812245,0.59812246]
k = 6
Probability of satisfaction lies in [0.66510206,0.66510207]
k = 7
Probability of satisfaction lies in [0.7209184,0.72091841]
k = 8
Probability of satisfaction lies in [0.76743201,0.76743202]
...
```

As you might have noticed, the runtimes quickly increase for larger BMC depths k. This is due to the quickly increasing search space of the problem. In most cases, runtimes can be reduced by using the idea of *thresholding*, where we are not interested in the exact probability but just in whether the exact probability is below some lower threshold lt or above some upper threshold ut. In our scenario, say we just want to know whether the probability of a gameover is below or above 75%. Then, we may run SiSAT with parameters `--lt 0.75 --ut 0.75`, where the output specifies whether the thresholds are missed or exceeded.

```
...
k = 7
Probability of satisfaction less than
lower target threshold! Witness: [0,0]
k = 8
Probability of satisfaction greater than
upper target threshold! Witness: [1,1]
Upper target threshold 0.75 is exceeded
by lower bound of probability (1).
```

We have found out that the probability of a game over rises quickly, e.g. after 8 rounds this probability is above 75%. What we are not aware of so far is the payout. As you can easily see from the game description, the payout is monotonically increasing in every possible run of the game. Say the stake for playing is 5 € and we want to know what the probability of winning at least 5 € is. That is, we need to check how likely is a payout of at least 10 €. We thus remove constraint `gameover;` from the `TARGET`, add instead the new constraint `payout >= 10;`, and call SiSAT. The resulting output is:

```
k = 0
Probability of satisfaction lies in [0,0]
k = 1
Probability of satisfaction lies in [0,0]
k = 2
Probability of satisfaction lies in [0.05555555,0.05555556]
k = 3
Probability of satisfaction lies in [0.15740741,0.15740742]
k = 4
Probability of satisfaction lies in [0.2013889,0.20138891]
k = 5
Probability of satisfaction lies in [0.27443417,0.27443418]
k = 6
Probability of satisfaction lies in [0.30774179,0.3077418]
k = 7
Probability of satisfaction lies in [0.34929415,0.34929416]
k = 8
Probability of satisfaction lies in [0.36411659,0.3641166]
...
```

We may again rephrase the problem into a decision problem. Assume that we are willing to play the game in case the chance of winning 5 € is greater than 35%. To answer this question, we need to call SiSAT with parameters `--lt 0.35 --ut 0.35`. Then, we obtain:

```
...
k = 7
Probability of satisfaction less than
lower target threshold! Witness: [0,0]
k = 8
Probability of satisfaction greater than
upper target threshold! Witness: [1,1]
Upper target threshold 0.35000001 is exceeded
by lower bound of probability (1).
```

Above analysis results are concerned with pure probabilities. However, we may be keen on more meaningful information like the expected payout after k rounds. As we want to consider all possible situations of the game, we delete all constraints from the `TARGET` and add constraint `true;`, the latter comprising all states of the game. After that, we call SiSAT with option `--cond-exp payout`. The output will be:

```
k = 0
Conditional expectation lies in [0,0]
k = 1
Conditional expectation lies in [1.75000003,1.75000004]
k = 2
Conditional expectation lies in [2.58333343,2.58333344]
k = 3
Conditional expectation lies in [3.91435208,3.91435209]
k = 4
Conditional expectation lies in [4.58950654,4.58950655]
k = 5
Conditional expectation lies in [5.59420066,5.59420067]
k = 6
Conditional expectation lies in [6.1300376,6.13003761]
k = 7
Conditional expectation lies in [6.88355827,6.88355828]
k = 8
Conditional expectation lies in [7.30218095,7.30218096]
...
```

The idea of thresholding, as mentioned above for satisfaction probabilities, is also availbale for expectations. Say that we just want to know whether the expected value of payout is below or above 7 €, then we can call SiSAT with thresholding options `--lt-exp 7 --ut-exp 7`. We get:

```
...
k = 7
Conditional expectation less than lower target threshold!
Witness: [6.74466936,6.74466937]
k = 8
Conditional expectation greater than upper target threshold!
Witness: [10000000,10000000]
Upper target threshold 7 is exceeded by lower bound
of expectation (10000000).
```

4.3 Controlling a heater

We will now consider a bit more complex example, namely the simple heater model from Fig. 8. The goal is to control a heater, i.e. to switch it on and off, in order to keep the temperature (of some room) between 60° and 100°. Initially, the temperature T is 80° and the heater is on, i.e. the automaton is in location "heat_on". If the temperature exceeds 100° then the location "fail" is entered. The latter encodes that a bad state is reached. If T is below 90° then we may opt between keeping the heater on or switching it off. That is, this model includes *non-deterministic* choices. This free choice of switching on or off can be encoded by existential quantification, i.e. by `E. switch {YES, NO}:`. If T is greater than 90° then we need to switch off the device. A similar behavior is given when the heater is off, i.e. if the automaton is in location "heat_off" (cf. Fig. 8). The probabilistic aspect here is that a mode change of the heating device does not work in all cases. More precisely, with a probability of 3% switching off fails, and with a bit higher probability of 6% switching on does.

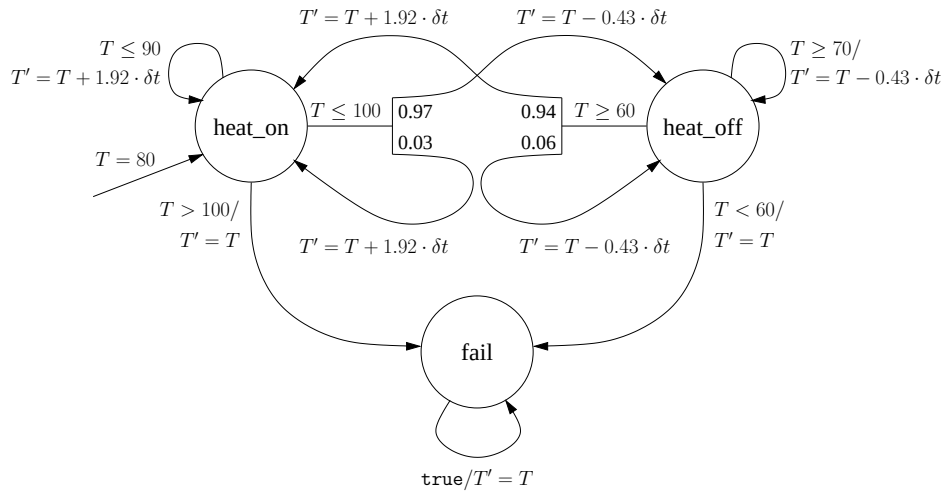

Fig. 8. Model of a heater

The continuous evolution of the temperature is specified by rate 1.92 if the heater is on, and by rate -0.43 if the heater is off. The discretization parameter in the model is given by δt. Let us fix $\delta t = 2.5$, i.e. we sample the system behavior each 2.5 time units. The heater model is encoded in the SiSAT language as follows.

```
DECL
    -- time step delta t
    define DT = 2.5;
    -- temperature
    float [-10000000,10000000] T;
    -- locations
    boole heat_on;
    boole heat_off;
    boole fail;
    -- global time
    float [0,1000000000] time;
    -- define symbolic values
    define YES = 1;
    define NO = 2;

INIT
    -- initial settings
    T = 80;
    heat_on;
    !heat_off;
    !fail;
    time = 0;

DISTR
    E. switch {YES, NO}:
    R. fail1 p = [YES -> 0.03, NO -> 0.97]:
    R. fail2 p = [YES -> 0.06, NO -> 0.94]:

TRANS
    -- automaton is in one location after next step
    heat_on' + heat_off' + fail' = 1;
    -- location: heat_on
    (heat_on and T > 100) -> (
        fail' and
        T' = T and
        time' = time
    );
    (heat_on and T <= 90 and switch = NO) -> (
        heat_on' and
        T' = T + 1.92*DT and
        time' = time + DT
    );
    (heat_on and T <= 100 and switch = YES and fail1 = YES) -> (
        heat_on' and
```

```
45          T' = T + 1.92*DT and
46          time' = time + DT
47      );
48      (heat_on and T <= 100 and switch = YES and fail1 = NO) -> (
49          heat_off' and
50          T' = T - 0.43*DT and
51          time' = time + DT
52      );
53      -- we need to switch if 90 < T <= 100.
54      (heat_on and 90 < T and T <= 100) -> (switch = YES);
55      -- location: heat_off
56      (heat_off and T < 60) -> (
57          fail' and
58          T' = T and
59          time' = time
60      );
61      (heat_off and T >= 70 and switch = NO) -> (
62          heat_off' and
63          T' = T - 0.43*DT and
64          time' = time + DT
65      );
66      (heat_off and T >= 60 and switch = YES and fail2 = YES) -> (
67          heat_off' and
68          T' = T - 0.43*DT and
69          time' = time + DT
70      );
71      (heat_off and T >= 60 and switch = YES and fail2 = NO) -> (
72          heat_on' and
73          T' = T + 1.92*DT and
74          time' = time + DT
75      );
76      -- we need to switch if 60 <= T < 70.
77      (heat_off and 60 <= T and T < 70) -> (switch = YES);
78      -- location: fail
79      (fail) -> (
80          fail' and
81          T' = T and
82          time' = time
83      );
84
85  TARGET
86      fail;
```

Constraint `fail;` in `TARGET` denotes that we are interested in the probability of reaching location "fail". Due to encoding the non-determinism by existential quantification, all non-deterministic choices are resolved such that the reachability probability is maximized. Running SiSAT yields:

```
1  ...
2  k = 5
3  Probability of satisfaction lies in [0,0]
4  k = 6
5  Probability of satisfaction lies in [0.00089999,0.00090001]
6  k = 7
7  Probability of satisfaction lies in [0.05560799,0.05560801]
8  k = 8
9  Probability of satisfaction lies in [0.89026971,0.89026973]
10 k = 9
11 Probability of satisfaction lies in [0.99023217,0.99023218]
12 k = 10
13 Probability of satisfaction lies in [0.99023217,0.99023218]
14 k = 11
15 Probability of satisfaction lies in [0.99050189,0.9905019]
16 ...
```

These results show that the maximum or worst-case probability of "fail" increases quickly. This could either mean that the discretization parameter δt is to coarse or that the whole model must be refined to work as desired.

5 A vital encoding trick

Consider the very simple probabilistic automaton from Fig. 9 and let us be interested in computing the probability of reaching location s_2. When having called SiSAT on this encoding

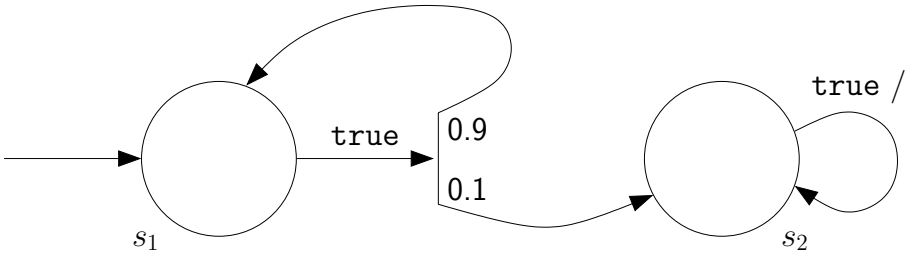

Fig. 9. A simple probabilistic (hybrid) automaton.

```
DECL
    boole s1, s2;

INIT
    s1 and !s2;

DISTR
    R. pc p = [1 -> 0.9, 2 -> 0.1]:

TRANS
    -- Transition relation.
    (s1 and pc = 1) -> ( s1' and !s2');
    (s1 and pc = 2) -> (!s1' and  s2');
     s2             -> (!s1' and  s2');

TARGET
    s2;
```

with tool option `--no-sdb`, i.e. disabling the optimization of *solution-directed back-jumping*, the tool solved the problem up to step depth 22 in about 387 seconds on 2.67 GHz Intel Core machine with 8 GByte physical memory, while it has detected more than 4 million solutions.

When using the idea of "disabling" quantifiers whenever their actual values become irrelevant with the objective of reducing the potential search space, as described in detail in Section 6.6.3 of [8], runtime can be improved significantly. For the above example, the modified SiSAT encoding looks like:

```
DECL
    boole s1, s2;
    -- Symbolic constant to "disable" quantifiers.
    define OFF = 0;

INIT
    s1 and !s2;

DISTR
    -- Add value OFF with probability 1 to domain.
    R. pc p = [OFF -> 1, 1 -> 0.9, 2 -> 0.1]:

TRANS
    -- Transition relation.
    (s1 and pc = 1) -> ( s1' and !s2');
    (s1 and pc = 2) -> (!s1' and  s2');
     s2             -> (!s1' and  s2');

    -- "Disable" randomized quantifier if system is in s2.
     s2 -> pc = OFF;
    -- "Enable" randomized quantifier if system is not in s2.
    !s2 -> pc != OFF;

TARGET
    s2;
```

Observe that the probabilities of the "relaxed" randomized quantifier `R. pc p = [OFF -> 1, 1 -> 0.9, 2 -> 0.1]:` add up to a value strictly greater than 1,

namely 2. Thus, the randomized quantifier does no longer specify a probability distribution. However, if there are some constraints in the `TRANS` section that exclude values from the domain of such a randomized variable such that all the remaining values have a probability mass at most 1 then a valid probability distribution is retrieved. In the example above, this is done by constraints `s2 -> pc = OFF;` and `!s2 -> pc != OFF;`.

When having called SiSAT with the same tool options but on the modified encoding, the tool solved the problem up to step depth 100 within 4 seconds. With regard to the tool performance up to step depth 22, solving time has reduced by *three* orders of magnitude to 0.07 seconds while the number of solutions has decreased by *five* orders of magnitude to 22.

That is to say, exploiting the idea of "disabling" quantifiers whenever they become irrelevant within SiSAT encodings of PHAs is a vital issue for tool performance. Note that the same trick is applicable to existential quantifiers.

References

1. Fränzle, M., Herde, C., Teige, T., Ratschan, S., Schubert, T.: Efficient solving of large non-linear arithmetic constraint systems with complex Boolean structure. Journal on Satisfiability, Boolean Modeling and Computation – Special Issue on SAT/CP Integration **1**(3–4) (2007) 209–236
2. Fränzle, M., Hermanns, H., Teige, T.: Stochastic satisfiability modulo theory: A novel technique for the analysis of probabilistic hybrid systems. In Egerstedt, M., Mishra, B., eds.: Proceedings of the 11th International Conference on Hybrid Systems: Computation and Control (HSCC'08). Volume 4981 of Lecture Notes in Computer Science., Springer (2008) 172–186
3. Teige, T., Fränzle, M.: Stochastic satisfiability modulo theories for non-linear arithmetic. In Perron, L., Trick, M.A., eds.: Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 5th International Conference (CPAIOR 2008). Volume 5015 of Lecture Notes in Computer Science., Springer (2008) 248–262
4. Fränzle, M., Teige, T., Eggers, A.: Engineering constraint solvers for automatic analysis of probabilistic hybrid automata. Journal of Logic and Algebraic Programming **79**(7) (2010) 436–466
5. Teige, T., Fränzle, M.: Constraint-based analysis of probabilistic hybrid systems. In Giua, A., Mahulea, C., Silva, M., Zaytoon, J., eds.: Proceedings of the 3rd IFAC Conference on Analysis and Design of Hybrid Systems (ADHS 2009), IFAC (2009) 162–167
6. Teige, T., Eggers, A., Fränzle, M.: Constraint-based analysis of concurrent probabilistic hybrid systems: An application to networked automation systems. Nonlinear Analysis: Hybrid Systems **5**(2) (2011) 343–366
7. Fränzle, M., Teige, T., Eggers, A.: Satisfaction meets expectations: Computing expected values of probabilistic hybrid systems with SMT. In Méry, D., Merz, S., eds.: Proceedings of the 8th International Conference on Integrated Formal Methods (IFM 2010). Volume 6396 of Lecture Notes in Computer Science., Springer (2010) 168–182
8. Teige, T.: Stochastic Satisfiability Modulo Theories: A Symbolic Technique for the Analysis of Probabilistic Hybrid Systems. Doctoral dissertation, Carl von Ossietzky Universität Oldenburg, Germany to appear.