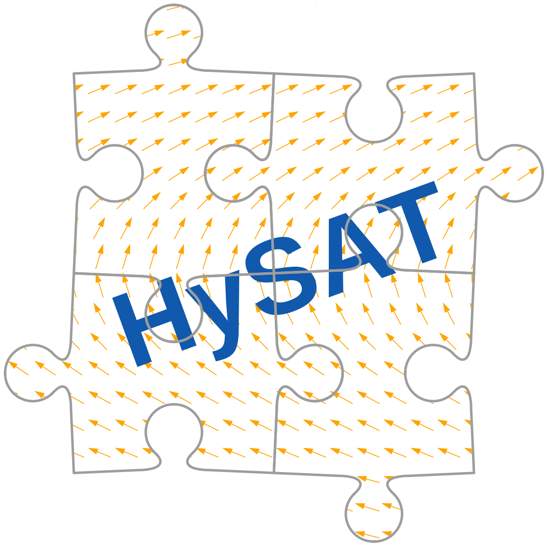


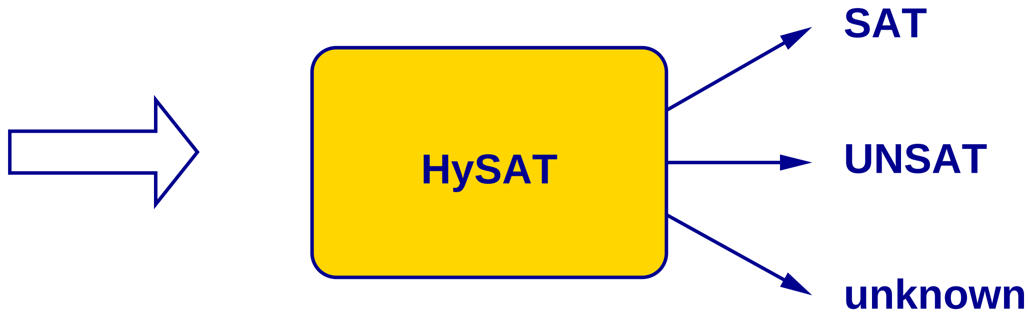
- what you can use it for
- how it works
- example from application domain
- final remarks

What you can use it for

- Satisfiability checker for quantifier-free Boolean combinations of arithmetic constraints over the reals and integers

- Can deal with nonlinear constraints and thousands of variables

$$
\begin{aligned}
& (\neg b \lor \neg c) \\
\land\ & (b \rightarrow \sin(x) \cdot y < 7.2) \\
\land\ & (\sqrt{2x - y} = 8 \lor c) \\
\land\ & (i^2 = 3j - 5)
\end{aligned}
$$

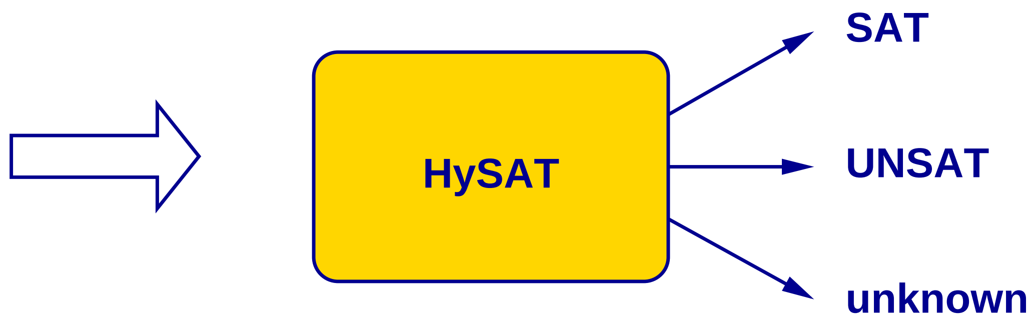


What you can use it for

- Satisfiability checker for quantifier-free Boolean combinations of arithmetic constraints over the reals and integers

- Can deal with nonlinear constraints and thousands of variables

$$
\begin{aligned}
&(\neg b \lor \neg c) \\
\land\ &(b \rightarrow \sin(x) \cdot y < 7.2) \\
\land\ &(\sqrt{2x - y} = 8 \lor c) \\
\land\ &(i^2 = 3j - 5)
\end{aligned}
$$



Allows mixing of Boolean, integer, and float variables in the same arithmetic constraint

$\rightarrow$ e.g. pseudo–Boolean constraints possible

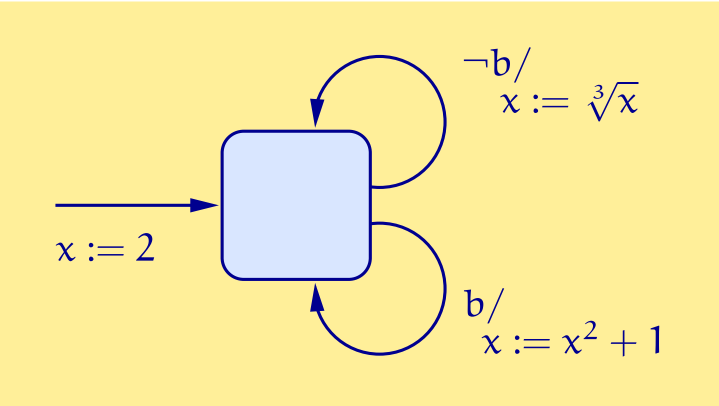
What you can use it for: Solving formulas

$$
\begin{aligned}
& (\neg b \lor \neg c) \\
\land\ & (b \rightarrow \sin(x) \cdot y < 7.2) \\
\land\ & (\sqrt{2x - y} = 8 \lor c) \\
\land\ & (i^2 = 3j - 5)
\end{aligned}
$$

```
DECL
    boole b, c;
    float [-100, 100] x, y;
    int [-100, 100] i, j;

EXPR
    !b or !c;
    b -> sin(x) * y < 7.2;
    nrt(2*x - y, 2) = 8 or c;
    i^2 = 3*j - 5;
```


```
SOLUTION:
    b (boole):
       @0:  [1, 1]

    c (boole):
       @0:  [0, 0]

    i (int):
       @0:  [-11, -11]

    j (int):
       @0:  [42, 42]

    x (float):
       @0:  [12.4357, 12.4357]

    y (float):
       @0:  [-39.1287, -39.1287]
```

What you can use it for: Bounded Model Checking




```
DECL
   boole b;
   float [0.0, 1000.0] x;

INIT
   -- Characterization of initial state.
   x = 2.0;

TRANS
   -- Transition relation.
   b -> x' = x^2 + 1;
   !b -> x' = nrt(x, 3);

TARGET
   -- State(s) to be reached.
   x >= 3.14 and x <= 3.15;
```

Safety property:

There's no sequence of input values such that $3.14 \leq x \leq 3.15$

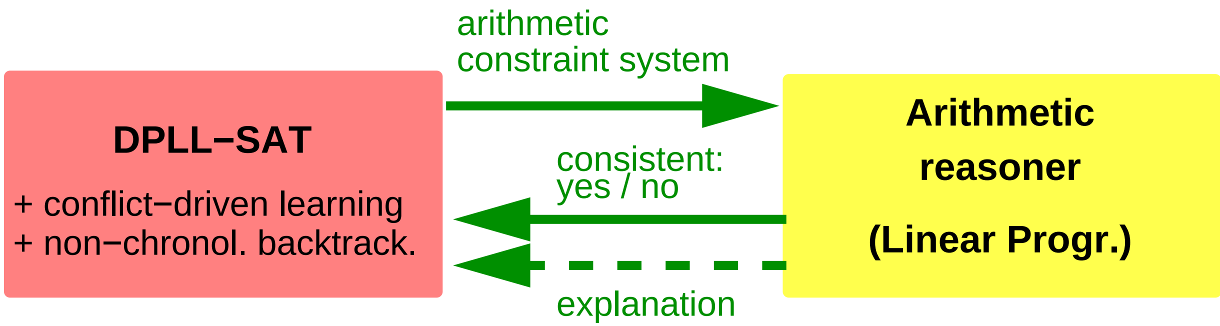
HySAT

COUNTEREXAMPLE

```
SOLUTION:
   b (boole):
      @0:  [0, 0]
      @1:  [1, 1]
      @2:  [1, 1]
      @3:  [0, 0]
      @4:  [1, 1]
      @5:  [1, 1]
      @6:  [0, 0]
      @7:  [1, 1]
      @8:  [0, 0]
      @9:  [1, 1]
      @10:  [1, 1]
      @11:  [0, 0]

   x (float):
      @0:  [2, 2]
      @1:  [1.25992, 1.25992]
      @2:  [2.5874, 2.5874]
      @3:  [7.69464, 7.69464]
      @4:  [1.97422, 1.97422]
      @5:  [4.89756, 4.89756]
      @6:  [24.9861, 24.9861]
      @7:  [2.92347, 2.92347]
      @8:  [9.5467, 9.5467]
      @9:  [2.12138, 2.12138]
      @10:  [5.50024, 5.50024]
      @11:  [31.2526, 31.2526]
      @12:  [3.14989, 3.14989]
```

How it works

- HySAT is **not** a SAT-Modulo-Theory solver:



- HySAT can be seen as a generalization of the DPLL procedure
  - ▷ deduction rule for n-ary disjunctions ('clauses'): unit propagation
  - ▷ deduction rules for arithmetic operators: adopted from interval constraint solving
  - ▷ search engine / branch-and-deduce framework inherited from DPLL

$\longrightarrow$ All acceleration techniques known for DPLL also apply to arithmetic constraints:

- ▷ conflict-driven learning
- ▷ backjumping
- ▷ lazy clause evaluation (watched literal scheme)

How it works: Example

$c_1 :$ $(\neg a \vee \neg c \vee d)$

$c_2 :$ $\wedge\ (\neg a \vee \neg b \vee c)$

$c_3 :$ $\wedge\ (\neg c \vee \neg d)$

$c_4 :$ $\wedge\ (b \vee x \geq -2)$

$c_5 :$ $\wedge\ (x \geq 4 \vee y \leq 0 \vee h_3 \geq 6.2)$

$c_6 :$ $\wedge\ h_1 = x^2$

$c_7 :$ $\wedge\ h_2 = -2 \cdot y$

$c_8 :$ $\wedge\ h_3 = h_1 + h_2$

- Use Tseitin-style (i.e. definitional) transformation to rewrite input formula into a conjunction of constraints:
  - ▷ n-ary disjunctions of bounds
  - ▷ arithmetic constraints having at most one operation symbol
- Boolean variables are regarded as 0-1 integer variables. Allows identification of literals with bounds on Booleans:

$$b \equiv b \geq 1$$
$$\neg b \equiv b \leq 0$$

- Float variables h_1, h_2, h_3 are used for decomposition of complex constraint $x^2 - 2y \geq 6.2$.

How it works: Example

c_1 : $(\neg a \lor \neg c \lor d)$

c_2 : $\land\ (\neg a \lor \neg b \lor c)$

c_3 : $\land\ (\neg c \lor \neg d)$

c_4 : $\land\ (b \lor x \geq -2)$

c_5 : $\land\ (x \geq 4 \lor y \leq 0 \lor h_3 \geq 6.2)$

c_6 : $\land\ h_1 = x^2$

c_7 : $\land\ h_2 = -2 \cdot y$

c_8 : $\land\ h_3 = h_1 + h_2$

DL 1: $a \geq 1$

How it works: Example

$c_1 :\quad (\neg a \lor \neg c \lor d)$

$c_2 :\quad \land\ (\neg a \lor \neg b \lor c)$

$c_3 :\quad \land\ (\neg c \lor \neg d)$

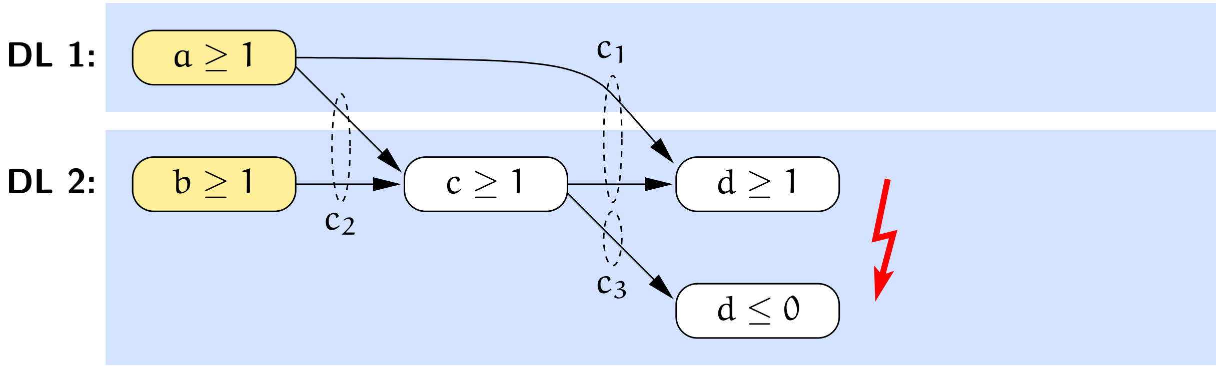
$c_4 :\quad \land\ (b \lor x \geq -2)$

$c_5 :\quad \land\ (x \geq 4 \lor y \leq 0 \lor h_3 \geq 6.2)$

$c_6 :\quad \land\ h_1 = x^2$

$c_7 :\quad \land\ h_2 = -2 \cdot y$

$c_8 :\quad \land\ h_3 = h_1 + h_2$



How it works: Example

$c_1 :$ $(\neg a \lor \neg c \lor d)$

$c_2 :$ $\land\ (\neg a \lor \neg b \lor c)$

$c_3 :$ $\land\ (\neg c \lor \neg d)$

$c_4 :$ $\land\ (b \lor x \geq -2)$

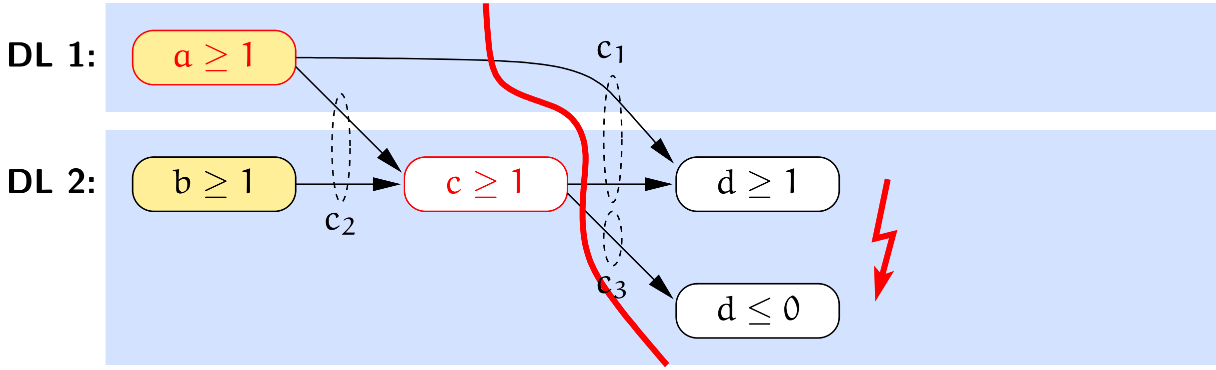
$c_5 :$ $\land\ (x \geq 4 \lor y \leq 0 \lor h_3 \geq 6.2)$

$c_6 :$ $\land\ h_1 = x^2$

$c_7 :$ $\land\ h_2 = -2 \cdot y$

$c_8 :$ $\land\ h_3 = h_1 + h_2$

$c_9 :$ $\land\ (\neg a \lor \neg c)$



How it works: Example

$c_1 :$ $(\neg a \lor \neg c \lor d)$

$c_2 :$ $\land\ (\neg a \lor \neg b \lor c)$

$c_3 :$ $\land\ (\neg c \lor \neg d)$

$c_4 :$ $\land\ (b \lor x \geq -2)$

$c_5 :$ $\land\ (x \geq 4 \lor y \leq 0 \lor h_3 \geq 6.2)$

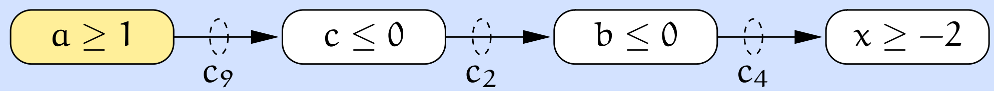
$c_6 :$ $\land\ h_1 = x^2$

$c_7 :$ $\land\ h_2 = -2 \cdot y$

$c_8 :$ $\land\ h_3 = h_1 + h_2$

$c_9 :$ $\land\ (\neg a \lor \neg c)$

DL 1:

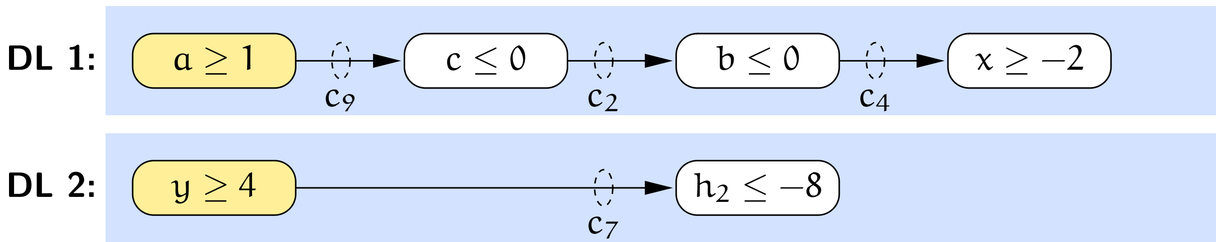


How it works: Example

$$c_1 : \quad (\neg a \lor \neg c \lor d)$$
$$c_2 : \land (\neg a \lor \neg b \lor c)$$
$$c_3 : \land (\neg c \lor \neg d)$$
$$c_4 : \land (b \lor x \geq -2)$$
$$c_5 : \land (x \geq 4 \lor y \leq 0 \lor h_3 \geq 6.2)$$

$$c_6 : \land h_1 = x^2$$
$$c_7 : \land h_2 = -2 \cdot y$$
$$c_8 : \land h_3 = h_1 + h_2$$

$$c_9 : \land (\neg a \lor \neg c)$$



How it works: Example

c_1 : $(\neg a \lor \neg c \lor d)$

c_2 : $\land\ (\neg a \lor \neg b \lor c)$

c_3 : $\land\ (\neg c \lor \neg d)$

c_4 : $\land\ (b \lor x \geq -2)$

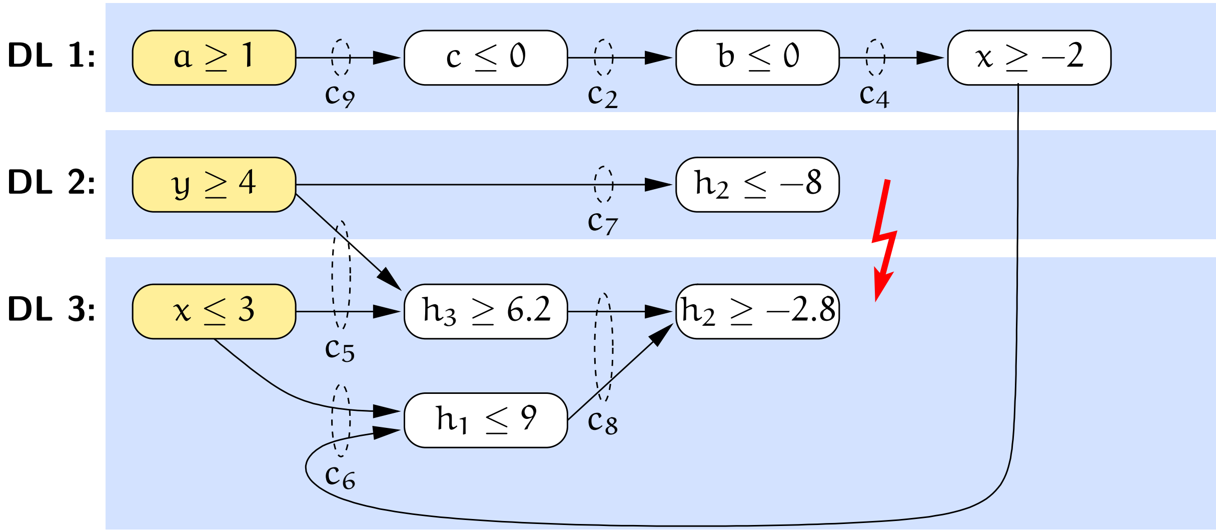
c_5 : $\land\ (x \geq 4 \lor y \leq 0 \lor h_3 \geq 6.2)$

c_6 : $\land\ h_1 = x^2$

c_7 : $\land\ h_2 = -2 \cdot y$

c_8 : $\land\ h_3 = h_1 + h_2$

c_9 : $\land\ (\neg a \lor \neg c)$

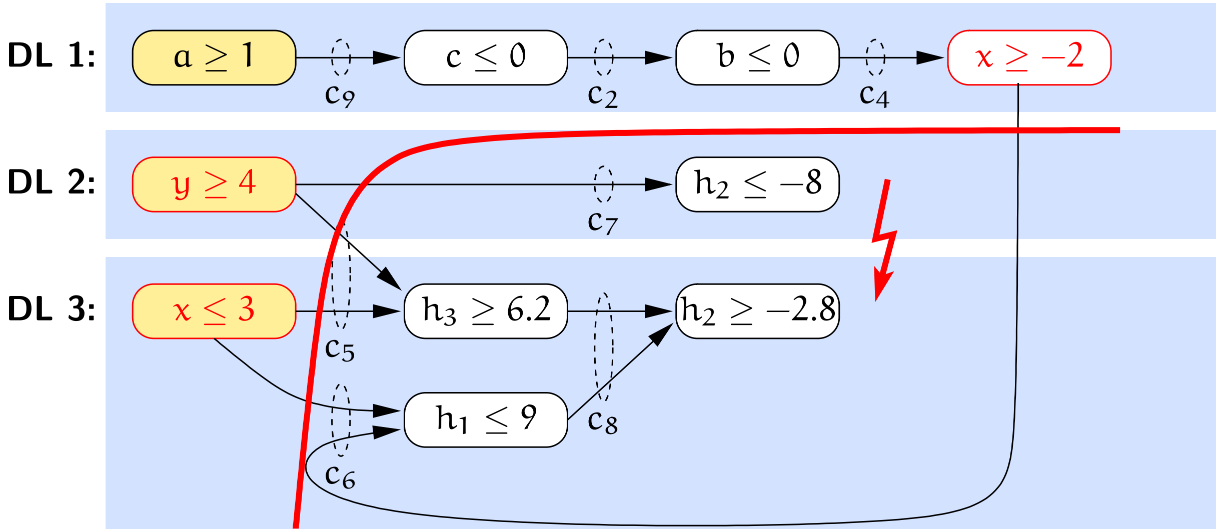


How it works: Example

c_1 : $(\neg a \lor \neg c \lor d)$
c_2 : $\land\ (\neg a \lor \neg b \lor c)$
c_3 : $\land\ (\neg c \lor \neg d)$
c_4 : $\land\ (b \lor x \geq -2)$
c_5 : $\land\ (x \geq 4 \lor y \leq 0 \lor h_3 \geq 6.2)$

c_6 : $\land\ h_1 = x^2$
c_7 : $\land\ h_2 = -2 \cdot y$
c_8 : $\land\ h_3 = h_1 + h_2$

c_9 : $\land\ (\neg a \lor \neg c)$
c_{10} : $\land\ (x < -2 \lor y < 3 \lor x > 3)$



$\leftarrow$ conflict clause = symbolic description

of a rectangular region of the search space

which is excluded from future search

How it works: Example

$c_1 : \quad (\neg a \lor \neg c \lor d)$

$c_2 : \quad \land\ (\neg a \lor \neg b \lor c)$

$c_3 : \quad \land\ (\neg c \lor \neg d)$

$c_4 : \quad \land\ (b \lor x \geq -2)$

$c_5 : \quad \land\ (x \geq 4 \lor y \leq 0 \lor h_3 \geq 6.2)$

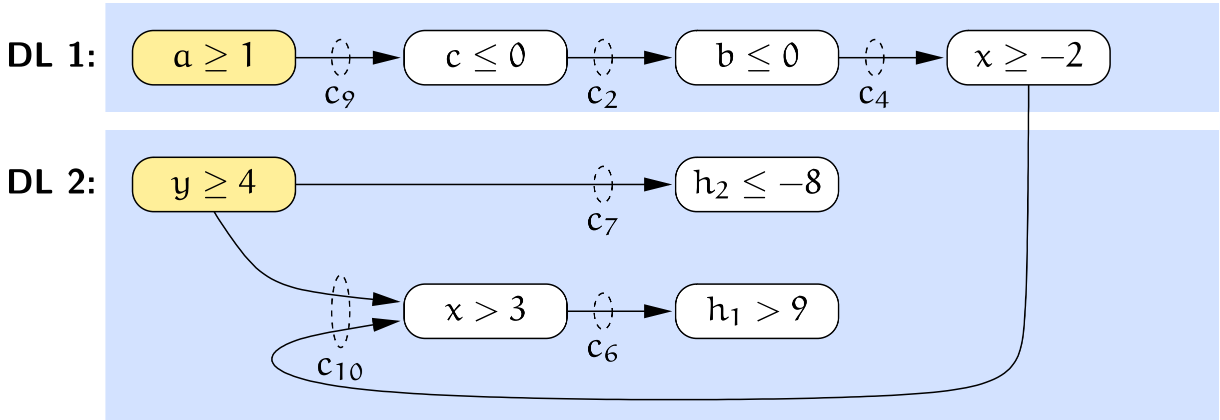
$c_6 : \quad \land\ h_1 = x^2$

$c_7 : \quad \land\ h_2 = -2 \cdot y$

$c_8 : \quad \land\ h_3 = h_1 + h_2$

$c_9 : \quad \land\ (\neg a \lor \neg c)$

$c_{10} : \quad \land\ (x < -2 \lor y < 3 \lor x > 3)$



How it works: Example

$c_1 :\quad (\neg a \vee \neg c \vee d)$

$c_2 :\quad \wedge\ (\neg a \vee \neg b \vee c)$

$c_3 :\quad \wedge\ (\neg c \vee \neg d)$

$c_4 :\quad \wedge\ (b \vee x \geq -2)$

$c_5 :\quad \wedge\ (x \geq 4 \vee y \leq 0 \vee h_3 \geq 6.2)$

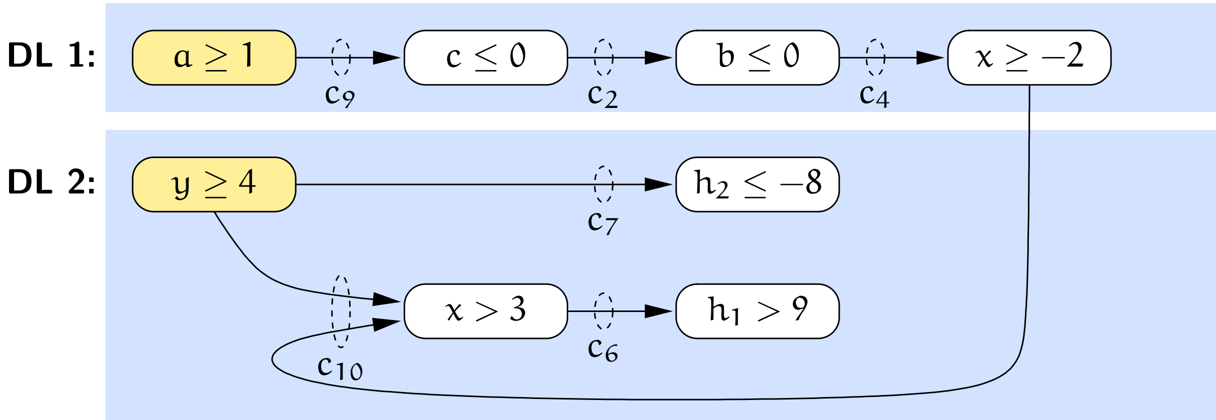
$c_6 :\quad \wedge\ h_1 = x^2$

$c_7 :\quad \wedge\ h_2 = -2 \cdot y$

$c_8 :\quad \wedge\ h_3 = h_1 + h_2$

$c_9 :\quad \wedge\ (\neg a \vee \neg c)$

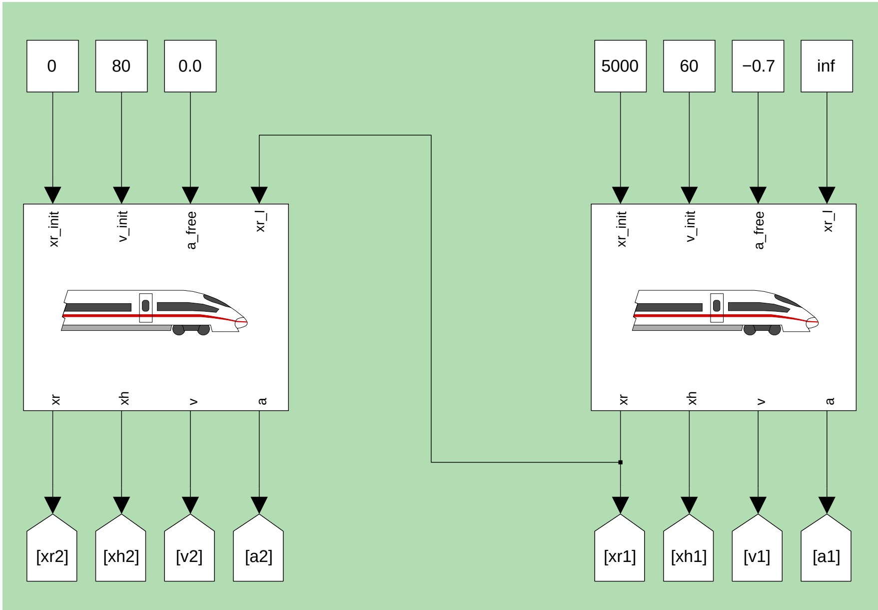
$c_{10} :\quad \wedge\ (x < -2 \vee y < 3 \vee x > 3)$



- Continue do split and deduce until either
  - ▷ formula turns out to be UNSAT (unresolvable conflict)
  - ▷ solver is left with 'sufficiently small' portion of the search space for which it cannot derive any contradiction

- Avoid infinite splitting and deduction:
  - ▷ minimal splitting width
  - ▷ discard a deduced bound if it yields small progress only

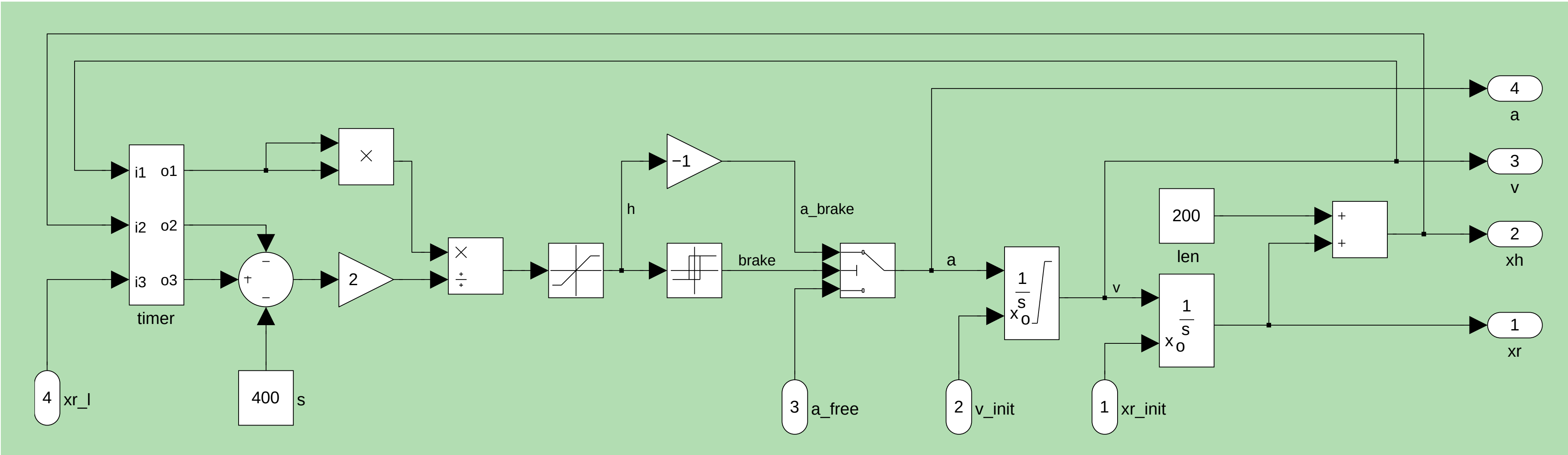
Example: Train Separation in Absolute Braking Distance



Minimal admissible distance d between two following trains equals braking distance d_b of the second train plus a safety distance S.

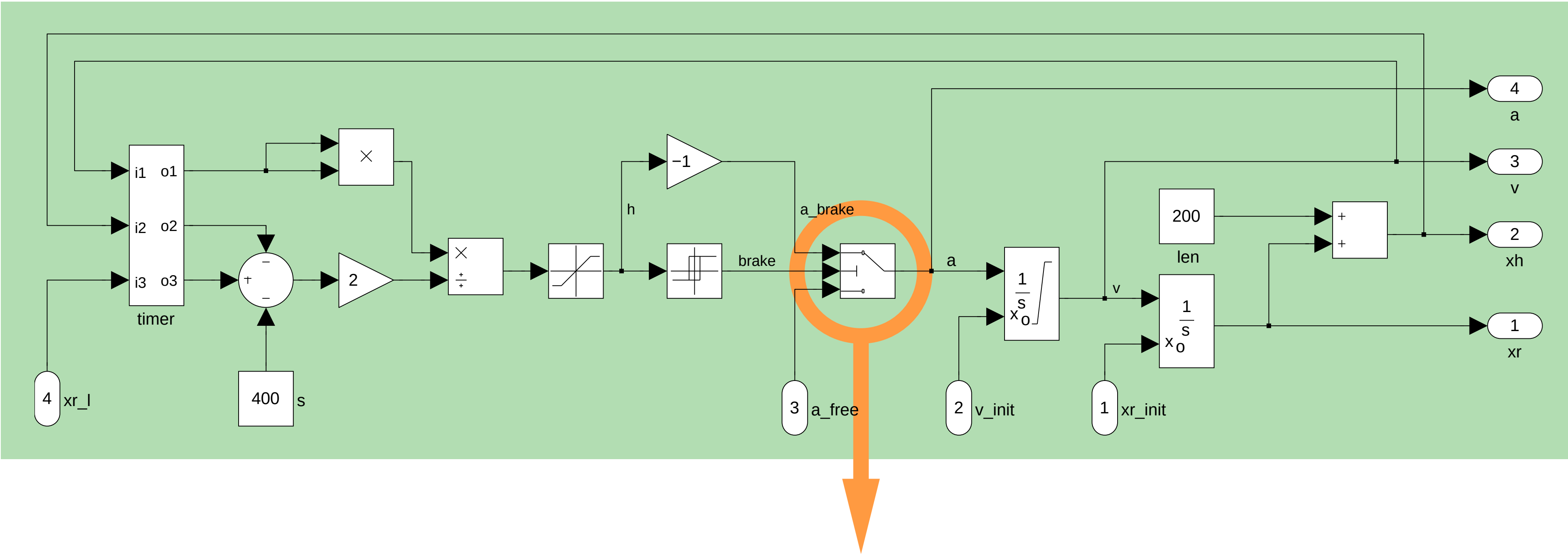
First train reports position of its end to the second train every 8 seconds.
Controller in second train automatically initiates braking to maintain a safe distance.

Model of Controller & Train Dynamics



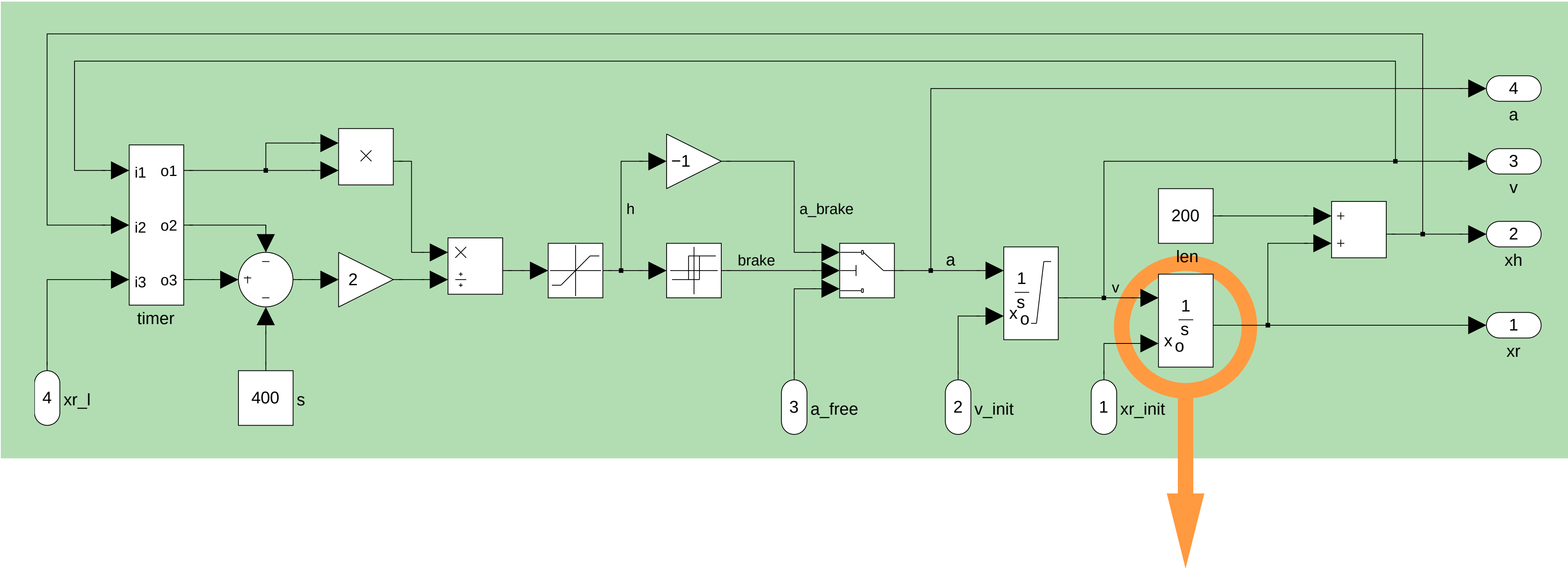
Property to be checked: Does the controller guarantee that collisions don't occur in any possible scenario of use?

Translation to HySAT



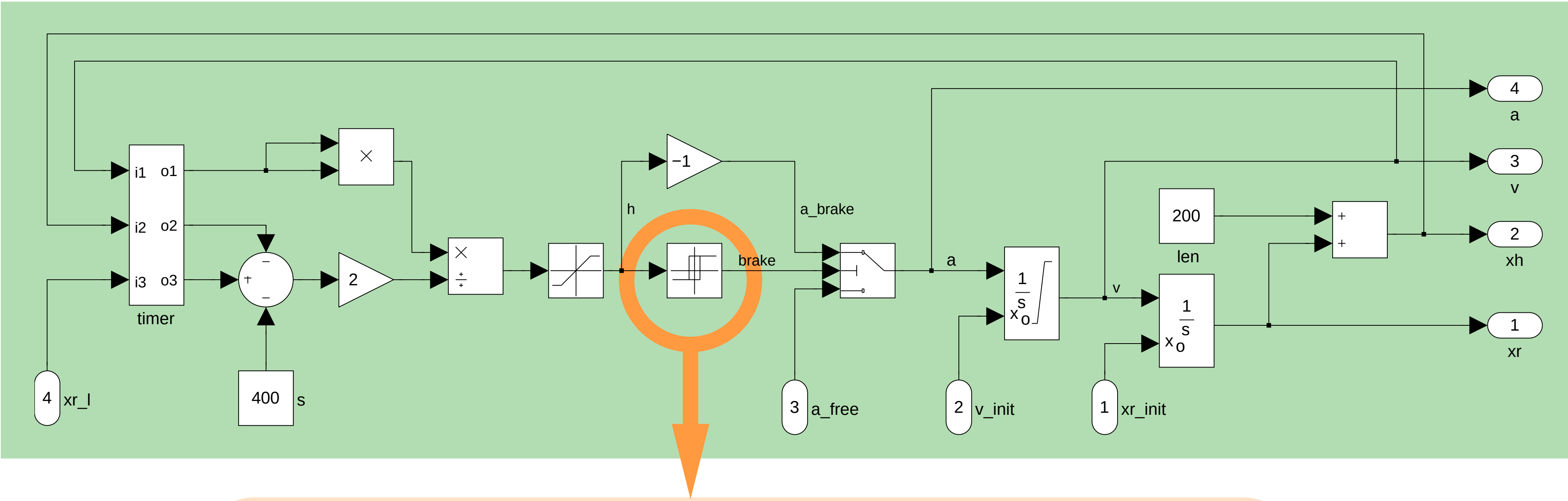

```
-- Switch block:  Passes through the first input or the third input
-- based on the value of the second input.

 brake -> a = a_brake;
!brake -> a = a_free;
```

Translation to HySAT




```
-- Euler approximation of integrator block

xr' = xr + dt * v;
```

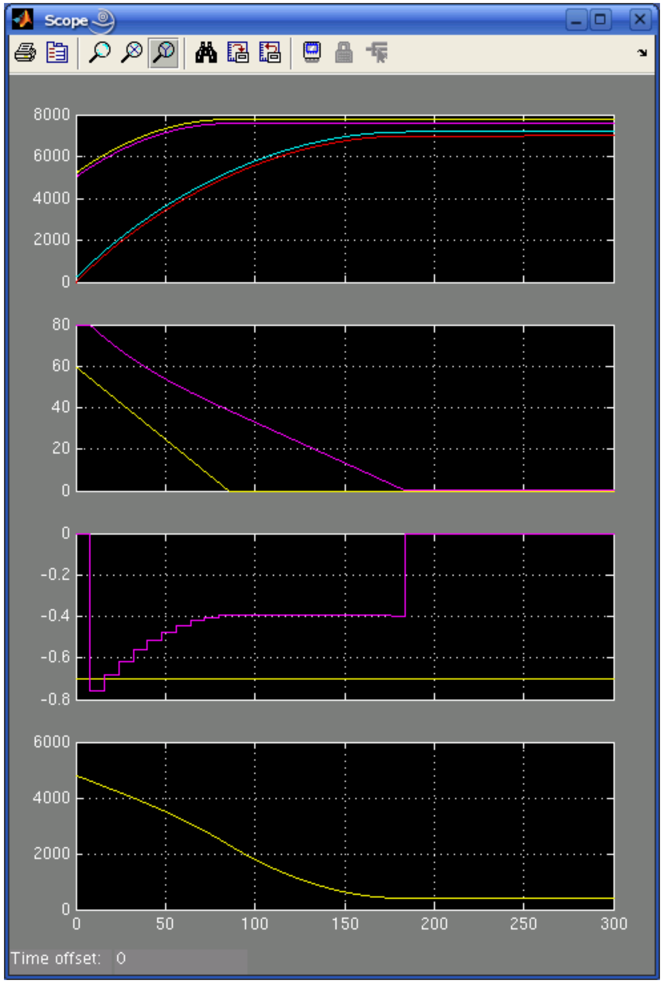
Translation to HySAT



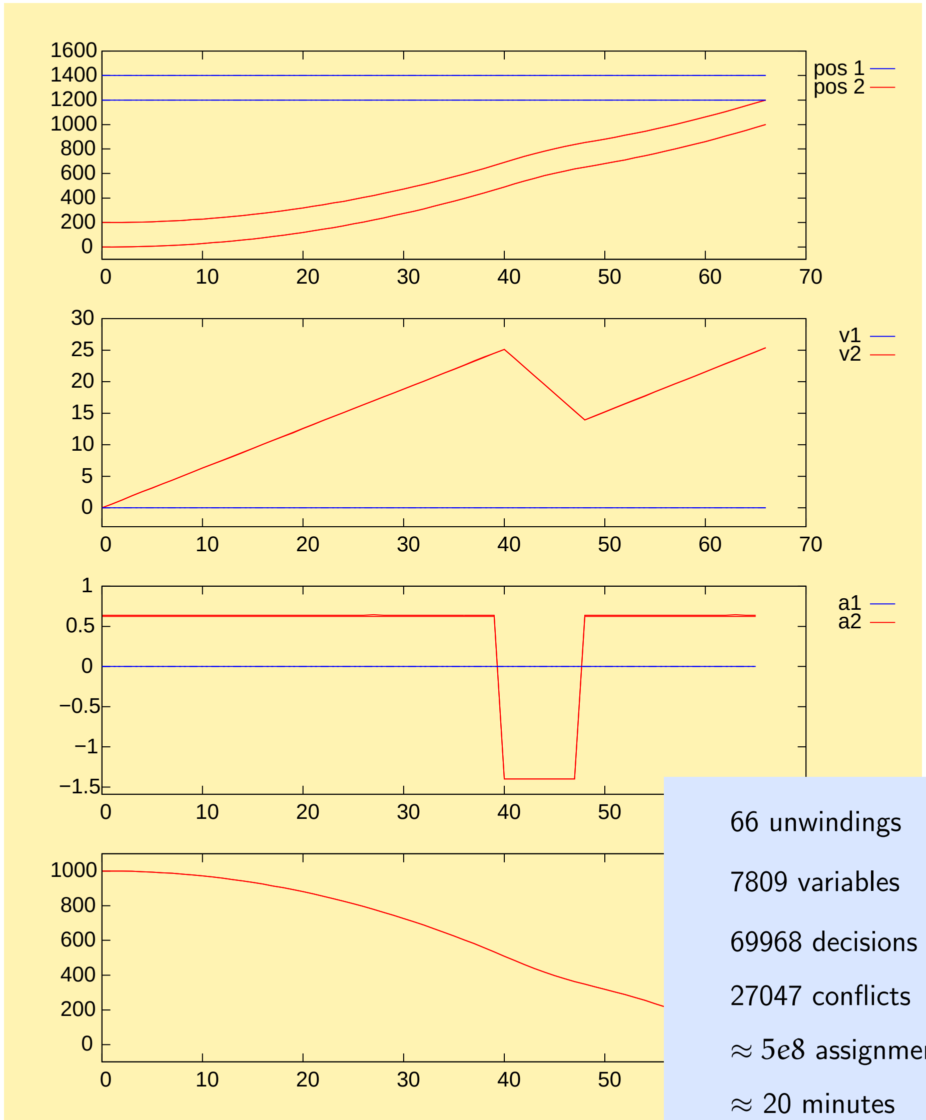

```
-- Relay block: When the relay is on, it remains on until the input
-- drops below the value of the switch off point parameter. When the
-- relay is off, it remains off until the input exceeds the value of
-- the switch on point parameter.

(!is_on and h >= param_on ) -> ( is_on' and  brake);
(!is_on and h <  param_on ) -> (!is_on' and !brake);
( is_on and h <= param_off) -> (!is_on' and !brake);
( is_on and h >  param_off) -> ( is_in' and  brake);
```

Application: BMC of Matlab/Simulink Model

Simulation of the Model



Error Trace found by HySAT



- 66 unwindings
- 7809 variables
- 69968 decisions
- 27047 conflicts
- ≈ 5e8 assignments
- ≈ 20 minutes

Final Remarks

- Prototype implementation. Still missing:
  - ▷ random restarts
  - ▷ activity–based splitting heuristics
  - ▷ 'forgetting' of learned clauses
  - ▷ low–level code optimizations (e.g. to improve caching)

- Future extensions:
  - ▷ (SMT–style) integration of Linear Programming
  - ▷ native (ICP–based) support for ODEs

- Tool & Papers available at
  - ▷ `http://hysat.informatik.uni-oldenburg.de`

Thank you!