Expressing and Solving Constraint Satisfaction Problems in B

Michael Leuschel, Daniel Plagge

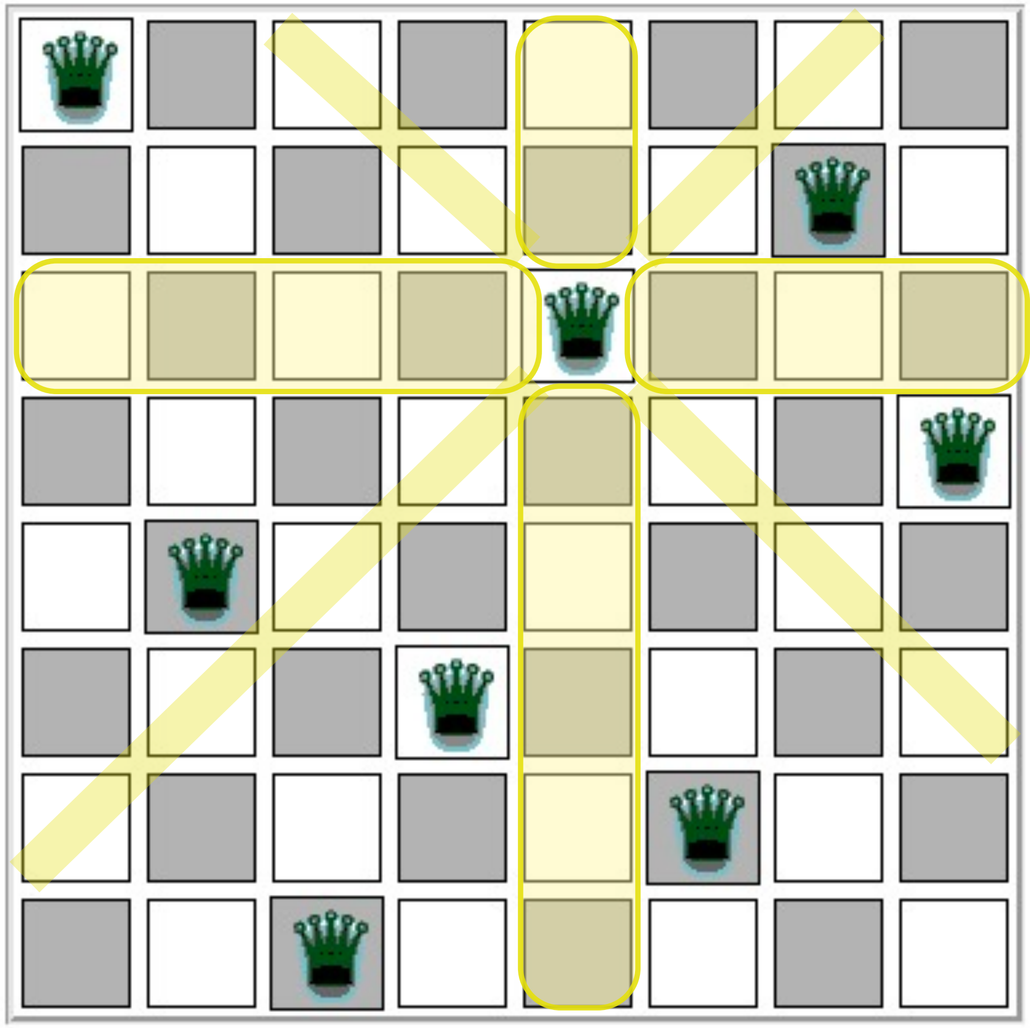
Rodin User & Developer Workshop
September 2010, Düsseldorf

Message of the Talk

- Many constraint satisfaction problems can be expressed very succinctly and elegantly in B / Event-B
- Some of these problems can be solved effectively using ProB

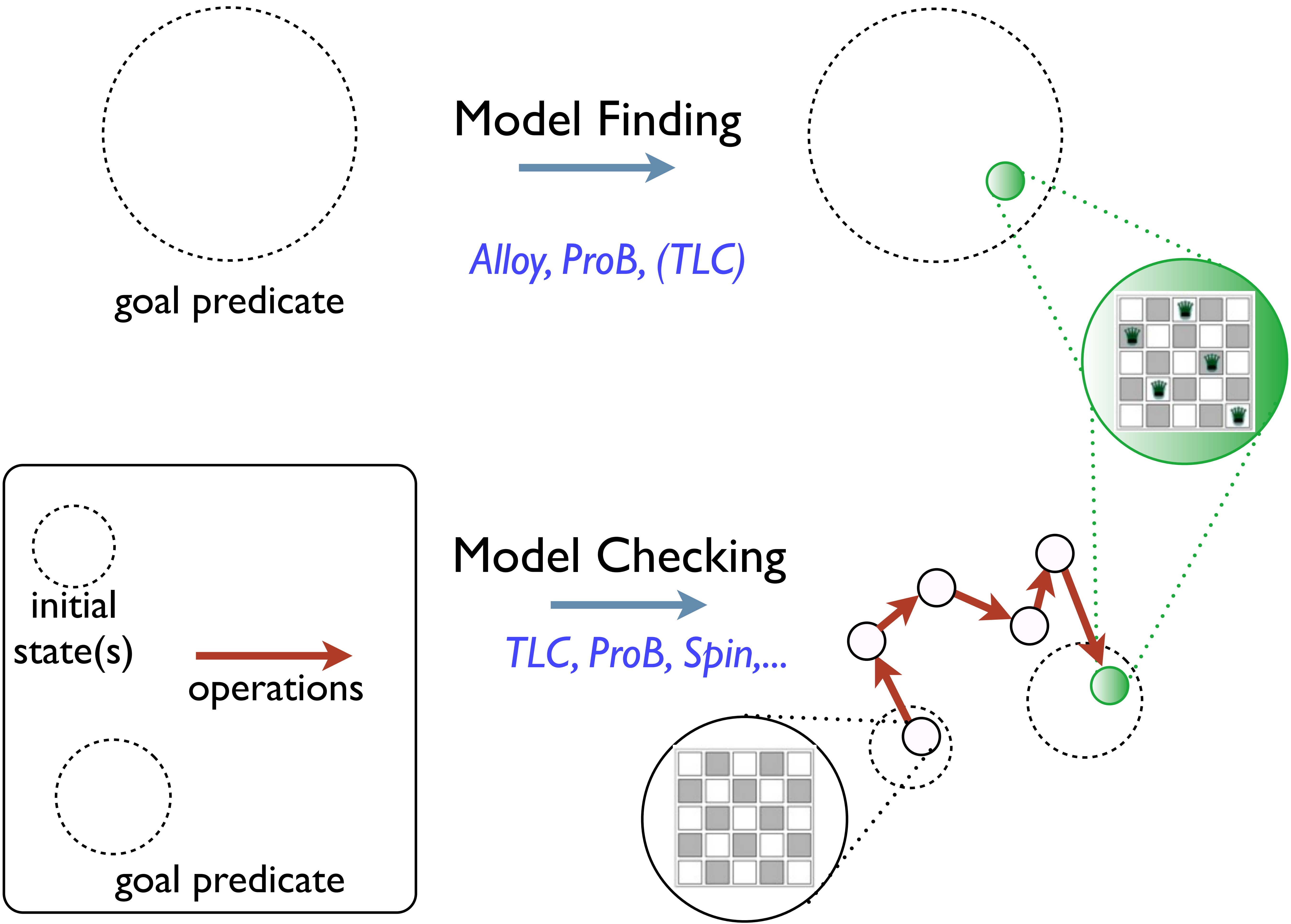
One Example Problem

- Standard Benchmark for Constraint Solving
- Place N queens on a N*N chessboard so that no two queens attach each other



Solving Constraint Satisfaction Problems:

- Write an Algorithm
- Semi Declarative
- Fully Declarative


goal predicate
Model Finding
Alloy, ProB, (TLC)
initial
state(s)
operations
goal predicate
Model Checking
TLC, ProB, Spin,...

Tools

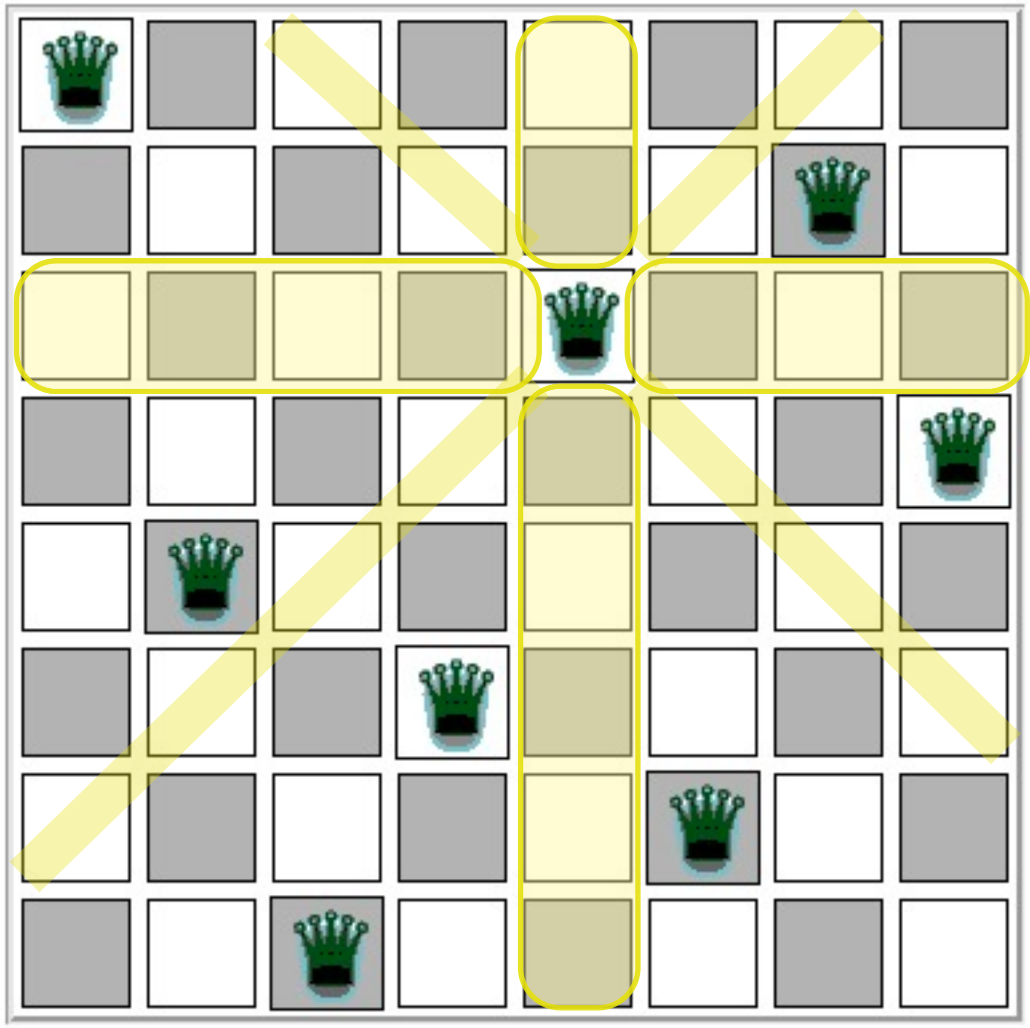
Tool	Input Language	Model Checking	Model Finding
Alloy	1st order Relational	- (BMC by hand)	SAT Symmetry
TLC	TLA+	Explicit Disk, Fingerprint	Enumeration (in Java)
AnimB	Event-B	- (animation)	Enumeration (in Java)
ProB	B, Event-B, Z CSP, CSP||B	Explicit Symmetry,...	Constraint Logic Programming
Spin	Low-level Promela	Explicit POR, Bitstate,...	-

As part of computing transitions

+ finding constants

N-Queens

- Let us try and solve it:
  - 1. Explicit Algorithm
  - 2. Using Model Checking
  - 3. Using Model Finding



Recursive Algorithm (Gnu Pascal)

```
PROGRAM Reines;

CONST
    MaxReines = 100;
VAR
    PosReine: ARRAY[1..MaxReines] OF INTEGER;
    LigneOccupe: ARRAY[1..MaxReines] OF BOOLEAN;

{--------------------------------------------------}

PROCEDURE PlacerReine(ligne,colonne: INTEGER);
BEGIN
    LigneOccupe[ligne] := TRUE;
    PosReine[colonne] := ligne;
END; {PlacerReine}

{--------------------------------------------------}

PROCEDURE EnleverReine(ligne,colonne: INTEGER);
BEGIN
    LigneOccupe[ligne] := FALSE;
END; {PlacerReine}
```

```
PROCEDURE ImprimerSolution(Dim: INTEGER);
VAR
    i: INTEGER;
BEGIN
    FOR i:= 1 TO Dim DO
    BEGIN
        Write(PosReine[i]:4);
    END; {FOR}
    Writeln;
END; {ImprimerSolution}

{--------------------------------------------------}

PROCEDURE InitReines;
VAR
    i: INTEGER;
BEGIN
    FOR i:= 1 TO MaxReines DO
        LigneOccupe[i] := FALSE;
END; {InitReines}
```

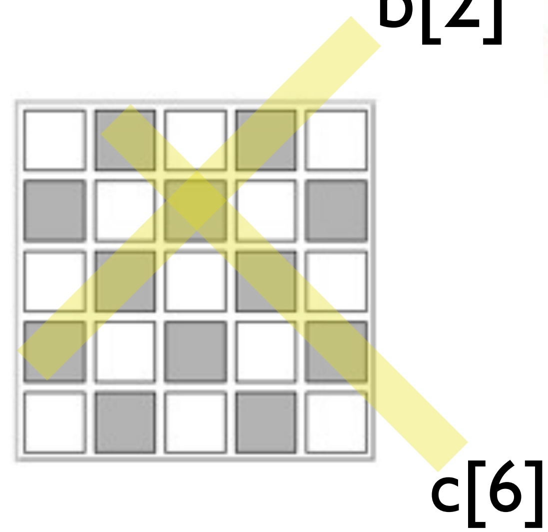
Algorithm (cont'd)

```
FUNCTION PlacementPossible(ligne,colonne: INTEGER): BOOLEAN;
VAR
    ReineCol: INTEGER;
BEGIN
    IF LigneOccupe[ligne] THEN
        PlacementPossible := FALSE  {ligne déjà occupée}
    ELSE
    BEGIN
        PlacementPossible := TRUE;

        {verifier les diagonales}
        ReineCol := 1;
        WHILE ReineCol<colonne DO
        BEGIN
            IF (PosReine[ReineCol]+(colonne-ReineCol) = ligne)
               OR(PosReine[ReineCol]-(colonne-ReineCol) = ligne)THEN
            BEGIN   {diagonale déjà occupée}
                PlacementPossible := FALSE;
                ReineCol := colonne; {sortir de la boucle}
            END
            ELSE    {pas d'interférence avec cette reine, avancer}
                ReineCol := ReineCol+1;
        END; {WHILE}
    END; {ELSE}
END; {PlacerReine}
```

```
PROCEDURE ReinesRec(Dim: INTEGER);
VAR
    col,nbreDeSol: INTEGER;

    PROCEDURE RReine;
    VAR
        i: INTEGER;
    BEGIN
        IF col>Dim THEN
        BEGIN
            nbreDeSol := nbreDeSol + 1;
            Write('Sol',nbreDeSol:3,':');
            ImprimerSolution(Dim);
            Halt;
        END
        ELSE
        BEGIN
            FOR i:= 1 TO Dim DO
            BEGIN
                IF PlacementPossible(i,col) THEN
                BEGIN
                    PlacerReine(i,col);
                    col := col+1;
                    RReine;
                    col := col-1;
                    EnleverReine(i,col);
                END;
            END; {FOR}
        END; {ELSE}
    END; {RReine}
BEGIN
    col := 1; nbreDeSol:= 0;
    RReine;
END; {ReinesRec}
```

Spin Solution (Ben-Ari)

```
inline Choose() {

    if
    :: row = 1
    :: row = 2
    :: row = 3
    :: row = 4
    :: row = 5
    :: row = 6
    :: row = 7
    :: row = 8
    fi
}

inline Write() {
    printf("result: %d,%d,%d,%d,%d,%d,%d,%d",
    result[0],result[1],result[2],result[3],
    result[4],result[5],result[6],result[7]);
}
```




```
byte result[8];
bool a[8];
bool b[15];
bool c[15];

active proctype Queens() {

    byte col = 1;
    byte row;

    do
    :: Choose();
       !a[row-1];
       !b[row+col-2];
       !c[row-col+7];
       a[row-1] = true;
       b[row+col-2] = true;
       c[row-col+7] = true;
       result[col-1] = row;
       if
       :: col == 8 -> break
       :: else -> col++
       fi
    od;
    Write();
    byte dummy = result[0];
    assert(false);
}
```

Diagonals

N=8: hard-coded into model!

TLC Solution

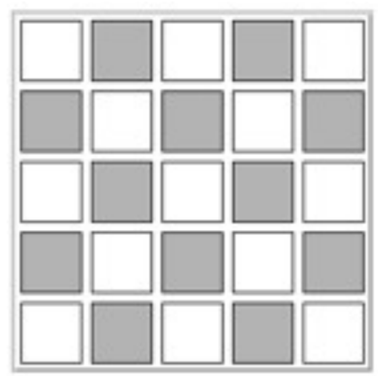
(Model checking)

MODULE *queensMC*

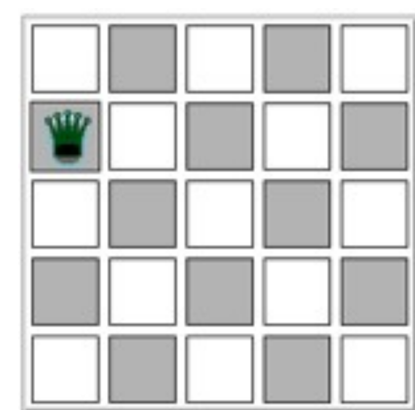
EXTENDS *Naturals*, *FiniteSets*

VARIABLE q, n, cur, pos

$$
\begin{aligned}
Init \triangleq\ & \land\ q = [i \in 1\,..\,8 \mapsto 0] \\
& \land\ n = 8 \\
& \land\ cur = 1 \\
& \land\ pos = 0
\end{aligned}
$$



$$
\begin{aligned}
Step \triangleq\ & \land\ cur \leq n \\
& \land\ pos' \in 1\,..\,n \\
& \land\ cur' = cur + 1 \\
& \land\ q' = [q \text{ EXCEPT } ![cur] = pos'] \\
& \land\ n' = n \\
& \land\ \forall i \in 1\,..\,(cur - 1) : q[i] \neq pos' \land q'[i] + i - cur \neq pos' \land q'[i] - i + cur \neq pos'
\end{aligned}
$$

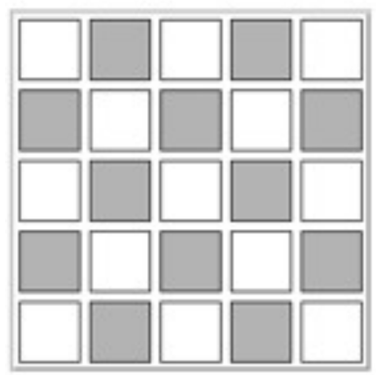


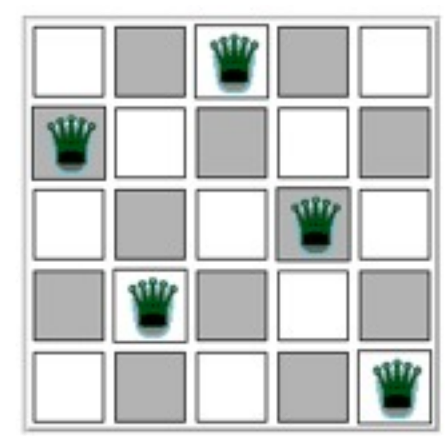
$$GINV \triangleq cur \leq n$$

$$Spec \triangleq Init \land \Box[Step]_{\langle n,\, q,\, cur,\, pos \rangle}$$

TLC Solution
(Model finding)

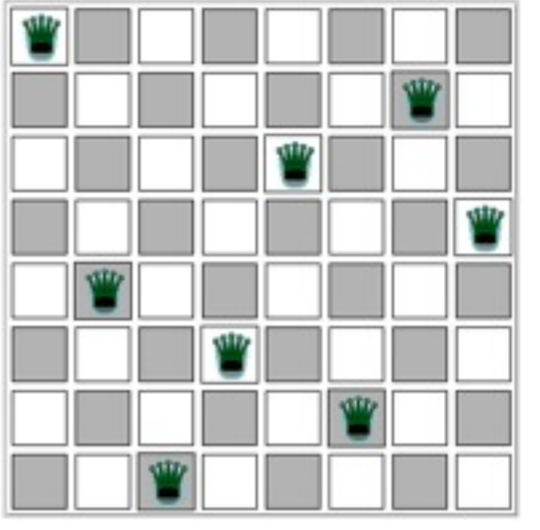
```
---- MODULE queens ----

EXTENDS Naturals, FiniteSets
VARIABLE q, n, solved
----

Init == /\ q=[i \in 1..2 |-> 0]
        /\ n=8
        /\ solved = 0

Solve == /\ solved=0
         /\ q' \in [1..n -> 1..n]
         /\ \A i \in 1..n : (\A j \in 2..n : i<j =>
              q'[i] # q'[j] /\ q'[i]+i-j # q'[j] /\ q'[i]-i+j # q'[j])
         /\ solved'=1
Spec == Init /\ [] [Solve]_<<n,q>>
=======
```





Alloy Solution

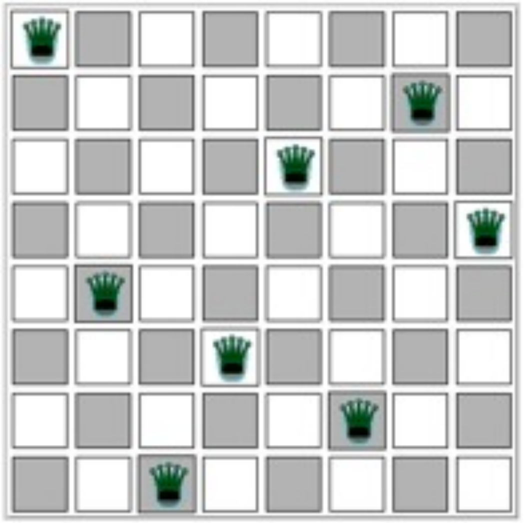
```
sig Queens {
  row : Int,
  col: Int
} {
 row >= 0 and row < #Queens
 and col >= 0 and col < #Queens
}

pred nothreat(q1,q2 : Queens) {
      q1.row != q2.row
      and q1.col != q2.col
      and q1.row+q2.col-q1.col != q2.row
    and q1.row-q2.col+q1.col != q2.row
}

pred valid { all q1,q2 : Queens |
    q1 != q2 => nothreat[q1, q2]
 }

fact card {#Queens = 8}
run valid for 8 Queens, 5 int
```



number of bits for integers

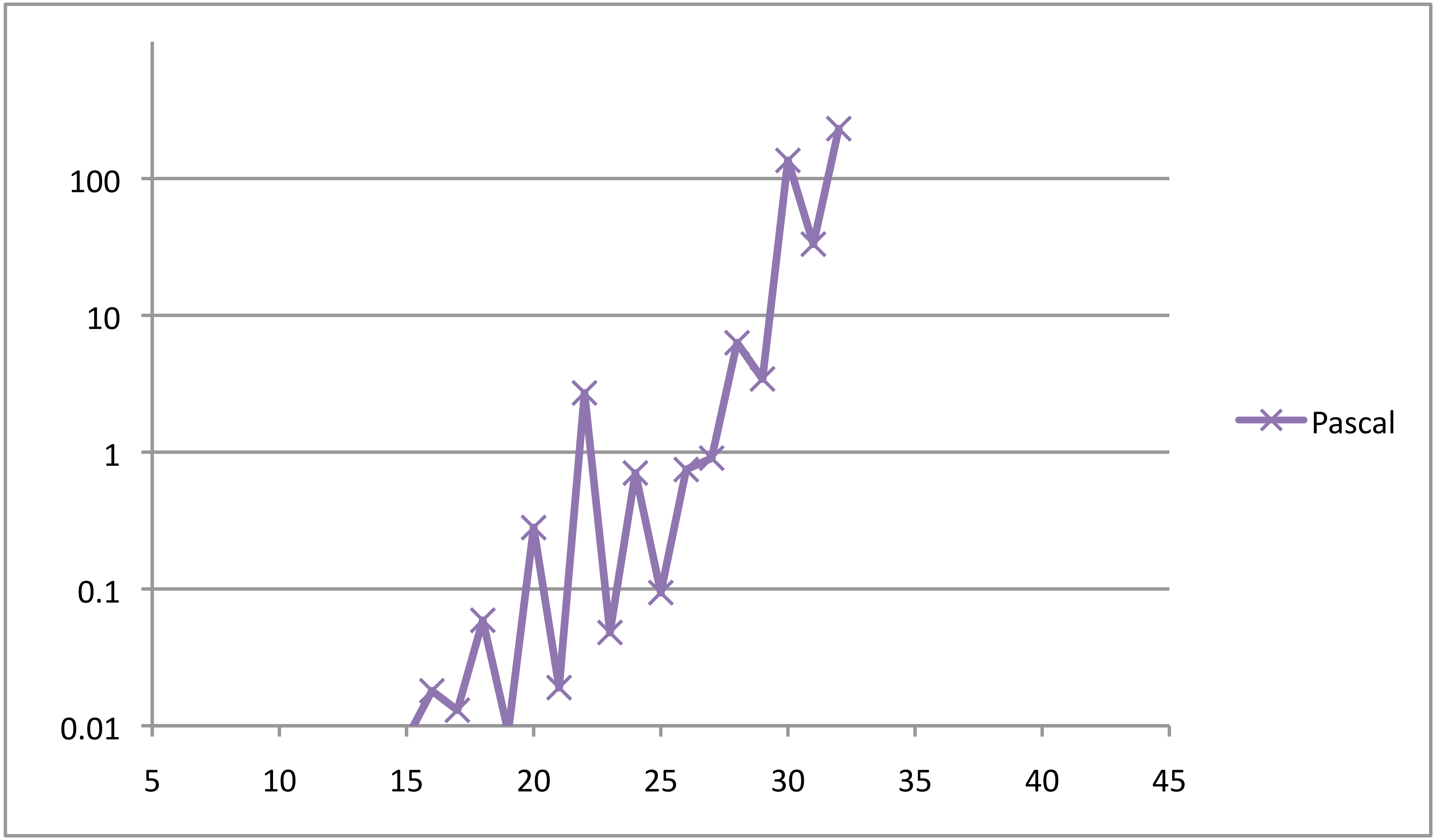
Event-B Solution

```
context NQueens
constants n queens
axioms
  @axm1 n=8
  @axm2 queens ∈ 1‥n ⤖ 1‥n
  @axm3 ∀q1,q2•(q1∈1‥n ∧ q2∈2‥n ∧ q2>q1
         ⇒ queens(q1)+q2−q1 ≠ queens(q2) ∧
            queens(q1)−q2+q1 ≠ queens(q2))
end
```



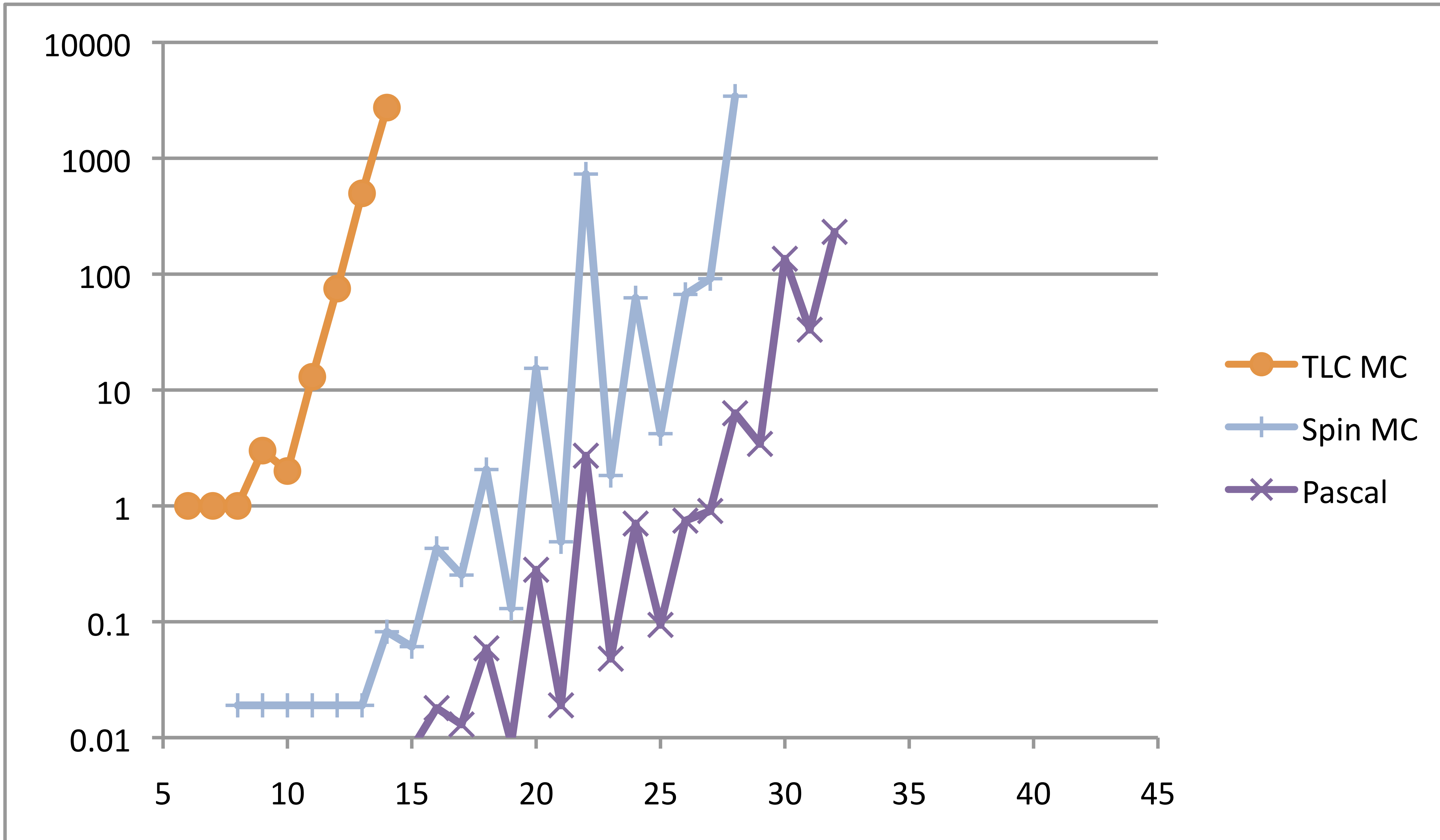
TLC solution very similar; no >->> in TLA+

Performance ?

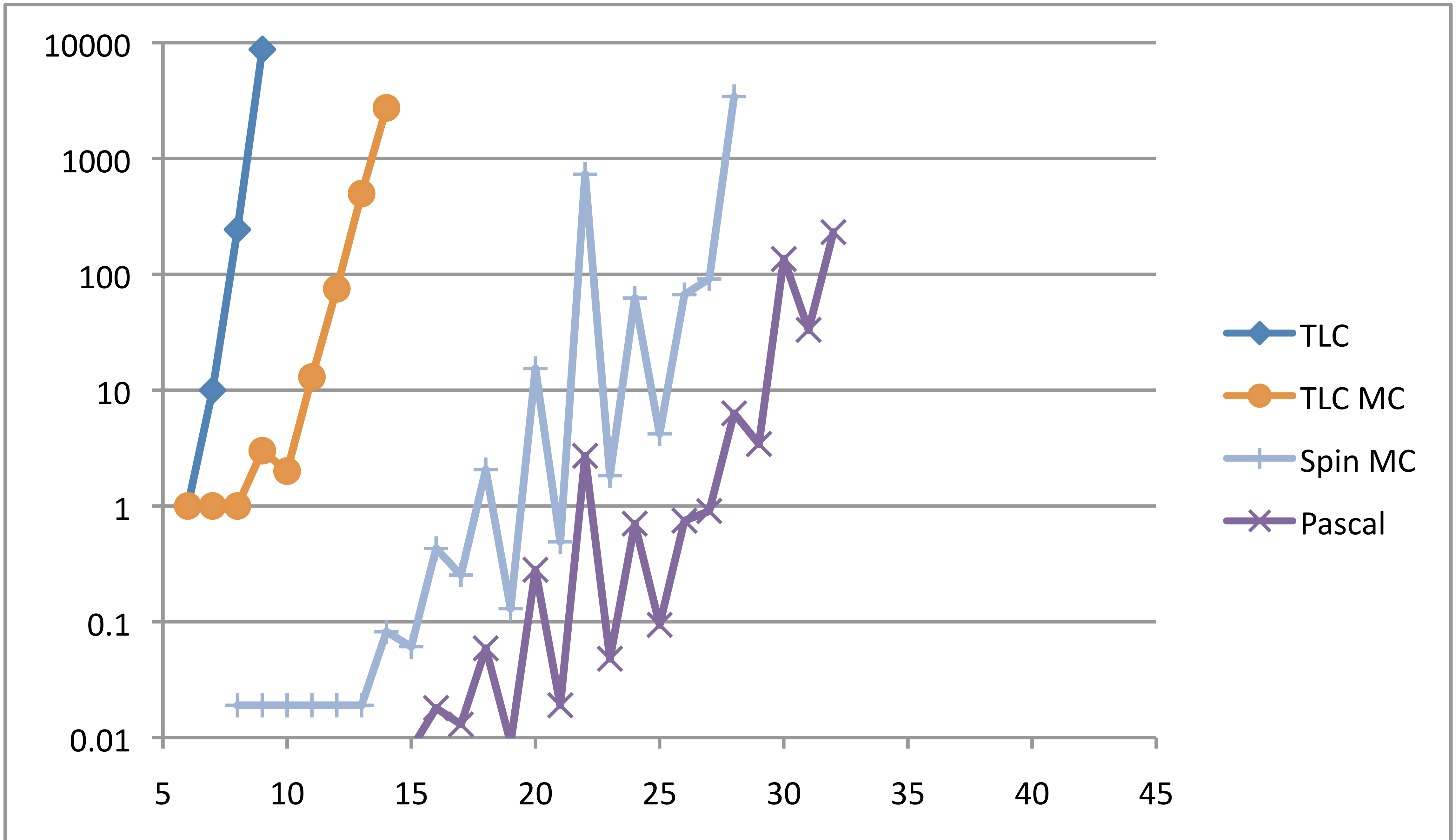


Pascal: no solution found after 90 minutes for n=40

Performance
Model Checking

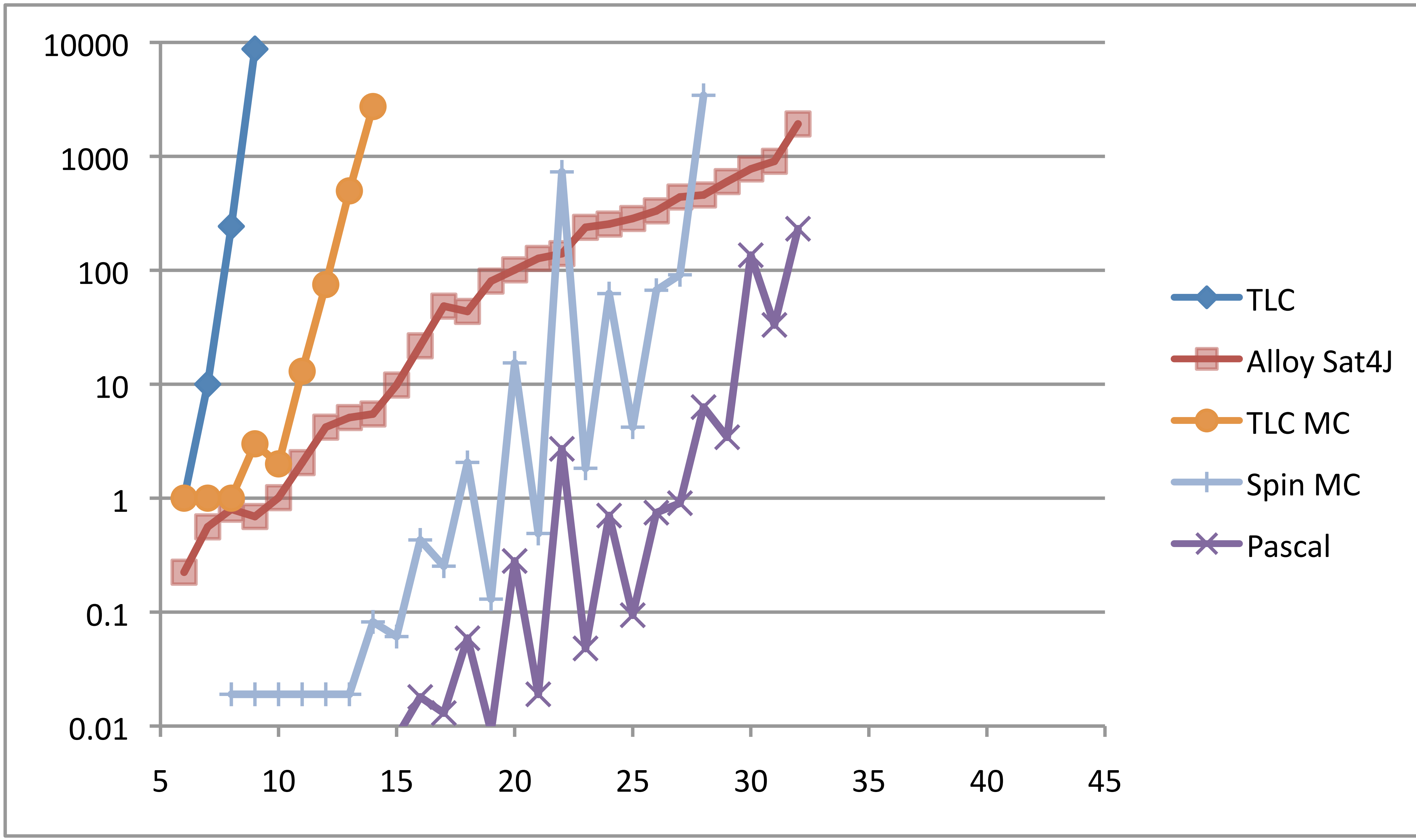


Performance: Model Finding I

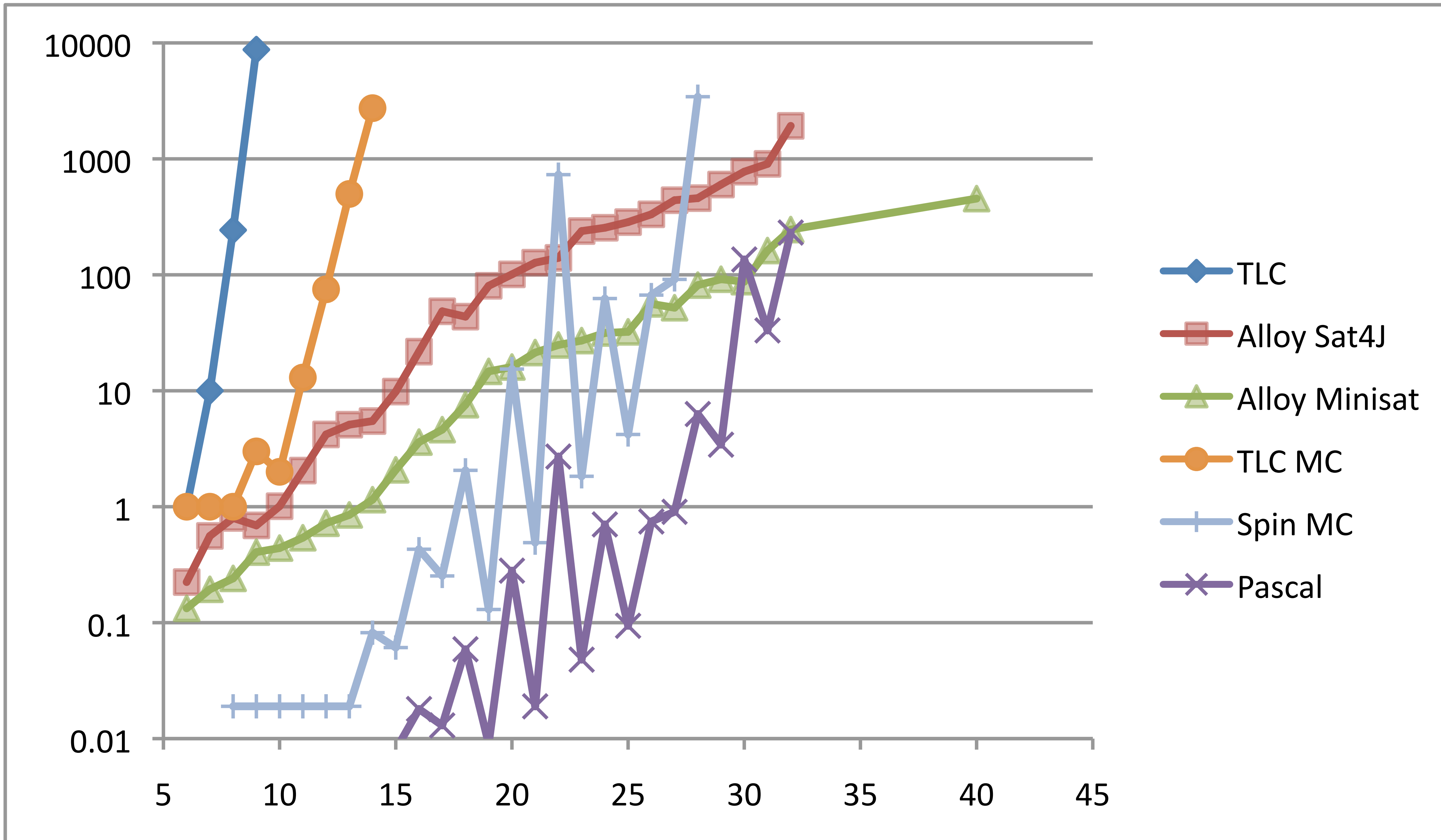


AnimB: only for n=5 solution found

Performance: Model Finding 2

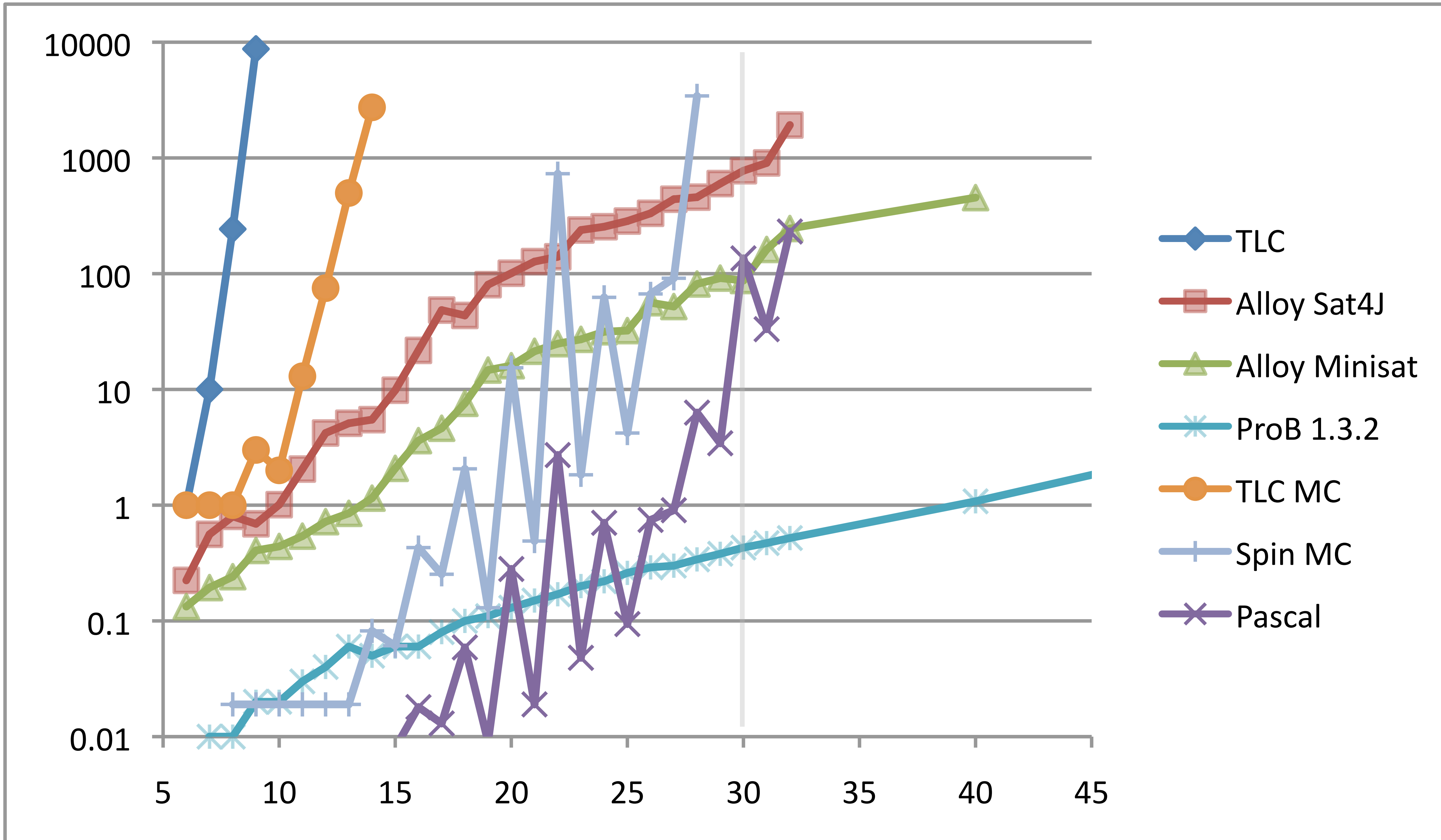


Performance: Model Finding 3



Pascal: no solution found after 90 minutes for n=40

ProB Performance

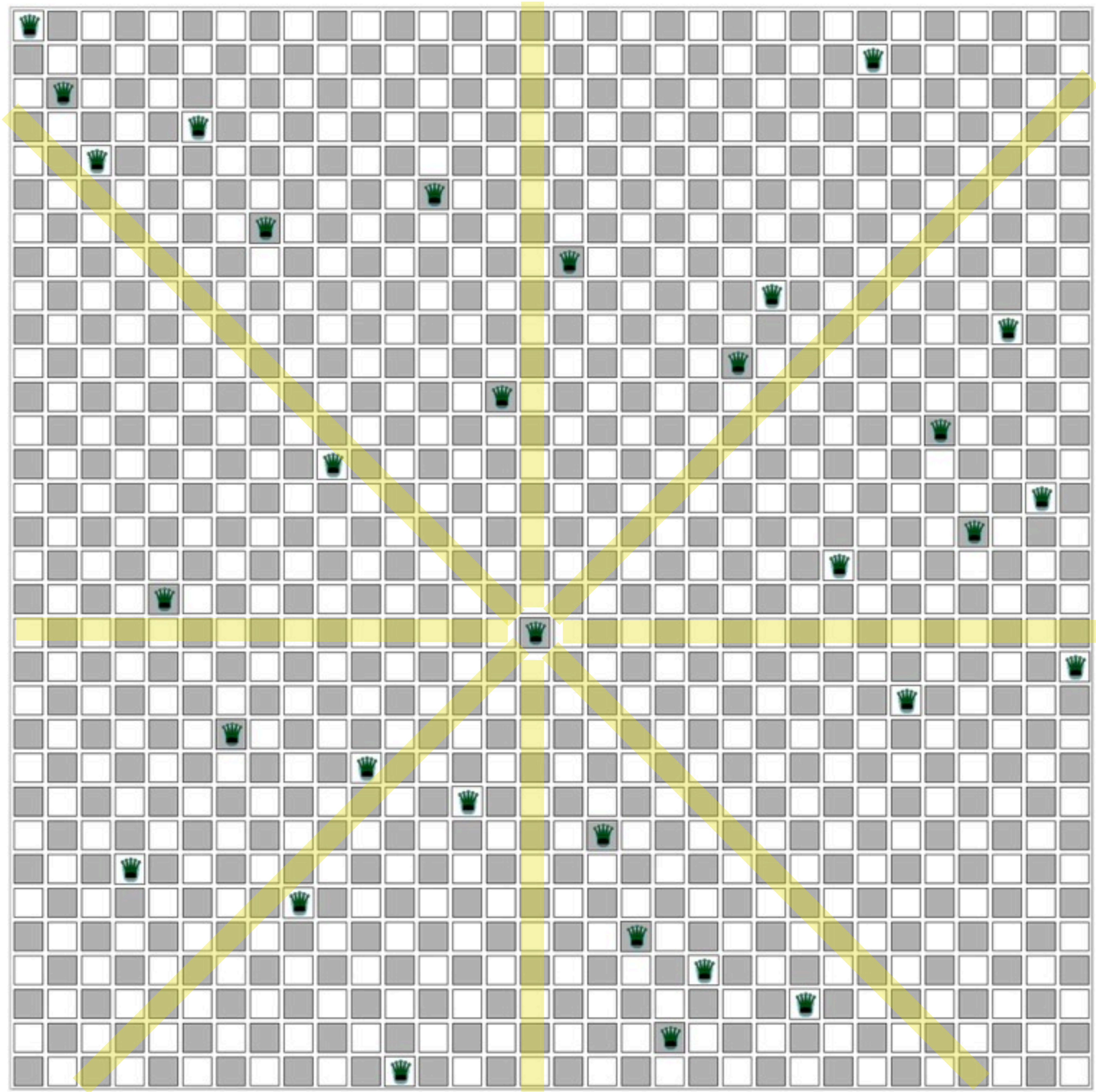


ProB: n=70: 9.09 secs, n=100 : 80.41 secs

ProB Solution for n=32

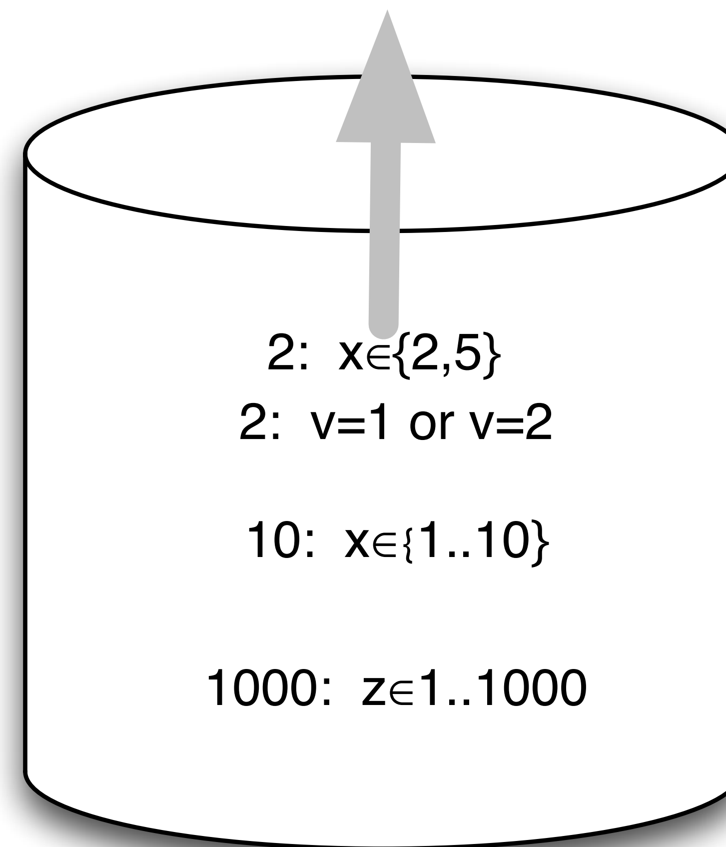
Time (seconds)

ProB:	0.5
C:	64.1
Pascal:	231.5
Alloy mini:	245.6
Alloy:	1925.1
Spin:	----
(3453 for n=28)	
TLC:	----
(2737 for n=14)	



ProB Constraint Solving Algorithm

- Priority Queue of Choice Points:
  - priority = estimated number of solutions
    - priority=1 ➔ deterministic



Priority Queue of Enumerations/Choice Points

Other Experiments

Model	ProB	Alloy	TLC
7Knights2Q	0.64 secs	1 min 53.9 secs	-
GraphIso	0.06 secs	0.05 secs	2h 6m 28s
Sudoku	0.46 secs	0.46 - 1.04 secs	-
Numerical	0.07 secs	1.8 secs *	-
CrewAlloc	1.24 secs	0.03 secs	-
Hanoi	0.3 secs	6.1 - 27.4 secs	-
Queens32	0.5 secs	4 min 6 secs	(**45 min** for n=14)
Switching	8.5 secs	too hard to model	-
SATLib (600 vars, 2137 clauses)	3.14 secs	-	-

*direct encoding in Kodkod, hard to model in Alloy ?

CrewAllocation

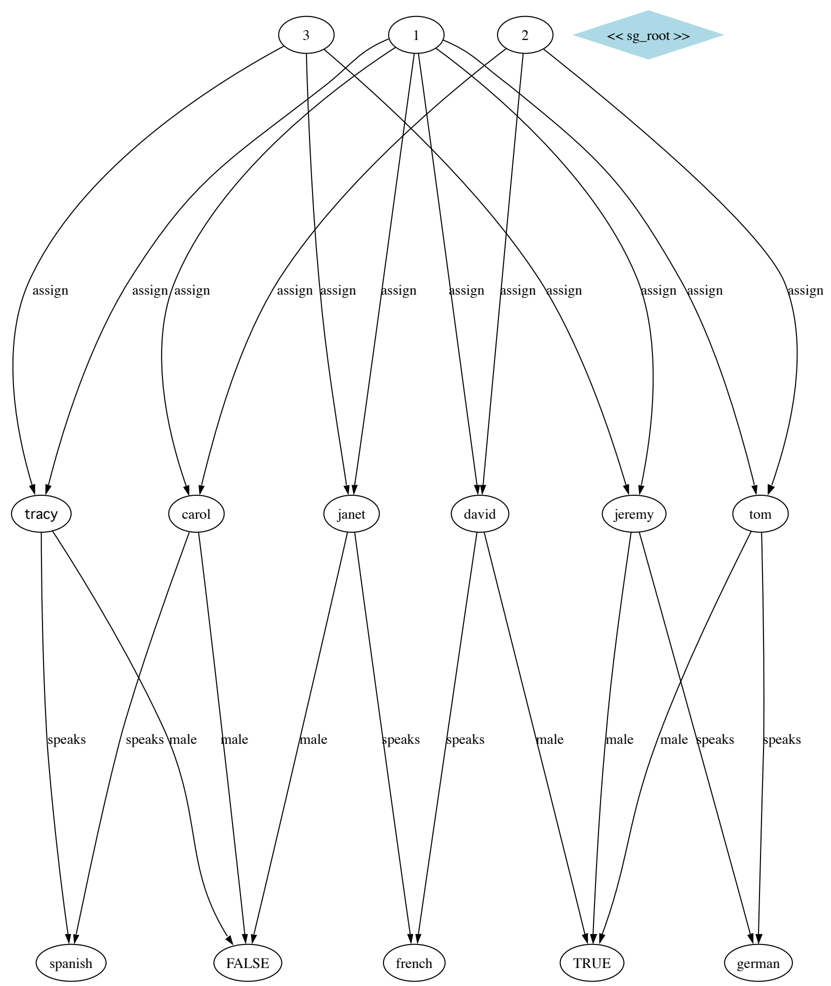
```
MACHINE CrewAllocationConstants
DEFINITIONS
 NRF == 3; FLIGHTS == 1..3;
 CONSTR1 == (!f.(f:FLIGHTS => speaks[assign[{f}]] = LANGUAGE));
                    /* all languages must be represented on all flights */
 CONSTR2 == (!f.(f:FLIGHTS => male[assign[{f}]] = BOOL)); /* both sexes must be on all flights */
 CONSTR3 == (!(f,p).(f:FLIGHTS & f<NRF-1 & p:PERSONNEL &  f|->p:assign & (f+1)|->p:assign
               => (f+2)|->p /: assign)); /* break of at least two after flight */
 CONSTR4 == (ran(assign) = PERSONNEL);
SETS
 PERSONNEL = {tom, david, jeremy, carol, janet, tracy};
 LANGUAGE = {french,german,spanish}
CONSTANTS male, speaks, assign
PROPERTIES
  male : PERSONNEL --> BOOL &
  speaks : PERSONNEL <-> LANGUAGE &
  ran(male) = BOOL & ran(speaks) = LANGUAGE &
  male = { tom|->TRUE, david|->TRUE, jeremy|->TRUE, carol|->FALSE, janet|->FALSE, tracy|-
>FALSE} &
  speaks = { tom|->german, david |-> french, jeremy |-> german, carol |-> spanish, janet |-> french,
tracy |-> spanish }
 &
  assign: FLIGHTS <-> PERSONNEL
  & CONSTR1 & CONSTR2 & CONSTR3 & CONSTR4
END
```

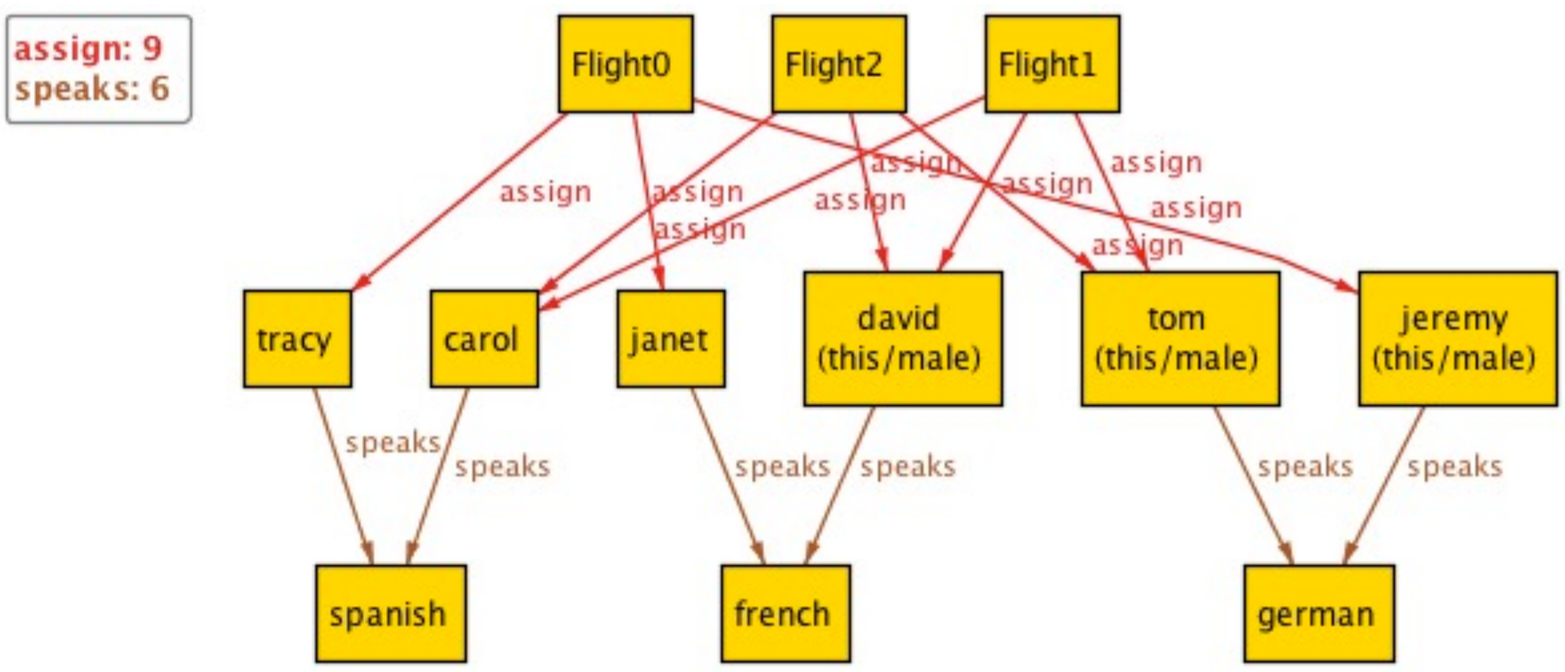
CrewAllocation in Alloy

```
open util/ordering [Flight]

abstract sig Language {}
one sig french, german, spanish extends Language {}
abstract sig Personell { speaks : set Language }
one sig tom, david, jeremy, carol, janet, tracy extends Personell {}
sig male in Personell {}
sig Flight {
  assign : set Personell
}

fact defmale {
  male = tom+david+jeremy
}
fact deflang {
  speaks =tom->german + david->french + jeremy->german + carol->spanish + janet->french + tracy->spanish
}
pred allLanguages {
  all f:Flight | f.assign.speaks = Language
}
pred allSexes {
  all f:Flight | some (f.assign-male) and some (f.assign & male)
}
pred scheduleOk {
  all p:Personell, f : Flight | p in f.assign and p in next[f].assign => not (p in next[next[f]].assign)
}
pred everybodyInSchedule {
  univ.assign = Personell
}
pred crewAlloc {
  allLanguages and allSexes and scheduleOk and everybodyInSchedule
}

run crewAlloc for 3 Flight
```

Crew Allocation Performance

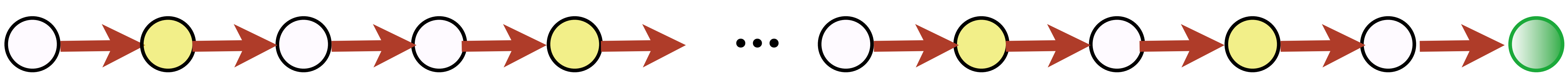
- ProB: 1.24 seconds
- Alloy: 0.03 seconds minisat



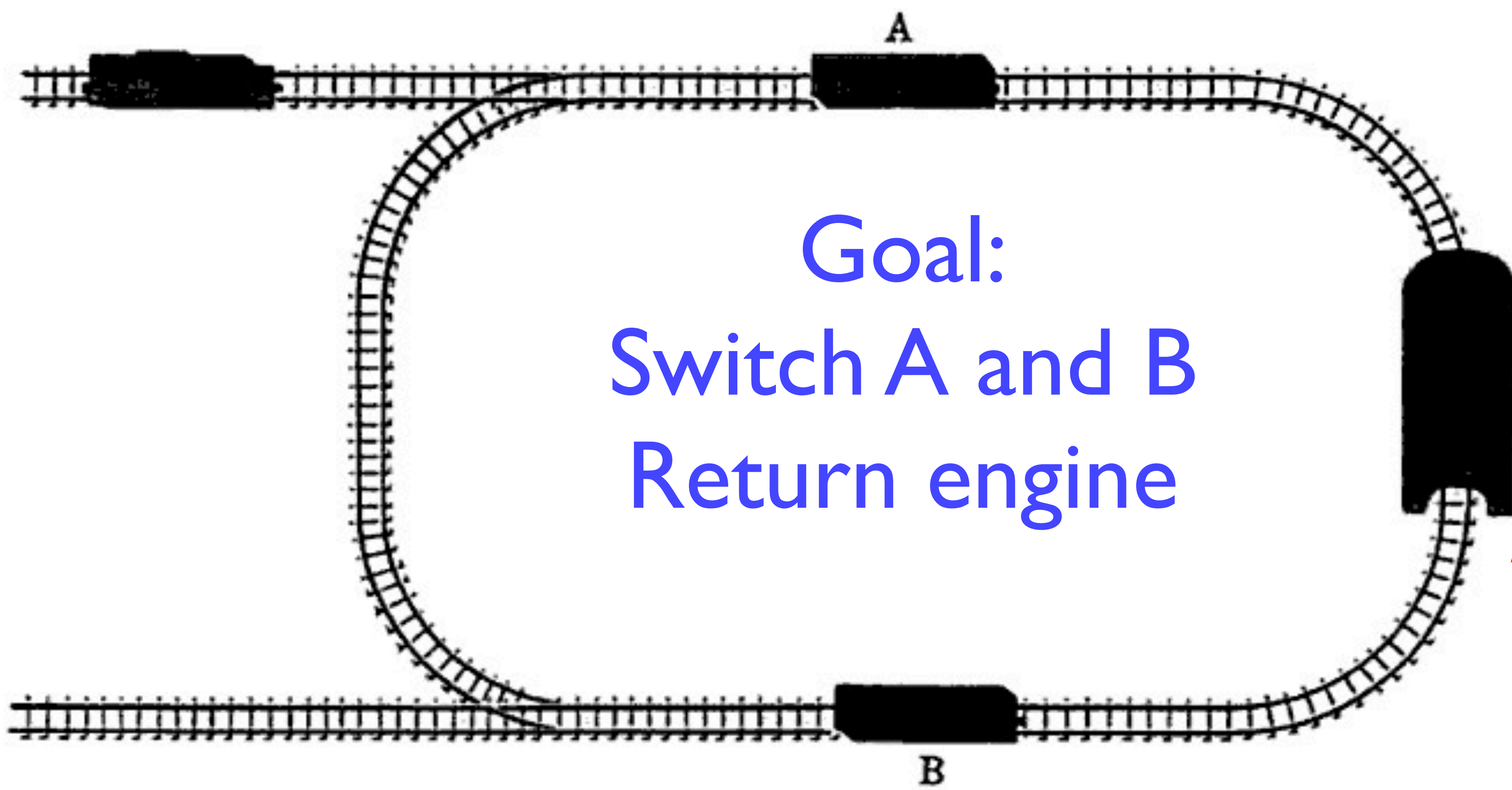


Hanoi Performance (5 Discs)

- Alloy: 27.4 secs Sat4J, 6.1 secs minisat; upper limit on solution had to be specified
- ProB: 0.3 secs
- Explanation:
    - BMC of Alloy does not memoize
- Some problems better suited to model checking



Gardner Switching Puzzle



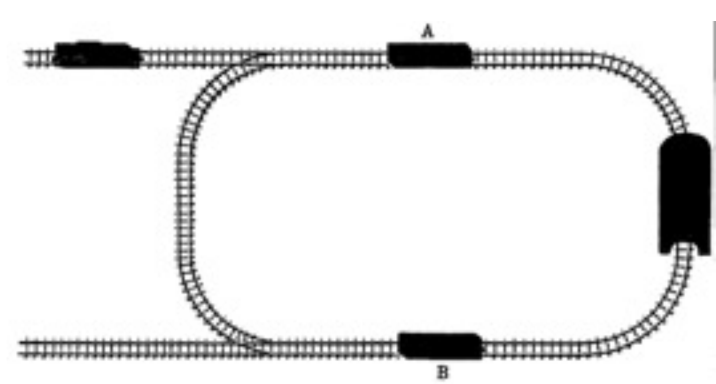
Goal:
Switch A and B
Return engine

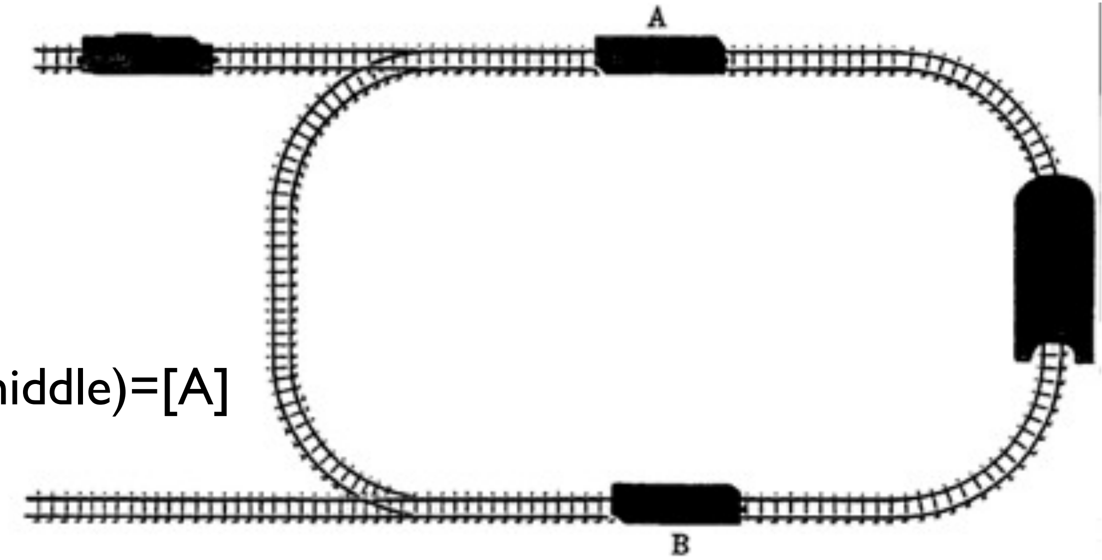
Tunnel not wide enough for A,B

Source: Martin Gardner, My Best Mathematical and Logical Puzzles, pages 22-23

Modelling the Puzzle

- Modelling in Alloy inconvenient; stopped
- Moddelling in Classical B:
  - 33 lines; shortest solution found in 8.5 seconds (initialisation+22 steps)
- Didn't attempt Event-B solution:
  - absence of Sequences
    (waiting for Math Extensions)

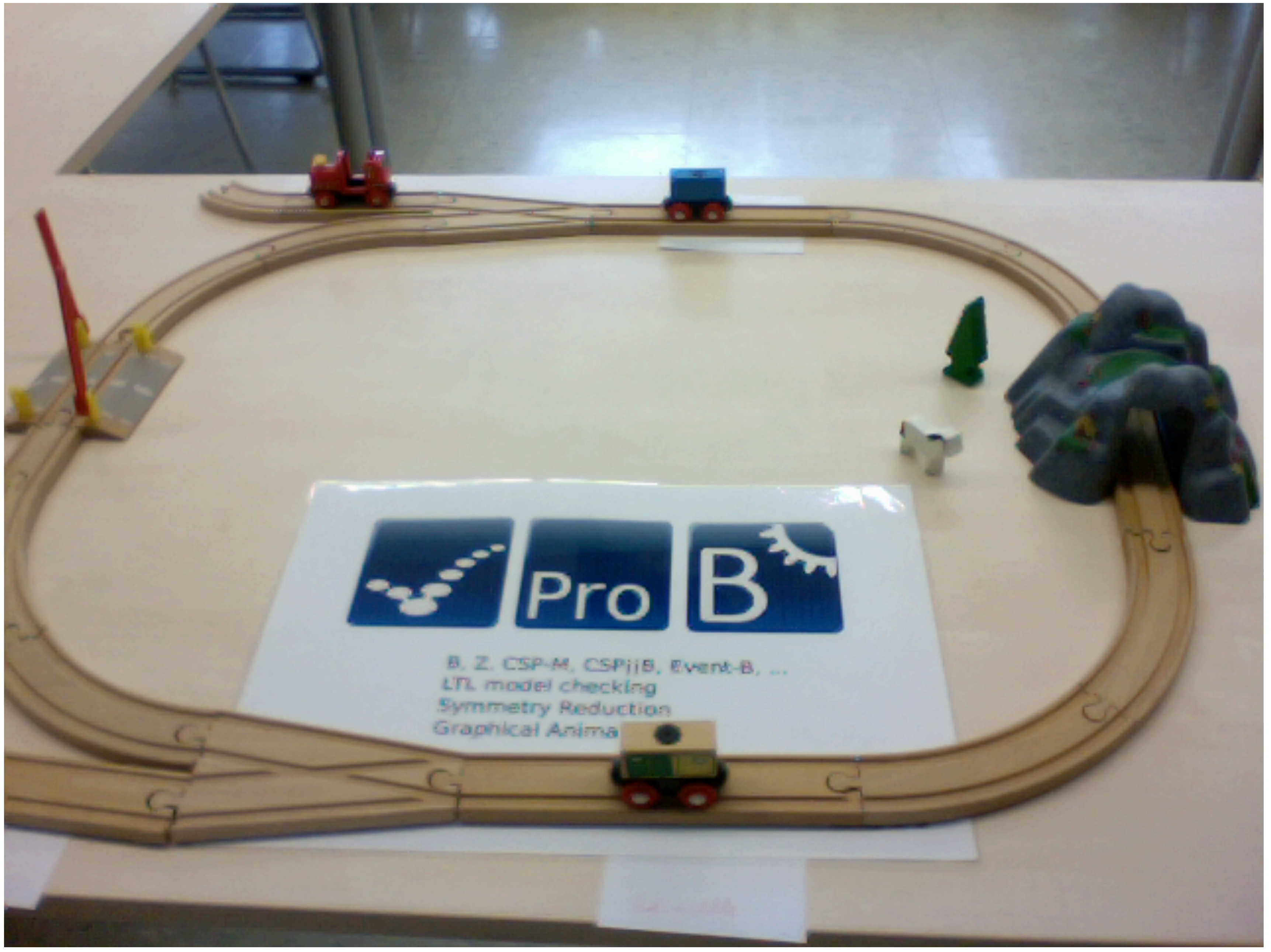





```
MACHINE GardnerSwitchingPuzzle_v2
SETS
 TRAINS={engine,A,B};
 TRACKS = {topleft,top_middle,bot_left,bot_middle,leftlink}
DEFINITIONS
 GOAL == occ(topleft) = [engine] & occ(top_middle)=[B] & occ(bot_middle)=[A]
CONSTANTS
  link, restrict
PROPERTIES
  link = {topleft |->top_middle, leftlink |->top_middle, top_middle |-> bot_middle, /* Tunnel */
          bot_middle|-> bot_left, bot_middle |-> leftlink} &
  restrict = (link*{{}}) <+ { (top_middle|->bot_middle) |-> {A,B} } /* A,B are not allowed to take the tunnel */
VARIABLES occ
INVARIANT
  occ: TRACKS --> iseq(TRAINS) &
  !(t1,t2).(t1:TRACKS & t1/=t2 => ran(occ(t1)) /\ ran(occ(t2)) = {} ) &
  UNION(t).(t:TRACKS|ran(occ(t))) = TRAINS
INITIALISATION occ := {topleft |-> [engine], top_middle |-> [A], bot_middle |->[B],
                  leftlink |-> <>, bot_left |-> <> }
OPERATIONS
  Move(Seq,T1,T2,Rest) = PRE Seq : iseq1(TRAINS) & Rest : iseq(TRAINS) &
         occ(T1)= Rest^Seq & engine:ran(Seq) & T1|->T2 : link &
         restrict((T1,T2)) /\ ran(Seq) = {} THEN
   occ := occ <+ {T1 |-> Rest, T2 |-> (Seq^occ(T2))}
  END;
  MoveRev(Seq,T1,T2,Rest) = PRE Seq : iseq1(TRAINS) & Rest : iseq(TRAINS) &
         occ(T1)= Seq^Rest & engine:ran(Seq) & T2|->T1 : link &
         restrict((T2,T1)) /\ ran(Seq) = {} THEN
   occ := occ <+ {T1 |-> Rest, T2 |-> (occ(T2)^Seq)}
  END
END
```

only two operations

Visualization of the Counterexample using the new Pro**B**rio Animation Plugin



Pro
B
B, Z, CSP-M, CSP||B, Event-B, ...
LTL model checking
Symmetry Reduction

Conclusions So Far

- B/Event-B often very compact and elegant
  - no example was easier in other language
  - some examples not done in Alloy (Train,7K2Q)
- ProB good at arithmetic & $\neq$ (Alphametic, Queens)
- Some problems are better expressed as model checking tasks (Hanoi, Train)
- For some relational constraints, Alloy is much faster than ProB (CrewAllocation)

Practical and Industrial Relevance ??

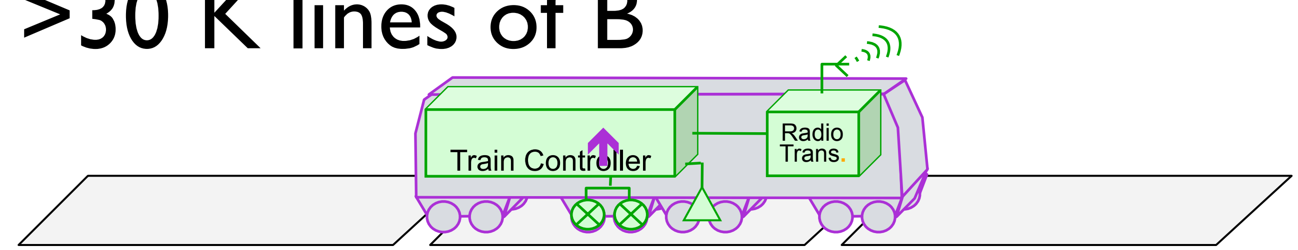
Dream

- Be able to use B/Event-B as a high-level constraint programming language

Initial Motivation: Property Validation

cf. [FM'09]

- Do properties (gluing invariant + concrete predicates extracted from ADA code) imply assumptions made during proof?
- Large B Models, large constants
  - San Juan: 79 files, >23 K Lines of B
    - 226 properties $\Rightarrow^?$ 147 assumptions
  - Paris L1: 74 Files, > 10K lines of B
  - Sao Paolo L4: 210 files, >30 K lines of B



Some of the 226 Properties

(solved in 0.56 seconds by ProB)

```
cfg_canton_cdv = {aa,bb|(aa : t_canton_acs & bb : t_cdv_acs) & bb : cfg_canton_cdv_liste_i
[cfg_canton_cdv_deb(aa) .. cfg_canton_cdv_fin(aa)]}
&
!(xx).(xx : t_aig_acs => cfg_aig_cdv_encl_indice(xx) <| cfg_aig_cdv_encl_liste : NATURAL >+>
t_cdv_acs)
&
!(xx).(xx : t_aig_acs & cfg_aig_cdv_encl_indice(xx) /= {} => cfg_aig_cdv_encl_indice(xx) = min
(cfg_aig_cdv_encl_indice(xx)) .. max(cfg_aig_cdv_encl_indice(xx)))
&
cfg_cdv_block = cfg_cdv_i~[{c_cdv_block}]
&
cfg_cdv_i = (0 .. 56) * {2} <+ {1 |-> 0,2 |-> 0,3 |-> 0,4 |-> 0,5 |-> 0,6 |-> 0,7 |-> 0,8 |-> 0,9 |-> 0,
10 |-> 0,11 |-> 0,12 |-> 0,13 |-> 0,14 |-> 0,15 |-> 0,16 |-> 0,17 |-> 0,18 |-> 0,19 |-> 0,
20 |-> 1,21 |-> 1,22 |-> 1,23 |-> 1,24 |-> 1,25 |-> 1,26 |-> 1,27 |-> 1,28 |-> 1,29 |-> 1,
30 |-> 1,31 |-> 1,32 |-> 1,33 |-> 1,34 |-> 1}
```

Use of the B Method developped by

CLEARSY
SYTEM ENGINEERING

Metros and Trains equipped with B SIL4 software

SIEMENS Use of ProB to validate toplogy and deployment configuration

Bosch
Adaptive Cruise Control

- Every possible situation anticipated in Model
  ⇛ Deadlock Freedom PO:
  - Disjunction of guards, >2 pages of Event-B
  - Proving tedious & difficult
- Idea: use ProB to solve constraint:
  - Axioms & INV & ¬ ($Guard_1 \vee \ldots \vee Guard_n$)

Axioms I

```
context c0

constants CONTROL // Mode of the CrCtl in which the Vehicle Speed
is controlled at a "fixed" speed
          ACONTROL // Mode of the CrCtl in which the Vehicle is
accelerated or decelerated
          NOCONTROL // Mode of the CrCtl in which the Vehicle
Speed is not influenced by the CrCtl
          T_AbstractMode // Name of the set of  the modes/abstact
states CONTROL, ACONTROL, NOCONTROL


sets T_Env_ControlSignals // Type of the control signals
     T_Env_Vehicle // Type of the vehicle signals
     T_Display // Type  of the CrCtl display signals
     T_Env_Output // Type of the Env Output signals
     T_Acceleration // Type for the acceleration - muss durch Z
ersetzt werden
     T_Speed // Type for the vehicle speed
     T_Para // Type for the parameters
     T_Mode // Tpye of the partiton of the abstract states
CONTROL, ACONTROL, NOCONTROL
     T_CrCtl_TimeStatus // Type for containerner concerning time


axioms
  @axm T_AbstractMode ⊆ ℙ(T_Mode)
  @axm6 partition(T_Mode, CONTROL, ACONTROL, NOCONTROL)
  @axm7 CONTROL ≠ ∅
  @axm8 ACONTROL ≠ ∅
  @axm9 NOCONTROL ≠ ∅
  @axm10 T_AbstractMode = {CONTROL, ACONTROL, NOCONTROL}
  theorem @thm1 CONTROL ≠ ACONTROL
  theorem @thm2 CONTROL ≠ NOCONTROL
  theorem @thm3 ACONTROL ≠ NOCONTROL
  theorem @thm4 T_Mode = CONTROL ∪ ACONTROL ∪ NOCONTROL
end
```

```
context c1 extends c0

constants UBAT_OFF // Batterie off
          INIT // init state
          OFF_BRAKE_READY // CrCtl Off, Brake pressed
          OFF_BRAKE_WAIT // CrCtl Off, waiting for brake presse
          STANDBY // CrCtl on no influence
          STD_BRAKE_WAIT // CrCtl on, waiting for brake, no
influence
          ERROR // non recoverable Error
          R_ERROR // reversible Error
          CRUISE // Maintaining a target speed
          RESUME // reaching a target speed
          RAMP_DOWN // comfort switch off
          ACC // maintaining a target acceleration
          DEC // maintaining a target acceleration


sets T_Env_Output_Mode // Type of the mode output


axioms
  @axm1 partition(NOCONTROL,{UBAT_OFF},{OFF_BRAKE_WAIT},
{OFF_BRAKE_READY},{ERROR}, {R_ERROR}, {STANDBY},
{STD_BRAKE_WAIT},{INIT})
  @axm2 partition(CONTROL,{CRUISE},{RESUME})
  @axm3 partition(ACONTROL,{ACC},{DEC},{RAMP_DOWN})
  theorem @thm1 NOCONTROL = {UBAT_OFF, OFF_BRAKE_WAIT,
OFF_BRAKE_READY, ERROR, R_ERROR, STANDBY, STD_BRAKE_WAIT, INIT}
  theorem @thm2 CONTROL = {CRUISE, RESUME}
  theorem @thm3 ACONTROL = {ACC, DEC, RAMP_DOWN}
  theorem @thm4 {UBAT_OFF, OFF_BRAKE_WAIT, OFF_BRAKE_READY,
ERROR, R_ERROR, STANDBY, STD_BRAKE_WAIT, INIT} ∪ {CRUISE, RESUM
∪ {ACC, DEC, RAMP_DOWN} = T_Mode
end
```

Axioms 2

```
axioms
  @axm11 PS_SET ⊆ T_Env_PedalSignals
  @axm14 PS_NO_ERROR ⊆ T_Env_PedalSignals
  @axm12 PS_NEUTRAL ⊆ T_Env_PedalSignals
  @axm13 PS_ERROR ⊆ T_Env_PedalSignals

  @axm_c3_25 PS_NO_ERROR = PS_SET ∪ PS_NEUTRAL//included in c2 as and axiom, has been proven before
  @axm_c3_10 PS_ERROR ∪ PS_NO_ERROR = T_Env_PedalSignals//included in c2 as and axiom, has been proven before
  @axm_c3_11 PS_SET ∪ PS_NEUTRAL ∪ PS_ERROR = T_Env_PedalSignals//included in c2 as and axiom, has been proven before
  @axm_c3_12 PS_ERROR ∩ PS_NO_ERROR = ∅//included in c2 as and axiom, has been proven before
  @axm_c3_13 PS_SET ∩ PS_NEUTRAL = ∅//included in c2 as and axiom, has been proven before
  @axm_c3_14 PS_SET ∩ PS_ERROR = ∅//included in c2 as and axiom, has been proven before
  @axm_c3_15 PS_NEUTRAL ∩ PS_ERROR = ∅//included in c2 as and axiom, has been proven before

  @axm35 VS_NOERRORCOND ⊆ T_Env_Vehicle_ErrorCond
  @axm45 VS_ERRORCOND ⊆ T_Env_Vehicle_ErrorCond
  @axm150 VS_NOERRORCOND ∪ VS_ERRORCOND = T_Env_Vehicle_ErrorCond
  @axm154 VS_NOERRORCOND ∩ VS_ERRORCOND = ∅

  @axm37 VS_NOSWITCHOFFCOND ⊆ T_Env_Vehicle_SwitchOffCond
  @axm46 VS_SWITCHOFFCOND ⊆ T_Env_Vehicle_SwitchOffCond
  @axm160 VS_NOSWITCHOFFCOND ∪ VS_SWITCHOFFCOND = T_Env_Vehicle_SwitchOffCond
  @axm161 VS_NOSWITCHOFFCOND ∩ VS_SWITCHOFFCOND = ∅

  @axm38 VS_NOCOMFORTSWITCHOFFCOND ⊆ T_Env_Vehicle_ComfortSwitchOffCond
  @axm50 VS_COMFORTSWITCHOFFCOND ⊆ T_Env_Vehicle_ComfortSwitchOffCond
  @axm170 VS_NOCOMFORTSWITCHOFFCOND ∪ VS_COMFORTSWITCHOFFCOND = T_Env_Vehicle_ComfortSwitchOffCond
  @axm171 VS_NOCOMFORTSWITCHOFFCOND ∩ VS_COMFORTSWITCHOFFCOND = ∅

  @axm17 CIS_ERROR ⊆ T_Env_ControlInterfaceSignals
  @axm18 CIS_NO_ERROR ⊆ T_Env_ControlInterfaceSignals
  @axm16 CIS_NEUTRAL ⊆ T_Env_ControlInterfaceSignals
  @axm15 CIS_SET ⊆ T_Env_ControlInterfaceSignals
  @axm43 CIS_MAIN_OFF ⊆ T_Env_ControlInterfaceSignals
  @axm44 CIS_MAIN_ON ⊆ T_Env_ControlInterfaceSignals
  @axm101 CIS_ERROR ∪ CIS_NO_ERROR ∪ CIS_NEUTRAL ∪ CIS_SET ∪ CIS_MAIN_OFF ∪ CIS_MAIN_ON =
T_Env_ControlInterfaceSignals
  @axm102 CIS_MAIN_ON ∩ CIS_MAIN_OFF = ∅
  @axm103  CIS_MAIN_OFF ∪ CIS_MAIN_ON ∪ CIS_ERROR = T_Env_ControlInterfaceSignals
  @axm104 CIS_SET ∩ CIS_NEUTRAL = ∅
  @axm20 T_CrCtl_TargetSpeed_Speed = ℤ
  @axm300 T_Env_TargetSpeed_Speed = ℤ
  @axm9 T_Env_IgnitionSignal = BOOL
  @axm2 partition(T_Env_Output_Mode_ECU, {ECU_INIT},{ECU_OFF}, {ECU_NOT_ACTIVE}, {ECU_ACTIVE}, {ECU_ERROR})
  @axm1 partition(T_Env_Output_Mode_Driver, {DISPLAY_ON}, {DISPLAY_OFF})
  @axm602 T_CrCtl_TargetSpeed_Speed = T_Env_TargetSpeed_Speed
  @axm19 partition(T_CrCtl_TargetSpeed_Status, {DEFINED}, {UNDEFINED})
  @axm400 partition(T_Env_TargetSpeed_Status, {DISPLAY_SPEED}, {NOT_DISPLAY_SPEED})
end
```

etc...

Deadlock Freedom PO

(P_Env_Vehicle_SwitchOffCond∈VS_NOSWITCHOFFCOND∧
P_Env_PedalSignals∈PS_NEUTRAL∧
P_Env_IgnitionSignal=TRUE ∧
P_Env_Vehicle_ErrorCond∈VS_NOERRORCOND ∧
P_CrCtl_Mode∈{STANDBY,CRUISE,RESUME,ACC,DEC} ∧
P_Env_ControlInterfaceSignals∈CIS_SET ∧
P_Env_ControlInterfaceSignals∈CIS_MAIN_ON ∧
P_Env_ControlInterfaceSignals∈CIS_ERROR)∨

((P_Env_Vehicle_InitRequest=TRUE∨
P_Env_Vehicle_ErrorCond∈VS_ERRORCOND∨
P_Env_Vehicle_SwitchOffCond∈VS_SWITCHOFFCOND∨
P_Env_Vehicle_ComfortSwitchOffCond∈VS_COMFORTSWITCHOFFCOND)∧
P_Env_IgnitionSignal=TRUE)∨

(P_Env_IgnitionSignal=TRUE ∧
P_CrCtl_Mode=ERROR)∨

(P_Env_Vehicle_SwitchOffCond∈VS_NOSWITCHOFFCOND∧
P_Env_PedalSignals∈PS_NEUTRAL∧
P_Env_IgnitionSignal=TRUE ∧
P_Env_Vehicle_ErrorCond∈VS_NOERRORCOND ∧
P_CrCtl_Mode∈{STANDBY,CRUISE,RESUME,ACC,DEC} ∧
P_Env_ControlInterfaceSignals∈CIS_NEUTRAL ∧
P_Env_ControlInterfaceSignals∈CIS_ERROR ∧
P_Env_ControlInterfaceSignals∈CIS_MAIN_ON ∧
P_Env_Vehicle_ComfortSwitchOffCond∈VS_NOCOMFORTSWITCHOFFCOND)∨

(P_Env_IgnitionSignal=TRUE ∧
P_CrCtl_Mode=UBAT_OFF)∨

(P_CrCtl_Mode=STD_BRAKE_WAIT ∧
P_Env_ControlInterfaceSignals∈CIS_MAIN_ON ∧
P_Env_IgnitionSignal=TRUE ∧
P_Env_PedalSignals∈PS_SET ∧
P_Env_Vehicle_ErrorCond∈VS_NOERRORCOND)∨

(P_Env_IgnitionSignal=TRUE ∧
P_Env_Vehicle_SwitchOffCond∈VS_NOSWITCHOFFCOND ∧
P_Env_ControlInterfaceSignals∈CIS_MAIN_ON ∧
P_Env_ControlInterfaceSignals∈CIS_ERROR ∧
P_Env_Vehicle_ErrorCond∈VS_NOERRORCOND ∧
P_CrCtl_Mode=RAMP_DOWN)∨

(P_Env_IgnitionSignal=TRUE ∧
P_Env_Vehicle_SwitchOffCond∈VS_NOSWITCHOFFCOND ∧
P_Env_ControlInterfaceSignals∈CIS_NEUTRAL ∧
P_Env_ControlInterfaceSignals∈CIS_MAIN_ON ∧
P_Env_ControlInterfaceSignals∈CIS_ERROR ∧
P_Env_Vehicle_ErrorCond∈VS_NOERRORCOND ∧
P_CrCtl_Mode=RAMP_DOWN)∨

etc...

Counter Example Found

2^{8231}

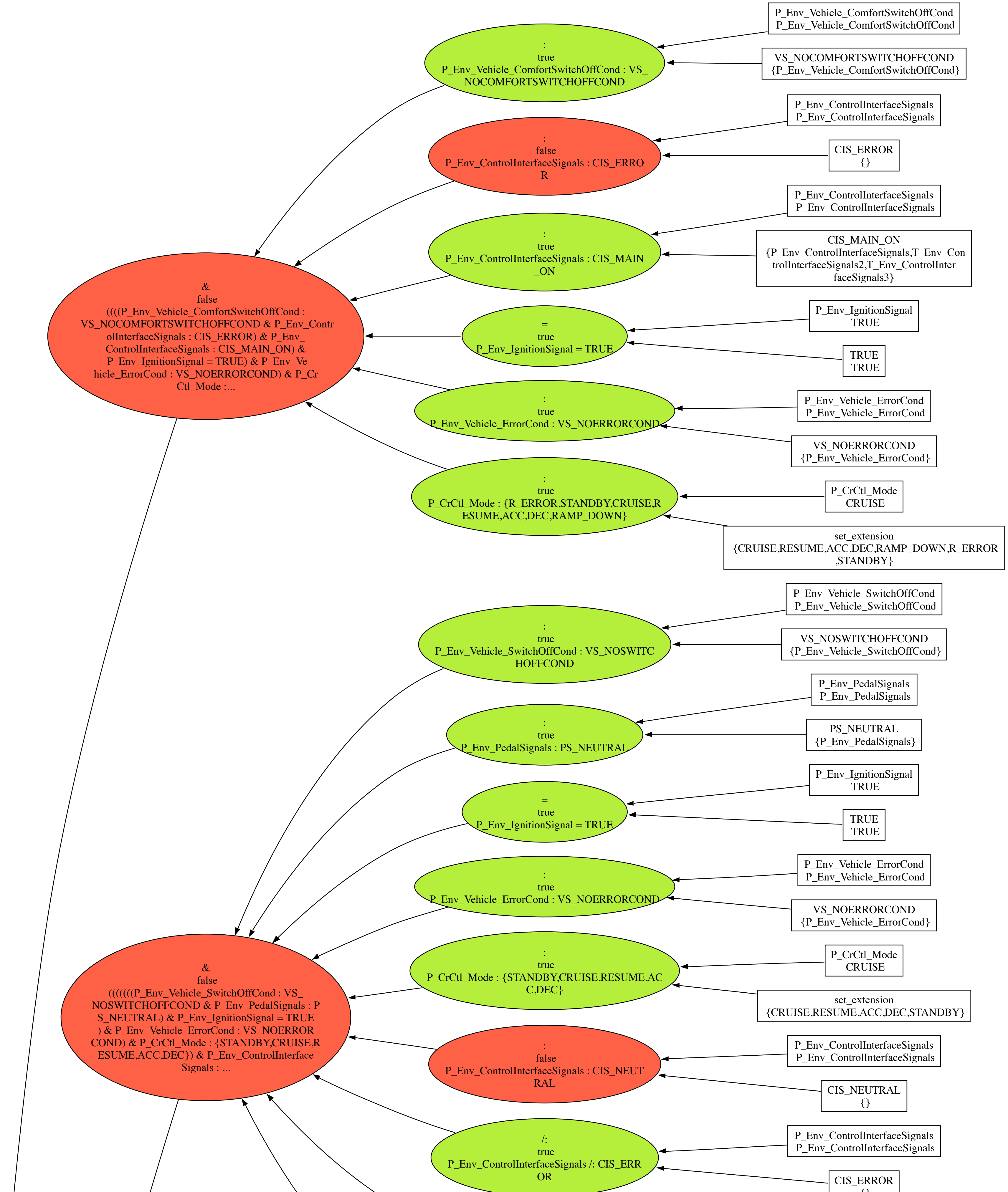
$\times$

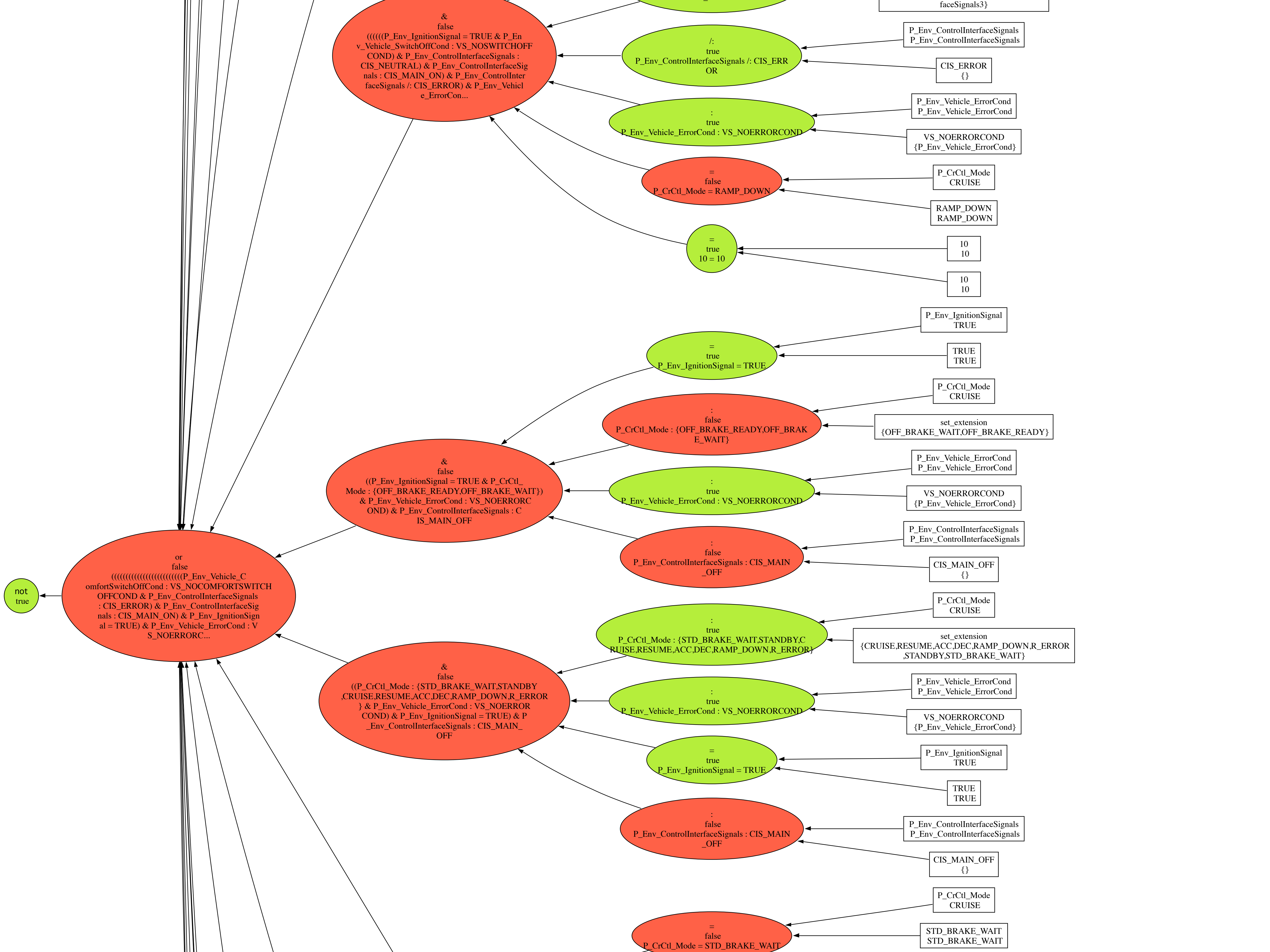
54,525,952

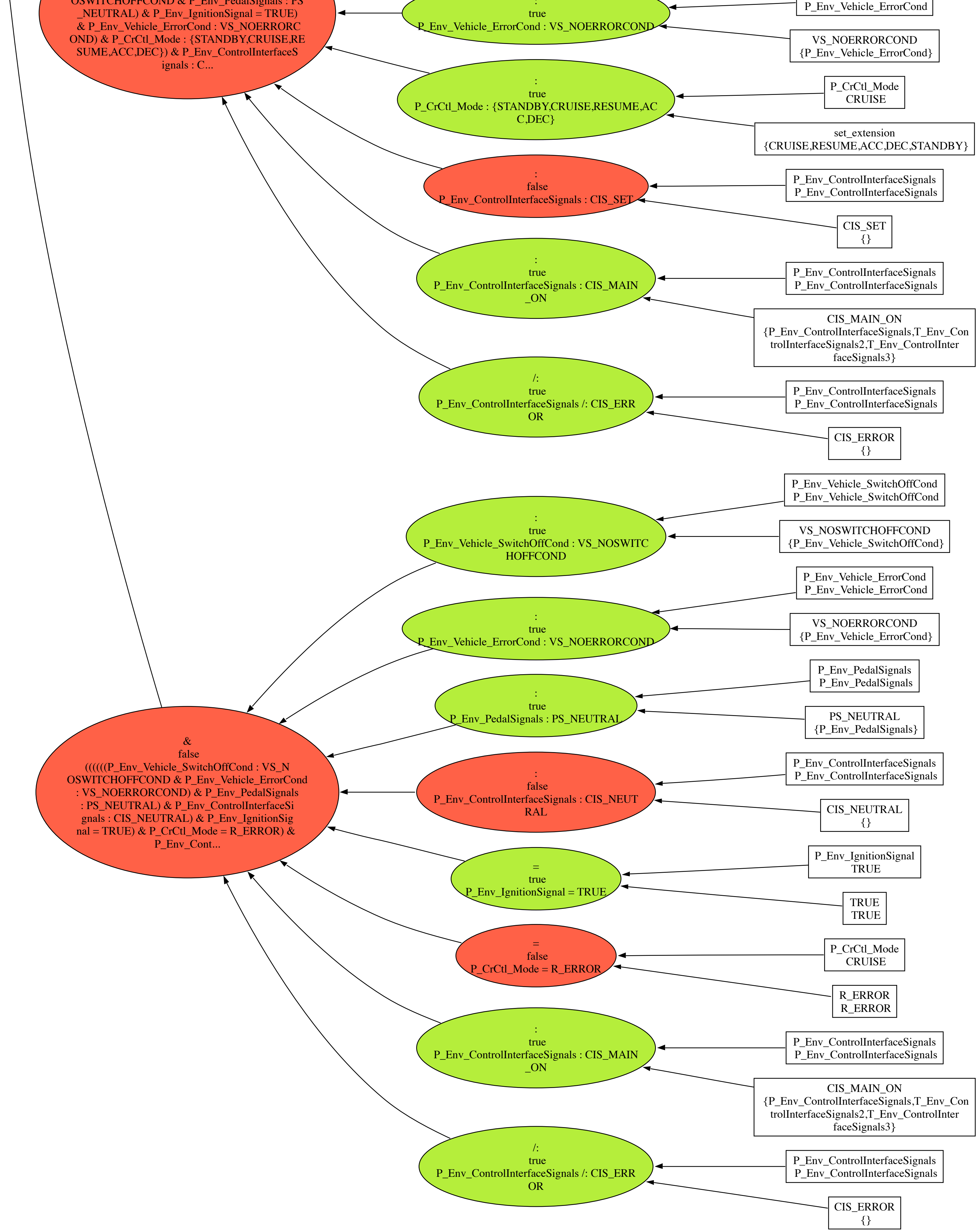
Possibilities

Name	Value
▼ c0	
ACONTROL	{ACC,DEC,RAMP_DOWN}
CONTROL	{CRUISE,RESUME}
NOCONTROL	ROR,STANDBY,STD_BRAKE_WAIT,INIT}
T_AbstractMode	:OR,STANDBY,STD_BRAKE_WAIT,INIT}}
▼ c2	
CIS_ERROR	ıals4,T_Env_ControlInterfaceSignals5}
CIS_MAIN_OFF	∅
CIS_MAIN_ON	∅
CIS_NEUTRAL	∅
CIS_NO_ERROR	∅
CIS_SET	ıals4,T_Env_ControlInterfaceSignals5}
PS_ERROR	∅
PS_NEUTRAL	∅
PS_NO_ERROR	,_PedalSignals4,T_Env_PedalSignals5}
PS_SET	,_PedalSignals4,T_Env_PedalSignals5}
T_CrCtl_TargetSpeed_Speed	Z
T_Env_IgnitionSignal	{FALSE,TRUE}
T_Env_TargetSpeed_Speed	Z
VS_COMFORTSWITCHOFFCO	∅
VS_ERRORCOND	∅
VS_NOCOMFORTSWITCHOFF	:nv_Vehicle_ComfortSwitchOffCond2}
VS_NOERRORCOND	:rrorCond,T_Env_Vehicle_ErrorCond2}
VS_NOSWITCHOFFCOND	{P_Env_Vehicle_SwitchOffCond}
VS_SWITCHOFFCOND	{T_Env_Vehicle_SwitchOffCond2}
▼ vars	
P_CrCtl_Mode	CRUISE
P_Env_IgnitionSignal	TRUE
P_Env_InitEnd	TRUE
P_Env_Vehicle_InitRequest	FALSE

Analysis of PO




faceSignals3}
&
false
(((((P_Env_IgnitionSignal = TRUE & P_Env_Vehicle_SwitchOffCond : VS_NOSWITCHOFFCOND) & P_Env_ControlInterfaceSignals : CIS_NEUTRAL) & P_Env_ControlInterfaceSignals : CIS_MAIN_ON) & P_Env_ControlInterfaceSignals /: CIS_ERROR) & P_Env_Vehicle_ErrorCon...
/:
true
P_Env_ControlInterfaceSignals /: CIS_ERROR
P_Env_ControlInterfaceSignals
P_Env_ControlInterfaceSignals
CIS_ERROR
{}
:
true
P_Env_Vehicle_ErrorCond : VS_NOERRORCOND
P_Env_Vehicle_ErrorCond
P_Env_Vehicle_ErrorCond
VS_NOERRORCOND
{P_Env_Vehicle_ErrorCond}
=
false
P_CrCtl_Mode = RAMP_DOWN
P_CrCtl_Mode
CRUISE
RAMP_DOWN
RAMP_DOWN
=
true
10 = 10
10
10
10
10
P_Env_IgnitionSignal
TRUE
=
true
P_Env_IgnitionSignal = TRUE
TRUE
TRUE
P_CrCtl_Mode
CRUISE
:
false
P_CrCtl_Mode : {OFF_BRAKE_READY,OFF_BRAKE_WAIT}
set_extension
{OFF_BRAKE_WAIT,OFF_BRAKE_READY}
&
false
((P_Env_IgnitionSignal = TRUE & P_CrCtl_Mode : {OFF_BRAKE_READY,OFF_BRAKE_WAIT}) & P_Env_Vehicle_ErrorCond : VS_NOERRORCOND) & P_Env_ControlInterfaceSignals : CIS_MAIN_OFF
P_Env_Vehicle_ErrorCond
P_Env_Vehicle_ErrorCond
:
true
P_Env_Vehicle_ErrorCond : VS_NOERRORCOND
VS_NOERRORCOND
{P_Env_Vehicle_ErrorCond}
P_Env_ControlInterfaceSignals
P_Env_ControlInterfaceSignals
:
false
P_Env_ControlInterfaceSignals : CIS_MAIN_OFF
CIS_MAIN_OFF
{}
or
false
((((((((((((((((((((((P_Env_Vehicle_ComfortSwitchOffCond : VS_NOCOMFORTSWITCHOFFCOND & P_Env_ControlInterfaceSignals : CIS_ERROR) & P_Env_ControlInterfaceSignals : CIS_MAIN_ON) & P_Env_IgnitionSignal = TRUE) & P_Env_Vehicle_ErrorCond : VS_NOERRORC...
not
true
P_CrCtl_Mode
CRUISE
:
true
P_CrCtl_Mode : {STD_BRAKE_WAIT,STANDBY,CRUISE,RESUME,ACC,DEC,RAMP_DOWN,R_ERROR}
set_extension
{CRUISE,RESUME,ACC,DEC,RAMP_DOWN,R_ERROR,STANDBY,STD_BRAKE_WAIT}
&
false
((P_CrCtl_Mode : {STD_BRAKE_WAIT,STANDBY,CRUISE,RESUME,ACC,DEC,RAMP_DOWN,R_ERROR} & P_Env_Vehicle_ErrorCond : VS_NOERRORCOND) & P_Env_IgnitionSignal = TRUE) & P_Env_ControlInterfaceSignals : CIS_MAIN_OFF
P_Env_Vehicle_ErrorCond
P_Env_Vehicle_ErrorCond
:
true
P_Env_Vehicle_ErrorCond : VS_NOERRORCOND
VS_NOERRORCOND
{P_Env_Vehicle_ErrorCond}
=
true
P_Env_IgnitionSignal = TRUE
P_Env_IgnitionSignal
TRUE
TRUE
TRUE
:
false
P_Env_ControlInterfaceSignals : CIS_MAIN_OFF
P_Env_ControlInterfaceSignals
P_Env_ControlInterfaceSignals
CIS_MAIN_OFF
{}
P_CrCtl_Mode
CRUISE
=
false
P_CrCtl_Mode = STD_BRAKE_WAIT
STD_BRAKE_WAIT
STD_BRAKE_WAIT



OSWITCHOFFCOND & P_Env_PedalSignals : PS _NEUTRAL) & P_Env_IgnitionSignal = TRUE) & P_Env_Vehicle_ErrorCond : VS_NOERRORC OND) & P_CrCtl_Mode : {STANDBY,CRUISE,RE SUME,ACC,DEC}) & P_Env_ControlInterfaceS ignals : C...
:
true
P_Env_Vehicle_ErrorCond : VS_NOERRORCOND
P_Env_Vehicle_ErrorCond
VS_NOERRORCOND
{P_Env_Vehicle_ErrorCond}
:
true
P_CrCtl_Mode : {STANDBY,CRUISE,RESUME,AC C,DEC}
P_CrCtl_Mode
CRUISE
set_extension
{CRUISE,RESUME,ACC,DEC,STANDBY}
:
false
P_Env_ControlInterfaceSignals : CIS_SET
P_Env_ControlInterfaceSignals
P_Env_ControlInterfaceSignals
CIS_SET
{}
:
true
P_Env_ControlInterfaceSignals : CIS_MAIN _ON
P_Env_ControlInterfaceSignals
P_Env_ControlInterfaceSignals
CIS_MAIN_ON
{P_Env_ControlInterfaceSignals,T_Env_ControlInterfaceSignals2,T_Env_ControlInterfaceSignals3}
/:
true
P_Env_ControlInterfaceSignals /: CIS_ERR OR
P_Env_ControlInterfaceSignals
P_Env_ControlInterfaceSignals
CIS_ERROR
{}
&
false
((((((P_Env_Vehicle_SwitchOffCond : VS_N OSWITCHOFFCOND & P_Env_Vehicle_ErrorCond : VS_NOERRORCOND) & P_Env_PedalSignals : PS_NEUTRAL) & P_Env_ControlInterfaceSi gnals : CIS_NEUTRAL) & P_Env_IgnitionSig nal = TRUE) & P_CrCtl_Mode = R_ERROR) & P_Env_Cont...
:
true
P_Env_Vehicle_SwitchOffCond : VS_NOSWITC HOFFCOND
P_Env_Vehicle_SwitchOffCond
P_Env_Vehicle_SwitchOffCond
VS_NOSWITCHOFFCOND
{P_Env_Vehicle_SwitchOffCond}
:
true
P_Env_Vehicle_ErrorCond : VS_NOERRORCOND
P_Env_Vehicle_ErrorCond
P_Env_Vehicle_ErrorCond
VS_NOERRORCOND
{P_Env_Vehicle_ErrorCond}
:
true
P_Env_PedalSignals : PS_NEUTRAL
P_Env_PedalSignals
P_Env_PedalSignals
PS_NEUTRAL
{P_Env_PedalSignals}
:
false
P_Env_ControlInterfaceSignals : CIS_NEUT RAL
P_Env_ControlInterfaceSignals
P_Env_ControlInterfaceSignals
CIS_NEUTRAL
{}
=
true
P_Env_IgnitionSignal = TRUE
P_Env_IgnitionSignal
TRUE
TRUE
TRUE
=
false
P_CrCtl_Mode = R_ERROR
P_CrCtl_Mode
CRUISE
R_ERROR
R_ERROR
:
true
P_Env_ControlInterfaceSignals : CIS_MAIN _ON
P_Env_ControlInterfaceSignals
P_Env_ControlInterfaceSignals
CIS_MAIN_ON
{P_Env_ControlInterfaceSignals,T_Env_ControlInterfaceSignals2,T_Env_ControlInterfaceSignals3}
/:
true
P_Env_ControlInterfaceSignals /: CIS_ERR OR
P_Env_ControlInterfaceSignals
P_Env_ControlInterfaceSignals
CIS_ERROR
{}

Conclusion

- Many Constraint Satisfaction Problems can be very conveniently expressed in B
- ProB can sometimes solve them very effectively
  - but further research to be carried out
- Model Checking useful for some problems

- Jens Bendisposto
- Carl Friedrich Bolz
- Nadine Elbeshausen
- Fabian Fritz
- Marc Fontaine
- Stefan Hallerstede
- Michael Jastram
- Li Luo
- Sebastian Krings
- Daniel Plagge
- Mireille Samia
- Corinna Spermann
- Dennis Winter

- Michael Butler
- Thierry Massart
- Edd Turner

Thanks !

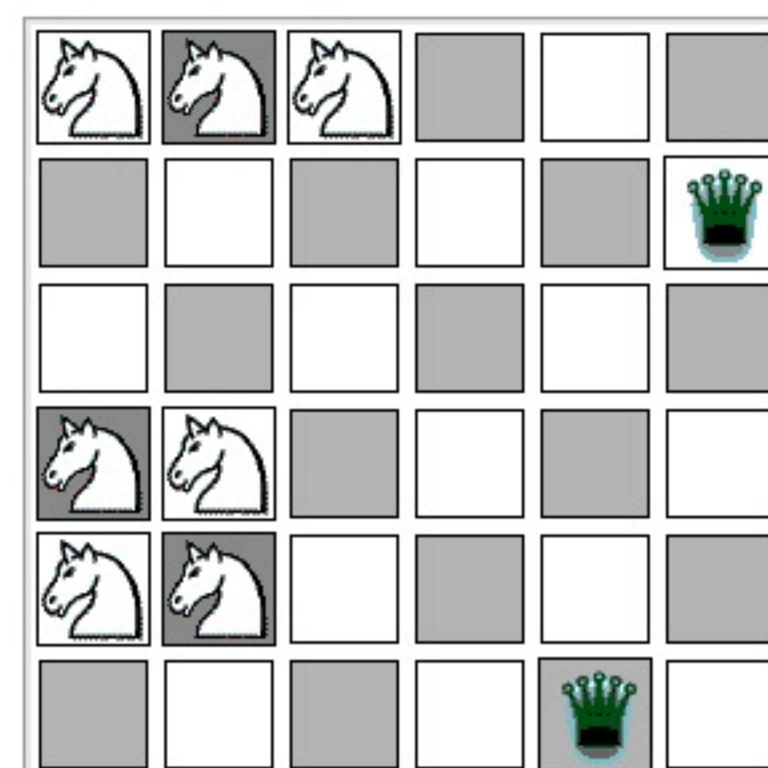
Extra Slides

```java
import java.util.Map;

import kodkod.ast.Formula;
import kodkod.ast.IntExpression;
import kodkod.ast.Relation;

/**
 * @author plagge
 */
public class CopyPasteSaveTools extends DigitPuzzle {
    public static void main(String[] args) {
        CopyPasteSaveTools cpst = new CopyPasteSaveTools();
        cpst.copyPastSaveTools();
    }

    public long copyPastSaveTools() {
        // the "variables"
        Relation c = Relation.unary("c");
        Relation o = Relation.unary("o");
        Relation p = Relation.unary("p");
        Relation y = Relation.unary("y");
        Relation a = Relation.unary("a");
        Relation s = Relation.unary("s");
        Relation t = Relation.unary("t");
        Relation e = Relation.unary("e");
        Relation v = Relation.unary("v");
        Relation l = Relation.unary("l");

        // the equation
        IntExpression copy = number(c, o, p, y);
        IntExpression paste = number(p, a, s, t, e);
        IntExpression save = number(s, a, v, e);
        IntExpression tools = number(t, o, o, l, s);

        Formula equation = copy.plus(paste).plus(save).eq(tools);

        // the formula oneDigit states that each relation contains exactl
        // one element, so we just have singleton sets
        // (this is necessary because Kodkod does not know types that are
        // no sets)
        Formula isDigit = isDigit(c, o, p, y, a, s, t, e, v, l);

        // the first digits should not be zero
        Formula notZero = notZero(c, p, s, t);

        // all digits are different
        Formula allDifferent = allDifferent(c, o, p, y, a, s, t, e, 

        // put all subformulas together
        Formula formula = allDifferent.and(isDigit).and(equation).an

        return solveIt(formula, c, o, p, y, a, s, t, e, v, l);
    }

    protected void checkSolution(Map<String, Integer> values) {
        int c = values.get("c");
        int o = values.get("o");
        int p = values.get("p");
        int y = values.get("y");
        int a = values.get("a");
        int s = values.get("s");
        int t = values.get("t");
        int e = values.get("e");
        int v = values.get("v");
        int l = values.get("l");

        int copy = c * 1000 + o * 100 + p * 10 + y;
        int past = p * 10000 + a * 1000 + s * 100 + t * 10 + e;
        int save = s * 1000 + a * 100 + v * 10 + e;
        int tools = t * 10000 + o * 1000 + o * 100 + l * 10 + s;

        StringBuilder sb = new StringBuilder();
        sb.append(c).append(o).append(p).append(y);
        sb.append(" + ");
        sb.append(p).append(a).append(s).append(t).append(e);
        sb.append(" + ");
        sb.append(s).append(a).append(v).append(e);
        sb.append(" = ");
        sb.append(t).append(o).append(o).append(l).append(s);

        System.out.println(sb.toString());

        if (copy + past + save != tools) {
            throw new IllegalStateException("no solution!");
        }
    }
```

return

A Variation: 7 Knights and 2 Queens

- More awkward to encode in Alloy (arithmetic)
- Still relatively easy in B
- ProB solves it in 0.64 secs



Graph Isomorphism

B:

```
@perm p∈ Nodes ⤖ Nodes
@iso ∀x,y•(x∈Nodes ∧ y∈Nodes ⇒
(x↦y∈graph1 ⇔ p(x)↦p(y) ∈ graph2))
```

TLA:

```
Solve == /\ solved = 0
         /\ solved' = 1
         /\ p' \in [1..n -> 1..n]
         /\ \A i \in 1..n : (\E j \in 1..n : p'[j]=i)
         /\ \A i \in 1..n : (p'[g1[i]] = g2[p'[i]])
         /\ UNCHANGED <<g1,g2,n>>
```

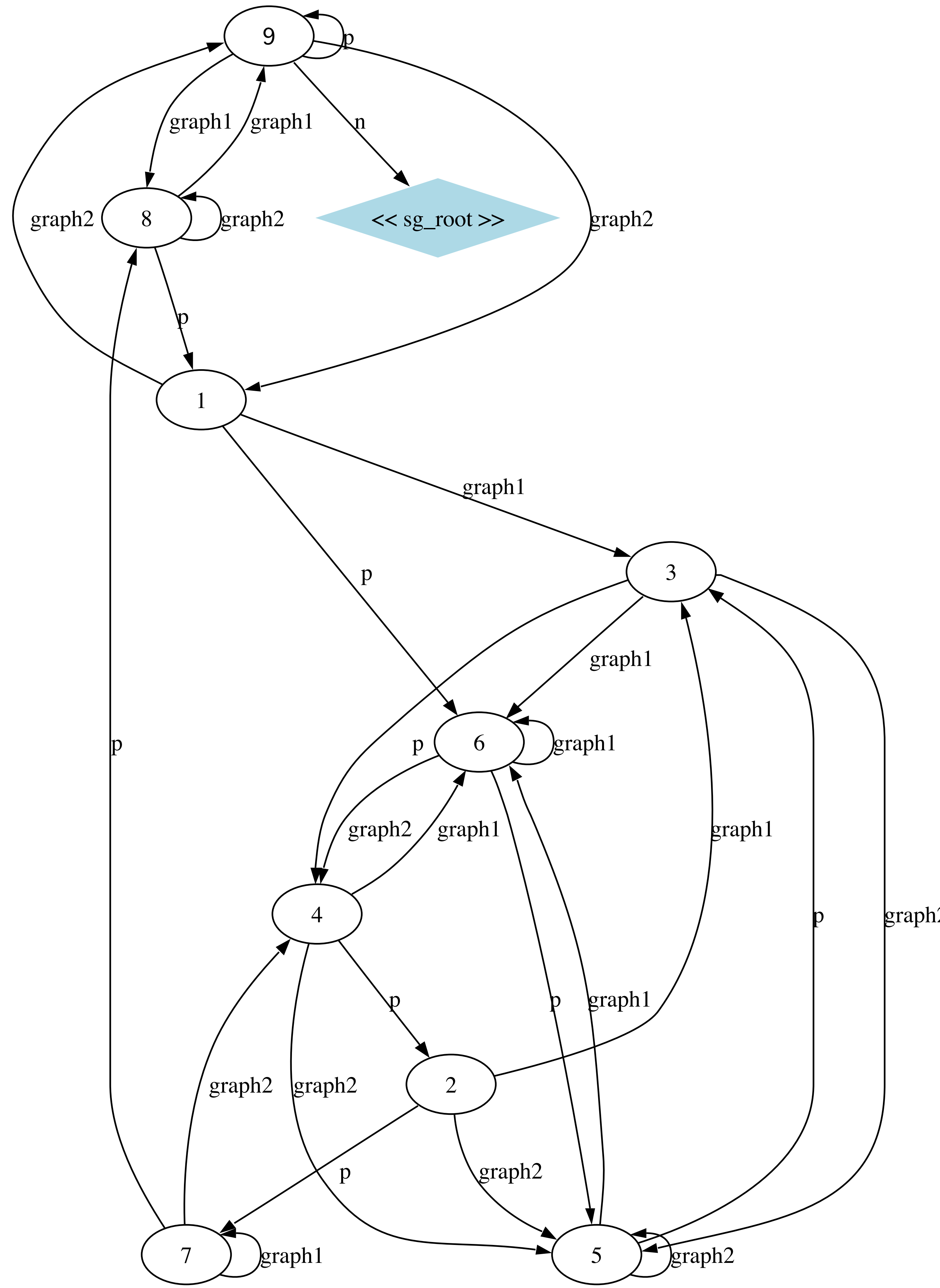
Alloy:

```
abstract sig Node {
  graph1 : set Node,
  graph2 : set Node,
  p : one Node
}
...
pred permutation {
  // p is already defined as a total function on Node
  // p is injective:
  p.~p in iden
  // p is surjective
  univ.p = Node
}
pred isomorph {
  permutation
  all n:Node | n.graph1.p = n.p.graph2
}
```

Performance

- graph1 = {1|->3,2|->3, 3|->6,4|->6,5|->6, 8|->9,9|->8, 6|->6, 7|->7}
- graph2 = {2|->5,3|->5,4|->5, 6|->4,7|->4, 1|->9,9|->1, 5|->5, 8|->8}
- TLC: 2 hours 6 minutes 27 seconds to find first solution [6, 7, 4, 2, 3, 5, 8, 1, 9]
- ProB: 0.1 secs (for all 8 solutions)
- Alloy: 0.11 secs (Sat4J), 0.05 secs (minisat)

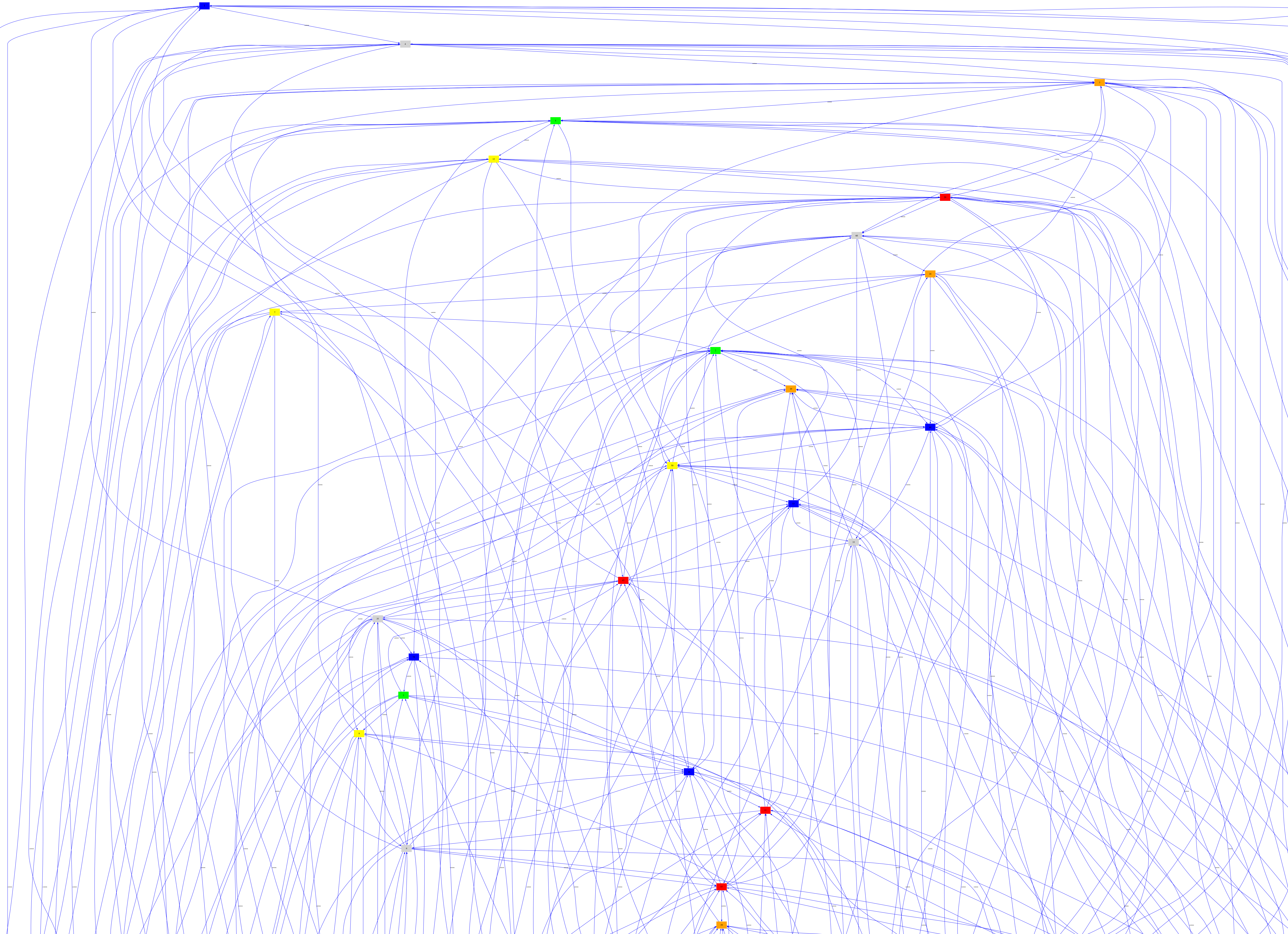
ProB Solution:

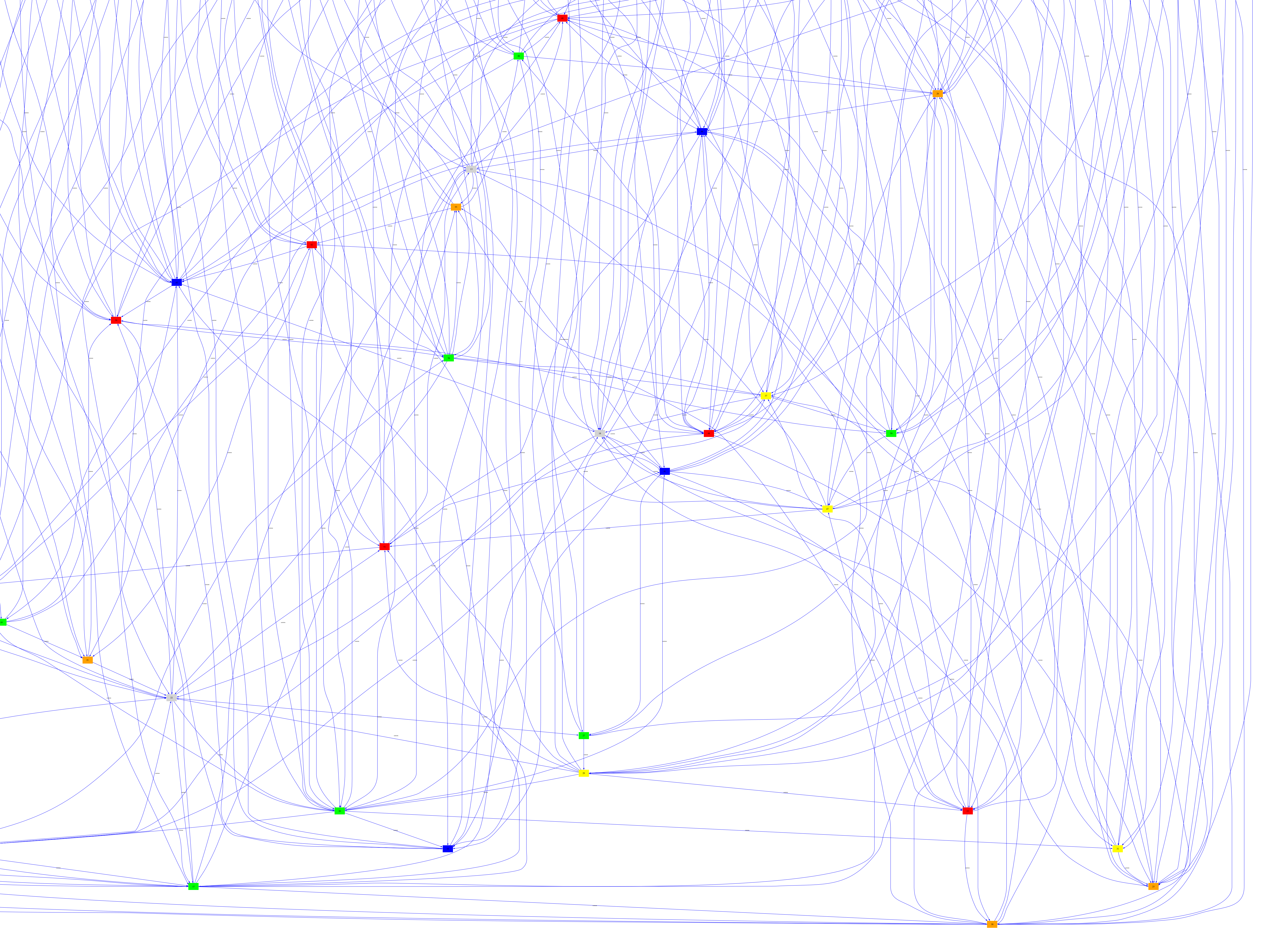


Graph Coloring

```
@ctype colour∈ Vtx ⟶ 1‥maxcol
@alldiff (∀i,j•i↦j∈Edge ⟹ colour(i) ≠  colour(j))
```

- ProB Performance:
- Graph with 60 nodes coloured with 6 colors in 0.25 seconds
- < 1 second to find out that no solution with 5 colours





Sudoku

- Alloy
  - Sudoku1.als : 1.036 seconds with Sat4J
  - Sudoku2.als: 0.455 seconds with minisat
- ProB: 0.46 seconds

```
axioms
  @axmd DOM = 1‥9
  @axm1 Board ∈ DOM ⟶ (DOM ⟶ DOM)
  @axms SUBSQ = { {1,2,3}, {4,5,6}, {7,8,9} }
  @axm2 ∀y·(y∈DOM ⇒ (∀x1,x2·(x1∈1‥8 ∧ x1<x2 ∧ x2∈2‥9
    ⇒ (Board(x1)(y) ≠ Board(x2)(y) ∧
       Board(y)(x1) ≠ Board(y)(x2)))))
  @axm3 ∀ s1,s2·(s1∈SUBSQ ∧ s2∈SUBSQ ⇒
       (∀x1,y1,x2,y2·( (x1∈s1 ∧ x2∈s1 ∧ x1≥x2 ∧
                       (x1=x2 ⇒ y1>y2) ∧
                       y1∈s2 ∧ y2∈s2 ∧ (x1↦y1) ≠ (x2↦y2))
       ⇒  Board(x1)(y1) ≠ Board(x2)(y2)
       )))
```

7	3	8	9	5	2	1	4	6
1	2	5	3	4	6	7	8	9
4	6	9	1	7	8	2	3	5
2	1	4	5	3	7	6	9	8
3	8	6	2	1	9	4	5	7
5	9	7	6	8	4	3	1	2
6	4	1	7	9	5	8	2	3
8	5	2	4	6	3	9	7	1
9	7	3	8	2	1	5	6	4

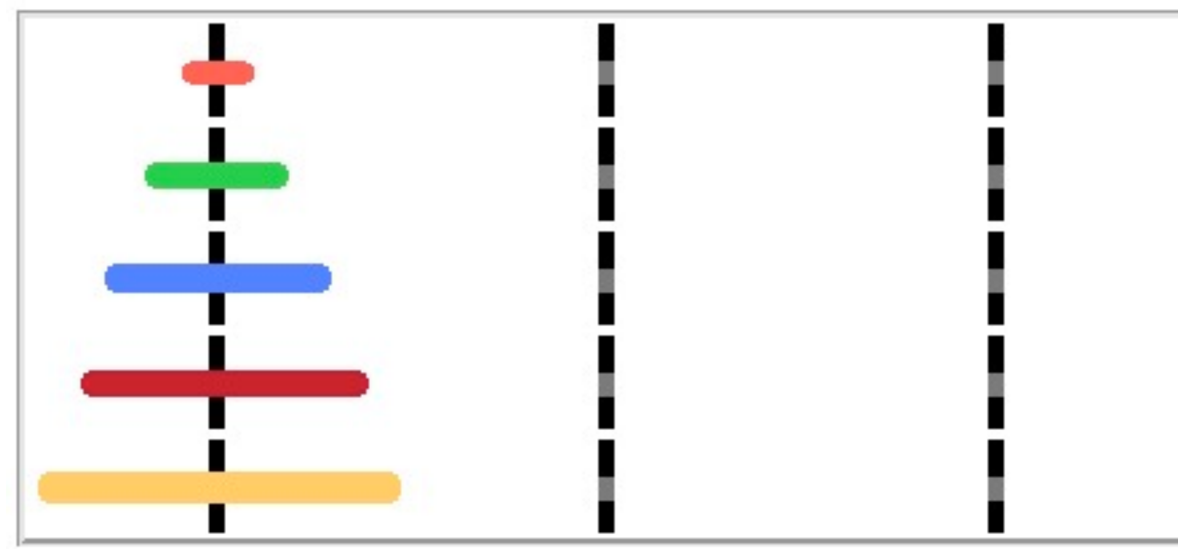
Numerical Constraints

```
constants C O P Y A S T E V L
axioms
  @axm1 C ∈ 1‥9 ∧ O ∈ 0‥9 ∧ P ∈ 1‥9 ∧
        Y ∈ 0‥9 ∧ A ∈ 0‥9 ∧ S ∈ 1‥9 ∧
        T ∈ 1‥9 ∧ E ∈ 0‥9 ∧ V∈ 0‥9 ∧ L∈0‥9
  @axm2 card({C,O,P,Y, A,S,T,E, V, L}) = 10 // all different
  @puzzleaxm
          C∗1000 + O∗100 + P∗10 + Y +
               P∗10000 + A∗1000 + S∗100 + T∗10 + E +
               S∗1000 + A∗100 + V∗10 + E
               =
               T∗10000 + O∗1000 + O∗100 + L∗10 + S
```

- For all 7 solutions:
  - Direct Kodkod Java Solution: 1.8 seconds (88 lines of Java)
  - ProB B Solution: 0.3 seconds
  - Direct CLP(FD) Encoding: 0.02 seconds

Hanoi

```
MACHINE Hanoi
SETS
 Stakes
DEFINITIONS
 GOAL == (!s.(s:Stakes & s/=dest => on(s) = <>))
CONSTANTS orig,dest,nrdiscs
PROPERTIES
 orig: Stakes & dest:Stakes &
 orig /= dest & nrdiscs = 5
VARIABLES on
INVARIANT
 on : Stakes --> seq(INTEGER)
INITIALISATION
  on := %s.(s:Stakes & s /= orig | <>) \/ {orig |-> %x.(x:1..nrdiscs|x)}
OPERATIONS
  Move(from,to,disc) = PRE from:Stakes & on(from) /= <> &
                        to:Stakes & to /= from &
                        disc:NATURAL1 & disc = first(on(from)) &
                        (on(to) /= <> =>  first(on(to))> disc )
                     THEN
    on := on <+ { from |-> tail(on(from)), to |-> (disc -> on(to)) }
  END
END
```



Hanoi in Alloy

```
open util/ordering[State] as states
open util/ordering[Stake] as stakes
open util/ordering[Disc] as discs

sig Stake { }

sig Disc { }

// sig State: the complete state of the system --
// which disc is on which stake.  An solution is a
// sequence of states.
sig State {
  on: Disc -> one Stake  // _each_ disc is on _exactly one_ stake
  // note that we simply record the set of discs on each stake --
  // the implicit assumption is that on each stake the discs
  // on that stake are ordered by size with smallest disc on top
  // and largest on bottom, as the problem requires.
}
```

```
fun State.discsOnStake[stake: Stake]: set Disc {
  // compute the set of discs on the given stake in this state.
  // ~(this.on) map the stake to the set of discs on that stake.
  stake.~(this.on)
}

fun State.topDisc[stake: Stake]: lone Disc {
  // compute the top disc on the given stake, or the empty set
  // if the stake is empty
  { d: this.discsOnStake[stake] | this.discsOnStake[stake] in discs/nexts[d] + d }
}
```

```
pred State.Move[ fromStake, toStake: Stake, s': State] {
  // Describes the operation of moving the top disc from stake fromStake
  // to stake toStake.  This function is defined implicitly but is
  // nevertheless deterministic, i.e. the result state is completely
  // determined by the initial state and fromStake and toStake; hence
  // the "det" modifier above.  (It's important to use the "det" modifier
  // to tell the Alloy Analyzer that the function is in fact deterministic.)

  let d = this.topDisc[fromStake] | {
    // all discs on toStake must be larger than d,
    // so that we can put d on top of them
    this.discsOnStake[toStake] in discs/nexts[d]
    // after, the fromStake has the discs it had before, minus d
    s'.discsOnStake[fromStake] = this.discsOnStake[fromStake] - d
    // after, the toStake has the discs it had before, plus d
    s'.discsOnStake[toStake] = this.discsOnStake[toStake] + d
    // the remaining stake afterwards has exactly the discs it had before
    let otherStake = Stake - fromStake - toStake |
      s'.discsOnStake[otherStake] = this.discsOnStake[otherStake]
  }
}
```

```
pred Game1 {
  // there is a leftStake that has all the discs at the beginning,
  // and a rightStake that has all the discs at the end
  Disc in states/first.discsOnStake[stakes/first]
  some finalState: State | Disc in finalState.discsOnStake[stakes/last]

  // each adjacent pair of states are related by a valid move of one disc
  all preState: State - states/last |
    let postState = states/next[preState] |
      some fromStake: Stake | {
        // must have at least one disk on fromStake to be able to move
        // a disc from fromStake to toStake
        some preState.discsOnStake[fromStake]
        // post- results from pre- by making one disc move
        some toStake: Stake | preState.Move[fromStake, toStake, postState]
      }
}
```