nuXmv & SAT/SMT Solver

Konkuk University
Department of Computer Science & Engineering
Kunha Kim

nuXmv Model Checker

- nuXmv is a new symbolic model checker for the analysis of synchronous finite-state and infinite-state systems.
- For the finite-state case, nuXmv features a strong verification engine based on state-of-the-art SAT-based algorithms.
- For the infinite-state case, nuXmv features SMT-based verification techniques, implemented through a tight integration with MathSAT5.
- nuXmv is targeting only finite- and infinite-state synchronous fair transition systems.
- nuXmv is used as a back-end in several other tools for requirements analysis, contract based design, model checking of hybrid systems, safety assessment, and software model checking.

Source of Infinity

- As you can see, the biggest difference between NuSMV and nuXmv is the infinite-case.
- There are many cases such as $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$, timed / hybrid systems.
- nuXmv supports the description of infinite transition systems through the type *integer* and *real*, also *clock*.
- Also, BMC and K-induction can be trivially extended to infinite-state system.
- They are still sound, but loose completeness.
- Because BMC looks for a loop-counterexample, and for K-induction, bad state reachable by infinite run without initial state.
- Use an SMT Solver able to handle required theories : integer and real.

Undecidability of Infinite-System

- When the domain is infinite, the verification problem is in general undecidable and this command may fail in proving the property.
- In particular, it may not terminate or it may terminate with an unknown result when it cannot refine the abstraction (this may be due to the presence of mixed integer/real predicates).
- Invariant, LTL, CTL checking are undecidable on infinite-transition system.
- This can be proved by a reduction from 2-counter machine’s halting problem.
- nuXmv provides many techniques such as IC3, liveness-to-safety, abstract-interpretation, etc.

Example : Equality using 2-counter machine

- Check if the 2 counters contain the same value.
- Write a program for a 2-counter machine that decides whether the counters contain the same value.

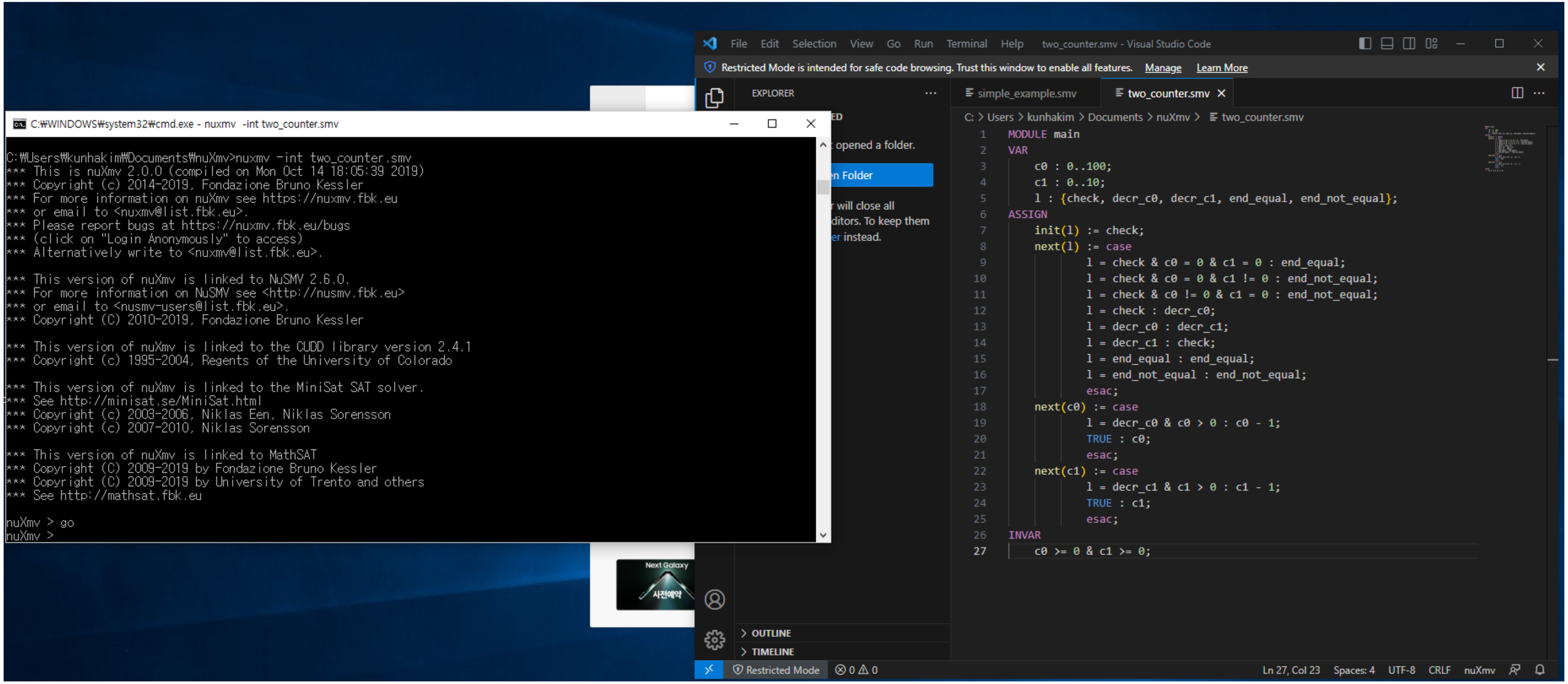
Simple pseudocode

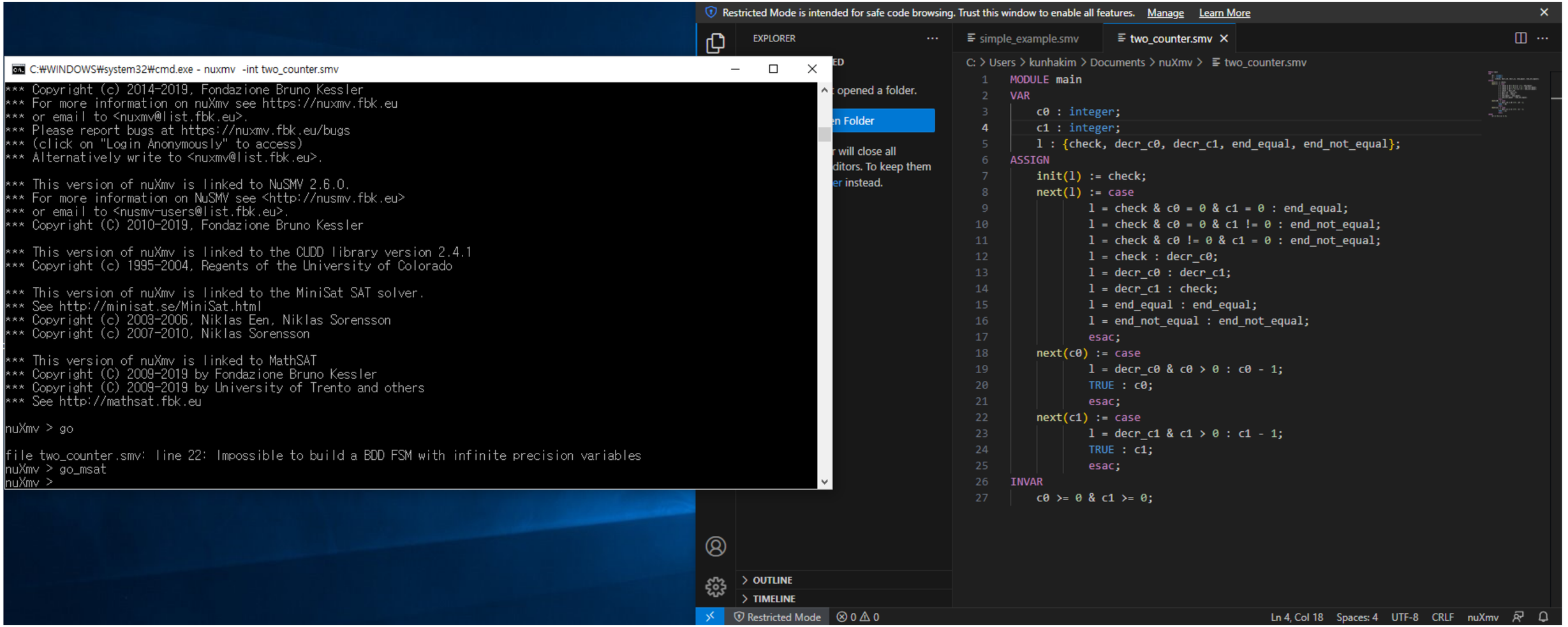
```
while(true):
  if c0 == 0:
    return c1 == 0
  if c1 == 0:
    return c0 == 0
  c0--;
  c1--;
```

```
MODULE main
VAR
    c0 : integer;
    c1 : integer;
    l : {check, decr_c0, decr_c1, end_equal, end_not_equal};
ASSIGN
    init(l) := check;
    next(l) := case
            l = check & c0 = 0 & c1 = 0 : end_equal;
            l = check & c0 = 0 & c1 != 0 : end_not_equal;
            l = check & c0 != 0 & c1 = 0 : end_not_equal;
            l = check : decr_c0;
            l = decr_c0 : decr_c1;
            l = decr_c1 : check;
            l = end_equal : end_equal;
            l = end_not_equal : end_not_equal;
            esac;
    next(c0) := case
            l = decr_c0 & c0 > 0 : c0 - 1;
            TRUE : c0;
            esac;
    next(c1) := case
            l = decr_c1 & c1 > 0 : c1 - 1;
            TRUE : c1;
            esac;
INVAR
    c0 >= 0 & c1 >= 0;
```

Some commands for nuXmv

- Most of the commands are same as NuSMV.
- But for infinite-system, we have to use tag *msat*.
- To build a model, we use *go_msat*, *msat_pick_state* to pick state and *msat_simulate* for simulation.
- Also, to check some specifications, use command like *msat_check_ltlspec_bmc* or other commands corresponding to other techniques.
- Some options may be unavailable, so you have to check with -h command.

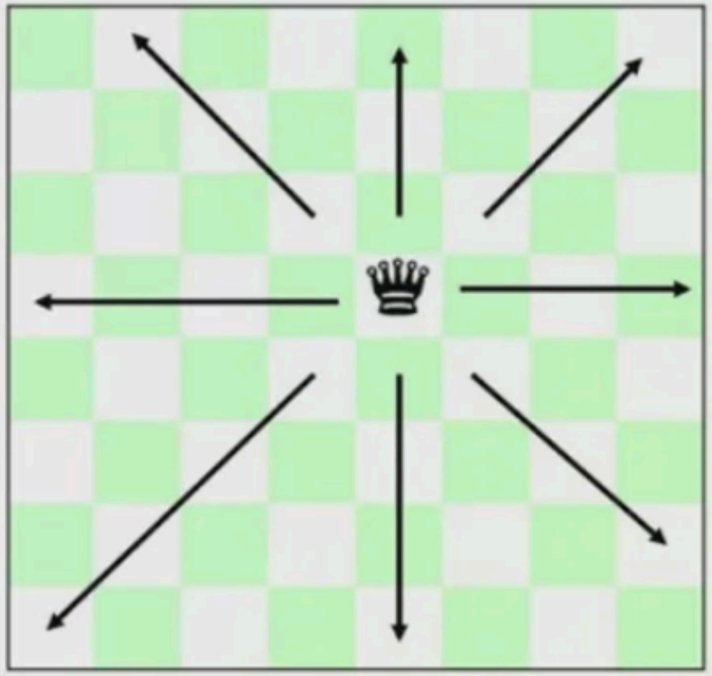

C:₩WINDOWS₩system32₩cmd.exe - nuxmv -int two_counter.smv
C:₩Users₩kunhakim₩Documents₩nuXmv>nuxmv -int two_counter.smv
*** This is nuXmv 2.0.0 (compiled on Mon Oct 14 18:05:39 2019)
*** Copyright (c) 2014-2019, Fondazione Bruno Kessler
*** For more information on nuXmv see https://nuxmv.fbk.eu
*** or email to <nuxmv@list.fbk.eu>.
*** Please report bugs at https://nuxmv.fbk.eu/bugs
*** (click on "Login Anonymously" to access)
*** Alternatively write to <nuxmv@list.fbk.eu>.
*** This version of nuXmv is linked to NuSMV 2.6.0.
*** For more information on NuSMV see <http://nusmv.fbk.eu>
*** or email to <nusmv-users@list.fbk.eu>.
*** Copyright (C) 2010-2019, Fondazione Bruno Kessler
*** This version of nuXmv is linked to the CUDD library version 2.4.1
*** Copyright (c) 1995-2004, Regents of the University of Colorado
*** This version of nuXmv is linked to the MiniSat SAT solver.
*** See http://minisat.se/MiniSat.html
*** Copyright (c) 2003-2006, Niklas Een, Niklas Sorensson
*** Copyright (c) 2007-2010, Niklas Sorensson
*** This version of nuXmv is linked to MathSAT
*** Copyright (C) 2009-2019 by Fondazione Bruno Kessler
*** Copyright (C) 2009-2019 by University of Trento and others
*** See http://mathsat.fbk.eu
nuXmv > go
nuXmv >
File Edit Selection View Go Run Terminal Help two_counter.smv - Visual Studio Code
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More
EXPLORER
opened a folder.
Folder
will close all
ditors. To keep them
er instead.
simple_example.smv
two_counter.smv
C: > Users > kunhakim > Documents > nuXmv > two_counter.smv
1 MODULE main
2 VAR
3 c0 : 0..100;
4 c1 : 0..10;
5 l : {check, decr_c0, decr_c1, end_equal, end_not_equal};
6 ASSIGN
7 init(l) := check;
8 next(l) := case
9 l = check & c0 = 0 & c1 = 0 : end_equal;
10 l = check & c0 = 0 & c1 != 0 : end_not_equal;
11 l = check & c0 != 0 & c1 = 0 : end_not_equal;
12 l = check : decr_c0;
13 l = decr_c0 : decr_c1;
14 l = decr_c1 : check;
15 l = end_equal : end_equal;
16 l = end_not_equal : end_not_equal;
17 esac;
18 next(c0) := case
19 l = decr_c0 & c0 > 0 : c0 - 1;
20 TRUE : c0;
21 esac;
22 next(c1) := case
23 l = decr_c1 & c1 > 0 : c1 - 1;
24 TRUE : c1;
25 esac;
26 INVAR
27 c0 >= 0 & c1 >= 0;
OUTLINE
TIMELINE
Restricted Mode
Ln 27, Col 23 Spaces: 4 UTF-8 CRLF nuXmv


C:₩WINDOWS₩system32₩cmd.exe - nuxmv -int two_counter.smv
*** Copyright (c) 2014-2019, Fondazione Bruno Kessler
*** For more information on nuXmv see https://nuxmv.fbk.eu
*** or email to <nuxmv@list.fbk.eu>.
*** Please report bugs at https://nuxmv.fbk.eu/bugs
*** (click on "Login Anonymously" to access)
*** Alternatively write to <nuxmv@list.fbk.eu>.
*** This version of nuXmv is linked to NuSMV 2.6.0.
*** For more information on NuSMV see <http://nusmv.fbk.eu>
*** or email to <nusmv-users@list.fbk.eu>.
*** Copyright (C) 2010-2019, Fondazione Bruno Kessler
*** This version of nuXmv is linked to the CUDD library version 2.4.1
*** Copyright (c) 1995-2004, Regents of the University of Colorado
*** This version of nuXmv is linked to the MiniSat SAT solver.
*** See http://minisat.se/MiniSat.html
*** Copyright (c) 2003-2006, Niklas Een, Niklas Sorensson
*** Copyright (c) 2007-2010, Niklas Sorensson
*** This version of nuXmv is linked to MathSAT
*** Copyright (C) 2009-2019 by Fondazione Bruno Kessler
*** Copyright (C) 2009-2019 by University of Trento and others
*** See http://mathsat.fbk.eu
nuXmv > go
file two_counter.smv: line 22: Impossible to build a BDD FSM with infinite precision variables
nuXmv > go_msat
nuXmv >
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More
EXPLORER
opened a folder.
Folder
will close all
ditors. To keep them
instead.
OUTLINE
TIMELINE
simple_example.smv
two_counter.smv
C: > Users > kunhakim > Documents > nuXmv > two_counter.smv
1 MODULE main
2 VAR
3 c0 : integer;
4 c1 : integer;
5 l : {check, decr_c0, decr_c1, end_equal, end_not_equal};
6 ASSIGN
7 init(l) := check;
8 next(l) := case
9 l = check & c0 = 0 & c1 = 0 : end_equal;
10 l = check & c0 = 0 & c1 != 0 : end_not_equal;
11 l = check & c0 != 0 & c1 = 0 : end_not_equal;
12 l = check : decr_c0;
13 l = decr_c0 : decr_c1;
14 l = decr_c1 : check;
15 l = end_equal : end_equal;
16 l = end_not_equal : end_not_equal;
17 esac;
18 next(c0) := case
19 l = decr_c0 & c0 > 0 : c0 - 1;
20 TRUE : c0;
21 esac;
22 next(c1) := case
23 l = decr_c1 & c1 > 0 : c1 - 1;
24 TRUE : c1;
25 esac;
26 INVAR
27 c0 >= 0 & c1 >= 0;
Restricted Mode
0 0
Ln 4, Col 18
Spaces: 4
UTF-8
CRLF
nuXmv

Some Common Problems & DPLL / CDCL / Simplex

The 8-queen Problem

Eight Queens Problem



Can we put 8 queens on the chess board in such a way that no two may hit each other?

Specifying Problem's Constraints

- When we use SAT / SMT Solver, we don't need to think about how to solve a problem. Think about how to specify the problem.
- For this case, it can be done only with boolean variable.
- We use p_{ij} to express every row and column of the chess board to check queen's existence.

p_{11}	p_{12}	p_{13}	p_{14}	p_{15}	p_{16}	p_{17}	p_{18}
p_{21}	p_{22}	p_{23}	p_{24}	p_{25}	p_{26}	p_{27}	p_{28}
p_{31}	p_{32}	p_{33}	p_{34}	p_{35}	p_{36}	p_{37}	p_{38}
p_{41}	p_{42}	p_{43}	p_{44}	p_{45}	p_{46}	p_{47}	p_{48}
p_{51}	p_{52}	p_{53}	p_{54}	p_{55}	p_{56}	p_{57}	p_{58}
p_{61}	p_{62}	p_{63}	p_{64}	p_{65}	p_{66}	p_{67}	p_{68}
p_{71}	p_{72}	p_{73}	p_{74}	p_{75}	p_{76}	p_{77}	p_{78}
p_{81}	p_{82}	p_{83}	p_{84}	p_{85}	p_{86}	p_{87}	p_{88}

Formal Specification of 8-queen Problem

- To satisfy 8-queen problem's constraints, there must be 1 queen on the row.
- This can be expressed by these notation : for row i, $\wedge_{i=1}^{8} \vee_{j=1}^{8} p_{ij}$, $\wedge_{i=1}^{8} \wedge_{1 \leq j < k \leq 8} (\neg p_{ij} \vee \neg p_{ik})$.
- For column, the requirements are exactly same, just change i and j.
- What about diagonal requirements? —> The sum or difference of i and j are same.
- In other words, $i + j = i' + j' \leftrightarrow p_{ij}$ and $p_{i'j'}$ or $i - j = i' - j' \leftrightarrow p_{ij}$ and $p_{i'j'}$ ($(i, j) \neq (i', j')$)are in the same diagonal direction.
- Thus, for all i, j, i', j' with $(i, j) \neq (i', j')$ satisfying $i + j = i' + j'$ or $i - j = i' - j'$, we have to satisfy $\neg p_{ij} \vee \neg p_{i'j'}$.
- The formal notation is denoted as $\wedge_{1 \leq i < i' \leq 8} (\wedge_{j, j' : i+j=i'+j' \vee i-j=i'-j'} \neg p_{ij} \vee \neg p_{i'j'})$

$$\bigwedge_{i=1}^{8} \bigvee_{j=1}^{8} p_{ij} \wedge$$

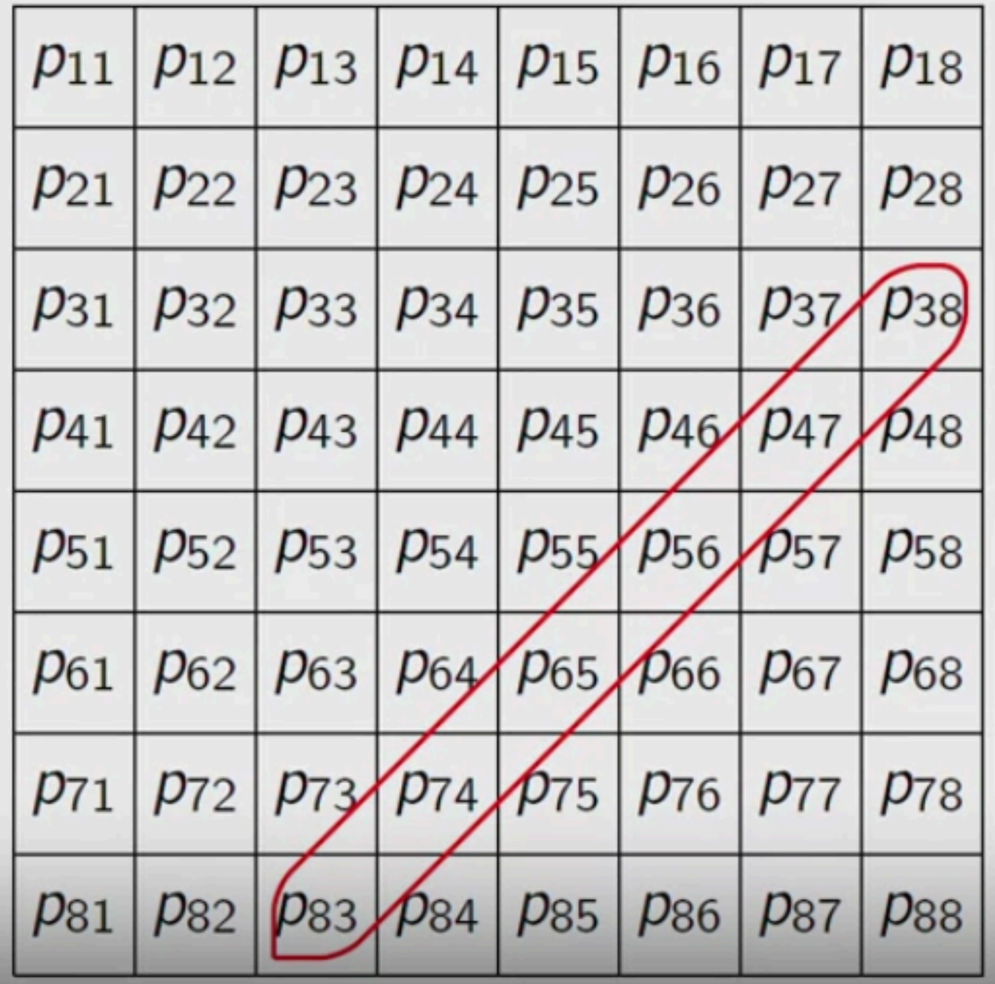
$$\bigwedge_{i=1}^{8} \bigwedge_{0<j<k\leq 8} (\neg p_{ij} \vee \neg p_{ik}) \wedge$$

$$\bigwedge_{j=1}^{8} \bigvee_{i=1}^{8} p_{ij} \wedge$$

$$\bigwedge_{j=1}^{8} \bigwedge_{0<i<k\leq 8} (\neg p_{ij} \vee \neg p_{kj}) \wedge$$

$$\bigwedge_{0<i<i'\leq 8} \Big(\bigwedge_{j,j' : i+j=i'+j' \vee i-j=i'-j'} \neg p_{ij} \vee \neg p_{i'j'} \Big)$$

At most one queen on every diagonal



p_{ij} and $p_{i'j'}$ on such a diagonal

$$\Longleftrightarrow$$

$$i + j = i' + j'$$

Diagonal in other direction:

p_{11}	p_{12}	p_{13}	p_{14}	p_{15}	p_{16}	p_{17}	p_{18}
p_{21}	p_{22}	p_{23}	p_{24}	p_{25}	p_{26}	p_{27}	p_{28}
p_{31}	p_{32}	p_{33}	p_{34}	p_{35}	p_{36}	p_{37}	p_{38}
p_{41}	p_{42}	p_{43}	p_{44}	p_{45}	p_{46}	p_{47}	p_{48}
p_{51}	p_{52}	p_{53}	p_{54}	p_{55}	p_{56}	p_{57}	p_{58}
p_{61}	p_{62}	p_{63}	p_{64}	p_{65}	p_{66}	p_{67}	p_{68}
p_{71}	p_{72}	p_{73}	p_{74}	p_{75}	p_{76}	p_{77}	p_{78}
p_{81}	p_{82}	p_{83}	p_{84}	p_{85}	p_{86}	p_{87}	p_{88}

p_{ij} and $p_{i'j'}$ on such a diagonal

$$\Longleftrightarrow$$

$$i - j = i' - j'$$

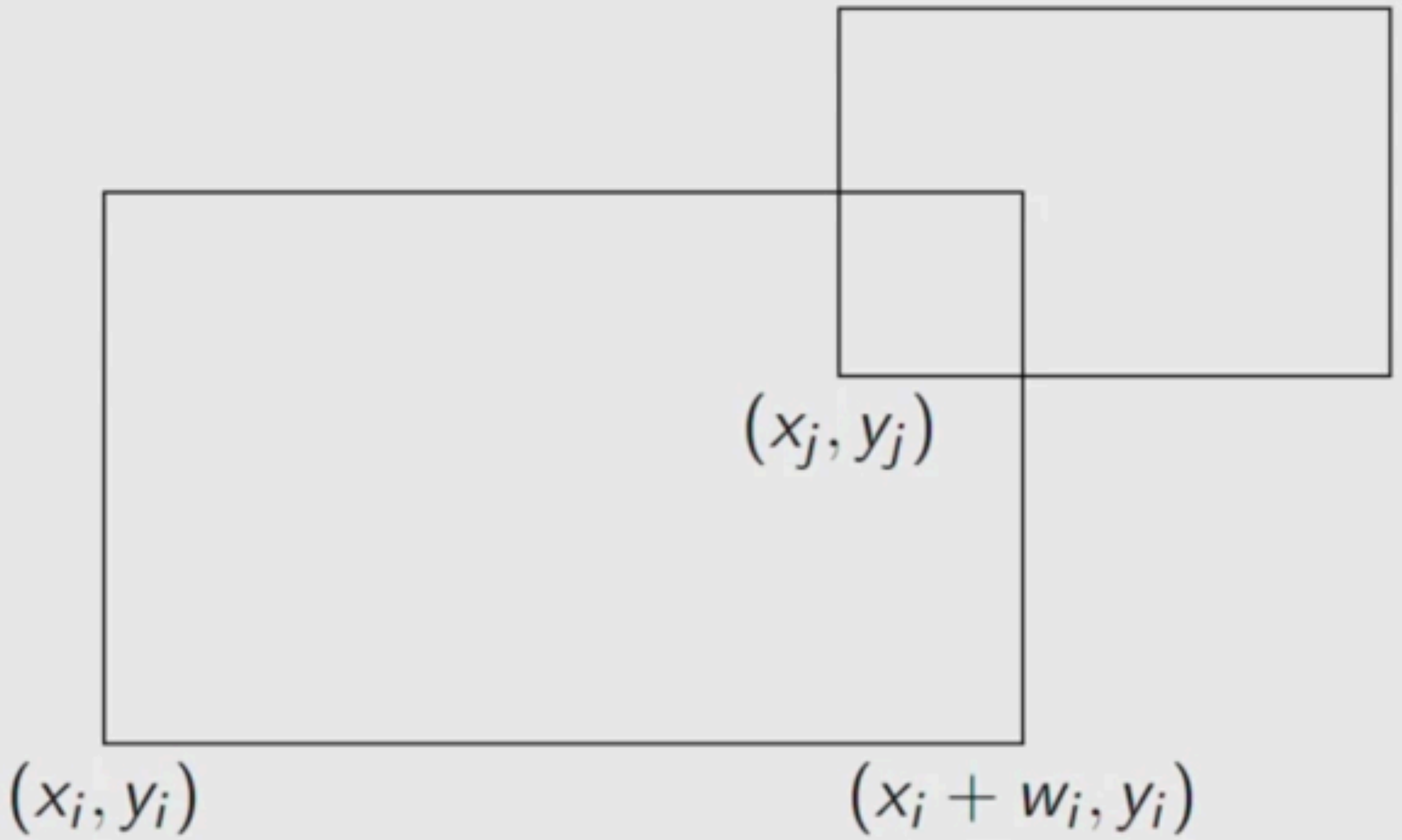
The Rectangle Fitting Problem

- Main question : Given a big rectangle and a number of smaller rectangles, can you fit the smaller rectangles in such a way that they all fit in a big rectangle without overlap?
- How can we specify the problem?
- For $i = 1, \cdots, n$, w_i is the width of rectangle i, h_i is the height of rectangle i, x_i is the x-coordinate of left lower corner of rectangle i, y_i is the y-coordinate of the left lower corner of rectangle i.
- We can easily check width and height requirements : for example, if rectangle 1's width and height are 3 and 5, $(w_1 = 3 \wedge h_1 = 5) \vee (w_1 = 5 \wedge h_1 = 3)$.
- This is similar for all $i = 1, \cdots, n$.

Fit with no overlapping

- Let W, H be the width and height of big rectangle, and (0,0) be the left lower corner of the big rectangle.
- Every rectangle should be in big rectangle. So, $x_i \geq 0 \wedge x_i + w_i \leq W \wedge y_i \geq 0 \wedge y_i + h_i \leq H$ for all $i = 1, \cdots, n$.
- Next, how can we check overlapping?
- This can be easily verified by this : Rectangles i and j overlap if $x_i + w_i > x_j, x_i < x_j + w_j, y_i + h_i > y_j, y_i < y_j + h_j$. Let's see an image.

No overlap?



If overlap, then $x_i + w_i > x_j$

So for all $i, j = 1, \ldots, n$, $i < j$, we should add the negation of this overlappingness:

$$\neg(x_i + w_i > x_j \ \wedge \ x_i < x_j + w_j \ \wedge \ y_i + h_i > y_j \ \wedge \ y_i < y_j + h_j)$$

or, equivalently

$$x_i + w_i \leq x_j \ \vee \ x_j + w_j \leq x_i \ \vee \ y_i + h_i \leq y_j \ \vee \ y_j + h_j \leq y_i$$

$$\bigwedge_{i=1}^{n}((w_i = W_i \wedge h_i = H_i) \vee (w_i = H_i \wedge h_i = W_i))$$

$$\wedge \bigwedge_{i=1}^{n}(x_i \geq 0 \ \wedge \ x_i + w_i \leq W \ \wedge \ y_i \geq 0 \ \wedge \ y_i + h_i \leq H)$$

$$\wedge \bigwedge_{1 \leq i < j \leq n} (x_i + w_i \leq x_j \ \vee \ x_j + w_j \leq x_i \ \vee \ y_i + h_i \leq y_j \ \vee \ y_j + h_j \leq y_i)$$

The Resolution Rule

- Resolution of propositional formulas can be applicable only if the formula has the form of CNF, conjunctive normal form.
- Clauses are properties that we know to be true.
- From these clauses new clauses are derived, trying to derive a contradiction : the empty clause $\bot$
- This rule states that if there are clauses of the shape $p \vee V$ and $\neg p \vee W$, then the new clause $V \vee W$ may be added.
- Formal Notation of resolution rule :

$$\frac{p \vee V, \quad \neg p \vee W}{V \vee W}$$

We prove that the CNF consisting of the following clauses 1 to 5 is unsatisfiable

1	$p \vee q$	
2	$\neg r \vee s$	
3	$\neg q \vee r$	
4	$\neg r \vee \neg s$	
5	$\neg p \vee r$	

6	$p \vee r$	$(1, 3, q)$
7	r	$(5, 6, p)$
8	s	$(2, 7, r)$
9	$\neg r$	$(4, 8, s)$
10	$\square$	$(7, 9, r)$

The DPLL Algorithm

- The idea of DPLL : First apply unit resolution as long as possible.
- If you cannot proceed by unit resolution or trivial observations then choose a variable p, introduce the cases p and $\neg p$, and do it recursively.
- Apply unit resolution as long as possible means : for all literal l (a unit clause), remove $\neg l$ from all clauses containing $\neg l$, and remove all clauses containing l.
- In pseudocode, we call this unit-resol.

The algorithm:

DPLL(X):
$X :=$ unit-resol(X)

if $X = \emptyset$ then return(satisfiable)

if $\bot \notin X$ then

choose variable p in X
DPLL($X \cup \{p\}$)
DPLL($X \cup \{\neg p\}$)

Example

Consider the CNF consisting of the following nine clauses

$$\begin{array}{lll} \neg p \vee \neg s & p \vee r & \neg s \vee t \\ \neg p \vee \neg r & p \vee s & q \vee s \\ \neg q \vee \neg t & r \vee t & q \vee \neg r \end{array}$$

No unit resolution possible: choose variable p

Add p, unit resolution:
$\neg s$, $\neg r$
q (use $\neg s$), t (use $\neg r$)
$\neg t$ (use q)
$\bot$

Add $\neg p$, unit resolution:
r, s
q (use r), t (use s)
$\neg t$ (use q)
$\bot$

Both branches yield $\bot$, so original CNF is unsatisfiable

Example:

Consider the CNF consisting of the following eight clauses

$$\begin{array}{lll} \neg p \vee \neg s & p \vee r & \neg s \vee t \\ \neg p \vee \neg r & p \vee s & q \vee s \\ \neg q \vee \neg t & r \vee t & \end{array}$$

No unit resolution possible: choose variable p

Add p
Unit resolution:
$\neg s$, $\neg r$
q (use $\neg s$), t (use $\neg r$)
$\neg t$ (use q)
$\bot$

Add $\neg p$
Unit resolution:
r, s
t (use s)
$\neg q$ (use t)

Yields satisfying assignment $p = q = \mathit{false}$, $r = s = t = \mathit{true}$

Basic Idea of CDCL

- A direct implementation of DPLL would make copies of the full CNF X at every recursive call.
- By unit-resol literals are removed from clauses, and clauses are removed.
- For efficiency, we need ways to mimic unit-resol and case analysis without changing the CNF.
- Keep track of a list M of literals that has been chosen and derived during the execution of DPLL, starts to be empty.
- The list M will be extended if a case analysis starts : $Decide$ or a literal is derived by unit resolution : $UnitPropagate$
- Part of it will be removed if case analysis is continued after finding a contradiction : $Backtrack$

Rules for CDCL

- For a literal l we write $M \vDash l$ if l occurs in M, $M \vDash \neg C$ if $\neg l$ occurs in M for every literal l in C, l is $undefined$ in M if neither l nor $\neg l$ occurs in M.
- $UnitPropagate : M \Longrightarrow Ml$ if l is undefined in M and the CNF contains a clause $C \vee l$ satisfying $M \vDash \neg C$
- $Decide : M \Longrightarrow Ml^d$ if l is undefined in M.
- $Backtrack : Ml^dN \Longrightarrow M\neg l$ if $Ml^dN \vDash \neg C$ for a clause C in the CNF and N contains no decision literals.
- $Fail : M \Longrightarrow fail$ if $M \vDash \neg C$ for a clause C in the CNF and M contains no decision literals

Example:

$\neg p \vee \neg s$	$p \vee r$	$\neg s \vee t$
$\neg p \vee \neg r$	$p \vee s$	$q \vee s$
$\neg q \vee \neg t$	$r \vee t$	$q \vee \neg r$

$p^d \ \neg s \ \neg r \ t \ q$

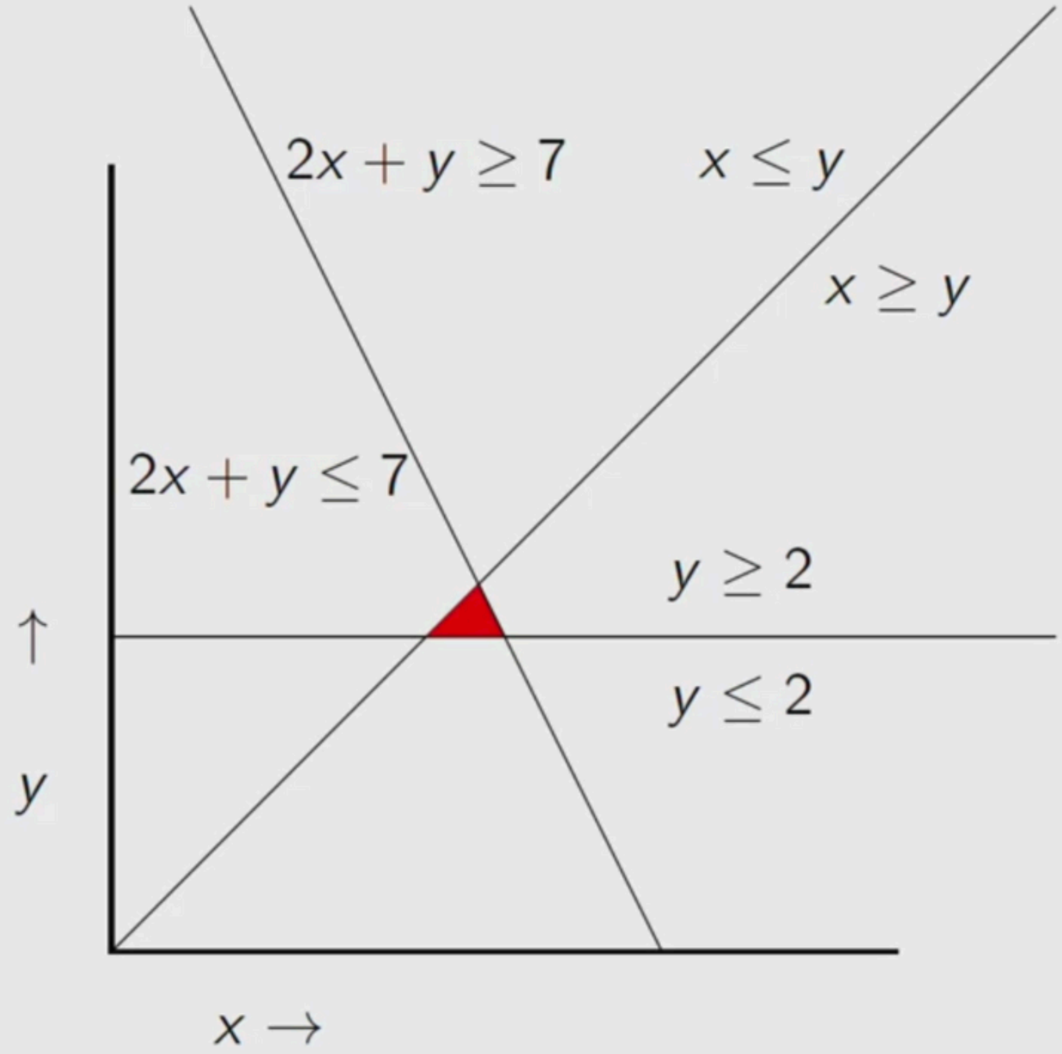
Backtrack:
$\neg p \ r \ s \ q \ t$

Fail

CDCL Optimization : Learning Clause

- When we do $Backtrack$, we initialize all assignments with reversing decision variable.
- This is inefficient, so we do $Backjump$ instead of $Backtrack$.
- $Backjump$: $Ml^dN \Longrightarrow Ml'$ if $Ml^dN \vDash \neg C$ for a clause C in the CNF and there is a clause $C' \vee l'$ derivable from the CNF such that $M \vDash \neg C'$ and l' is undefined in M.
- The new clause $C' \vee l'$ will be added to the CNF.
- You can check an example here : https://en.wikipedia.org/wiki/Conflict-driven_clause_learning or https://ece.uwaterloo.ca/~vganesh/TEACHING/W2013/SATSMT/grasp-1999.pdf

Deal with Inequalities



So indeed the **red** part describes the values x, y satisfying

$$x \geq y$$

$$y \geq 2$$

$$2x + y \leq 7$$

The Simplex method

- But if we have 100, 1000, … variables to check, how can we solve this problem?
- For SMT the underlying approach is the simplex method for linear optimization.
- Among all real values $x_1, \cdots, x_n \geq 0$ find the maximal value of a linear goal function $v + c_1 x_1 + \cdots + c_n x_n$ satisfying k linear constraints $a_{i1} x_1 + a_{i2} x_2 + \cdots + a_{in} x_n \leq b_i$ for $i = 1,2,\cdots, k$.
- Here v, a_{ij}, c_i and b_i are given real values, satisfying $b_i \geq 0$ for $i = 1,2,\cdots, k$
- Trivially if we choose 0 for all x_i, it can be a solution.
- Starting from here, we want to maximize the goal function.

Some terminologies and consideration

- An equality $A = B$ is replaced by $A \geq B$ and $A \leq B$. And if a variable x runs over all reals, replace it by $x_1 - x_2$ for fresh variables x_1, x_2 satisfying $x_1, x_2 \geq 0$.
- For every linear inequality $a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n \leq b_i$, a fresh variable $y_i \geq 0$ is introduced, $y_i = b_i - a_{i1}x_1 - a_{i2}x_2 - \cdots - a_{in}x_n$. This format is called the slack form.
- The solution $y_i = b_i$ for $i = 1,2,\cdots,k$ and $x_j = 0$ for $j = 1,2,\cdots,n$ is called the basic solution
- The variables y_i for $i = 1,2,\cdots,k$ are called $basic$
- The variables x_j for $j = 1,2,\cdots,n$ are called $non - basic$.
- The approach of Simplex method is swapping $basic$ and $non - basic$ with responsibility. Let's see with example.

Example

Maximize $z = 3 + x_1 + x_3$ satisfying the constraints

$$\begin{aligned} -x_1 + x_2 - x_3 &\leq 2 \\ x_1 + x_3 &\leq 3 \\ 2x_1 - x_2 &\leq 4 \end{aligned}$$

Slack form:

$$\begin{aligned} y_1 &= 2 + x_1 - x_2 + x_3 \\ y_2 &= 3 - x_1 - x_3 \\ y_3 &= 4 - 2x_1 + x_2 \end{aligned}$$

Start by **basic solution**:

$$x_1 = x_2 = x_3 = 0$$

$$y_1 = 2,\ y_2 = 3,\ y_3 = 4$$

Goal: maximize $z = 3 + x_1 + x_3$

Observation: if we increase x_1, and x_2 and x_3 remain 0, then the goal function z will increase

We want to increase x_1 as much as possible, keeping $x_2 = x_3 = 0$, while in the equations

$$\begin{array}{lcrrrr} y_1 & = & 2 & +x_1 & -x_2 & +x_3 \\ y_2 & = & 3 & -x_1 & & -x_3 \\ y_3 & = & 4 & -2x_1 & +x_2 & \end{array}$$

all variables should be ≥ 0

$y_1 = 2 + x_1 \geq 0$: OK if x_1 increases

$y_2 = 3 - x_1 \geq 0$: only OK if $x_1 \leq 3$

$y_3 = 4 - 2x_1 \geq 0$: only OK if $x_1 \leq 2$

So $y_i \geq 0$ only holds for all i if $x_1 \leq 2$

The highest allowed value for x_1 is 2, and then y_3 will get the value 0

Highest allowed value for x_1 is 2, then y_3 will be 0

pivot: swap x_1 and y_3

Recall that

y_3 is **basic**: left from '$=$', possibly ≥ 0 in basic solution

x_1 is **non-basic**: right from '$=$', $= 0$ in basic solution

By the pivot

- the basic variable y_3 will become non-basic, and
- the non-basic variable x_1 will become basic

Swap x_1 and y_3 in

$$\begin{array}{rcrrrr} y_1 & = & 2 & +x_1 & -x_2 & +x_3 \\ y_2 & = & 3 & -x_1 & & -x_3 \\ y_3 & = & 4 & -2x_1 & +x_2 & \end{array}$$

In the equation $y_3 = 4 - 2x_1 + x_2$ move x_1 to the left and y_3 to the right, yielding the equivalent equation

$$x_1 = 2 + \frac{1}{2}x_2 - \frac{1}{2}y_3$$

Next replace every x_1 by $2 + \frac{1}{2}x_2 - \frac{1}{2}y_3$ in the equations for z, y_1, y_2, yielding the following slack form:

maximize $z = 5 + \frac{1}{2}x_2 + x_3 - \frac{1}{2}y_3$ satisfying

$$\begin{array}{rcrrrr} x_1 & = & 2 & +\frac{1}{2}x_2 & & -\frac{1}{2}y_3 \\ y_1 & = & 4 & -\frac{1}{2}x_2 & +x_3 & -\frac{1}{2}y_3 \\ y_2 & = & 1 & -\frac{1}{2}x_2 & -x_3 & +\frac{1}{2}y_3 \end{array}$$

This is the end of the first pivot

Observe

- all equations are replaced by equivalent equations, so the new optimization problem is equivalent to the original one
- Now the basic variables are x_1, y_1, y_2 and the non-basic variables are x_2, x_3, y_3
- By construction again we have a slack form with a basic solution in which all non-basic variables are 0, and all basic variables are ≥ 0
- In this new basic solution the goal function

$$z = 5 + \frac{1}{2}x_2 + x_3 - \frac{1}{2}y_3$$

has value 5, improving the original value 3

In this goal function

$$z = 5 + \frac{1}{2}x_2 + x_3 - \frac{1}{2}y_3$$

the non-basic variable x_2 has a positive factor $\frac{1}{2}$: increasing x_2 is the goal of the next pivot, while the other non-basic variables remain 0

$$\begin{array}{rcrrrr} x_1 & = & 2 & +\frac{1}{2}x_2 & & -\frac{1}{2}y_3 \\ y_1 & = & 4 & -\frac{1}{2}x_2 & +x_3 & -\frac{1}{2}y_3 \\ y_2 & = & 1 & -\frac{1}{2}x_2 & -x_3 & +\frac{1}{2}y_3 \end{array}$$

$x_1 = 2 + \frac{1}{2}x_2 \geq 0$, OK if x_2 increases
$y_1 = 4 - \frac{1}{2}x_2 \geq 0$, only OK if $x_2 \leq 8$
$y_2 = 1 - \frac{1}{2}x_2 \geq 0$, only OK if $x_2 \leq 2$

So the maximal allowed value for x_2 is 2, in which case y_2 will get the value 0
$\Longrightarrow$ pivot swapping x_2 and y_2

TU/e

eit Digital

$$y_2 = 1 - \frac{1}{2}x_2 - x_3 + \frac{1}{2}y_3$$

yields $x_2 = 2 - 2x_3 - 2y_2 + y_3$, so in the goal function and in the other equations we replace x_2 by $2 - 2x_3 - 2y_2 + y_3$, yielding

Maximize $z = 6 - y_2$

satisfying

$$\begin{array}{lcrrrr} x_1 & = & 3 & -x_3 & -y_2 & \\ x_2 & = & 2 & -2x_3 & -2y_2 & +y_3 \\ y_1 & = & 3 & +2x_3 & +y_2 & -y_3 \end{array}$$

Observation: since in

$$z = 6 - y_2$$

y_2 should be ≥ 0, the value of z will always be ≤ 6

On the other hand, in the basic solution with $x_3 = y_2 = y_3 = 0$ and $x_1 = 3$, $x_2 = 2$ and $y_1 = 3$ we obtain $z = 6$, so this basic solution yields the **maximal value** for z

Simplex method: solve a linear optimization problem in slack form

maximize $z = v + \sum_{j=1}^{n} c_j x_j$ under a set of constraints of the shape

$$y_i = b_i + \sum_{j=1}^{n} a_{ij} x_j$$

with $b_i \geq 0$, for $i = 1, \ldots, k$

As long there exists j such that $c_j > 0$ do a **pivot**, that is

- find the highest value for x_j for which $b_i + a_{ij} x_j \geq 0$ for all i, and $b_i + a_{ij} x_j = 0$ for one particular i
- swap x_j and y_i and bring the result in slack form

At the end $c_j \leq 0$ for all j, from which can be concluded that the basic solution of this last slack form yields the maximal value for z

TU/e

eit Digital

Existence of Solution

- We want to know not only optimal solution but also existence of solution. This is called feasible if there is a solution.
- For a set of linear inequalities $a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n \leq b_i$ for $i = 1,2,\cdots, k$ and $x_j \geq 0$ for $j = 1,\cdots, n$ (not all $b_i \geq 0$), introduce a fresh variable $z \geq 0$ and extend the set of inequalities $a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n - z \leq b_i$ for $i = 1,\cdots, k$.
- This extended problem is always feasible, even if $b_i \leq 0$: choose z very large.
- Original problem is feasible if and only if maximal value of $-z$ is 0 in extended problem.
- $(\Longleftarrow)$ If extended problem has solution with $-z = 0$, then it is solution of the original problem, so feasible.
- $(\Longrightarrow)$ If original problem feasible, then extended one has solution with $-z = 0$.
- Since $z \geq 0$, this is the maximal value for $-z$

Difference between basic problem

- So now we want to maximize $-z$ by the Simplex method.
- Since b_i may be negative, we have no basic solution with all variables ≥ 0.
- As the first pivot swap the non-basic variable z with the basic variable y_i for i for which b_i is the most negative.
- After this pivot, the resulting equivalent problem in slack form has $b_i \geq 0$, so has a basic solution. From here, repeat pivots as before.
- If at the end the value for $-z$ is 0, then original problem is feasible, otherwise it is not.

Find values $x, y \geq 0$ satisfying

$$\begin{array}{rrcr} -x & -3y & \leq & -12 \\ x & +y & \leq & 10 \\ -x & +y & \leq & -7 \end{array}$$

Introduce z for maximizing $-z$:

$$\begin{array}{rrrcr} -x & -3y & -z & \leq & -12 \\ x & +y & -z & \leq & 10 \\ -x & +y & -z & \leq & -7 \end{array}$$

Slack form:

$$\begin{array}{rcrrrr} y_1 & = & -12 & +x & +3y & +z \\ y_2 & = & 10 & -x & -y & +z \\ y_3 & = & -7 & +x & -y & +z \end{array}$$

Slack form:

$$\begin{array}{rcrrrr} y_1 & = & -12 & +x & +3y & +z \\ y_2 & = & 10 & -x & -y & +z \\ y_3 & = & -7 & +x & -y & +z \end{array}$$

Indeed $x = y = z = 0$ does not yield a basic solution, since then $y_1 = -12$ and $y_3 = -7$ do not satisfy $y_i \geq 0$

Most negative b_i is $b_1 = -12$, so do pivot on z and y_1

replace every z by $z = 12 - x - 3y + y_1$:

Maximize $-12 + x + 3y - y_1$ satisfying

$$\begin{array}{rcrrrr} z & = & 12 & -x & -3y & +y_1 \\ y_2 & = & 22 & -2x & -4y & +y_1 \\ y_3 & = & 5 & & -4y & +y_1 \end{array}$$

Indeed now we have a basic solution
$x = y = y_1 = 0$, $z = 12$, $y_2 = 22$ and $y_3 = 5$, all ≥ 0

From now on: proceed by simplex algorithm as before

Maximize $-12 + x + 3y - y_1$, so swap x with z, y_2 or y_3

z: $12 - x \geq 0$, so $x \leq 12$
y_2: $22 - 2x \geq 0$, so $x \leq 11$
y_3: no bound on x

So next pivot: swap x with y_2, yielding

Maximize $-1 + y - \frac{1}{2}y_1 - \frac{1}{2}y_2$ satisfying

$$\begin{array}{rcrrrr} x & = & 11 & -2y & +\frac{1}{2}y_1 & -\frac{1}{2}y_2 \\ z & = & 1 & -y & +\frac{1}{2}y_1 & +\frac{1}{2}y_2 \\ y_3 & = & 5 & -4y & +y_1 & \end{array}$$

Next pivot: swap y and z

Doing so yields:
Maximize $-z$ satisfying

$$\begin{array}{rcrrrr} x & = & 9 & \cdots z & \cdots y_1 & \cdots y_2 \\ y & = & 1 & -z & +\frac{1}{2}y_1 & +\frac{1}{2}y_2 \\ y_3 & = & 1 & \cdots z & \cdots y_1 & \cdots y_2 \end{array}$$

Surprise: we are back at our original maximization function $-z$

Since it has no positive factors any more, the resulting basic solution

$$z = y_1 = y_2 = 0,\ x = 9,\ y = 1,\ y_3 = 1$$

yields the optimal value $-z = 0$

Since $-z = 0$, the original set of inequalities

$$-x - 3y \leq -12 \ \wedge\ x + y \leq 10 \ \wedge\ -x + y \leq -7$$

is satisfiable: we found the solution $x = 9$, $y = 1$

Apply Simplex to SMT

- For SAT, the CDCL approach applies on CNF and atom is just a boolean variable.
- For SMT, an atom is an expression in the corresponding theory, in our case a linear inequality.
- In SAT being contradictory means a combination of p and $\neg p$ for some variable p.
- If literals consist of linear inequalities, being contradictory coincides with being unfeasible.
- So the CDCL approach can be extended to CNFs on linear inequalities, where for applicability of the rules the Simplex method is applied.

Consider the CNF of the following three clauses

$$
\begin{array}{ll}
(1) & x \geq y+1 \vee z \geq y+1 \\
(2) & y \geq z \\
(3) & z \geq x+1
\end{array}
$$

$(y \geq z)$	UnitPropagate on (2)
$(y \geq z)\ (x \geq y+1)$	UnitPropagate on (1)
	since $y \geq z \wedge z \geq y+1$ is unfeasible by Simplex
	FAIL on (3), since $y \geq z \wedge x \geq y+1 \wedge z \geq x+1$
	is unfeasible by Simplex

This proves that the given CNF is unsatisfiable